ISSN 1975-4736 ©MITA2023

DFF: Distributed Forward-Forward Algorithm for Large Model Training in Low-Performance Devices

Qikang Deng, Guang Jin, Yohwan Noh, Subin Yu, DoHoon Lee
Department of Information Convergence Engineering
Pusan National University
Busan, Korea
{dengqikang, geemguang, ubseeyou, sea1498, dohoon}@pusan.ac.kr

Abstract—**The advancement of Artificial Intelligence(AI) technology, especially with the success of Large Language models(LLMs) like ChatGPT and image generation models such as Stable Diffusion, signifies that AI has entered a new era of large-scale models. However, the rapid development of AI models has brought about the disaster of unlimited growth in model size such as the latest LLM model, PanGu-Sigma, which has reached an astonishing size of 1085B. For ordinary individuals, the enormous hardware costs for these large-scale models make it impossible to train and infer a large model. Fortunately, Geoffrey Hinton has proposed the Forward-Forward(FF) algorithm, which aims to deconstruct a large model into smaller layers and each layer updates its weights through its own backpropagation process, which means that each layer is independent. Therefore, inspired by Geoffrey Hinton's work, this paper introduces a new Distributed Forward-Forward(DFF) algorithm, which distributes each layer of the Forward-Forward algorithm model to different devices based on ROS2's publisher-subscriber communication mechanism. Then, the model can be computed across different devices. The Distributed Forward-Forward(DFF) Algorithm System in this paper theoretically allows multiple low-performance devices to jointly run large models that a single device is hard to run. Classification model tests on MNIST and CIFAR10 datasets show that the accuracy is 0.9292 and 0.4249 respectively, which has the comparable performance of the pure forward-forward algorithm (0.9315 and 0.4310). Our implementation is available at [Github].**

Keywords—Distributed AI model, Forward-Forward algorithm, ROS2 **publisher-subscriber*****, MNIST, CIFAR10***

I. Introduction

Since the explosive development of Artificial Intelligence Generated Content(AIGC) models in 2022, the demand for GPU capacity has been increasing. A large AIGC model may require tens, hundreds, or even thousands of gigabytes of GPU memory, which is unimaginable for ordinary users. However, the forward-forward(FF)[1] algorithm, proposed by Geoffrey Hinton, splits the model into layers, and each layer can learn and perform backpropagation independently. Meanwhile, it means these layers can be run on different devices independently.

Therefore, We propose a Distributed Forward-Forward (DFF) Algorithm System based on the publisher-subscriber communication mechanism of ROS2. DFF enables the deployment of independent layers of the FF model on various devices, allowing each layer to train and infer on its respective device. The computed results of each layer are transmitted to the required layers through ROS2 pub/subs system.

The DFF system is composed of five parts: service creation, model layers distribution, training, inference, and model weight saving. These parts allow a FF model to be distributed to multiple devices for training, inference, and saving thereby sharing their computing resources to run the large model. Observing the test results on MNIST[2] data and CIFAR10[3] dataset shows that the accuracy of the model with DFF is 0.9292 for MNIST and 0.4249 for CIFAR10, which is essentially the same as the FF (0.9315 and 0.4310).

II. Related words

FF algorithm is inspired by Boltzmann machines[4] and Noise Contrastive Estimation[5]. It replaces the forward and backward passes of backpropagation with two forward passes. Positive data and negative data are separately forward passed the model to train a FF model. FF algorithm is an independent layer learning method that aims to improve the training efficiency of neural networks. FF algorithm has shown promising results on small problems like MNIST handwriting digital classification, encouraging further investigation and research. For example, The SymBa[6] algorithm, based on the FF algorithm, addresses the issue of asymmetric gradients and loss of class information during training which has improved the convergence behavior and performance of the FF algorithm. Moreover, FFCL[7] proposed a new Contrastive Learning method based on the FF algorithm, approaching outperforms in pneumonia classification of chest X-ray images. However, previous research attempted to improve the FF algorithm performance and run the model on a single device and omitted the independent layer learning characteristics of the FF algorithm. In this paper, based on the concept of independent layer learning from the FF algorithm, we explored the possibility of distributing a model to multiple devices based on the FF algorithm.

III. Methodology

In general, the BP algorithm is the common algorithm to create and train a model as shown in Figure 1. a), during the

ISSN 1975-4736 ©MITA2023

training, input training data is forward passed the whole network from the first layer to the output layer. Then, the loss function calculated the loss between the prediction and ground truth. After that, the loss information is propagated back from the output layer to each layer of the model to update the weights and biases. Due to the backpropagation processes, the model is unable to be split into independently local trainable layers. However, the FF algorithm enables local training and introduces the concept of goodness metrics for model training and inference as shown in Figure 1. b). Moreover, In this paper, we leverage the characteristics of the FF algorithm to attempt the deployment of the original FF model, which runs on a single device, across different devices as shown in Figure 1. c), thus verifying the possibility of allocating large models to multiple low-performance devices.

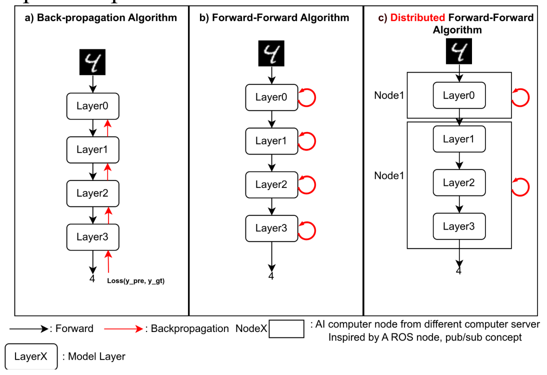
a) Back-propagation Algorithm
b) Forward-Forward Algorithm
c) Distributed Forward-Forward Algorithm
Layer0
Layer1
Layer2
Layer3
Loss(y_pre, y_gt)
Node1
: Forward
: Backpropagation
NodeX
: AI computer node from different computer server Inspired by A ROS node, pub/sub concept
LayerX
: Model Layer

Figure 1. a) Back Propagation, b) Forward-Forward, and c) Distributed Forward-Forward Algorithm.

The DFF algorithm system comprises five components, depicted in Figure 2.

- Part 1: Service creation
- Part 2: Model layers distribution
- Part 3: Training
- Part 4: Inference
- Part 5: Model Weight saving

A. *Service creation*

To start using the DFF algorithm system, a couple of ROS server nodes and a ROS master node should be created as in Figure 3. The main capability of the server node is to use ROS services to listen for layer creation requests and, based on the information in the requests, build layer instance and their corresponding ROS subscribers and publishers. Additionally, the server node also has the ability to delete layers. When the server node receives a request to delete a model, all layers distributed in the node that is associated with that model will be deleted.

Correspondingly, the main functionality of the master node is to read the layer configuration file and model configuration file, send layer creation requests to the service nodes based on configuration files, and process the training and inference results of the model. Basically, the master node acts as the controller for distributing model layers, initiating model training, and performing model inference.

Besides, when we initialize a server node, the server node will scan the created server nodes and try to find out the latest index *idx* of the created server nodes. Then the new server node will be created following the created server nodes using the name *Server_{idx+1}*. The creation of the master node is very similar to that of the server node. It also requires scanning the created server nodes. However, the purpose of scanning all the server nodes for the master node is to record all the server nodes and provide a server node list for the next step of allocating model layers.

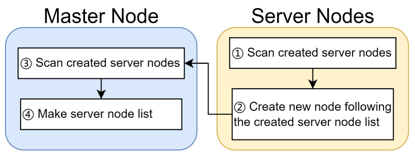
Master Node
Server Nodes
③ Scan created server nodes
④ Make server node list
① Scan created server nodes
② Create new node following the created server node list

Figure 3. Service creation: when we initialize a service node, we need to scan all existing nodes to get the necessary information.

B. *Model layers distribution*

In Section A, we mentioned that the master node constructs a layer distribution list by reading the layer configuration file and the model configuration file.The layer configuration file defines the input and output of each layer type in the DFF algorithm. The inputs correspond to the topic name and msg type of the layer's ROS2 subscriber, while the outputs correspond to the ROS topic name and msg type. This is shown in Code 1. Furthermore, the model configuration file is similar to the format used in YOLOv5[8], and it defines the model architecture and other hyperparameters. This format is illustrated in Code 2.

Thus, In this section as shown in Figure 4, the master node automatically converts the model configuration into the layer distribution format required by the server node using the layer and model configuration files. The layer distribution information, as shown in Code 3, is sent to each server node following a certain rule. Once a server node receives a layer

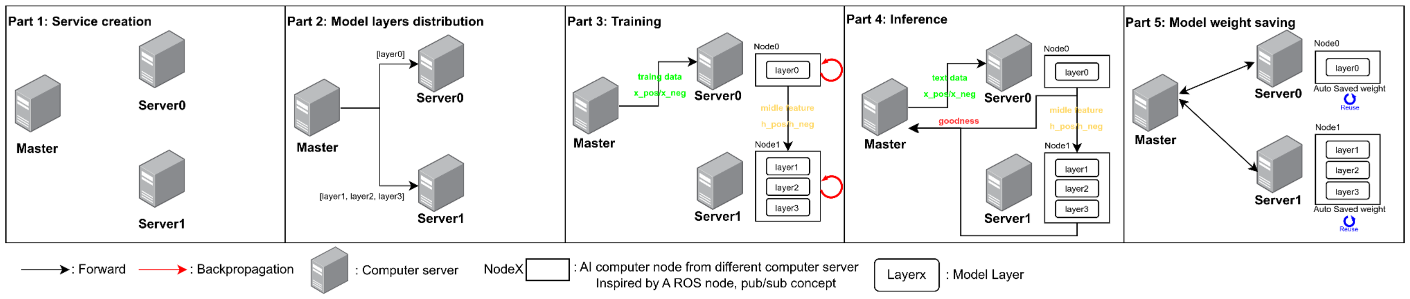

Figure 2. Distributed forward-forward algorithm system

distribution request, it creates the corresponding layer instance and the associated publishers/subscribers based on the layer distribution request. Lastly, the server node sends a confirmation message to the master nodes to indicate the success or failure of the layer creation.

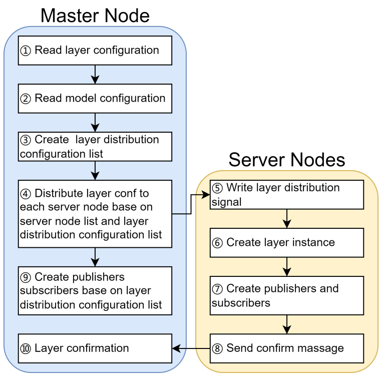
Master Node
① Read layer configuration
② Read model configuration
③ Create layer distribution configuration list
④ Distribute layer conf to each server node base on server node list and layer distribution configuration list
⑨ Create publishers subscribers base on layer distribution configuration list
⑩ Layer confirmation
Server Nodes
⑤ Write layer distribution signal
⑥ Create layer instance
⑦ Create publishers and subscribers
⑧ Send confirm massage

Figure 4: Model layers distribution: The master node creates model distribution information and makes a layer creation request to the service node.

C. Training

Training is the core part of this DFF algorithm system shown in Figure 5. The master node will load the training data, send the training data to the next layer, and wait for the training results of the layers in each server node while gathering necessary data for the latest necessary processing. After the model training is completed, additional necessary processing is performed. Meanwhile, after the master node sends out the training data, the server nodes sequentially receive the training results of the previous layer and start training the local layer. Once a layer training is completed, the server node sends the training result to the next layer until the final output layer. The result data that the master node needs to gather during training can be defined in the model configuration file.

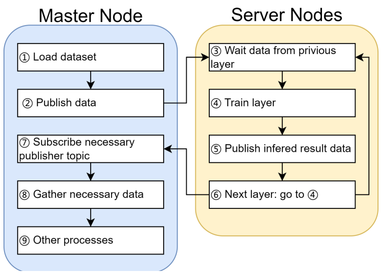

Figure 5: Training: The master node sends training data to each layer and gets the training results

D. Inference

The inference part is almost identical to the model training part. However, due to the diversity of models, the output of each layer during inference may differ from the output during training. Therefore, we define a separate set of twin processes for training and inference, as shown in Figure 6.

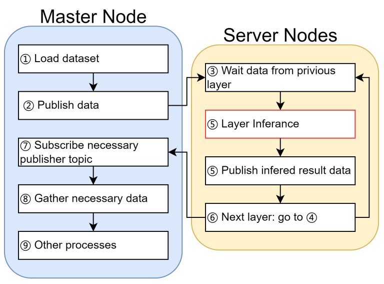

Figure 6: Inference: Similar to training but just do the inference process.

E. Model weight saving

The model weights are automatically stored in the local device of the server node. If the model is redistributed to a server node, the server node will check if the corresponding layers' weights of the model already exist in its local device. If they exist, the server node can reload the weights of the layers from the previous training session. Therefore, The pre-trained models can be reused effectively.

IV. Experience

A. MNIST Supervised Learning

Referring to Hinton's paper, when training the FF algorithm model, we need to construct positive data and negative data. We create training data by substituting the first 10 pixels of an image with a one-hot label. If the label is correct, it becomes positive data; if the label is incorrect, it becomes negative data, as shown in Figure 7. During each training iteration, both positive data and negative data are fed into the model simultaneously. Each layer of the model calculates its own goodness, where the goodness value for positive data should be very high, while the goodness value for negative data should be very low. By applying this rule, we compute the loss value and use the backpropagation method to train the weights and biases of each layer.

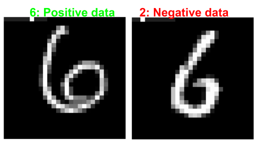

Figure 7: Forward-Foward algorithm Supervised Learning positive data and negative data

During the inference stage, the model only needs to aggregate the goodness values for the image under each label after passing through all layers. The label corresponding to the largest goodness value is then considered as the predicted result, as shown in Figure 8.

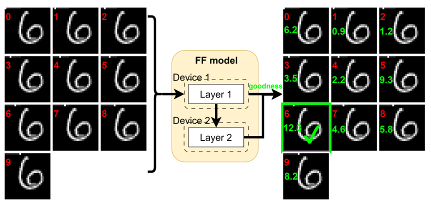

Figure 8: Forward-Forward algorithm model inference, find the highest goodness(the green number is goodness)

In this article, we deploy two dense layers on two separate server nodes for training. The results obtained are equivalent to those computed by the FF algorithm on a single device, as shown in Table 1.

Table 1. MNIST Accuracy of FF and DFF

	FF	**DFF**
Accuracy	**0.9315**	**0.9292**

B. CIFAR-10 Unsupervised Learning

The CIFAR-10 FF classifier is divided into two parts: a contrastive model for unsupervised learning, and a classification model for supervised learning. In the first part, the contrastive model also needs to construct positive and negative data. However, the negative data in CIFAR-10 is composed of hybrid images formed by multiplying one image with a mask and another image with the reverse of the mask, as shown in Figure 9. The mask is created by repeatedly blurring an image and applying a threshold of 0.5 to divide the blurred image pixels into values of 0 and 1.

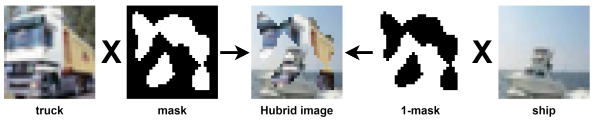

Figure 9: Negative data(Hybrid image) for negative data

By training the contrastive model with the constructed positive and negative data, as shown in Figure 10, the contrastive model learns the features of positive data. Then the features inferred by the contrastive model from positive data are passed to the second part, the classification model. The classification model is a regular supervised model, taking input from the contrastive model and outputting classification results. The results of the DFF algorithm system and the pure FF accuracy algorithm are as follows in Table 2.

Table 2. CIFAR-10 Accuracy of FF and DFF

	FF	**DFF**
Accuracy	**0.4310**	**0.4249**

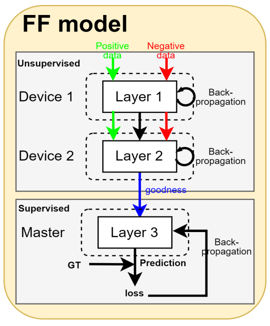

Figure 10: CIFAR-10 Unsupervised Learning model

V. Limitation

The DFF algorithm system has validated the possibility of training and inferring models across multiple devices. However, there are still many limitations compared to using the DP algorithm.

Data transmission time consumption: As the DFF algorithm system utilizes the publisher-subscriber mechanism of ROS to transmit input and output data of layers between various devices, there is a significant time overhead involved in the data transmission across the network.

Date reconstruction: Besides, only basic float arrays can be used to pass data when using the ROS publisher-subscriber for data transmission. This requires frequent reconstruction of data between the msg type specified by the ROS topic and the tensor data required by the model, resulting in additional CPU and memory overhead.

VI. Conclusion and Future work

In this paper, Although our experiments focused on small-size models, we anticipate that as the FF algorithm progresses, larger models will become applicable to both the FF algorithm and the DFFA algorithm system. Additionally, we acknowledged the current limitation of the DFF algorithm system in slow data transmission between devices. To address this, future work should prioritize the development of a system for data transmission. We highlighted the benefits of the FF algorithm in distributing models into local self-training layers, enabling the training and inference of large models on diverse devices. Building upon the FF algorithm, we successfully demonstrated the effectiveness of the DFF algorithm system by distributing a model across multiple ROS nodes.

References

[1] Hinton, G. (2022). The forward-forward algorithm: Some preliminary investigations. arXiv preprint arXiv:2212.13345.

[2] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.

[3] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[4] Montúfar, G. (2018). Restricted boltzmann machines: Introduction and review. In Information Geometry and Its Applications: On the Occasion of Shun-ichi Amari's 80th Birthday, IGAIA IV Liblice, Czech Republic, June 2016 (pp. 75-115). Springer International Publishing.

[5] Smith, N. A., & Eisner, J. (2005, June). Contrastive estimation: Training log-linear models on unlabeled data. In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05) (pp. 354-362).

[6] Lee, H. C., & Song, J. (2023). SymBa: Symmetric Backpropagation-Free Contrastive Learning with Forward-Forward Algorithm for Optimizing Convergence. arXiv preprint arXiv:2303.08418.

[7] Ahamed, M., Chen, J., & Imran, A. A. Z. (2023). Forward-Forward Contrastive Learning. arXiv preprint arXiv:2305.02927.

[8] Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., Kwon, Y., Michael, K., et al.. (2022). Ultralytics/yolov5: v7. 0-YOLOv5 SotA realtime instance segmentation. Zenodo.