IEEE TRANSACTIONS ON
SIGNAL PROCESSING
A PUBLICATION OF THE IEEE SIGNAL PROCESSING SOCIETY
www.signalprocessingsociety.org
Indexed in PubMed and MEDLINE, products of the United States National Library of Medicine

Federated Learning with Spiking Neural Networks

Yeshwanth Venkatesha, Youngeun Kim, Leandros Tassiulas, and Priyadarshini Panda
Department of Electrical Engineering, Yale University, USA

Abstract*—As neural networks get widespread adoption in resource-constrained embedded devices, there is a growing need for low-power neural systems. Spiking Neural Networks (SNNs) are emerging to be an energy-efficient alternative to the traditional Artificial Neural Networks (ANNs) which are known to be computationally intensive. From an application perspective, as federated learning involves multiple energy-constrained devices, there is a huge scope to leverage energy efficiency provided by SNNs. Despite its importance, there has been little attention on training SNNs on a large-scale distributed system like federated learning. In this paper, we bring SNNs to a more realistic federated learning scenario. Specifically, we design a federated learning method for training decentralized and privacy preserving SNNs. To validate the proposed method, we experimentally evaluate the advantages of SNNs on various aspects of federated learning with CIFAR10 and CIFAR100 benchmarks. We observe that SNNs outperform ANNs in terms of overall accuracy by over 15% when the data is distributed across a large number of clients in the federation while providing up to $4.3\times$ energy efficiency. In addition to efficiency, we also analyze the sensitivity of the proposed federated SNN framework to data distribution among the clients, stragglers, and gradient noise and perform a comprehensive comparison with ANNs.

***Index Terms*—Neuromorphic Computing, Federated Learning, Spiking Neural Network, Energy-efficient Deep Learning, Event-based Processing**

I. Introduction

NEUROMORPHIC learning is being explored as a resilient low-power alternative to conventional Deep Learning in the perspective of both algorithm and hardware [1], [2]. With the release of neuromorphic chips like IBM TrueNorth [3] and Intel Loihi [4], we are not very far from neuromorphic processors becoming a part of everyday life. A class of Neuromorphic Learning algorithms called Spiking Neural Networks (SNNs) have gained widespread attention for their ability to achieve comparable performance to that of deep neural networks at considerably less computation cost and extreme energy efficiency [5], [6]. SNNs, analogous to the electrical activity in the human brain, utilize the discrete spike mechanism with Integrate-and-Fire (IF) or Leaky-Integrate-and-Fire (LIF) neuron units to transmit information.

Considering their energy efficiency, SNNs have a huge potential to be deployed in applications on embedded devices. However, these models need to be continuously trained and updated according to the new data collected at these devices. Conventional cloud-based machine learning collects data from the devices, transfers it to a central location, and trains the model offline. However, owing to increasing privacy concerns, it is unwise to transfer data out of the device. Modern collaborative on-device learning methods such as Federated Learning offers a solution for this. Federated Learning [7]–[12] is used for private and data-secure distributed learning among mobile devices (or clients) and a cloud server. This enables model training without the transfer of data from the client [7] to the cloud. This is particularly helpful when the data is sensitive and privacy is of paramount importance. Federated learning has been deployed in day-to-day applications such as Google Gboard [9] and is likely to be deployed in multiple practical applications in healthcare, retail, finance, wireless communication, etc [10], [11]. It provides the advantage of continuously learning and updating the model as and when the new data is available which is more practical compared to offline training. It can also enable user personalization as we train models locally on user data [13].

A major distinguishing factor of federated learning over traditional distributed training lies in the properties of the data distribution in the clients. Fig. 1 summarizes the difference between conventional machine learning, distributed learning and federated learning. In conventional machine learning, the data is collected into a single location and a model is trained offline on a single machine. However, as modern systems deal with enormous amount of data it is not always feasible to train a model on a single machine. Hence, distributed learning aims to accelerate the training process by parallelizing the training across multiple workers by distributing the training dataset. Here, since all the data is available at one location (i.e. cloud server), it can be Independent and Identically Distributed (IID) among the workers. This results in each worker having nearly the same amount of data and equal distribution of all the classes resulting in a stable learning method and faster convergence. On the other hand, in federated learning, the clients are not allowed to share their local data to the cloud server. The clients can only share their local model updates with the server. Furthermore, owing to the heterogeneity in the client devices, the IID assumption is not valid in the case of federated learning. Handling the non-IID data among the clients is one of the key aspects of federated learning that is widely studied [14]–[16]. This problem is further amplified as the number of clients increases.

Another important factor to consider while designing a federated learning system is the impact of stragglers [17], [18]. As the devices are not controlled by the central server and typically the updates happen over an unreliable network, it is not guaranteed that all the clients will send their updates to the server within an acceptable time limit. Hence, the server might end up with fewer updates than expected. Further, recent studies have shown that some of the data can be recovered from the model updates [19], [20]. To counter this data leakage, there

Yeshwanth Venkatesha, Youngeun Kim, Leandros Tassiulas, and Priyadarshini Panda are with the Department of Electrical Engineering, Yale University, New Haven, CT, USA (Corresponding author: Yeshwanth Venkatesha, e-mail: yeshwanth.venkatesha@yale.edu).

Digital Object Identifier 10.1109/TSP.2021.3121632

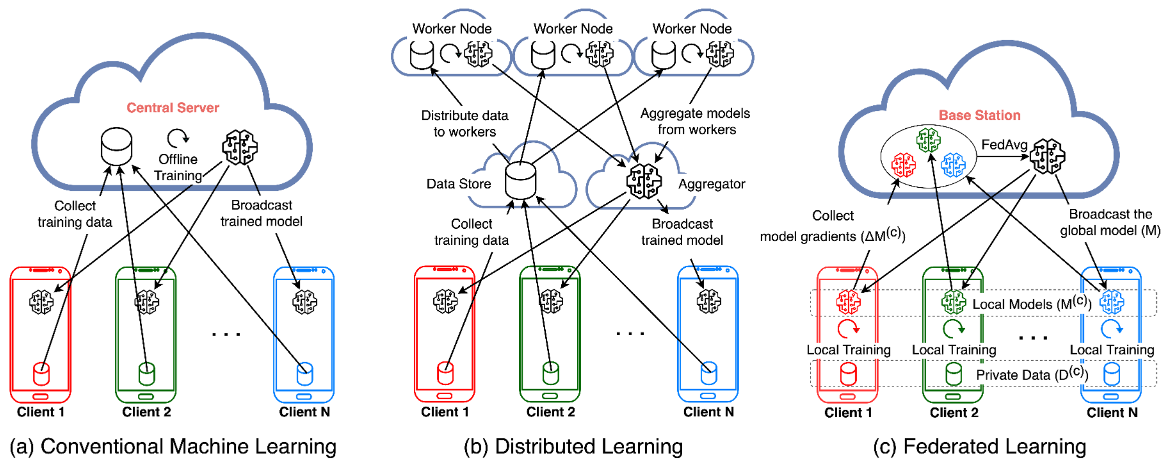


Fig. 1: Schematic diagram to illustrate distributed learning and federated learning. (a) Conventional Learning — the data is collected from the clients and the model is trained in an offline manner. (b) Distributed Learning — the dataset is collected at a single data store and then it is distributed across multiple worker nodes for training. (c) Federated Learning — In this case, the training is performed at the client, and model gradients are communicated to the base station which aggregates the gradients and updates the global model, and broadcasts it back to all the clients. Each of these processes is iterative and models at the clients are periodically updated.

is a body of work involving differential privacy [21] which obfuscates the gradients of the model with noise [22]. Hence, the training algorithm for federated learning must be resilient to noise in the model updates communicated to the server.

A majority of existing SNN literature examines SNNs only through the lens of energy efficiency and is limited to standard machine learning settings where the entire dataset is available in one place. There has been little attention on utilizing SNNs in more practical systems beyond classic image recognition task. A potential reason for this limitation is the lack of robust training algorithms for SNNs. Traditionally, the non-linear thresholding functionality in LIF or IF spiking neurons is known to prohibit the conventional backpropagation-based training methods for SNNs. Hence, the training of SNNs was limited to unsupervised learning methods like Spike Timing Dependent Plasticity [23] and conversion of fully trained ANN to SNN [24], [25]. However, recent findings illustrate methods to reliably train SNNs using an approximate gradient backpropagation method. This method unrolls the network and performs a Backpropagation Through Time (BPTT) similar to that of recurrent neural networks [5], [26]–[29]. In addition to this, a recent work uses batch normalization in the time dimension—Batch Normalization Through Time (BNTT) in order to achieve high performing SNNs by training from scratch [30]. Such gradient-based training methods enable SNNs to be used in distributed training methods such as federated Learning. Further, recent studies suggest that SNNs are robust to adversarial attacks [31], [32]. As security is a major concern in federated learning applications, we believe SNNs can be a viable alternative to ANNs [33].

In contrast to previous works which are limited to extracting computation and energy efficiency with the spiking model, we look at SNNs from the perspective of robust distributed training. We propose a method to train SNNs in a federated learning paradigm and conduct a series of structured experiments to study the robustness of SNNs trained in a federated manner with respect to the different aspects of federated learning. Previously, the authors of [34] propose an on-device training method for SNNs in a federated setting using a probabilistic SNN model. In contrast to [34], we use a mainstream surrogate gradient based backpropagation approach, called BNTT, that has been shown to train SNNs from scratch in a faster manner with lower latency and scales to larger networks and complex datasets. Moreover, since the reported experiments in [34] were performed on the MNIST-DVS dataset with only 2 clients, it may not be straightforward to extend it to mainstream computer vision problems with large number of clients. To the best of our knowledge, our work is one of the first to train SNNs with federated learning on complex image datasets and perform a comprehensive analysis of robustness, performance, and energy efficiency.

Contributions: In this paper, we design a federated learning method to train low-power SNNs in a distributed and privacy preserving manner. We experimentally evaluate the feasibility of federated learning with SNNs for standard image classification tasks with CIFAR10 and CIFAR100 datasets. We highlight that, as the number of devices in the federation increases, SNNs outperform ANNs by $> 15\%$ on CIFAR10 and $> 25\%$ on CIFAR100 with VGG9 model. We study the effect of skewness in the class distribution among the clients on the performance of the system. We evaluate and compare the performance of SNNs to ANN counterparts with respect to robustness to stragglers and robustness to gradient noise and finally, provide estimated energy efficiency of SNNs compared to ANNs.

II. Background

In this section, we provide a brief background on federated learning, spiking neural networks and the batch normalization technique for SNNs, called Batch Normalization Through Time (BNTT).

A. Federated Learning

A typical federated learning system consists of a set of N clients and a base station. The base station broadcasts the initial model M_0 to all the clients to start the training process. The initial model can be a model trained offline on a public dataset. Each of the clients $c = 1, 2, ..., N$, maintain their own private dataset $D^{(c)}$ and locally train a model by iterating over the local dataset $D^{(c)}$ to obtain a locally trained model at client c (say $M^{(c)}$). The size of the data retained in the client depends on the storage and computation capacity of the device. The model update from each client is the accumulated gradients over the course of local training. These updates are periodically communicated to the base station for aggregation. Each iteration of communication of the model updates from clients to the server is defined as a federated learning *round*. At *round* r, the gradients $\Delta M_r^{(c)}$ from a subset of participating clients $P_r(\ll N)$ is sent to the base station. The participating clients can be selected at random or based on a defined scheduling algorithm according to the needs of the specific application. The base station performs a weighted average of the gradients to obtain the updated global model as shown in Eqn. 1.

$$M_{r+1} = M_r + \frac{1}{\sum_{c \epsilon P_r} |D_r^{(c)}|} \sum_{c \epsilon P_r} |D_r^{(c)}| \Delta M_r^{(c)}, \tag{1}$$

where $|D_r^{(c)}|$ denotes the number of data samples used for local training in client c at round r. This aggregation of the model updates from the clients is described as $FedAvg$ algorithm in [7]. There have been multiple enhancements to the $FedAvg$ algorithm such as $FedProx$, that uses a regularization term to stabilize federated training [35] and $FedMA$, which performs a layer-wise matching and averaging of the models to handle heterogeneous data [36]. For simplicity, in this article, we shall be using only $FedAvg$. The process of federated learning is illustrated in Fig. 1.

B. Spiking Neural Networks

A spiking neuron is modeled by an Integrate-and-Fire (IF) mechanism — each neuron keeps track of its membrane potential by accumulating the incoming spikes scaled by their corresponding synaptic weights and generates a spike when its membrane potential reaches a certain threshold. In the Leaky-Integrate-and-Fire (LIF) variant, the membrane potential also leaks at a constant rate. As illustrated in Fig. 2, for a given neuron i with a set of input neurons N, the incoming spikes from the input neurons are weighted by the parameters w_{ij} for all $j \in N$ and accumulated as the membrane potential of the neuron. Once the membrane potential reaches a certain threshold v, it generates an output spike. Following a spike, the membrane potential is reset to the resting potential u_{rest} or in case of a soft reset, it is reduced by the threshold value. This process is repeated for T timesteps. In discrete form, the LIF mechanism is modeled as follows [29], [37]:

$$\begin{gathered} u_i^t = \lambda u_i^{t-1} + \sum_{j \in N} w_{ij} o_j^{t-1}, \\ \text{where} \quad \lambda < 1 \quad \text{and} \quad o_i^{t-1} = \begin{cases} 1 & \text{if } u_i^{t-1} > v \\ 0 & \text{otherwise} \end{cases}. \end{gathered} \tag{2}$$

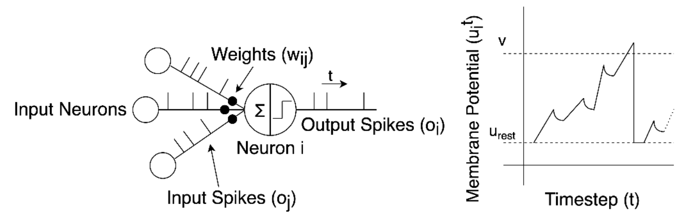


Fig. 2: The illustration of leaky-integrate-and-fire behavior of a spiking neuron.

Here, o_i^t is the binary output that takes the value 1 if the neuron i produces a spike at timestep t and 0 otherwise, u_i^t represents the membrane potential of neuron i at timestep t, λ is a constant leak factor by which membrane potential reduces every timestep, v is the threshold, and N specifies the set of input neurons connected to neuron i with w specifying their corresponding weights.

The operation of a spiking neuron can be unrolled along the time dimension similar to a recurrent neural network (RNN) [26]. Hence, Δw_{ij}, the gradient of the weight connecting neuron i and j is accumulated over T timesteps and can be calculated as follows:

$$\Delta w_{ij} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial o_i^t} \frac{\partial o_i^t}{\partial u_i^t} \frac{\partial u_i^t}{\partial w_{ij}}, \tag{3}$$

where, $\mathcal{L}$ is the loss function being optimized. In case of image classification, categorical cross entropy loss is widely used. However, as o_i^t is a thresholding function, it's derivative is a Dirac-Delta function that is not defined at the time of spike and zero everywhere else. Thus, computing Δw_{ij} is intractable. To overcome this, the derivative of threshold function is approximated with several surrogate functions such as piece-wise linear function or the exponential function [26], [38]. The surrogate gradient descent method using a piece-wise linear approximation is defined as:

$$\frac{\partial o_i^t}{\partial u_i^t} = \xi \max\{0, 1 - |\frac{u_i^t - v}{v}|\}, \tag{4}$$

where, ξ is a decay factor for back-propagated gradients and v is the threshold value. The hyperparameter ξ should be set based on the total number of timesteps T. As gradients are accumulated at every time step, it is recommended to use a smaller ξ for large T to avoid exploding gradient problem.

C. Training Spiking Neural Networks with BNTT

The authors of [30] propose a method to leverage batch normalization in temporal dimension to improve the training performance of SNNs. With this method, it is possible to train low latency and high-performing SNNs from scratch without the need for a fully trained ANN model. This is achieved by associating a local learning parameter to each time-step and thereby expanding the batch norm layer through time. During the forward propagation, the BNTT layer is applied after each convolutional/linear layer as:

$$\begin{aligned} u_i^t =& \lambda u_i^{t-1} + \text{BNTT}_{\gamma_i^t}(\sum_{j \epsilon N} w_{ij} o_j^t) \\ =& \lambda u_i^{t-1} + \gamma_i^t (\frac{\sum_{j \epsilon N} w_{ij} o_j^t - \mu_i^t}{\sqrt{(\sigma_i^t)^2 + \epsilon}}), \end{aligned} \tag{5}$$

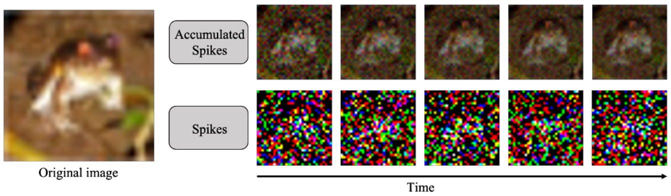


Fig. 3: Example of rate coding. As time goes on, the accumulated spikes represent similar image to original image. We use an image from the CIFAR10 dataset.

where μ_i^t and σ_i^t are the mean and variance from the samples in a mini-batch $\mathcal{B}$ at time step t. A global mean and variance is calculated by computing an exponential moving average over training iterations which are used to normalize the validation data at inference. The parameter γ_i is learnt using backpropagation and different γ_i^t values are used for each time-step t for efficient inference.

III. Federated SNN Training

Given a base station and a set of N clients with each client c having its own private local dataset $D^{(c)}$, we train a single SNN model without transferring the dataset out of the client. The base station initially broadcasts a randomly initialized SNN model to each of the N clients. As described in Algorithm 1, each client trains a local copy of the model $M^{(c)}$ in parallel using the BNTT training technique. The BNTT training process (described in procedure $BNTT\text{-}Training$ in Algorithm 1) starts by encoding the pixel values into spike trains of length T using Poisson rate coding. In Poisson coding, the number of spikes generated is matched with the corresponding pixel intensity. At each timestep, a random number is sampled between the minimum and maximum possible pixel intensity (I_{min}, I_{max}), and a spike is generated if this random number is less than the real pixel intensity. Hence, the spike generated at each timestep corresponding to a pixel is stochastic while the total number of spikes generated over the timesteps is proportional to the pixel intensity. In Fig. 3, we illustrate the process of spike encoding using poisson generator. Note that, if we accumulate spikes over time, we get an image similar to the original.

For a given timestep $t = 1, 2, ..., T$, for each layer $l = 1, 2, ..., L-1$ with weight tensor W_l, BNTT parameter γ_l and threshold v_l, we perform forward propagation to update the set of membrane potentials U_l^t and when the membrane potentials exceed the layer threshold v_l, output spikes O_l^t are generated:

$$U_l^t = \lambda U_l^{t-1} + \text{BNTT}_{\gamma_l^t}(W_l, O_{l-1}^{t-1}),$$
$$\text{where} \quad \lambda < 1 \quad \text{and} \quad O_l^t = \begin{cases} 1 & \text{if } U_l^{t-1} > v_l, \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Note that we use upper case notation to denote vectors and lower case to denote scalar values. Here, U_l^t and O_l^t is the set of membrane potentials and output spikes respectively of all the neurons of layer l at timestep t. W_l is the set of weights w_{ij} connecting neurons of layer $l-1$ to neurons of layer l. λ is the constant leak factor of membrane potential. The membrane potential at the last layer is accumulated without the leak factor

Algorithm 1: Federated SNN Training Method

Input : Set of N clients with Local Datasets $D^{(c)} \forall c = 1, 2, ..., N$
Output : Trained Model M
Hyperparams : Number of rounds (R), Number of local epochs (E), Number of timesteps (T)

```
Algorithm Fed-SNN()
    Initialize Model M with random weights
    for round r ← 0 to R do
        Broadcast the current model M to all clients
        for Client c ← 0 to N (in parallel) do
            for local_epoch ← 0 to K do
                perform BNTT-Training() on M_r^(c).
            end
        end
        Randomly select P participating devices
        Aggregate the gradients (Eqn. 1).
    end
return
Procedure BNTT-Training()
    foreach mini-batch B do
        for time step t ← 0 to T do
            O ← PoissonGenerator(B)
            for Layer l ← 1 to L − 1 do
                Forward propagate (Eqn. 6).
            end
            Accumulate membrane potential at the last layer.
        end
        Compute Loss (Eqn. 7) and Backpropagate the model gradient δM^(c).
    end
return
```

(i.e. $\lambda = 1$ in Eqn. 6) and the threshold functionality is not used. This makes the output values continuous which are then passed through a softmax layer to get the model predictions.

The model gradient $\delta M^{(c)}$ at client c is calculated backpropagating the loss $\mathcal{L}$ through the unrolled network by iterating over the local dataset $D^{(c)}$ according to Eqn. 3 and 4. For the case of image classification, the loss $\mathcal{L}$ is defined by categorical cross-entropy function between the accumulated membrane potentials of the last layer U_L^T and the ground truth labels Y as follows:

$$\mathcal{L}(U_L^T, Y) = -log(\frac{exp(U_L^T[Y])}{\sum_j exp(U_L^T[j])}), \tag{7}$$

The BNTT training process updates each of the local client model $M_e^{(c)}$ at epoch e as:

$$M_e^{(c)} = M_{e-1}^{(c)} - \eta_e^{(c)} \delta M_e^{(c)}, \tag{8}$$

where $\eta_e^{(c)}$ is the local learning rate and $\delta M_e^{(c)}$ is the model gradients of client c respectively at epoch e. A full pass through the local dataset constitutes a *local epoch* and the gradients accumulated over K local epochs forms the aggregated model

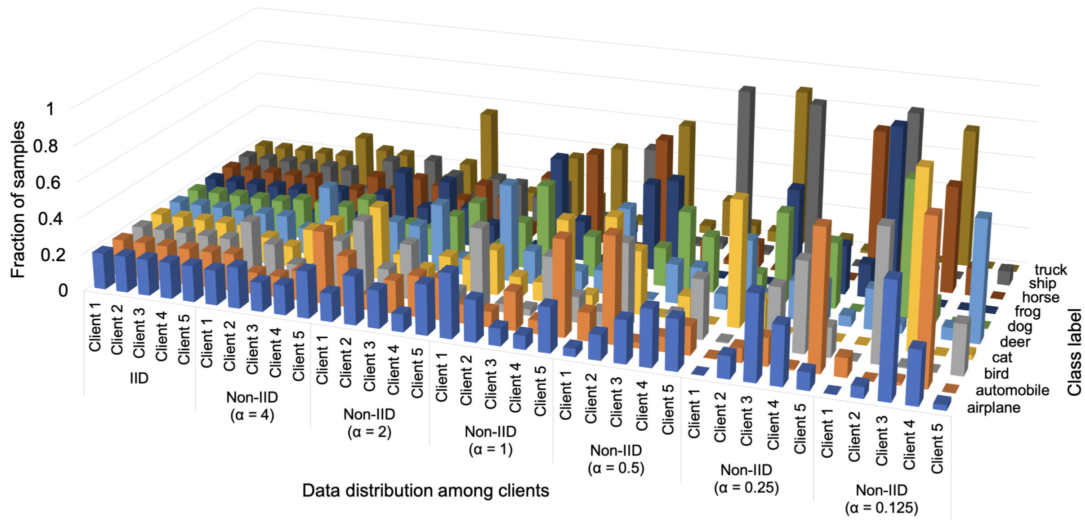


Fig. 4: Distribution of CIFAR10 dataset among 5 clients for different degrees of non-IID. In the case of IID, the 10 classes are equally distributed among all the clients with each client getting 1/5 of the samples of each class. As we increase the parameter alpha to the Dirichlet Distribution, the class composition at each client becomes more varied. The extreme case with $\alpha = 0.125$ results in no examples of certain classes in some clients.

update $\Delta M^{(c)} = \sum_{e=1}^{K} \eta^{(c)} \delta M_e^{(c)}$. The number of *local epochs* K is set depending on the volume of data and the available compute resources on each client. After every K *local epochs*, a randomly chosen subset of P clients will send their aggregated model updates $\Delta M^{(i)}$ to the base station which in turn updates the global model according to the FedAvg Algorithm (Eqn. 1). This cycle of broadcast, local model update, gradient communication, and global model update constitutes one *round* of federated learning. This process is repeated for R *rounds* to train the model. The process is detailed in Algorithm 1. In production systems, this is a lifelong process to keep the model up to date with the latest data distribution. This is extremely helpful as many modern applications rely on quickly adapting to the latest trends. Note that instead of sending the model updates $\Delta M^{(c)}$ to the base station, the clients can also send the full model $M^{(c)}$. The base station can, then, perform an equivalent computation and get the same updated model. However, transferring gradients can result in efficient communication if a relevant gradient compression techniques are used [39].

IV. Experiments

To evaluate our method, we perform structured experiments with VGG9 network [40] on CIFAR10 and CIFAR100 benchmarks [41]. We use the standard training and validation datasets provided in the original dataset with 50000 32x32 RGB images of training data and 10000 images of validation data. We use two strategies to distribute the training dataset among N clients: (i) IID − each client has an equal number of samples and nearly the same proportion of all classes. (ii) non-IID − sample the proportion of each class using a Dirichlet distribution with concentration parameter $\alpha = 0.5$ to obtain non-identical datasets similar to previous federated learning works [36], [42].

In Fig. 4, we illustrate the difference between IID and non-IID and describe how we control the α parameter in Dirichlet distribution to generate different extent of non-IID distribution. Although non-IID is more realistic, we use the IID strategy to establish a baseline to compare against. For simplicity, we assume the local datasets do not change between rounds, i.e. $D_r^{(i)} = D^{(i)} \; \forall r$. The validation set is held out and is used to evaluate all the models. We use the validation accuracy on this held-out dataset as the performance metric throughout our experiments. All the reported validation accuracy values are calculated by averaging across three repetitions of the experiment. We use a batch size of 32 across all the experiments. We train locally on each client for 5 epochs which is termed as *local epochs* and train the global model for 40 *rounds*. We consider different number of total and participating clients to study how the algorithm scales with the number of devices. We use the convention of N/P to specify the client split where N denotes the total number of clients and P denotes the number of participating devices chosen uniformly at random in each round.

We use VGG9 as the base model for our experiments. VGG9 consists of 7 convolutional layers and two fully connected layers with pooling layers to reduce the dimension. Traditionally max pooling is used in ANN literature and average pooling is more common in the SNN literature. To make the comparison simple, we use average pooling for both ANN and SNN. We use the same architecture for SNN with 25 timesteps across all our experiments. We use SGD optimizer with an initial learning rate of 0.001 and weight decay of 5e-4 for ANN while we use SGD with an initial learning rate of 0.1 and momentum of 0.95 for SNN. The learning rate is reduced by a factor of 10 after 20 and 30 rounds. Note that in theory each client can have a different set of local training hyperparameters such as number of local epochs, choice of optimizer, local learning rate, momentum and weight decay. However, in our experiments we are using the same hyperparameters across all clients. Note that while state of the art federated learning methods achieve

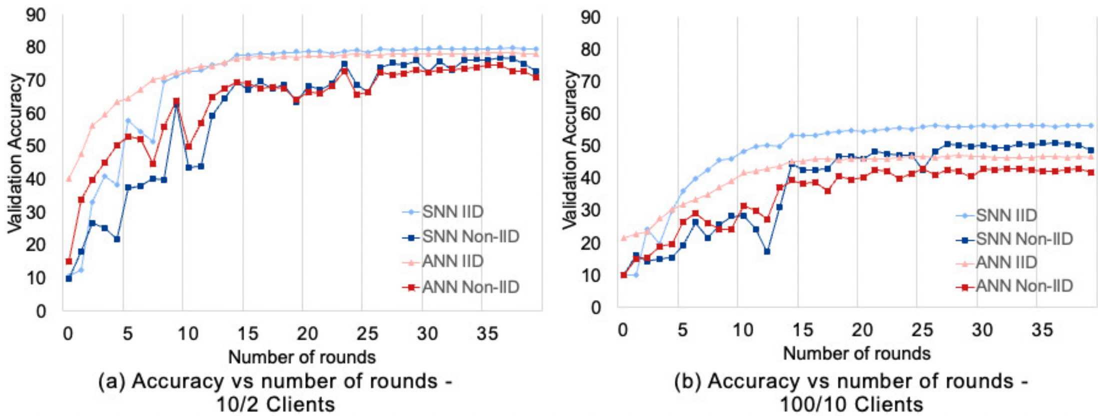


Fig. 5: (a) Training progress of the VGG9 model trained on CIFAR10 dataset for 10 clients with 2 clients participating in each round. (b) Training progress of the VGG9 model trained on CIFAR10 dataset for 100 clients with 10 clients participating in each round. We observe a similar trend in convergence for both SNNs and ANNs on IID and non-IID data implying that with federated SNN training with our method is equally robust in handling non-IID data as ANNs.

higher accuracy on CIFAR10 in specific federated learning environments [7], [43], in our setup we focus on objective evaluation of our proposed method rather than on achieving the best possible accuracy on the given task at every scenario. Hence, we fix the local learning hyperparameters such as the number of local epochs, choice of optimizer, local learning rate, weight decay and momentum based on the best choice for 1/1 client case which is equivalent to centralized learning and use the same parameters across all our experiments. We vary only the hyperparameters pertaining to federated learning such as the number of clients, degree of skewness in data, probability of stragglers, and noise in the gradients in our experiments. This helps in isolating the impact of each parameter and facilitates a fair comparison between ANNs and SNNs. We believe better regularization methods for federated learning [15], [35], [43]–[45] and custom hyperparameters can improve the accuracy for specific cases.

In Fig. 5, we plot the training curve for two instances of federated learning. Fig. 5(a) describes the training progress with 10 total clients and 2 randomly chosen clients participating in each round. In Fig. 5(b) we plot the training progress when the number of total and participating clients is increased to 100 and 10 respectively. We observe a similar trend in training both ANNs and SNNs in case of both IID and non-IID validating our approach can handle non-IID data with SNNs. While the final models reach similar accuracy in both ANNs and SNNs when the number of clients is low (10/2), there is a significant difference in final performance of SNNs over ANNs when the number of clients is high (100/10 clients).

In subsequent sections, we study in detail, the performance of our method on various facets of federated learning. In Section IV-A, we evaluate the performance of the model as the number of clients increases. In Section IV-B, we study the effect of the number of participating clients in each round. In Section IV-C, we expand on the details of non-IID and study the impact of skewness in the data distribution on the performance of the model. In Section IV-D and Section IV-E we investigate if there is a significant drop in accuracy with respect to having stragglers and noise in gradients respectively. Finally, in Section IV-G, we estimate the energy consumption of SNNs and compare it to that of ANNs.

A. Scalability

Since real-world federated systems often involve a large number of devices, a federated learning model must be scalable with the number of devices. To evaluate the scalability of our method, we perform a series of experiments by steadily increasing the number of total and participating clients in the federated learning process (Fig. 6 and Table I). We report the performance on several combinations of clients for both CIFAR10 and CIFAR100 datasets in Table I and provide plots in Fig. 6 to visualize the trend. While the ANN achieves superior accuracy in the baseline case of 1/1 (N/P) configuration, the accuracy drops sharply as the number of clients increases. On the other hand, the performance degradation is not as steep in the case of SNN. SNNs surpass the performance of ANNs when the total number of clients is increased to 100. This phenomenon is more pronounced in the case of CIFAR100 (shown in Fig. 6b) where the ANN model accuracy goes below 20% for 100/10 clients which is practically unusable. Therefore, as the number of clients increase, we observe that SNNs outperform the corresponding ANNs. To explain this, we analyze the trend of Relative Model Deviation which quantifies the amount by which the client model deviates from the server model in each communication round. Formally, we define the Relative Model Deviation (RMD) of client c at round r as:

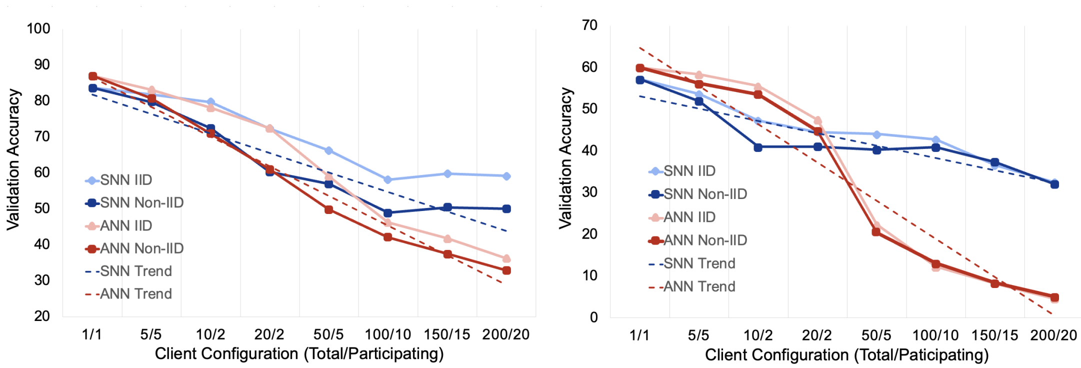
$$RMD_r^{(c)} = \frac{||M_r^{(c)} - M_{r-1}||}{||M_{r-1}||} \tag{9}$$

Here, $M_r^{(c)}$ is the final model at client c at round r and M_{r-1} is the model broadcasted from the server before the start of round r. $||\cdot||$ denotes the Frobenius norm.

We identify the cases with 10/2 clients and 100/10 clients as representatives for low clients and high clients respectively and observe the trend of RMD of one of the clients as the training progresses. Since the magnitude of weights of ANNs and SNNs will be different, we normalize the RMD to enable better comparison. As shown in Fig. 7, when the number of

TABLE I: Final validation accuracy reached by the models after 40 rounds for different numbers of total and participating clients on CIFAR10 and CIFAR100. The key observation to note is that the performance deterioration is steeper in the case of ANN compared to that of SNN as N/P increases.

Clients (N/P)	CIFAR10				CIFAR100			
	ANN		SNN		ANN		SNN	
	IID	*non-IID*	*IID*	*non-IID*	*IID*	*non-IID*	*IID*	*non-IID*
1/1	87.09	87.09	83.74	83.74	59.98	59.98	57.17	57.17
5/5	83.27	80.85	81.93	79.82	58.39	56.16	48.54	43.49
10/2	78.23	70.98	79.80	72.48	55.56	53.55	47.25	41.00
20/2	72.49	61.06	72.40	60.36	47.47	44.80	44.56	41.08
50/5	59.06	49.88	66.30	56.99	22.34	20.41	44.08	40.32
100/10	46.28	42.19	58.14	48.97	12.29	13.12	42.79	40.91
150/15	41.86	37.56	59.86	50.51	8.25	8.39	36.61	37.35
200/20	36.34	32.90	59.26	50.15	4.61	5.08	32.52	32.13



(a) Accuracy vs Client Configuration - CIFAR10 (b) Accuracy vs Client Configuration - CIFAR100

Fig. 6: Performance trend of ANN vs SNN for VGG9 model trained on CIFAR10 and CIFAR100 as we increase the number of clients. We observe a similar trend on both the datasets where ANNs perform better than SNNs when the number of clients is low but as the number of clients is scaled up, the performance of ANNs plummet.

clients is low i.e., in the case with 10/2 clients, we observe a continuously decreasing trend in RMD for both SNNs and ANNs. However, when the number of clients is high (in the case of 100/10 client combination), we observe an increasing trend in RMD of ANNs whereas the trend is downward in case of SNNs. This implies that the federated averaging diverges in ANNs when the number of clients is high while SNNs perform a stable averaging despite having a large number of clients.

B. Impact of Number of Participating Clients

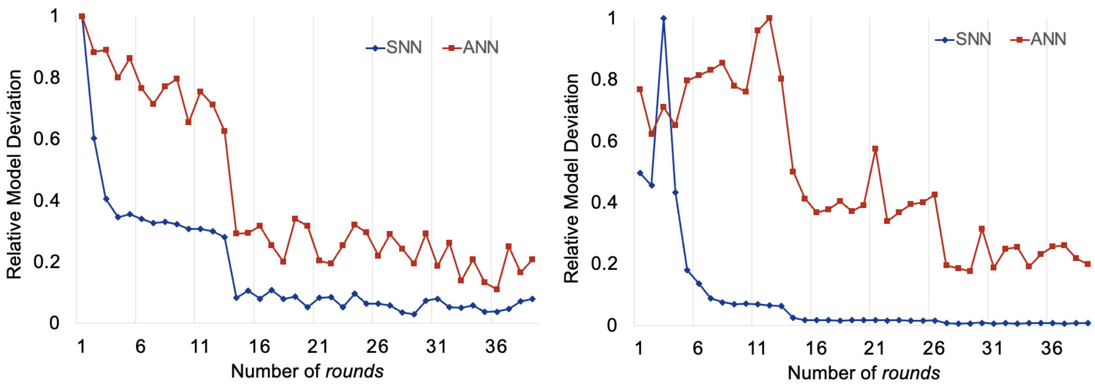
While total number of clients is one aspect of scalability, the fraction of clients participating in each round also plays an important role in the performance of a federated learning system. Especially in the case of non-IID, it is desirable to have a model that can generalize well with less number of clients participating in each round. To study the impact of number of participating clients, we use the case with CIFAR10 trained with 100 clients and progressively decrease the number of participating clients from 100 to 1. We observe, in Fig. 8a, that there is a gradual performance drop as the number of participating clients decrease in case of non-IID in both ANNs and SNNs. In case of IID, since all the clients contain similar data distribution, both ANNs and SNNs are preserving the performance as the number participating clients reduce. This implies that SNNs are robust in generalizing the model using updates from only a fraction of the total clients.

C. Sensitivity to Data Distribution

In addition to scalability with the number of devices, the nature of data distribution across the devices is another factor that impacts the performance of a federated learning system. In this section, we study how SNNs perform as the distribution of classes among the clients is varied. As the non-IID data distribution is synthetically generated by sampling from Dirichlet distribution, we can control the skewness by varying the parameter α of the Dirichlet distribution [36], [42]. Fig. 4 illustrates how the fraction of samples of different classes are distributed among different clients. As the value of α decreases, the class composition gets more skewed. This skewness will be further amplified as the number of clients increase. In Fig. 8b, we consider the models trained on CIFAR10 with 100/10 clients at different extent of non-IID. We start with $\alpha = 4$ and progressively reduce it by the factor of 2 as long as the model does not diverge and observe the performance of the system. We note that there is a steady decline in the performance of both ANNs and SNNs as the data becomes more non-IID. This implies that federated training of SNNs with our method can produce equally robust models with non-IID data as that of ANNs.

D. Sensitivity to Stragglers

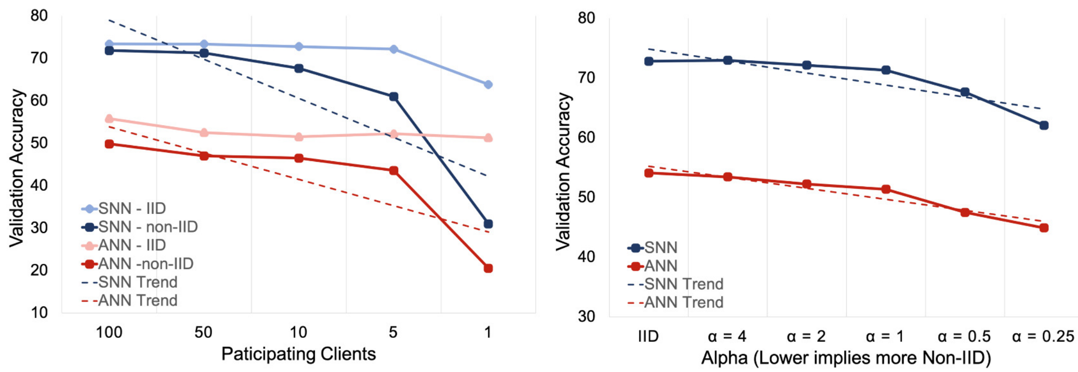
Since real-world federated learning applications involve training with millions of devices, it is impractical to assume the communication of gradients will be successful from all



(a) Relative Model Deviation vs Number of *rounds* - 10/2 Clients

(b) Relative Model Deviation vs Number of *rounds* - 100/10 Clients

Fig. 7: Trend of relative model deviation for the cases with 10/2 clients and 100/10 clients.



(a) Impact of Number of Participating Clients - CIFAR10 (100 Clients)

(b) Sensitivity to Non-IID Data - CIFAR10 (100/10)

Fig. 8: (a) Impact of number of clients participating in each round. For the case of CIFAR10 with 100 clients, we decrease the total participating devices from 100 to 1 and study the performance. We observe a similar trend with both ANNs and SNNs. (b) Effect of having more skewed class distribution among the clients. We increase the non-IID nature of the data distribution and observe the validation accuracy of the final model. We use the case with CIFAR10 divided among 100 clients and 10 clients participating in each round. We observe a similar trend of decreasing performance as the distribution becomes more non-IID in case of both ANNs and SNNs.

the selected devices. Hence, the model needs to be robust to handle devices failing to communicate the gradients. These devices are referred to as stragglers [17], [18]. In this section, we analyze the impact of stragglers on the performance of the final SNN model. We use the case with 100 total clients and 10 participating clients for a VGG9 model trained on CIFAR10 data. We evaluate the model performance by varying the probability of a device failing to communicate the model updates to the server. To consider a *round* to be successful, at least one out of P participating devices has to communicate its update to the base station. Hence, one out of P participating clients is guaranteed to send the update, while the rest of the clients drop out with a probability value. We consider different levels of probabilities and plot the trend in Fig. 9a. Since the updates are similar across the clients in the case of IID, the impact of the missing updates is minimal resulting in a nearly flat curve. However, the impact of stragglers is prominent in the case of non-IID data for both ANN and SNN. We observe a similar trend between ANN and SNN up to a probability value of 0.5. The steep drop in performance of SNNs in the extreme case of stragglers (for probabilities > 0.5) suggests that SNNs are more sensitive to stragglers when the data distribution is non-IID. However, it is worth noting that SNN is still performing better than the corresponding ANN in presence of stragglers. Hence, we conclude SNNs are robust in handling stragglers to a considerable extent on par with the corresponding ANNs on CIFAR10 dataset.

E. Sensitivity to Noise in Gradients

Straightforward federated learning is not enough to preserve data privacy as there are methods by which some of the data and its attributes can be recovered from the gradients communicated between the clients and the server. Hence, the gradients are often obfuscated with added noise before sending to the base station. In this section, we evaluate the performance of the federated model with respect to the added noise in the gradients. As in the previous section, we use the case with 100 total clients and 10 participating clients for a model trained on CIFAR10 data. We add gaussian noise $N(0,1)$ multiplied by noise strength. The noise strength is increased up to 4 implying

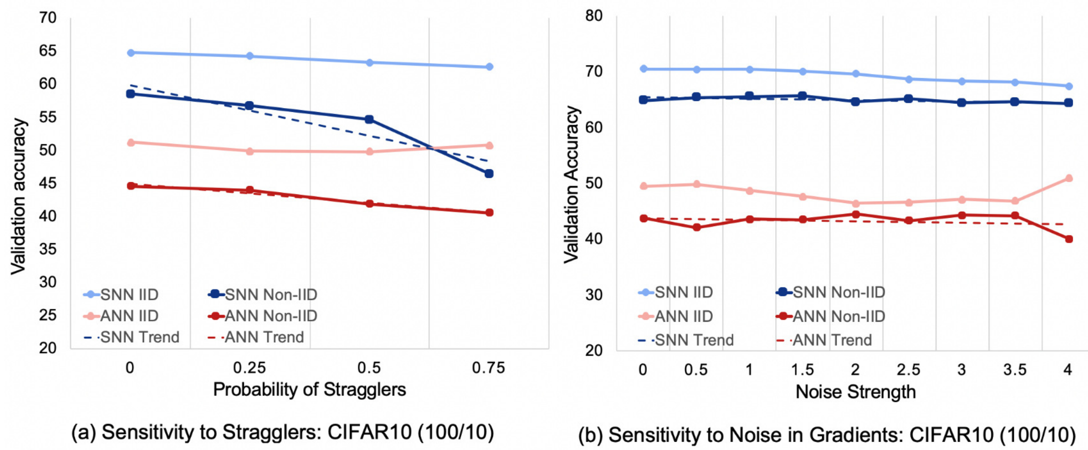


Fig. 9: (a) Impact of stragglers on the performance. We observe training SNNs with our method is robust to stragglers. There is a noticeable drop in performance in extreme cases. However, 75% straggler probability is extreme. (b) Impact of noise in the gradients. Both SNNs and ANNs show equal robustness with added noise to the gradients.

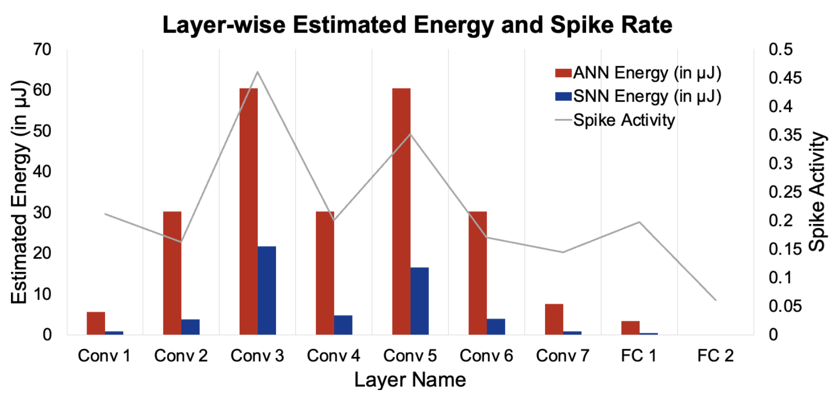


Fig. 10: Estimated inference energy across layers for SNN and ANN VGG9 model trained on CIFAR10 dataset. We see that SNNs are considerably more energy efficient compared to ANNs.

an added noise with a mean of 0 and standard deviation of 4. The observation is captured in Fig. 9b. For the IID case, in both SNN and ANN, we observe a slight impact of noise as the performance is gradually decreasing. On the contrary, there is no visible impact on the performance in the case of non-IID resulting in a nearly flat line. This can be explained by the fact that gradient updates from non-IID is inherently diverse. Hence, adding noise to the gradients has minimal effect on the performance implying that differential privacy techniques [22] can be seamlessly applied to SNNs to make the system more secure.

F. Impact of number of timesteps

The number of timesteps in SNNs control the information processing in the temporal dimension. In this section we measure the impact of the number of timesteps on performance of the final model. Similar to previous experiments, we use the case with 100/10 clients with VGG9 model trained on CIFAR10 dataset and provide the baseline case of training the model on single client for comparison (Table II). We evaluate on three different values of timesteps 20, 25, and 30. While we observe a significant improvement in accuracy when the number of timesteps are increased from 20 to 25, the improvement is marginal when increased further to 30. Owing to increased stochasticity and additional gradient updates, more timesteps typically result in better performance in SNNs. At the same time, increasing the number of timesteps also increases the inference latency and energy consumption of the network. Hence, there is a natural tradeoff between latency and accuracy. Note that since the working mechanism of ANNs does not rely on temporal information, this section does not contain a comparison to ANNs.

G. Energy Estimation

We estimate the energy consumption for 32-bit integer arithmetic operations on traditional hardware based on a 45nm CMOS process [46] using energy calculations described in [47]. Note that, this is rather a rough estimate as we are considering only Multiply and Accumulate (MAC) operations and neglecting the energy consumed by softmax computation, memory access and any peripheral circuit energy. The energy for multiply and accumulate operations are summarized in Table III. For a $k \times k$ convolution layer with I input channels, O output channels operating on an input feature map size $N \times N$ and resulting in output feature map of size $M \times M$ number of operations (OPS) is given by: $OPS = M^2 \times I \times k^2 \times O$.

TABLE II: Impact of number of timesteps. We vary the number of timesteps in SNNs for 100/10 clients (IID and Non-IID) on CIFAR10 with VGG9 and observe the impact on final validation accuracy reached by the model after 40 communication rounds. We provide the baseline centralized learning case with single client for comparison. We observe an improvement in performance as the number of timesteps is increased. However, increasing the timesteps has an adverse impact on inference latency and energy consumption.

Timesteps	Baseline (1/1)	100/10 (IID)	100/10 (Non-IID)
20	82.38	56.73	46.14
25	83.74	59.42	49.12
30	84.53	60.43	51.20

TABLE III: Energy estimation for multiply and accumulate operations.

Operation	Estimated Energy (pJ)
32-bit Multiply (E_{Mult})	3.1
32-bit Add (E_{Add})	0.1
32-bit Multiply and Accumulate ($E_{MAC} = E_{Mult} + E_{Add}$)	3.2
32-bit Accumulate (E_{AC})	0.1

For a fully connected layer with I inputs giving O outputs, the number of operations (OPS) is given by $OPS = I \times O$. Since SNNs operate with binary spikes the MAC operation reduces to an accumulate (AC) operation which leads to significant energy efficiency. The energy consumption of ANN is straight-forward $E_{ANN} = OPS \times E_{MAC}$. We calculate the energy consumption of SNN by multiplying OPS with spiking rate (activity) R across total timesteps, T *i.e.*, $E_{SNN} = OPS \times R \times T \times E_{AC}$. In our experiments, we computed the average spiking rate by performing inference on the entire validation set. Fig. 10 shows the estimated energy for each layer of ANN and SNN with VGG9 model trained on the CIFAR10 dataset for 10/2 clients with non-IID data distribution. We also observe the spike rate of the corresponding layers for the final model. The total estimated energy by ANN is $\mathbf{227.99\mu J}$ while that of SNN is $\mathbf{53.24\mu J}$ which is $\mathbf{4.3\times}$ more efficient. The energy for ANN is constant for all the instances while that of SNN varies for each instance depending upon the spike activity and the number of timesteps.

V. Discussion

We perform a comprehensive study on the feasibility of training spiking neural networks in a federated learning paradigm. We show how to train SNNs from scratch in a federated setting using the BNTT method. We experimentally evaluate the performance and estimated energy of the VGG9 SNN trained on CIFAR10 and CIFAR100 datasets and compare them with corresponding ANNs. We study different facets of federated learning such as data distribution across the clients, stragglers, and gradient noise to find potential strengths and weaknesses of SNNs. We find that SNNs outperform ANNs when the data is distributed across a large number of clients and are equally robust to ANNs in handling non-IID data, stragglers, and noise in the gradients. We conclude that SNNs are a viable alternative to ANNs in large-scale federated learning with embedded devices, owing to their energy efficiency and superior performance when the number of clients is scaled up.

From the series of experiments we identify that while SNNs are equally robust to ANNs in all the facets of federated learning, the key advantage of SNNs is the ability to preserve the accuracy at scale. In our experiments, as the size of the dataset is fixed, dividing it among a large number of clients results in each client having fewer samples. Naturally, the updates from these clients will overfit to their local dataset. Hence, it is difficult to aggregate these local models into a robust global model. Therefore, we can infer that federated training of SNNs with our method is more sample efficient i.e. the local SNN model in each client can generalize well with fewer data samples as compared to ANNs. Consequently, the aggregated global model is preserving the performance. This is supported by the analysis on relative model deviation where we observe that the federated averaging in ANNs show divergence behaviour when the number of clients is high. We hypothesize that this phenomenon can be attributed to the fact that since SNNs contain more gradient updates due to the unrolling of the network in the time dimension, the gradients updates are more likely to become smoother providing a form of regularization to the local model. This regularization occurs intrinsically due to temporal processing in SNNs is contributing to their robustness in handling variability in the gradients from a large number of clients.

In addition to having promising results with SNNs, this study opens up many questions to explore in future work. For example, scheduling the federated learning updates with more temporal granularity is an open problem and has potential to significantly improve the performance of the system. While it is out of scope of this work as it involves rethinking of the BNTT learning rule, we hope this work will serve as a baseline for future works on large scale federated learning with SNNs incorporating finer temporal learning. Additionally, developing a theoretical basis to explain the observation of a relatively lower decline in performance in SNNs compared to ANNs as the number of clients is increased. This would help in manifesting SNNs as a more scalable alternative to ANNs in federated learning. There are multiple studies on federated learning in ANN domain focusing on improving convergence of federated averaging when the data distribution among the clients is non-IID [15], [35], [43]–[45]. FedOpt [44] generalizes the FedAvg [7] by proposing adaptive optimization algorithms on the server. Since the process of communicating the gradients of weights and performing federated averaging in SNNs is similar to that of ANNs, FedOpt can be applied to SNNs to improve the performance. FedDyn [43], SCAFFOLD [45] and FedProx [35] perform regularization while training on the client by restricting the client models from significantly deviating from the server model. Extending these methods to SNNs is not trivial as it involves a modified loss function which advocates for further refinement in local BNTT training method. Implementing and evaluating these methods on SNNs will be helpful in tackling non-IID challenges. Additionally, batch normalization is traditionally avoided in federated learning literature as it can perform poorly with non-IID data [15]. While the authors in [44] replace batch normalization with group normalization, an in-depth analysis on batch normalization in federated learning will be helpful. Note that since this work is focused on comparing ANNs and SNNs, our experiments are limited to standard image datasets. Benchmarking SNNs for neuromorphic datasets such as N-MNIST [48], N-Caltech101 [48], DVSGesture [49], MNIST-DVS [50] and CIFAR10-DVS [51] in federated environments is still an open problem. Additionally, exploring novel training methods of SNNs based on local loss functions [52], [53] in the context of federated learning is a promising direction.

Federated learning involves many more engineering intricacies such as handling delays/failures in the client updates, the communication cost of models and gradients to and from the clients, and the policy of selecting the clients in each round. SNNs can provide unique advantages in each of these directions. Exploring temporal datasets beyond static vision for federated learning with SNNs is also a potential direction. Since the spikes are binary, SNNs are also close to binarized ANNs which are in turn a special case of quantized ANNs. In addition to quantized models being more robust to adversarial attacks [54], [55], there has been recent work on quantization of ANNs in federated learning providing additional communication efficiency [12], [56], [57]. It would be intriguing to compare and evaluate these methods and draw connections to SNNs.

References

[1] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[2] B. Rajendran, A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou, "Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 97–110, 2019.

[3] 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, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: https://science.sciencemag.org/content/345/6197/668

[4] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[5] E. Hunsberger and C. Eliasmith, "Training spiking deep networks for neuromorphic hardware," *CoRR*, vol. abs/1611.05141, 2016. [Online]. Available: http://arxiv.org/abs/1611.05141

[6] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in Neuroscience*, vol. 13, p. 95, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2019.00095

[7] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017. [Online]. Available: http://arxiv.org/abs/1602.05629

[8] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[9] A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *CoRR*, vol. abs/1811.03604, 2018. [Online]. Available: http://arxiv.org/abs/1811.03604

[10] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3298981

[11] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 46–51, 2020.

[12] N. Shlezinger, M. Chen, Y. C. Eldar, H. V. Poor, and S. Cui, "Uveqfed: Universal vector quantization for federated learning," *IEEE Transactions on Signal Processing*, 2020.

[13] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," *arXiv e-prints*, pp. arXiv–2002, 2020.

[14] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[15] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.

[16] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1698–1707.

[17] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[18] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," *arXiv preprint arXiv:1705.10467*, 2017.

[19] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2512–2520.

[20] L. Zhu and S. Han, "Deep leakage from gradients," in *Federated Learning*. Springer, 2020, pp. 17–31.

[21] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[22] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[23] N. Caporale and Y. Dan, "Spike timing–dependent plasticity: A hebbian learning rule," *Annual Review of Neuroscience*, vol. 31, no. 1, pp. 25–46, 2008, pMID: 18275283. [Online]. Available: https://doi.org/10.1146/annurev.neuro.31.060407.125639

[24] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*. ieee, 2015, pp. 1–8.

[25] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[26] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," 2019.

[27] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[28] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature communications*, vol. 11, no. 1, pp. 1–15, 2020.

[29] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," 2020.

[30] Y. Kim and P. Panda, "Revisiting batch normalization for training low-latency deep spiking neural networks from scratch," 2020.

[31] S. Sharmin, N. Rathi, P. Panda, and K. Roy, "Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations," 2020.

[32] Y. Kim and P. Panda, "Visual explanations from spiking neural networks using interspike intervals," *arXiv preprint arXiv:2103.14441*, 2021.

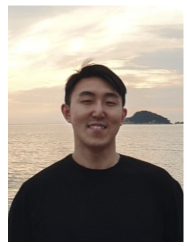
[33] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[34] N. Skatchkovsky, H. Jang, and 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.

[35] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous

networks," *CoRR*, vol. abs/1812.06127, 2018. [Online]. Available: http://arxiv.org/abs/1812.06127
[36] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, 2020.
[37] 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, p. 331, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00331
[38] S. M. Bohte, "Error-backpropagation in networks of fractionally predictive spiking neurons," in *Artificial Neural Networks and Machine Learning – ICANN 2011*, T. Honkela, W. Duch, M. Girolami, and S. Kaski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 60–68.
[39] A. Albasyoni, M. Safaryan, L. Condat, and P. Richtárik, "Optimal gradient compression for distributed and federated learning," 2020.
[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[41] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.
[42] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, T. N. Hoang, and Y. Khazaeni, "Bayesian nonparametric federated learning of neural networks," *arXiv preprint arXiv:1905.12022*, 2019.
[43] D. A. E. Acar, Y. Zhao, R. Matas, M. Mattina, P. Whatmough, and V. Saligrama, "Federated learning based on dynamic regularization," in *International Conference on Learning Representations*, 2020.
[44] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.
[45] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for on-device federated learning." 2019.
[46] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.
[47] P. Panda, S. A. Aketi, and K. Roy, "Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization," *Frontiers in Neuroscience*, vol. 14, Jun 2020. [Online]. Available: http://dx.doi.org/10.3389/fnins.2020.00653
[48] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.
[49] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.
[50] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-dvs and mnist-dvs. their history, how they were made, and other details," *Frontiers in neuroscience*, vol. 9, p. 481, 2015.
[51] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: an event-stream dataset for object classification," *Frontiers in neuroscience*, vol. 11, p. 309, 2017.
[52] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.
[53] H. Jang, N. Skatchkovsky, and O. Simeone, "Vowel: A local online learning rule for recurrent networks of probabilistic spiking winner-take-all circuits," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 4597–4604.
[54] P.Panda et al., "Quanos- adversarial noise sensitivity driven hybrid quantization of neural networks," *ISLPED*, Aug 2020. [Online]. Available: http://dx.doi.org/10.1145/3370748.3406585
[55] A. Galloway, G. W. Taylor, and M. Moussa, "Attacking binarized neural networks," *arXiv preprint arXiv:1711.00449*, 2017.
[56] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2021–2031.
[57] M. M. Amiri, D. Gunduz, S. R. Kulkarni, and H. V. Poor, "Federated learning with quantized global model updates," *arXiv preprint arXiv:2006.10672*, 2020.

Yeshwanth Venkatesha received B.Tech. in Computer Science and Engineering from Indian Institute of Technology and Science Kharagpur, India, in 2017. He joined Yale University, USA, in 2020 as a Ph.D. student in the Electrical Engineering department. Prior to joining Yale University, he worked as a Data Scientist and Software Engineer at Walmart Labs India and Samsung Advanced Institute of Technology India respectively. His research interests lie in the areas of efficient processing of neural networks and distributed machine learning.



Youngeun Kim is currently working toward the Ph.D. degree in Electrical Engineering at Yale University, New Haven, CT, USA. Prior to joining Yale, he worked as a full-time student internship at T-Brain, AI Center, SK telecom, South Korea. He received his B.S. degree in Electronic Engineering from Sogang University, South Korea, in 2018 and M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST), in 2020. His research interests include neuromorphic computing, computer vision, and deep learning.

Leandros Tassiulas received the Ph.D. degree in electrical engineering from the University of Maryland, College Park, MD, USA, in 1991. He held faculty positions at Polytechnic University, New York, NY, USA, the University of Maryland, and the University of Thessaly, Greece. He is currently the John C. Malone Professor of electrical engineering with Yale University. His research interests include computer and communication networks, with an emphasis on fundamental mathematical models and algorithms of complex networks, architectures and protocols of wireless systems, sensor networks, novel internet architectures, and experimental platforms for network research. His most notable contributions include the max-weight scheduling algorithm and the back-pressure network control policy, opportunistic scheduling in wireless, the maximum lifetime approach for wireless network energy management, and the consideration of joint access control and antenna transmission management in multiple antenna wireless systems. Dr. Tassiulas's research has been recognized by several awards, including the IEEE Koji Kobayashi Computer and Communications Award, the Inaugural INFOCOM 2007 Achievement Award for fundamental contributions to resource allocation in communication networks, the INFOCOM 1994 and 2017 Best Paper Awards, the National Science Foundation (NSF) Research Initiation Award in 1992, the NSF CAREER Award in 1995, the Office of Naval Research Young Investigator Award in 1997, and the Bodossaki Foundation Award in 1999.

Priyadarshini Panda obtained her Ph.D. from Purdue University, USA, in 2019. She joined Yale University, USA, as an assistant professor in the Electrical Engineering department in August, 2019. She received the B.E. degree in Electrical & Electronics Engineering and the M.Sc. degree in Physics from B.I.T.S. Pilani, India, in 2013. She was the recipient of outstanding student award in physics for academic excellence. From 2013-14, she worked in Intel, India on RTL design for graphics power management. She has also worked with Intel Labs, USA, in 2017 and Nvidia, India in 2013 as research intern. She has been awarded the Amazon Research Award for 2019-20. Her research interests lie in robust and energy efficient neuromorphic computing, deep learning and related applications through algorithm-hardware co-design.