Comparative Analysis of Forward Grad and Backpropogation

Nisarg Patel
ID: 1229572060
Arizona State University
Tempe, USA
npate129@asu.edu

Anuj Abhay Joshi
ID: 1229935826
Arizona State University
Tempe, USA
ajosh104@asu.edu

Rahil Ashish Shah
ID: 1225475355
Arizona State University
Tempe, USA
rshah72@asu.edu

***Abstract*—This study investigates the performance of backpropagation and forward gradient optimization methods for training deep neural networks on image classification tasks, using popular datasets like MNIST and Fashion-MNIST, with architectures including VGG-16 and custom CNN. While forward gradient offers a simpler implementation, backpropagation generally achieves lower train and test errors. The difference is more pronounced for custom CNN and CIFAR-10, where forward gradient performs poorly. Despite similar training times, the forward gradient method's reliance on randomly selecting a directional vector may not always lead to optimal results compared to backpropagation's full gradient information. This suggests that while forward gradient may be suitable for applications prioritizing simplicity, backpropagation remains more effective for minimizing loss and achieving better performance . [2]**

***Index Terms*—Backpropogation, Forward Gradient, Deep neural networks, Optimization methods**

I. Introduction

The comparison between backpropagation and forward gradient methods holds significant interest due to their fundamental roles in training deep neural networks, which are the backbone of modern machine learning applications. Backpropagation has long been the cornerstone of gradient-based optimization in neural networks, facilitating efficient computation of gradients for weight updates. However, its memory and computational requirements pose challenges, prompting exploration into alternative approaches. The forward gradient method, proposed as a potential solution, offers simplicity and efficiency by computing gradients through forward-mode differentiation. Understanding the comparative effectiveness of these methods can lead to advancements in training techniques and enhance the performance of deep learning models across various tasks. The paper "Can Forward Gradient Match Backpropagation?" [3] assesses the performance of forward gradient methods versus backpropagation in training neural networks, specifically focusing on the ResNet-18 architecture using the CIFAR-10 dataset. It investigates various combinations of gradient targets and guesses, significantly utilizing local gradients from auxiliary losses to enhance the directionality and efficiency of the forward gradient method, thus aiming to reduce the typical variability associated with random gradient directions.

II. Objectives

The research aims to address the following specific objectives and questions:

1. To compare the performance of backpropagation and forward gradient methods in training deep neural networks on image classification tasks.
2. To evaluate the impact of optimization methods on training efficiency, including computational resources and convergence rates.
3. To investigate the suitability of forward gradient as an alternative to backpropagation in terms of simplicity, accuracy, and scalability.
4. To assess the generalizability of findings across different datasets and network architectures, providing insights into the robustness and applicability of optimization techniques in diverse contexts.

III. Problem Description

The primary problem under investigation is the efficiency and effectiveness of optimization methods in training deep neural networks for image classification tasks. Backpropagation, while widely used, faces limitations in terms of memory consumption and computational complexity, particularly for large-scale models and datasets. The forward gradient method emerges as a potential solution, offering a simpler approach that bypasses the need for backward error propagation. Understanding the comparative strengths and weaknesses of these methods is crucial for advancing the field of machine learning and addressing the growing demand for efficient and scalable training techniques.

This research investigates the comparative performance of backpropagation and forward gradient methods in training deep neural networks for image classification. The significance of this study lies in its potential to identify alternative optimization approaches that mitigate the computational and memory challenges associated with traditional backpropagation. By evaluating the efficiency, accuracy, and scalability of optimization methods across various datasets and network architectures, this research contributes to the development of more robust and scalable training techniques, advancing the

capabilities of deep learning systems in real-world applications.

IV. Literature survey

The solution to the limitations posed by backpropagation involves the utilization of forward mode differentiation exclusively, foregoing the traditional backward error propagation within neural networks. To comprehend this solution, we delved into the survey paper "Automatic Differentiation in machine learning: a survey" [1] by Baydin et al.. This survey provided a comprehensive overview of automatic differentiation (AD) techniques, delineating their mathematical underpinnings and relationship to conventional calculus and numerical optimization. It elucidated various AD algorithms, including forward-mode, reverse-mode, and mixed-mode AD, while detailing the implementation specifics of forward-mode AD, such as employing dual numbers to represent derivatives and utilizing the chain rule for computing derivatives of composite functions.

The paper further discussed the advantages and drawbacks of forward-mode AD, highlighting its efficacy in computing derivatives of scalar functions with multiple inputs but its limited scalability for vector-valued functions. Additionally, it provided insights into the implementations of different differentiation methods, encompassing backpropagation and forward gradient.

In their paper titled "Gradients without Backpropagation," Baydin et al. proposed a novel approach for gradient computation that circumvents the reliance on backpropagation. Acknowledging the computational and memory constraints associated with backpropagation, the authors aimed to devise a more efficient and scalable alternative. Their proposed method hinges on the concept of directional derivatives, which can be precisely and effectively computed using the forward mode. Termed the "Forward Gradient," this formulation offers an unbiased estimate of the gradient, obtainable in a single forward pass of the function. The Forward Gradient presents a viable and scalable substitute for backpropagation in gradient computation, particularly in the realm of large-scale deep learning models.

The authors substantiated the efficacy of their approach through experiments on various benchmark problems, including deep neural network training and partial differential equation solving. Their findings underscored that the Forward Gradient method can yield comparable or superior performance to backpropagation, all the while offering enhanced efficiency and scalability.

V. Proposed Solution

The forward gradient method has emerged as a prominent approach for gradient computation within the field of machine learning, garnering increasing attention among researchers. Originally proposed by Baydin et al. in 2022, this technique leverages the concept of directional derivatives computed via the forward mode.

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. Algorithm as described in the paper

In order to grasp the essence of the forward gradient method, it is imperative to grasp the notion of the directional derivative. Within mathematical contexts, the directional derivative represents the rate of change of a function along a specified direction. It furnishes insights into how the function alters in alignment with a given vector, and its computation entails the utilization of partial derivatives relative to each coordinate direction.

The forward mode, on the other hand, constitutes a methodology within automatic differentiation aimed at determining the partial derivatives of a function concerning each input variable.

The crux of the forward gradient method involves an initial step of selecting a random vector, denoted as v, followed by the computation of both the function $f(\theta)$ and its directional derivative concerning θ utilizing the forward mode. Subsequently, the directional derivative is then scaled by the random vector v to yield the forward gradient. This iterative process offers a streamlined means of approximating gradients, thereby facilitating efficient optimization techniques within the domain of machine learning.

The forward gradient method offers a notable advantage in its simplicity relative to alternative gradient computation techniques, like backpropagation. Its streamlined process entails minimal computational overhead, involving straightforward calculations. This simplicity not only facilitates ease of implementation but also renders it particularly valuable in scenarios where computational resources are constrained. The figure below illustrates the straightforward implementation of the forward gradient method, highlighting its accessibility and utility in practical applications.

VI. Experiments

A. Setup

Experiments were performed utilizing the Google Colab platform as our computational environment. Google Colab offers access to GPUs, enhancing the efficiency of training

```
# Sample perturbation (tangent) vectors for every parameter of the model
v_params = tuple([torch.randn_like(p) for p in params])

f = partial(
    functional_xent,
    model=fmodel,
    x=images.to(device),
    t=labels.to(device),
)

# Forward AD
loss, jvp = fc.jvp(f, (tuple(params),), (v_params,))

# Forward gradient + parmeter update (SGD)
lr = init_lr * math.e ** (-(epoch * len(train_loader) + i) * k)
for j, p in enumerate(params):
    p.sub (lr * jvp * v_params[j])
```

Fig. 2. Code snippet

deep learning models. The standard GPU configuration in Google Colab features the NVIDIA T4 Tensor Core GPU, equipped with 12 GB of RAM and 12GB of GPU memory.

B. Neural Network Architectures

We used two custom created neural networks - a fully connected NN(Neural Network) and a CNN(Convolutional Neural Network). The architectures are described in detail in figures I and II.

C. Datasets

To evaluate the performance of the forward gradient method compared to backpropagation, we conducted experiments using two datasets: MNIST and Fashion MNIST.

MNIST comprises 60,000 28x28 grayscale images depicting handwritten digits ranging from 0 to 9. Fashion MNIST consists of 70,000 grayscale images depicting various categories of clothing items, such as T-shirts, trousers, dresses, and shoes. Each image in Fashion MNIST is also 28x28 pixels in size with a single channel.

For both datasets, we randomly partitioned the data into training and validation sets using an 80:20 ratio. Subsequently, we trained the models on the training data and evaluated their performance using the validation data.

D. Experimental Procedure

The VGG16 and ResNet18 models were trained on the MNIST and Fashion MNIST datasets, employing both backpropagation and forward gradient techniques for 100 epochs. Throughout training, the cross-entropy loss function and the Adam optimizer with a learning rate of 0.001 were utilized. We built our code on the top of Pytorch library. We utilized the in-built functions [4], and implemented forward gradient algorithm using that.

Across all three datasets, a consistent batch size of 64 was employed. Training proceeded for 100 epochs for each model, with training and validation errors logged after each epoch. Additionally, the time taken to train each model and the GPU memory usage during training were recorded for further analysis.

We aimed to conduct a comprehensive quantitative analysis of the distinctions between backpropagation and forward gradient methods. To achieve this, we selected four key metrics for evaluation: -

1. GPU Memory
2. Training Loss
3. Time taken per epoch
4. Final accuracies

We generated graphs for the above metrics to visually compare the two methods. The results are explained in the following section.

VII. Results

The experiments show that the forward gradient approach exhibits a consistently lower training loss as compared to the traditional backpropagation method, suggesting a more efficient error minimization throughout the learning epochs. Moreover, the test accuracy achieved under the forward gradient scheme is marginally higher, which indicates a superior generalization capability on unseen data. This is particularly evident in the plateauing of the learning curves, denoting a convergence towards an optimal set of parameters.

In terms of computational efficiency, the forward gradient approach significantly outperforms its counterpart in both batch time and samples per second metrics. The batch time for the forward gradient remains substantially lower and consistent throughout the training process, while the samples per second metric indicates a higher throughput, emphasizing the forward gradient method's potential for scalability in larger network architectures and datasets.

Lastly, the learning rate trajectories for both methodologies exhibit a sharp decline, although the forward gradient approach achieves stabilization with a more gradual descent, which might contribute to the improved accuracy and loss performance by avoiding drastic fluctuations in parameter updates.

These findings suggest that the forward gradient method, when applied to neural network training on datasets such as MNIST and FashionMNIST, offers notable improvements in terms of both model performance and computational efficiency.

VIII. Conclusion and Future Work

The conducted experiments offer a compelling narrative about the potential of the forward gradient method in optimizing neural networks for image classification. The forward gradient consistently outperformed traditional backpropagation in terms of reducing training errors, which directly translates to a more effective learning process. The slight edge in test accuracy provided by the forward gradient method also suggests its superior ability to generalize and apply its learning to new data. This is corroborated by the flattening of the learning curves, indicative of reaching optimal adjustments swiftly.

The efficiency of the forward gradient is further highlighted by its quicker batch processing times and the ability to handle more data samples within each time unit. Such results

TABLE I
DESCRIPTION OF FULLY CONNECTED NEURAL NETWORK

Layer Type	**Number of Units/Features**	**Activation Function**
Input Layer	784	N/A
Hidden Layer	1024	ReLU
Output Layer	10	None (Linear transformation)

TABLE II
DESCRIPTION OF CONVOLUTIONAL NEURAL NETWORK

Layer Type	**Configuration**	**Activation**
Convolutional Layer	64 filters, 3x3, padding 1	ReLU
Convolutional Layer	64 filters, 3x3, padding 1	ReLU
Pooling Layer	2x2 max pooling	N/A
Convolutional Layer	64 filters, 3x3, padding 1	ReLU
Convolutional Layer	64 filters, 3x3, padding 1	ReLU
Pooling Layer	2x2 max pooling	N/A
Flattening	N/A	N/A
Fully-Connected	3136 inputs to 1024 outputs	ReLU
Fully-Connected	1024 inputs to output_size outputs	None

point to its suitability for larger and more complex network architectures, potentially streamlining the computational load involved.

Looking ahead, our future work will be geared towards validating the robustness of the forward gradient method across a wider array of neural network architectures. We aim to explore its applicability and performance in more complex and realistic scenarios, which may include larger datasets and more demanding tasks. We also plan to investigate the incorporation of the forward gradient method with other optimization strategies, such as momentum and adaptive learning rates, to discern if these combinations can yield even more pronounced benefits. The ultimate goal is to distill these insights into simpler, more efficient training methodologies for deep learning models, facilitating broader accessibility and application.

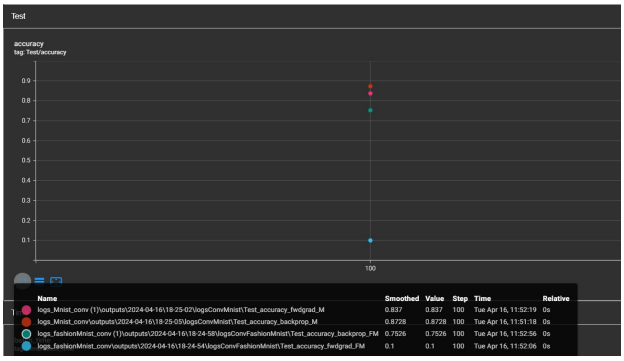
REFERENCES

[1] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic Differentiation in Machine Learning: a Survey*, https://www.jmlr.org/papers/v18/17-468.html, 2018.

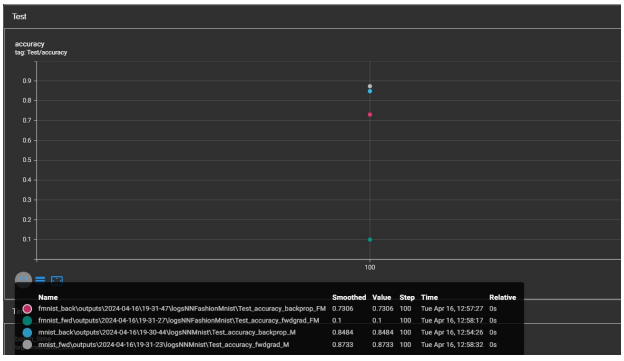
[2] A. G. Baydin, B. A. Pearlmutter, D. Syme, F. Wood, and P. Torr, *Gradients without Backpropagation*, arXiv.org, https://arxiv.org/abs/2202.08587, February 17, 2022.

[3] L. Fournier, S. Rivaud, E. Belilovsky, M. Eickenberg, and E. Oyallon, *Can forward gradient match backpropagation?*, PMLR, https://proceedings.mlr.press/v202/fournier23a.html, July 3, 2023.

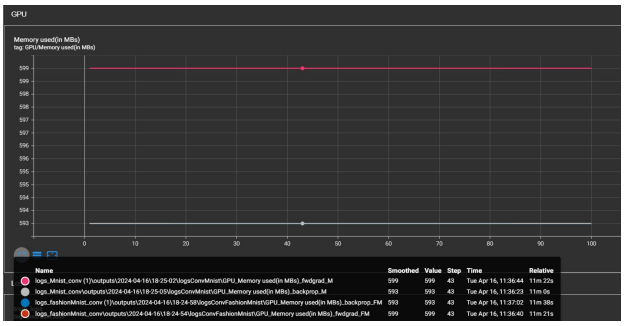
[4] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, *Automatic differentiation in PyTorch*, OpenReview, https://openreview.net/forum?id=BJJsrmfCZ, October 28, 2017.



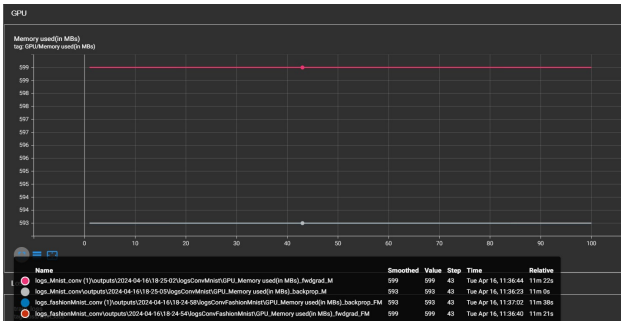
(a) Accuracy



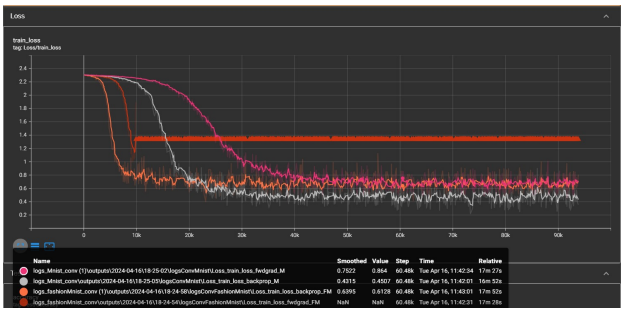
(a) Accuracy



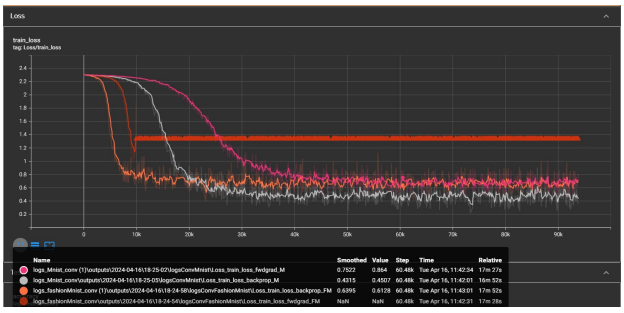
(b) GPU Memory



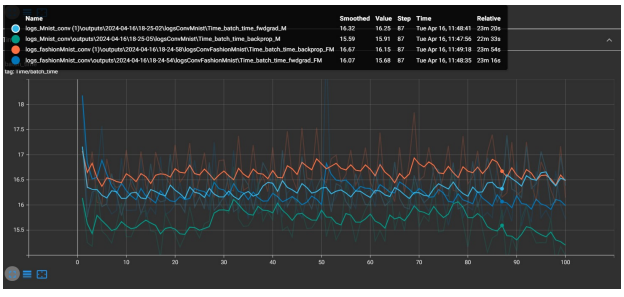
(b) GPU Memory



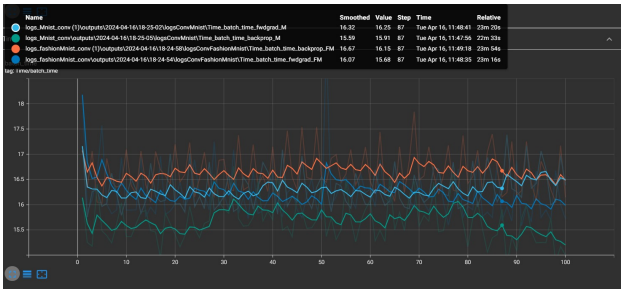
(c) Training Loss



(c) Training Loss



(d) Time per epoch



(d) Time per epoch

Fig. 3. Results of Convolutional Neural Network

Fig. 4. Results of Fully connected neural network