CSE 598: Data Intensive and Machine Learning System

Forward Gradient in a Distributed Setting

Written By

Chirag Jitesh Rana, Yukta Sarode
Arizona State University

Abstract

The purpose of this project is to study and analyze the performance of using Forward Gradient in a distributed setting. In order to assess the tradeoffs, we have executed the algorithm to evaluate its suitability through benchmarking. We found that the batch size played a significant role in the model performance and theorize that it is due to the nature of the algorithm which hurdles with the randomized perturbation vector. We also claim that the forward gradient's convergence reduces significantly with an increase in the complexity of the weights. Traditional methods, like noise addition, have accuracy trade-offs. Forward gradient claims to eliminate backpropagation and offers significant advantages in computation and time complexity. The project has huge significance as it has wide-ranging applications, from virtual keyboards to processing private information on the client side. The algorithm has promising future scope for improving privacy and accuracy in machine learning.

Relevant keywords*: *Forward Gradient, Distributed Machine Learning, Jacobian Vector Product Aggregation.

Problem Statement

The efficient computation of the forward gradient is a critical component in performing Deep learning mode without backpropagation. However, in a distributed setting, where data is spread across multiple nodes, the computation of the forward gradient becomes challenging due to communication and synchronization overheads. Existing solutions for computing the forward gradient in a centralized setting display good convergence on simple algorithms but real-life distributed machine learning systems accommodate the use of complex architectures. Therefore, the problem that this project aims to address is to understand the tradeoff and possible working of the forward gradient in a distributed setting, which can lead to faster and more accurate optimization results along with securing the privacy of the training data. Our objective is to build a necessary framework architecture for performing forward gradients algorithm in a distributed setting and analyze the system using Convolutional Neural Networks (CNN) and Multilayer Neural Networks (MNN). We have chosen these 2 models as complex and simple parameters to be updated by the optimization algorithm. The objective also includes using different merging algorithms for the Jacobian vector product (jvp) while training and how the forward Gradient algorithm performs in comparison to the Back Propagation algorithm.

Motivation

Machine learning models trained on sensitive data can pose privacy risks if the data is leaked or accessed by unauthorized parties. One solution to this problem is to use distributed machine learning, where data is distributed across multiple devices or servers and processed locally, without ever leaving the user's control. This approach can help preserve privacy while still allowing for effective machine learning. Distributed machine learning algorithms use techniques such as federated learning or secure multi-party computation to train models across multiple devices without sharing the data. This enables users to retain control over their data and keep it private. In federated learning, a central server coordinates the training of a model, but the actual training occurs on local devices. Moreover, some differential privacy implementations include the use of noise in the model parameters which reduces the performance of the model drastically. In secure multi-party computation, multiple parties collaborate to compute a function without revealing their inputs to each other.

Another advantage of distributed machine learning is that it can improve model accuracy and performance. By training models on larger, more diverse datasets, distributed machine learning algorithms can produce more accurate models than those trained on a single, centralized dataset.

Forward gradient is a popular optimization algorithm that has several benefits in terms of optimization and computational complexity. One advantage of a forward gradient is that it is computationally efficient, as it only requires the computation of the gradient in the forward direction. This can lead to faster convergence and lower computational complexity compared to other optimization algorithms. Another benefit of the forward gradient is that it is suitable for large-scale optimization problems, as it can be easily parallelized and distributed across multiple machines. This allows for efficient computation of the

gradient on large datasets and can lead to improved scalability and faster training times.

Forward gradient also has the benefit of being easily adaptable to non-linear optimization problems and can be used in conjunction with other optimization techniques such as conjugate gradient or Newton's method to further improve optimization performance.

There are several applications where these models can be used for training such as:

1. Healthcare - Healthcare data is often sensitive and subject to privacy regulations, making it difficult to centralize and process. Distributed Machine learning Techniques can be used to train models on data from multiple healthcare providers, leading to improved disease detection and diagnosis, personalized treatment plans, and more efficient drug development, all while maintaining privacy.

2. Internet of Things (IoT): Distributed machine learning training can be used to train models on this data, leading to more efficient and effective device management, a better understanding of user behavior, and more intelligent automation.

3. Financial Services: Distributed Machine learning can be used to train models on this data while maintaining privacy, leading to improved fraud detection, risk management, and investment strategies.

4. Autonomous Vehicles: Distributed machine learning can be used to train and deploy ML models in a decentralized manner, enabling autonomous vehicles to operate safely and efficiently in complex environments.

Moreover, we had a personal motivation of understanding the working of Distributed Machine learning using different algorithms and got to learn more about Federated Learning, understand the working of different optimization algorithms like Backpropagation, Forward Gradient and think about the impact of such amazing model architectures in the industry especially in bringing personalized training on multiple devices.

Literature Survey

Gradients Without Backpropagation[1]: In this paper, the authors have introduced a new method by eliminating the use of backpropagation for the calculations of gradients. Figure 1 consists of the algorithm proposed by the authors of the paper. Firstly we need to initialize the parameter space for the calculations of the gradients. In (FGD)**,** we deal with a perturbation vector which is a randomly generated vector based on Normal Distribution. We then compute $f(\theta)$ and $J_f(\theta_t).v$ using the Forward Mode AD[4] where $J_f(\theta_t)$ is the Jacobian Vector Product and $\nabla f(\theta_t) \cdot v$ is the directional derivative is denoted as (d_t). These values are then used for the calculations of the gradients (g_t) by multiplying with the perturbation vector. Later in

Algorithm 1 Forward gradient descent (FGD)

Require: η: learning rate
Require: f: objective function
Require: $\boldsymbol{\theta}_0$: initial parameter vector
  $t \leftarrow 0$ ▷ Initialize
  while $\boldsymbol{\theta}_t$ not converged **do**
    $t \leftarrow t + 1$
    $\boldsymbol{v}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ ▷ Sample perturbation
    Note: *the following computes f_t and d_t simultaneously and without having to compute ∇f in the process*
    $f_t,\ d_t \leftarrow f(\boldsymbol{\theta}_t),\ \nabla f(\boldsymbol{\theta}_t) \cdot \boldsymbol{v}$ ▷ Forward AD (Section 3.1)
    $\boldsymbol{g}_t \leftarrow \boldsymbol{v}_t d_t$ ▷ Forward gradient
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\, \boldsymbol{g}_t$ ▷ Parameter update
  end while
  return $\boldsymbol{\theta}_t$

Fig 1. Forward Gradient Descent Algorithm

is the next step, the gradients are then used for updating the parameters. To understand the implementation of the algorithm, we made use of the "fwdgrad" library built by Orobix, an AI service company[3]. We also found a few bugs in their implementation and raised issues on their GitHub page about the resolution.

In our project, we have used the MNIST dataset[5]. It is a well-known benchmark dataset used in the field of machine learning for image recognition tasks. It consists of a set of 70,000 handwritten digits that are each represented by a 28x28 pixel grayscale image. The digits are divided into a training set of 60,000 examples and a test set of 10,000 examples. The MNIST dataset has been widely used as a standard benchmark for evaluating image recognition models, particularly for training and testing simple feedforward neural networks. Because the dataset is relatively small and easy to preprocess, it is often used as an initial step in the development of more complex models or for testing new algorithms. Despite its simplicity, the MNIST dataset remains a useful resource for machine learning researchers and practitioners and has helped to advance the field of deep learning over the years. It has also inspired the development of other benchmark datasets for image recognition tasks, including CIFAR-10, ImageNet, and COCO.

For model training we have used Convolutional Neural Network CNN[6] and Multilayer Neural Network (MNN) for the comparison of complex and simple Machine learning algorithms. Convolutional Neural Networks (CNNs) are a class of deep learning models that are widely used in computer vision tasks such as image and video recognition, object detection, and image segmentation. CNNs are designed to automatically learn and extract relevant features from input images, by applying a series of convolutional filters to the input data. In a typical CNN architecture, the input image is passed through a series of

convolutional layers, which apply a set of learned filters to the input data. The output of each convolutional layer is then passed through a non-linear activation function, such as ReLU, to introduce non-linearity into the model. This process is repeated for several layers, with the output of each layer serving as the input to the next layer. CNNs are also often combined with pooling layers, which downsample the output of each convolutional layer to reduce the spatial dimensions of the feature maps. This helps to reduce the computational complexity of the model and also makes it more robust to variations in the input data.[7]

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Fig 2. Accuracy Formulae

To analyze the performance of the model, we have used accuracy as the metric and considered the time to benchmark the results in comparison with Backpropagation.

Proposed Solution

In order to solve the privacy issue and work on improving the model accuracy in a distributed setting, we found that if the server and the clients had the same random generator seed value then the client would only need the (d_t) i.e. the directional derivative for generating the final gradients. Moreover, the clients would use the updated model parameters from the server for the next iteration. As the seed would be shared securely between the clients and server, there would be no way any third-party attacker would be able to generate the gradients to back-calculate the data used for training, and hence privacy would be ensured.

Steps:

1) Server Generates the Model Parameters.
2) Distribute the model params and seed with clients.
3) Clients would generate a random perturbation vector.
4) Use the data on the client's end to train the model.
5) Generate the Jacobian vector Product (jvp) and return.
6) Server would generate the perturbation vector (v_t).
7) Aggregate the vector (jvp) using different merge algos.
8) Multiply (v_t) with (jvp) to generate the gradients.
8) Update model parameters using the gradients.
9) Test the model and repeat step 2.

The aggregation step 7 includes the use of 3 merging algorithms that we have implemented in this project.

1) Naive Method: This is very similar to normal centralized training. In this approach, the received (jvp) would be used one by one to update the model parameters from every client.
2) Simple Average: In this approach, the mean value of the returned jvp is taken into consideration for a 1-time model update.
3) Weighted Average: Here, every client is given a weight based on the number of data points it has used for training in a batch. [Note: We have considered this weight as the batch size in the initial implementation lead us to have significant differences in the convergence speed of the model]

Usual federated methods include training the model on the client's end for several epochs and returning the final trained model to the server. However, in this project, we have tried a different approach to understand its working and reliability for convergence.

To build the framework, we have used the PyTorch framework for coding and Tensorboard for analyzing the results and convergence.

Experimental Setup

Our experimental setup involves a distributed system with clients and servers. We are assuming the clients already have their personal training data and model. Similarly, the server also has the model. We have used google colab for the development. We simulated a distributed environment by using a server class and making multiple objects of the client class. To replicate a distributed environment, we first distributed the data after shuffling the number of clients. During training, the clients ran one after another using the model parameters available with the server and appended the (jvp) vectors in an array. To consider the real-world scenario of parallel processing we have considered the mean time the clients took while training at each epoch. After receiving the vectors, aggregation algorithms were used to calculate the gradients and update the model parameters.

System Architecture

We have used 10 client objects for the proof of concept and there is 1 server object as shown in Figure 3. The system architecture was designed to facilitate the processing of large datasets. We utilized PyTorch and TensorBoard to implement various deep learning models like Convolutional Neural Networks (CNN) and Multilayer Neural Networks (MNN) to analyze image data. To enhance the processing power of our system, we utilized an NVIDIA Tesla T4 GPU provided by Google colab. We have also used a seed value of 48 to generate the same perturbation vectors between the server and the client.

Model and Dataset

We have used 2 models: convolutional neural networks (CNN) and Multilayer Neural Networks (MNN). The CNN model has four convolution layers with size 64, two max-pooling layers, one flatten layer, and one dense layer and we have used the Relu activation function. For MNN: There are two dense layers with size 1024, and the activation function is Relu. We have used the seed value as 48. The learning rate is constant (0.001).

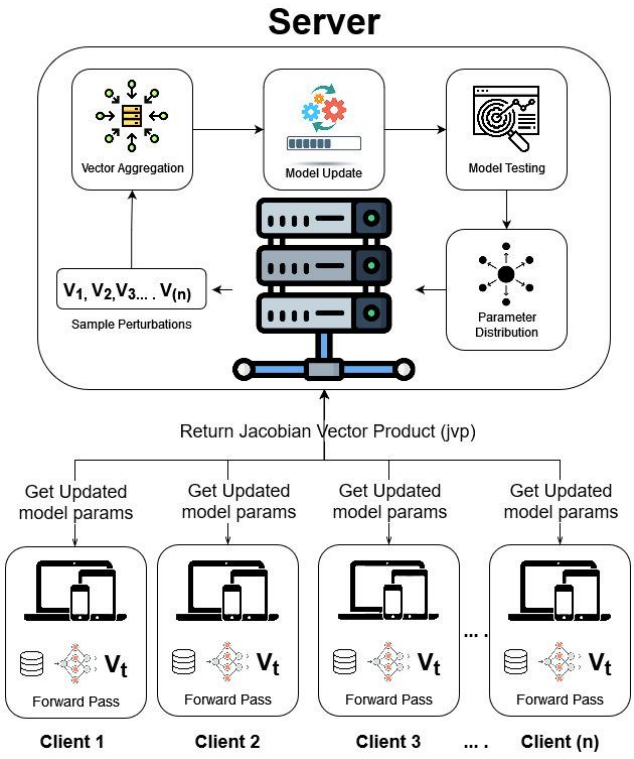


Fig 3. System Architecture Diagram

For training and testing data we have used the MNIST dataset. The training data has 60000 images and the testing data has 10000 images. After a lot of trial and error, we divided the dataset into a batch size of 64 images for each client. We also performed the training using 6000 images as a batch size while training and no momentum in convergence. At the same time while reducing the sample size we found some momentum and finalized the batch size as 64 for efficiently loading the data on the GPU.

Results

We compared the performance of backpropagation and forward gradient.

Centralized Back Propagation

	Test Accuracy	Time
CNN	0.8839	1hr 4 min
MNN	0.8763	53.23 min

Centralized Forward Gradient

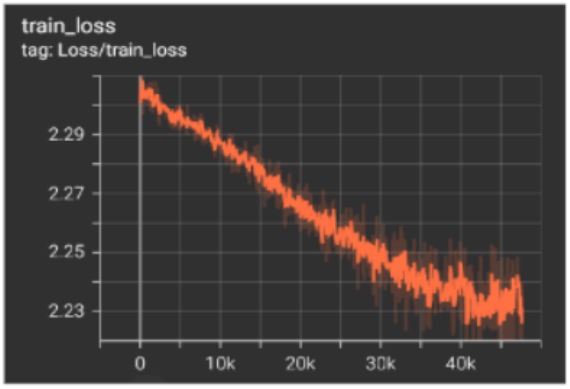
	Test Accuracy	Time
CNN	0.6474	59.23 min
MNN	0.8823	57.64 min

De-centralized Forward Gradient

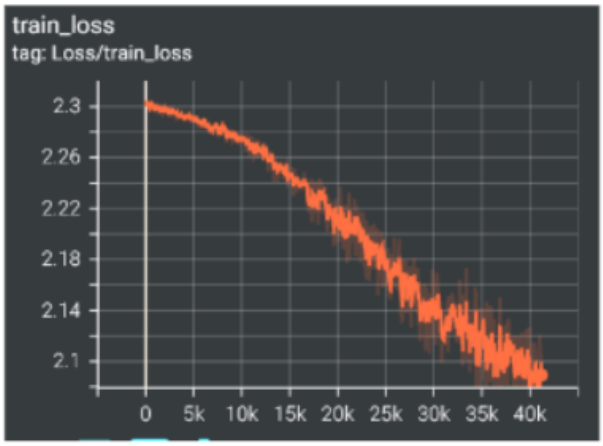
	Test Accuracy	Time
CNN	0.4821	1hr 17 min
MNN	0.5016	1 hr 15 min

What we found was that the accuracy for decentralized methods is way less than for the centralized method and even the training time required is more. That is because of the high communication overhead of saving and loading the model parameters on a pickle file. If we increase the number of clients we can increase the accuracy and reduce the comparative time taken for training by scalability. Between backpropagation and forward gradient, no significant speedup was observed. However, the accuracy of CNN has decreased significantly as the model is highly complex to converge. While MNN performed better because it was less complex.

We also took a look at the loss of CNN and MNN in a forward gradient under a weighted average setting. The results are as follows:



CNN Forward Gradient



MNN Forward Gradient

We can observe from the loss functions that CNN reached the plateau of convergence at a higher loss value of around 2.23 while the MNN model is still converging to a lower value of the loss.

Parameter Aggregation in Decentralized Setting

CNN:

	Time	Accuracy
Naive Method	59.31 min	0.471
Weighted Average	1 hr 17	0.485
Average	59. 42 min	0.482

MNN:

	Time	Accuracy
Naive Method	57.49 min	0.497
Weighted Average	1 hr 15 min	0.519
Average	57.22 min	0.502

From the results of parameter aggregation in a decentralized setting, we found that the weighted average performs better with the highest accuracy but the time required is also more.

Analysis

We found that the batch size affects the accuracy of the model drastically. The model performed better with smaller batch sizes and converged faster. When a batch size of 60000 images was used the model took a lot of time to converge, while a batch size of 64 images performed much better and converged relatively fast.

The next observation was that CNN took more time to converge than MNN. Since the model is based on the randomization of perturbation vectors, a complex model like CNN would bring more noise in the directional derivative while converging. This is because while using forward gradient the no.of parameters in CNN are more as compared to MNN. Moreover, our implementation involved merging (jvp) from different clients by iterating the data once. This leads the model to give poor performance by including wider information.

Furthermore, in a centralized setting, we also found that the convergence is slower on the Forward gradient as compared to backpropagation in both models.

Conclusion

We implemented forward gradient Descent in a distributed setting. Initially, we faced some difficulties while setting up the distributed environment and converging the model. With the use of constant learning rates and appropriate batch size, we were able to converge the model. With the help of aggregation methods, we found the tradeoffs between accuracy and time among the three methods.

For both centralized and decentralized versions, a significant speed optimization was not seen after using Forward Gradient. Which is in contrast to the paper[1]. However, the paper used less complex test functions to check the convergence speed and therefore we believe that forward gradient benchmarking highly depends on the model complexity. Further, including more clients could help in speeding the process but might also bring underfitting due to the wide amount of data.

We observed a high communication overhead due to extensive steps including loading and saving the model parameters. Since in our implementation, we have included sharing of model parameters by saving the model in a pickle file, it costs more communication overhead. However, in a different setting when the model is shared directly would reduce this overhead.

Future Work

One possible future work is to look at the possibility of lowering the high overhead communication costs. We can employ techniques like quantization, compressed sensing, differential privacy, and model parallelization to accomplish this. This could potentially reduce the quantity of data sent between clients and the server, lowering communication overhead.

Since the model performance of Forward Gradient Descent is proportional to the model complexity, research might be conducted to evaluate the tradeoffs by evaluating the model complexity and performance in the same experimental scenario.

Further, we can also use selective aggregation of parameters. In our algorithm, all the local model updates are aggregated on the server to compute the global model update. However, not all local updates are equally vital, and some may be unnecessary or noisy. Selective aggregation can be used to discover and aggregate only the most essential local model updates on the server, reducing communication overhead.

References

[1] Baydin, A.G., Pearlmutter, B.A., Syme, D., Wood, F. and Torr, P., 2022. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*.

[2] Werbos, P.J., 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), pp.1550-1560.

[3] OrobiX (2021). Forward Gradient Descent. GitHub. https://github.com/orobix/fwdgrad

[4] Shi, J. (2021, February 25). Automatic Differentiation (Part 1) - Introduction. Jingnan Shi's Blog. https://jingnanshi.com/blog/autodiff.html

[5] PyTorch. "TorchVision Datasets: MNIST." 2021. https://pytorch.org/vision/stable/datasets.html#mnist

[6] Bhogendra Sharma, Nutan. "Pytorch Convolutional Neural Network with MNIST Dataset." Medium, 21 May.2021, https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118.

[7] Nair, Nikhil. "Convolutional Neural Networks Explained." Towards Data Science, 28 Sep. 2020, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[8] Aldrich, Chris; Zeinolabedini Hezave, Ali; Ghiasi, Mohammad Mahdi "Multilayer Neural Networks." Encyclopedia of Chemical Processing, 2019, pp. 1-13, via ScienceDirect website: https://www.sciencedirect.com/topics/chemical-engineering/multilayer-neural-networks.