# Enabling Privacy-Preserving Edge AI: Federated Learning Enhanced with Forward-Forward Algorithm

1st Mohammadnavid Ghader
*Department of Computer Science*
Shahid Beheshti University
Tehran, Iran
m_ghader@sbu.ac.ir

2nd Saeed Reza Kheradpisheh
*Department of Computer Science*
Shahid Beheshti University
Tehran, Iran
s_kheradpisheh@sbu.ac.ir

3rd Bahar Farahani
*Cyberspace Research Institute*
Shahid Beheshti University
Tehran, Iran
b_farahani@sbu.ac.ir

4th Mahmood Fazlali
*Cybersecurity and Computing Systems Research Group*
University of Hertfordshire
Hatfield, United Kingdom
m.fazlali@herts.ac.uk

2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS) | 979-8-3503-4959-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/COINS61597.2024.10622150

***Abstract*—Artificial Intelligence (AI) has emerged as a pivotal technology across various sectors, including healthcare, transportation, and the development of smart cities, revolutionizing service delivery and operational efficiency. However, the adoption and introduction of new data-driven services leveraging centralized training models have been hindered by significant concerns over privacy and data security, as these traditional techniques potentially expose sensitive information to breaches. Federated Learning (FL) presents a compelling solution to this dilemma, enabling decentralized data processing without compromising privacy. Integrating Edge AI into this framework, FL enables the collaborative training of models based on data distributed across different clients. Nevertheless, implementing FL on edge devices introduces a set of challenges due to the limited computational and memory resources available on such tiny devices. Specifically, the backpropagation (BP) phase of training models is notably resource-intensive, posing a barrier to efficient deployment. To address this, we replaced the backpropagation phase with a Forward-Forward (FF) algorithm. Moreover, we integrated and compared several loss functions, namely Hinton, Symba, and Swish, to assess their compatibility and efficiency in the context of forward-forward training within the federated learning framework. The study indicates that our novel method leads to a slight decrease in accuracy for large and complex datasets compared to the traditional BP technique. However, it has the potential to enhance runtime and reduce memory overhead. The proposed technique represents a promising path toward the broader adoption of Edge AI by effectively addressing critical technical challenges, namely privacy concerns and on-chip model training.**

***Index Terms*—Federated Learning, Forward-Forward Algorithm, Edge AI, Privacy-Preserving ML**

## I. Introduction

Artificial Intelligence (AI) — including Machine Learning (ML) and Deep Learning (DL) — has gained significant traction across various vertical industries in recent years. The traditional approach to the development of ML/DL models relies on centralized training, which begins with the ingestion and collection of data from distinctive remote sources and subsequently combing all of them in a central repository for processing and training. However, this conventional approach poses significant security and privacy threats. In response, Federated Learning (FL) has emerged as a paradigm shift to mitigate this issue by striking a balance between AI utility and privacy/security [1]. FL is a distributed technique for training machine learning models, wherein the learning process takes place locally on individual clients or edge devices, rather than on a centralized server [2]. This decentralized strategy, especially pertinent in the context of Edge AI, significantly enhances security as data remains distributed across edge devices. Despite its advantages, FL also faces several challenges, particularly in the training process on resource-constrained edge devices. The limited computational power and insufficient memory of these edge devices significantly hinder the efficiency and feasibility of the FL training procedure.

DL models are typically very complex due to their numerous parameters and densely interconnected layers of neurons, presenting significant challenges for storage and processing on edge and tiny devices [3]. Particularly, the backpropagation (BP) algorithm, the established standard for training DL models, requires substantial computational resources and consumes a large part of the client's memory for its intermediate computations. To address this issue, recently, a few works started exploring alternative training techniques targeting resource-constrained devices. In this context, notably, the Forward-Forward (FF) algorithm has been introduced as a promising approach to potentially replace the BP algorithm [4].

The FF algorithm represents a novel technique for training neural networks, offering several potential advantages compared to the traditional BP approach: (i) First, it has

979-8-3503-4959-7/24/$31.00 ©2024 IEEE

been shown that it could be faster for specific tasks as it requires fewer calculations. (ii) Moreover, it might also be more efficient for handling large models and datasets due to its simpler nature. (iii) Additionally, The FF algorithm can handle models containing components that aren't entirely understood (black boxes components), a feature lacking in traditional BP. (iv) Furthermore, some preliminary results and early evidence indicate that the FF algorithm might even improve accuracy. It is important to note that the FF algorithm may not always outperform BP in performance, and its effectiveness in addressing particular challenges is under investigation by ongoing research. Overall, FF has the potential to be a faster and more scalable training method for Edge AI. To do so, it demands further development and evaluation against the well-established BP technique. Particularly, the performance of the FF algorithm in comparison to BP in the FL setup should be thoroughly and deeply investigated. By addressing this significant research gap, we integrate the FF algorithm into the FL framework. We summarize our contributions as follows:

- For the first time, we analyze various flavors of different loss functions for training in the FF method comprehensively.
- We propose to use a new loss function, namely Swish, and advocate for the Swish function as an alternative option to previously proposed loss functions, including Hinton and Symba, by demonstrating its improvements across various datasets.
- Additionally, we investigate the FF algorithm in FL by utilizing all these three loss functions and compare their efficiency to that of BP. We demonstrate and discuss that FF offers comparable accuracy to BP, with the potential to be deployed on resource-constrained edge devices.

The rest of this paper is organized as follows: Section II overviews related work on the FF algorithm. Section III discusses the necessary background knowledge of the FF method. Section IV presents the proposed method. Section V elaborates on the experimental setup and results. Finally, Section VI concludes the paper and suggests a direction for future work.

## II. Related Work

The original paper [4], which has explained the FF algorithm, was introduced for the first time in 2022. The primary goal of this paper is to replace the traditional forward and backward passes of BP with a more streamlined approach. In this algorithm, two separate forward passes are used: one with positive (real) data and the other with negative data, which can be generated by the network itself.

Another research presents a novel algorithm named SymBa [5], designed to achieve more biologically plausible learning in neural networks compared to the commonly used BP method. The algorithm is built upon the FF algorithm, which is a BP-free approach for training neural networks. SymBa addresses a specific issue in the FF algorithm related to asymmetric gradients caused by conflicting converging directions for positive and negative samples. It aims to improve the convergence behavior of the FF algorithm by balancing positive and negative losses, thus enhancing both performance and convergence speed. Additionally, the algorithm introduces the concept of Intrinsic Class Pattern (ICP) containing class information, which is incorporated to prevent the loss of class-related information during training. The experiments of this study were conducted on the MNIST, CIFAR-10, and CIFAR-100 datasets.

There is a study comparing the internal representations generated by the FF algorithm and standard BP [6]. It discusses how the FF algorithm produces category-specific ensembles with high sparsity, resembling the organization observed in cortical sensory areas. The baseline method for the FF has a threshold in loss definition, which is a hyperparameter. There is a study [7] that presents a new pyramidal optimization technique for adjusting the loss threshold, a particular hyperparameter associated with the FF. This illustrates that employing an efficient thresholding strategy can result in a significant improvement in test error. From the point of view of memory consumption, a study [8] has been conducted that examines the quantitative enhancements in complexity and memory usage achieved by PEPITA and FF computing methods compared to backpropagation.

A study has been done about the use of the FF algorithm in FL [9]. This research discusses a challenge in FL where clients with limited resources can disrupt training efficiency. To address this issue, the author proposes a new approach called FedFwd, which leverages a learning procedure that doesn't rely on BP. Instead, it employs the Forward Forward algorithm in the local training process. FedFwd reduces computational requirements for updating parameters by implementing layer-wise local updates. This approach eliminates the need to store all intermediate activation values during training, making it more efficient for clients with limited resources. The paragraph highlights that the proposed FedFwd method is competitive with other FL methods that depend on BP, as demonstrated in various experiments conducted on standard datasets like MNIST and CIFAR-10.

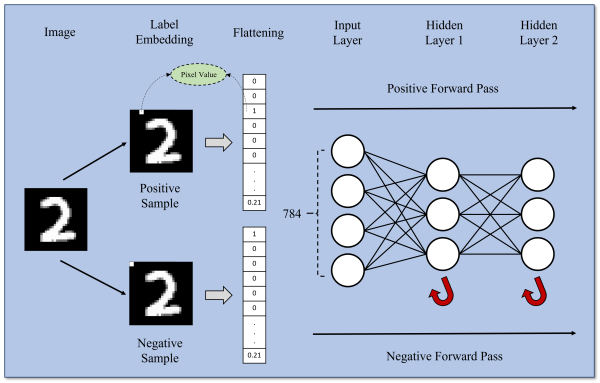


Fig. 1: The architecture of the proposed forward-forward algorithm.

## III. Background

### A. Forward-Forward Algorithm

BP is the most common learning algorithm used in the training procedure of deep learning models. In a new research, the FF algorithm proposed by Geoffrey Hinton was introduced to replace BP. Unlike BP, which uses a forward pass to calculate errors and a backward pass to adjust weights, the FF algorithm relies on two forward passes. This algorithm incorporates both positive and negative data as part of its conceptual framework. The first pass of the algorithm uses "positive data," which is real, correctly labeled data, and the second pass uses "negative data," which can be artificially generated by the network itself. Another characteristic of the FF algorithm is layer-level learning. Each layer in the network has its own objective function. This function aims to achieve high "goodness" for positive data and low "goodness" for negative data.

Goodness can be defined in various ways. for instance, in the original paper of this algorithm, the goodness function for positive and negative data was defined as follows:

$$Goodness_{pos} = \sum_j y_{pos,j}^2 \tag{1}$$

$$Goodness_{neg} = \sum_j y_{neg,j}^2 \tag{2}$$

where $y_{pos,j}$ and $y_{neg,j}$ are the outputs of the hidden neuron of j when the input data of the layer is positive and negative data, respectively. In the FF algorithm, every layer has its own loss function. The loss function for positive and negative data and also total loss in the original paper was computed as follows:

$$Loss_{pos} = \log(1 + e^{\theta - Goodness_{pos}}) \tag{3}$$

$$Loss_{neg} = \log(1 + e^{Goodness_{neg} - \theta}) \tag{4}$$

$$Loss_{total} = Loss_{pos} + Loss_{neg} \tag{5}$$

where $\theta$ is a hyperparameter for loss value modification. The steps involved in the FF algorithm are as follows:

- **Positive Pass:** This pass uses real data with correct labels. The positive pass aims to push a specific layer-wise "goodness" function to a high value. This function essentially measures how well the network activations represent the positive data. Common choices for the goodness function include the sum of squares of the activations in a layer. Higher activation values generally indicate a better representation of the data. The positive data is fed through the network layer by layer in the usual forward-pass fashion. At each hidden layer, the activations are evaluated using the chosen goodness function. The network doesn't explicitly calculate errors or adjust weights during this pass.
- **Negative Pass:** This pass utilizes "negative data." These are not necessarily incorrect examples but rather data crafted to be dissimilar or misleading compared to the positive data. The goal is to push the layer-wise "goodness" function (introduced in the positive pass) to a low value for the negative data. This encourages the network to differentiate between positive and negative examples. The negative data is fed forward through the network. At each layer, the activations are evaluated using the same goodness function as in the positive pass. Similar to the positive pass, there is no explicit error calculation or weight update here.

When we try to maximize the value of $Goodness_{pos}$ and its value tends to infinity, the $Loss_{pos}$ tends to zero. On the opposite side, When we try to minimize the value of $Goodness_{neg}$ and its value tends to zero, the $Loss_{neg}$ tends to $log(1 + e^{-\theta})$. It is obvious that the results of $Loss_{pos}$ and $Loss_{neg}$ are not the same. Another research [6] tries to mitigate this issue and change the way the loss function was calculated in such a way that both $Loss_{pos}$ and $Loss_{neg}$ tend to have identical values. They suggested a superseded loss function as follows:

$$\begin{aligned} \Delta &= Goodness_{\text{pos}} - Goodness_{\text{neg}} \\ Loss_{\text{Symba}} &= \log(1 + e^{-\alpha\Delta}) \end{aligned} \tag{6}$$

where $\alpha$ is a hyperparameter for scaling value of $\Delta$. The FF algorithm is a new proposal and is under active research. Initial investigations show promise, but they require further development for practical applications. Challenges include computational efficiency and achieving good generalization performance compared to BP.

### B. Federated Learning

FL is a machine learning technique that allows training a model on a decentralized network of devices, such as smartphones or smart devices, without directly accessing the private data stored on those devices. This approach is beneficial for applications where privacy is a concern because user data often contains sensitive information, and FL keeps that data on the devices. In addition, sometimes data is geographically distributed, and collecting data from all devices to a central server can be expensive and impractical.

The first step of FL is that a central server distributes an initial model (usually a basic neural network) to all participating devices. Next, each device uses its own local data to train the model locally for a specified number of iterations using techniques like Stochastic Gradient Descent (SGD). This training only updates the model parameters (weights and biases) on the device itself, not the raw data. After local training, each device sends its updated model parameters back to the central server. The central server aggregates the received model updates from multiple devices. This can be done by simply averaging the updates (FedAvg) or using more sophisticated methods. The aggregated model update is then incorporated into the global model on the server. This process repeats for a predefined number of rounds or until a stopping criterion like desired accuracy is met. The FedAvg algorithm is widely used in FL tasks. The scheme of FedAvg's operations can be seen in Algorithm 1.

**Algorithm 1** Proposed Method (Fed-FF)

**Input:** Global model parameters $w_0$, Number of communication rounds $T$, Number of local epochs E, Number of model's layers L
**for** $t = 1$ **to** $T$ **do**
  **Client Update Phase:**
  **for** each client $k = 1$ **to** $K$ in parallel **do**
    Receive global model parameters $w_{t-1}$ from server
    **for** $e \in \{1, \ldots, E\}$ **do**
      **for** $l \in \{1, \ldots, L\}$ **do**
        ***Positive Pass:***
        $a_l^+ = \text{Activate}(w_l \cdot x^+)$
        $g_l^+ = \text{Goodness function}(a_l^+)$
        ***Negative Pass:***
        $a_l^- = \text{Activate}(w_l \cdot x^-)$
        $g_l^- = \text{Goodness function}(a_l^-)$
        ***Weight Update***
      **end for**
    **end for**
  **end for**
  **Server Aggregation Phase (FedAvg):**
  Aggregate local model updates:
  $w_t = \frac{1}{K} \sum_{k=1}^{K} w_t^k$
  Send updated global model parameters $w_t$ to all clients
**end for**
**Output:** Final global model parameters $w_T$

## IV. Proposed Method

Algorithm 1 illustrated the proposed FL method. In this study, we utilize the Swish function as the loss function of models that are trained in the clients. The Swish function [10] is usually used as an activation function. In deep learning, activation functions introduce non-linearity into the model, allowing it to learn complex patterns in the data. The Swish function is a smooth, non-monotonic function. It is based on the logistic sigmoid function, which squashes the input into the range [0, 1]. The sigmoid function introduces non-linearity and ensures that the output of the Swish function is bounded. The mathematical formulation of the Swish function is as follows:

$$Swish(x) = x \cdot \text{sigmoid}(\beta x) \tag{7}$$

where x is the input to the function and $\beta$ is a hyperparameter that controls the smoothness of the function.

The Swish function is smooth and differentiable everywhere, which is crucial for training neural networks using techniques like gradient descent. Smoothness ensures that small changes in the input result in small changes in the output, making it easier to optimize the network parameters. The Swish loss function is defined as follows:

$$\begin{aligned} \Delta &= Goodness_{\text{pos}} - Goodness_{\text{neg}} \\ Loss_{\text{Swish}} &= \frac{x}{1 + e^{-\alpha\Delta}} \end{aligned} \tag{8}$$

It is important to mention that in calculating the goodness of layers, we use the mean squared of neuron activities instead of their sum.

Fig. 1 demonstrates the flow of the FF algorithm, which is utilized by our clients. For better and more tangible transmission of the details of the process used in the algorithm, an example from the MNIST dataset has been used. It is assumed that we have a sample of digit two and we are going to use it as an input of our neural network. In the proposed architectures for the FF algorithm, despite the BP, there is no output layer for the softmax computing and inferring the label that is predicted by the model. Hence, we have to embed the label of the sample in the input.

In the FF algorithm, we have positive data, which is assigned by a true label, and negative data, which is assigned by a false label. Therefore, for label embedding, the first 10 pixels of each sample are allocated for the one-hot representation of labels. In our example, the third and first pixels of the image are set to one for the positive sample and negative data, respectively.

After label embedding, both the positive and negative samples are flattened to prepare for passing to the fully connected neural network. Since the dimension of the figure's sample is 28 by 28, the size of the flattened data is 784. Although the number of hidden layers placed in the figure is equal to 2, it can be increased depending on the need.

During the first epoch and in the first forward pass, called the positive forward pass, the data in batches moves from the input layer to the first hidden layer, and for the first batch, the $Goodness_{pos}$ is calculated. After the positive forward pass, it is the turn to perform the second forward pass, called the negative forward pass. The negative data also moves from the input layer to the first hidden layer, and after that, the $Goodness_{neg}$ is calculated. Now that we have the values of $Goodness_{pos}$ and $Goodness_{neg}$, it is time to calculate the loss value with the desired objective function and update the weights of the first layer. The outputs of activation of the first layer are normalized for both positive and negative data and ready to enter the next hidden layer.

Similar to the process of the first hidden layer, The positive and negative data generate their activations again, and $Goodness_{pos}$ and $Goodness_{neg}$ of the second layer and then the loss amount of the second layer are calculated. The weights of the second hidden layer are updated, and if there are further hidden layers, this process is continued. After the first batch, the rest of the batches enter the neural network and undergo the mentioned steps. With the completion of the above operations, the first epoch will end, and the next epoch will begin. The model is trained under the number of assumed epochs, and after that, The training phase is completed.

In the testing phase, each sample is considered to have duplicates corresponding to the number of classes in the dataset. However, for each duplicate, one of the possible labels is embedded in it. Each of them enters the trained neural network, and the goodness value of every layer is calculated. The total goodness value corresponding to each label equals

the sum of all the calculated goodness values in each layer. The final predicted label for the desired input sample is the label that has a larger total goodness value than the others.

## V. Experiments

### A. Experimental Setup

To evaluate the proposed method, three datasets were utilized: MNIST [11], Fashion-MNIST [12], and CIFAR-10 [13]. Both MNIST and Fashion-MNIST datasets comprise 60,000 training samples, whereas CIFAR-10 contains 50,000. For each dataset, we also considered 10,000 samples for testing. To ensure a fair comparison, the same architecture as the FedFwd model was adopted. Experiments were conducted under both IID and non-IID data distributions among clients. In the non-IID distribution, All three datasets contain 10 labels, which were divided into five different sets, each comprising two distinct labels. In the neural network architecture of FL, we have used 2 and 3 layers of a fully connected network. The number of neurons in each layer across all experiments was set to 500. The total number of considered clients is 100, and 10 of them are randomly selected for participation in each round. The number of rounds and local epochs are 600 and 10, respectively. The learning rate in all experiments is set to 0.004. The Symba and Swish loss have $\alpha$, and the Hinton loss has $\theta$, which are hyperparameters. In all experiments, we set the $\alpha$ and $\theta$ equal to 4 and 2, respectively.

### B. Experimental Results and Analysis

*a) Centralized Setting:* In FL, the "centralized setting" refers to a traditional machine learning setup where all the data is collected and stored in a central server or data center. In this setting, the training process typically involves bringing all the data to a central location where a global model is trained using all available data. In the MNIST, the best result is obtained when the model has 2 layers, and the Swish is the loss function, reaching an accuracy of 98.58. This accuracy exceeds that of the same model employing the Symba and Hinton loss functions by 0.23% and 4.68%, respectively.

When the number of neural network layers changes from two to three, the accuracy of the Symba and Swish is slightly reduced. However, the Hinton loss has experienced a 2.15% increase in accuracy.

In the CIFAR-10, the optimal outcome is achieved with a model comprising three layers and utilizing the Symba loss function. The accuracy under the specified condition stands at 53.31%, marking an improvement of 0.36% compared to the model using Swish and 2.01% compared to the one employing the Hinton loss function. When the network consists of two layers, the accuracy decreases slightly for both the Symba and Swish functions. However, the Hinton loss achieves a marginally higher accuracy of 0.4%.

In Fashion-MNIST, the best results are obtained when a model with three layers is employed, utilizing the Swish loss function. The accuracy achieved under this configuration is 90.22%, indicating an enhancement of 1.14% compared to the model using the Symba loss function, and a 2% improvement compared to the model utilizing the Hinton loss function. When the model includes two layers, The accuracy experiences a scant decline when utilizing the Swish function. However, the Symba and Hinton loss achieves higher accuracy of 0.12% and 1.09%, respectively. All the values of the accuracies obtained from the centralized setting are shown in Table 1.

TABLE I: Model performance in the centralized setting.

| Dataset | # Layers | Loss | # Params | Test Acc (%) |
|---|---|---|---|---|
| MNIST | 2 | Hinton | 0.64M | 93.9 |
| | | Symba | 0.64M | 98.35 |
| | | Swish | 0.64M | **98.58** |
| | 3 | Hinton | 0.89M | 96.05 |
| | | Symba | 0.89M | 98.12 |
| | | Swish | 0.89M | 98.45 |
| CIFAR-10 | 2 | Hinton | 1.78M | 51.7 |
| | | Symba | 1.78M | 53.04 |
| | | Swish | 1.78M | 52.41 |
| | 3 | Hinton | 2.03M | 51.3 |
| | | Symba | 2.03M | **53.31** |
| | | Swish | 2.03M | 52.95 |
| Fashion-MNIST | 2 | Hinton | 0.64M | 89.31 |
| | | Symba | 0.64M | 89.2 |
| | | Swish | 0.64M | 90.1 |
| | 3 | Hinton | 0.89M | 88.22 |
| | | Symba | 0.89M | 89.08 |
| | | Swish | 0.89M | **90.22** |

*b) Federated Learning Setting:* This work was compared with the FedAvg, which employs BP, as well as FedFwd and the centralized FF algorithm. The results of the experiments on the MNIST dataset show that the best performance is achieved when we have 3 layers in the network, and the Swish loss is the objective function. In the IID distribution, Fed-FF with the Swish has 98.80% accuracy on test data, which is even better than the FedAvg, which has 98.25%. In the Non-IID distribution of data among clients, Fed-FF with the Swish has the best outcome again and has 98.56% accuracy. The FedFwd, which uses the Hinton loss function, performed relatively significantly worse than the other methods both in 2 and 3-layer architectures. At its best condition, the FedFwd achieved 96.78%, which has a difference of 2.02% with the best performance on the MNIST.

The experimental results obtained from the CIFAR-10 dataset indicate that optimal performance is achieved using the FedAvg, employing a 3-layer neural network architecture and cross-entropy as the loss function. In the context of an Independent and Identically Distributed (IID) data distribution, this approach yields an accuracy of 59.91%, representing a significant improvement of 7.22% over the most proficient FeedForward (FF) model. Notably, among FF-based models, a 2-layer neural network incorporating the Swish activation function achieves an accuracy of 52.69%, surpassing the performances of models employing Symba and Hinton loss functions by margins of 2.13% and 6.93%, respectively. Transitioning to a Non-IID data distribution, the FedAvg methodology featuring a three-layer neural network achieves

TABLE II: Comparison of the results obtained in FL setting on the MNIST, CIFAR-10, and Fashion-MNIST.

| Dataset | # Layers | Learning Procedure | Loss | # Params | Test Acc IID (%) | Test Acc Non-IID (%) |
|---|---|---|---|---|---|---|
| MNIST | 2 | FedAvg (BP) | CE | 0.64M | 98.19 | 97.43 |
| | | FedFwd | Hinton | 0.64M | 96.63 | 92.47 |
| | | Fed-FF | Symba | 0.64M | 98.07 | 97.53 |
| | | Fed-FF | Swish | 0.64M | 98.71 | 98.39 |
| | 3 | FedAvg (BP) | CE | 0.89M | 98.25 | 97.46 |
| | | FedFwd | Hinton | 0.89M | 96.78 | 92.57 |
| | | Fed-FF | Symba | 0.89M | 98.10 | 97.71 |
| | | Fed-FF | Swish | 0.89M | **98.80** | **98.56** |
| CIFAR-10 | 2 | FedAvg (BP) | CE | 1.79M | 58.86 | 50.67 |
| | | FedFwd | Hinton | 1.78M | 45.51 | 37.06 |
| | | Fed-FF | Symba | 1.78M | 50.31 | 43.31 |
| | | Fed-FF | Swish | 1.78M | 52.69 | 48.07 |
| | 3 | FedAvg (BP) | CE | 2.04M | **59.91** | **52.00** |
| | | FedFwd | Hinton | 2.03M | 45.76 | 38.10 |
| | | Fed-FF | Symba | 2.03M | 50.56 | 41.89 |
| | | Fed-FF | Swish | 2.03M | 51.61 | 47.39 |
| Fashion-MNIST | 2 | FedAvg (BP) | CE | 0.64M | 89.75 | 85.25 |
| | | Fed-FF | Hinton | 0.64M | 88.96 | 82.62 |
| | | Fed-FF | Symba | 0.64M | 88.62 | 86.06 |
| | | Fed-FF | Swish | 0.64M | **90.06** | **87.74** |
| | 3 | FedAvg (BP) | CE | 0.64M | 88.32 | 84.17 |
| | | Fed-FF | Hinton | 0.89M | 88.35 | 82.25 |
| | | Fed-FF | Symba | 0.89M | 87.48 | 84.96 |
| | | Fed-FF | Swish | 0.89M | 89.55 | 86.84 |

an accuracy of 52%, marking a noteworthy improvement of 3.93% compared to the top-performing FF-based model utilizing a 2-layer neural network with the Swish loss function. Noteworthy trends emerge within FF-based models: as the number of layers increases from two to three, the accuracies of models employing Swish and Symba loss functions experience slight declines of 0.68% and 1.42%, respectively. Intriguingly, the accuracy of models employing the Hinton loss function exhibits a modest increase of 1.04%.

The findings from experiments on the Fashion-MNIST dataset underscore the efficacy of specific model configurations and loss functions in achieving optimal performance. Notably, the pinnacle of performance is reached with a 2-layer neural network architecture combined with the Swish loss function. Within an Independent and Identically Distributed (IID) distribution context, the 2-layer neural network approach incorporating the Swish loss attains an accuracy of 90.06% on test data, surpassing the highest-performing FedAvg model by a margin of 0.31% (89.75%). Among FF-based models, the superiority of the Swish loss function is evident, with a 2-layer neural network exhibiting 1.1% higher accuracy when compared to its counterpart employing the Hinton loss function. Remarkably, within the Fashion-MNIST dataset, the performance of the Hinton loss function outpaces that of the Symba loss function across both 2-layer and 3-layer neural network architectures. Specifically, in the 2-layer model, the Hinton loss achieves an accuracy of 88.96%, outstripping the Symba loss by 0.34%. Similarly, within the 3-layer model, the Hinton loss achieves an accuracy of 88.35%, surpassing the Symba loss by 0.84%. Transitioning to a Non-Independent and Identically Distributed (Non-IID) data distribution scenario, a 3-layer neural network employing the Swish loss function demonstrates superior performance, achieving an accuracy of 87.74%, outperforming the top-performing FedAvg model by 2.49%. Among FF-based models, the efficacy of the Swish loss function remains evident, with a 2-layer neural network exhibiting 1.68% and 5.12% higher accuracy when compared to its counterparts utilizing the Symba and Hinton loss functions, respectively. Despite the marginal superiority of the Hinton loss function over the Symba loss function in IID distributions, its performance significantly deteriorates in non-IID distributions. Notably, a 2-layer neural network employing the Symba loss function achieves an accuracy 3.44% higher than its Hinton loss counterpart, while within the 3-layer architecture, this margin amounts to 2.71%. The details of the obtained results in the FL setting are given in Table 2.

### C. Complexity Analysis

In the field of complexity analysis of methods in deep learning, three key items can be considered for evaluating and comparing the efficiency of algorithms. These items include MACC, activations, and RAM usage [8]. For the MNIST dataset, we consider that we have three layers with the size of 784, 500, and 500 neurons, respectively. The number of parameters is calculated as follows:

$$\begin{aligned} Weights &= 784 \times 500 + 500 \times 500 = 642000 \\ Biases &= 500 + 500 = 1000 \\ Parameters &= Weights + Biases = 643000 \end{aligned} \tag{9}$$

In the Forward-Forward and the backpropagation algorithm, the activations' size for each sample is:

$$\begin{aligned} activations_{FF} &= Max(784, 500, 500) \\ &= 784Byte \simeq 0.8KB \end{aligned} \tag{10}$$

$$activations_{BP} = 784 + 500 + 500 \\ = 1.784Byte \simeq 1.8KB \tag{11}$$

The RAM usage of the Forward-Forward and the backpropagation algorithm for each sample is:

$$RAM_{FF} = Parameters + activations_{FF} \\ = 643 + 0.8 = 643.8KB \tag{12}$$

$$RAM_{BP} = Parameters + activations_{BP} \\ = 644 + 0.8 = 644.8KB \tag{13}$$

The $activations_{FF}$ is less than $activations_{BP}$ and this affects the efficiency of RAM usage in the Forward-Forward algorithm. For the other datasets like CIFAR10 and Fashion-MNIST, a similar procedure can be followed for the calculation of the RAM usage in both algorithms.

## VI. Conclusion

This study introduced a novel federated learning technique by exploiting the forward-forward algorithm. The proposed method provides potential advantages in terms of memory utilization and computational efficiency compared to BP-based FL algorithms. The performance of this new method was evaluated using three distinct datasets, namely MNIST, CIFAR-10, and Fashion-MNIST. Results indicated that the proposed method achieves test accuracies and training times comparable to those of FedAvg, the traditional BP-based federated learning algorithm. According to the findings, it performs slightly better on MNIST and Fashion-MNIST. However, in CIFAR-10, which has a greater complexity of data compared to MNIST and Fashion-MNIST, FedAvg achieves marginally better accuracy. For future work, it can be investigated how we can modify the objective function or some other part of the FF algorithm to achieve better performance compared to BP in higher-dimensional datasets like CIFAR10,

## References


[1] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, 'Communication-efficient learning of deep networks from decentralized data', Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, vol. 54, 2017.

[2] M. Ghader, B. Farahani, Z. Rezvani, M. Shahsavari, and M. Fazlali, 'Exploiting Federated Learning for EEG-based Brain-Computer Interface System', in 2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS), 2023, pp. 1–6.

[3] D. P. Pau and F. M. Aymone, 'Forward Learning of Large Language Models by Consumer Devices', Electronics, vol. 13, no. 2, 2024.

[4] G. Hinton, 'The Forward-Forward Algorithm: Some Preliminary Investigations', 2022.

[5] H.-C. Lee and J. Song, 'SymBa: Symmetric Backpropagation-Free Contrastive Learning with Forward-Forward Algorithm for Optimizing Convergence', 2023.

[6] N. Tosato, L. Basile, E. Ballarin, G. de Alteriis, A. Cazzaniga, and A. Ansuini, 'Emergent representations in networks trained with the Forward-Forward algorithm', ArXiv, vol. abs/2305.18353, 2023.

[7] S. Gandhi, R. Gala, J. Kornberg, and A. Sridhar, 'Extending the Forward Forward Algorithm', ArXiv, vol. abs/2307.04205, 2023.

[8] D. P. Pau and F. M. Aymone, 'Suitability of Forward-Forward and PEPITA Learning to MLCommons-Tiny benchmarks', in 2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS), 2023, pp. 1–6.

[9] S. Park, D. Shin, J. Chung, and N. Lee, 'FedFwd: Federated Learning without Backpropagation', arXiv [cs.LG]. 2023.

[10] P. Ramachandran, B. Zoph, and Q. Le, 'Swish: a Self-Gated Activation Function', 10 2017.

[11] L. Deng, 'The mnist database of handwritten digit images for machine learning research', IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141–142, 2012.

[12] H. Xiao, K. Rasul, and R. Vollgraf, 'Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms', ArXiv, vol. abs/1708.07747, 2017.

[13] A. Krizhevsky, V. Nair, and G. Hinton, 'Cifar-10 (canadian institute for advanced research)'. 2010.