Testing forward gradients performance for Adam and SGD with momentum

Benjamin Rike, Olav Førland
École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

***Abstract*—Training deep natural networks is a resource- and time-consuming process. The networks are usually trained using backpropagation, a case of reverse-mode automatic differentiation that differentiates functions using a forward and backward pass. Baydian et al. [1] propose using forward-mode automatic differentiation to eliminate the backward pass and calculate an unbiased estimator of the actual gradient, the *forward gradient*. The authors show strong results where forward gradient trains twice as fast as backpropagation while reaching the same or better performance. In this paper, we investigate the authors' results, test whether they scale to more complex tasks, and explore the performance of the Adam optimizer and SGD with Nesterov momentum on the forward gradient. We find that the original paper uses too small learning rates for backpropagation, leading to unfair comparisons. Furthermore, our results indicate that backpropagation outperforms forward gradients and that forward gradient does not scale to increased complexity, regardless of the optimizer used.**

I. Introduction

Training neural networks (NNs) is a resource- and time-consuming process and novel methods are continuously proposed to reduce training time and improve accuracy. Gradient descent is the backbone of most modern optimization methods for neural networks. This simple, yet powerful concept computes the gradient of an arbitrary loss function with respect to the weights and biases of a NN, and takes a step in the opposite direction to minimize the loss. Most applications implement gradient descent by backpropagation. Backpropagation is a two-step algorithm that calculates the loss by passing the input through the network and then computes the gradients through a backward pass. Most software packages implement this with reverse-mode automatic differentiation, a member of the set of differentiation techniques called Automatic Differentiation (AD)[2].

To reduce the time per step, Baydin et al. [1], in the paper *Backpropagation without gradients*, suggest instead calculating an unbiased estimate of the gradient in only a forward pass using forward-mode AD. The authors claim that their implementation of forward gradients cuts the training time in half without sacrificing the quality of their model. To support their claims, the authors pairwise compare backpropagation and forward gradients for logistic regression, dense neural networks, and convolutional neural networks, all trained and tested on the MNIST dataset[3]. Their results show significant speedup and, in the case of dense neural networks, even decreased loss. If the results scale and prove valid, it can save significant resources and time. However, the experiments use the same learning rate for both backpropagation and forward gradients, which does not necessarily yield a fair comparison. Through this paper, we will therefore perform the following:

1) Repeat the original paper's experiments, but with optimized learning rates for each algorithm.
2) Investigate whether forward gradients are compatible with state-of-the-art optimization methods, in particular, SGD with Nesterov momentum and Adam.
3) Apply forward gradient descent to more complex tasks and networks. Concretely, we train and test on the CIFAR-10 dataset[4].

Our results indicate that although forward gradient descent performs well on simpler tasks, it struggles when the task gets more complex and the networks deeper. Moreover, we find that the original paper [1] neglects to optimize the learning rate for backpropagation, resulting in an invalid comparison. Finally, on both the MNIST and CIFAR-10 datasets, we find that backpropagation significantly outperforms forward gradients on both final loss and the number of iterations until convergence.

II. Methods

A. Automatic Differentiation

Automatic Differentiation (AD) is a set of methods to calculate a function's derivative. The method exploits the fact that all computer programs consist of compositions of simple functions and operations, which have analytical solutions [2]. Today's most used gradient method for neural networks, backpropagation, is implemented through reverse-mode AD. Reverse-mode AD computes the analytical gradient by one forward pass through a function and then one backward pass. As an alternative, one can use forward-mode AD to calculate the function value and the directional derivative along a stochastic direction $\mathbf{v}$, and use the directional derivative to get an unbiased estimate of the gradient. A short introduction is outlined in Appendix A for the curious reader.

B. Forward Gradient

The forward gradient proposed by Baydin et al. [1] utilizes forward-mode AD to create an unbiased estimator of the gradient defined as:

$$\mathbf{g}(\boldsymbol{\theta}) = (\nabla \mathbf{f}(\boldsymbol{\theta})\mathbf{v})\mathbf{v} \tag{1}$$

where $\mathbf{v}$ is a perturbation vector sampled from a multivariate Gaussian with expectation zero and unit variance. In other words, the forward gradient samples a random direction, calculates the directional derivative along this direction and multiplies it by the direction again to get an unbiased estimate for the gradient. For more information, see Baydin et al. [1].

C. Optimization algorithms

Baydin et al. [1] implement forward gradient for stochastic gradient descent (SGD), the backbone of most optimization algorithms today. In this work, we extend to two popular algorithms which build on SGD: SGD with Nesterov momentum and Adam. More information about the algorithms and their parameter update rules can be found in Appendix A.

III. Experiments

We perform our experiments on two datasets, MNIST and CIFAR-10. For both datasets, we split the training set into training and validation sets to optimize hyperparameters. For MNIST, we implement forward gradient and backpropagation for all three optimization algorithms: SGD, SGD with Nesterov momentum, and Adam. For CIFAR-10, we only implemented vanilla SGD. The forward gradient is implemented in functorch[5], while we use Pytorch [6] for backpropagation. All the networks use ReLU as activation functions in the hidden layers and softmax on the output. For forward gradients, we used gradient clipping and capped the Jacobian-vector product's norm at one to avoid exploding gradients. In all experiments, we use a batch size of 64.

A. MNIST

The MNIST dataset consists of 70 000 images of handwritten digits [3]. For MNIST, we implement both a neural network and a convolutional network.

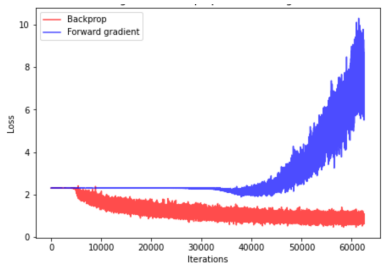

Figure 1: Forward gradient and backpropagation trained using SGD on the CIFAR-10 dataset. Hyperparameters is found in Table I

1) Vanilla neural network: The vanilla neural network uses three fully connected layers with sizes 1024, 1024, and 10.

2) Convolutional neural network: The convolutional neural network consists of four convolutional layers of size 64 with a 3 by 3 kernel, stride 1, and zero padding. At the end, we have two dense layers.

B. CIFAR-10

The CIFAR-10 dataset contains 60 000 32x32 images from 10 classes. In this case, we only implement a convolutional network. The network consists of three blocks of two convolutional layers after the VGG architecture [7] with sizes 64, 128, and 256. Each block has two convolutional layers and a maxpooling operation. The convolutional layers have a 3 by 3 kernel, 1 stride, and 1 padding. The maxpooling layers are 2 by 2 with zero padding.

IV. Results

Our experiments found that training with backpropagation outperformed training with forward gradients in every aspect. Table I depicts each test loss and hyperparameters for the different configurations we tested. All networks trained with backpropagation on the MNIST dataset converge to

Table I: The full results of our experiments. The test loss is the test loss after e epochs, γ the learning rate, and decay the exponential learning rate decay.

Gradient type	**Model**	**Dataset**	**Optimizer**	γ	**decay**	**Momentum**	β_1	β_2	**epochs**	**Test loss**
Forward gradient	NN	MNIST	SGD	0.000464	1e-5	0.000	0.000	0.0000	50	0.332
Backpropagation	NN	MNIST	SGD	0.100000	0	0.000	0.000	0.0000	50	0.000
Forward gradient	NN	MNIST	Nesterov	0.000320	1e-5	0.367	0.000	0.0000	50	0.270
Backpropagation	NN	MNIST	Nesterov	0.100000	0	0.500	0.000	0.0000	50	0.000
Forward gradient	NN	MNIST	ADAM	0.000500	5e-5	0.000	0.990	0.9990	50	0.283
Backpropagation	NN	MNIST	ADAM	0.000100	0	0.000	0.900	0.9990	50	0.000
Forward gradient	CNN	MNIST	SGD	0.000398	1e-5	0.000	0.000	0.0000	50	0.548
Backpropagation	CNN	MNIST	SGD	0.100000	0	0.000	0.000	0.0000	50	0.000
Forward gradient	CNN	MNIST	Nesterov	0.000100	1e-4	0.900	0.000	0.0000	50	0.559
Backpropagation	CNN	MNIST	Nesterov	0.010000	0	0.783	0.000	0.0000	50	0.000
Forward gradient	CNN	MNIST	ADAM	0.000250	1e-4	0.000	0.900	0.9999	50	0.669
Backpropagation	CNN	MNIST	ADAM	0.000236	0	0.000	0.900	0.9990	50	0.000
Forward gradient	CNN	CIFAR-10	SGD	0.000433	1e-6	0.000	0.000	0.0000	100	6.955
Backpropagation	CNN	CIFAR-10	SGD	0.010000	1e-4	0.000	0.000	0.0000	100	1.685

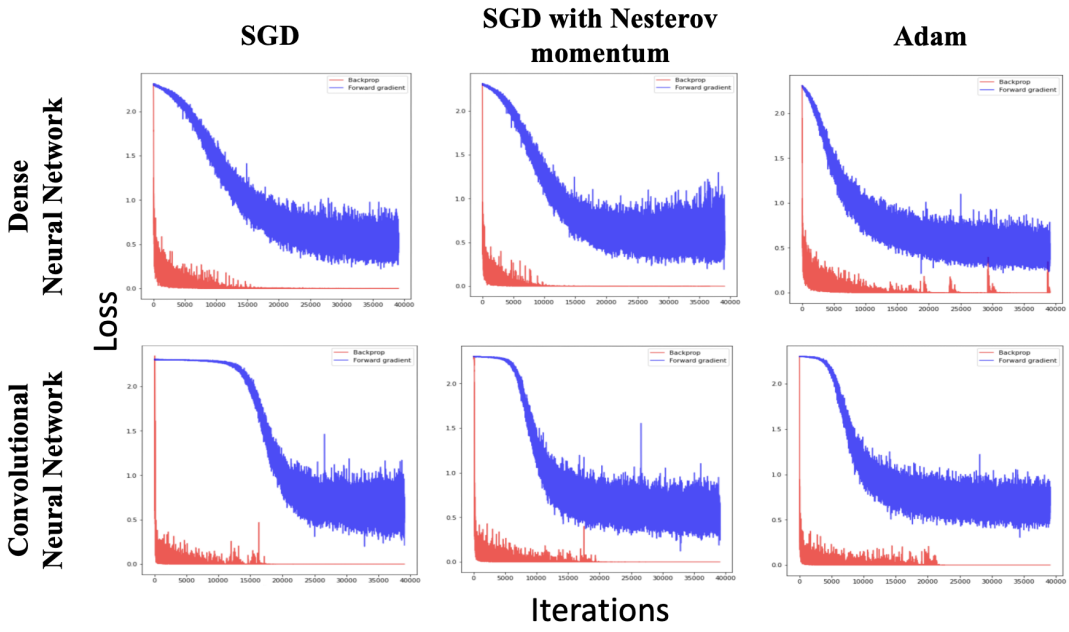

Figure 2: The results from training backpropagation and forward gradient on the MNIST dataset. The hyperparameters used is found in Table I

zero loss regardless of the optimization methods used, while none of the forward gradient networks do. From Figure 2, it is also clear that the loss of the forward gradient networks is significantly more noisy than their backpropagation counterparts. For the networks trained on the CIFAR-10 dataset, the forward gradient network fails to learn anything, and the loss explodes after 40000 iterations, as seen in Figure 1. The backpropagation network ends up with a loss of 1.685.

V. Discussion

In the original paper by Baydin et al. [1], the authors find that forward gradient performs at least as well as backpropagation in loss and convergence speed. Interestingly, we observe that when the learning rate for backpropagation and forward gradients is optimized separately, backpropagation achieves a significantly lower loss on the MNIST data. This is valid for all optimizers and nets we used on the MNIST data. We believe this indicates that the forward gradient is too noisy and has too much variance. Furthermore, we see that backpropagation converges 2-4 times faster. In the original paper, Baydin et al.'s [1] results show that forward gradients are around two times faster to compute than backpropagation, and reach the same level of performance. However, they used the same learning rates for backpropagation and forward gradients. When we optimize the learning rates, backpropagation can use a much higher learning rate than the forward gradient. This causes the backpropagation to converge faster, despite the forward gradient's advantage in time per iteration.

For the CIFAR-10 dataset, the forward gradients performed poorly. As seen in Figure 1, the algorithm can almost not achieve any training. This result indicates that forward gradients struggle when the task gets more complex and the network bigger. This could prove a significant disadvantage as the trend is that networks and tasks get more complex.

Another problem we encountered with forward gradients was that they were very unstable and suddenly exploded. Figure 1 shows an example of this. To mitigate it, we implemented gradient clipping and capped the norm of the Jacobian-vector product to one. This stabilized the training process and allowed us to get the same performance as Baydin et al. [1]. The instability and exploding gradient problem showed up in all our models, and we strongly recommend using some form of gradient clipping when using forward gradients.

VI. Summary

This paper studied the performance of forward gradients compared to standard backpropagation when training neural networks, as described by Baydin et al. [1]. We were able to reproduce the authors' results. However, a fair comparison reveals that the forward gradient performs weaker and needs more iterations than backpropagation when training on the MNIST dataset with SGD. This also holds for SGD with Nesterov momentum and the Adam optimization algorithm. Furthermore, we observe that forward gradients perform very weak when we increase the task and network complexity by training and testing a convolutional network on the CIFAR-10 dataset.

APPENDIX

Automatic Differentiation:
Automatic Differentiation is a set of general techniques for differentiating an arbitrary function. In the following paragraphs, we denote $\mathbf{f} : R^n \rightarrow R^m$ as the function to differentiate and $\mathbf{J_f}$ the Jacobian of partial derivatives of $\mathbf{f}$. The two techniques utilized in this paper are reverse-mode AD and forward-mode AD.

1) Reverse-mode AD: Reverse mode AD works by passing through a function twice, one forward pass and one backward pass. During the forward pass, the method saves intermediary variables and records dependencies in the computational graph [2], before it outputs the function value $\mathbf{f}(\boldsymbol{\theta})$. Then, the method makes a backward pass to evaluate the vector-Jacobian product $\mathbf{v}^T\mathbf{J_f}$. In most neural network applications, the functions we evaluate is scalar objective functions. In that case, the vector-Jacobian product corresponds to the gradient.

2) Forward-mode AD: On the other hand, forward mode AD evaluates both the function value $\mathbf{f}(\boldsymbol{\theta})$ and the Jacobian-vector product $\mathbf{J_f v}$ in only one forward pass. For the scalar objective case, the Jacobian-vector product corresponds to the directional derivative along the direction $\mathbf{v}$.

SGD:
Stochastic gradient descent (SGD) is the backbone of most optimization algorithms in deep learning today. The algorithm is a stochastic version of gradient descent which estimates the gradient over a batch of n samples. The parameter update equation for the weights $\boldsymbol{\theta}$, the function $\mathbf{f}$, and with a step size γ is:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \gamma g(\boldsymbol{\theta}),\ where\ \ g(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \mathbf{f}_i(\boldsymbol{\theta}) \quad (2)$$

SGD with Nesterov momentum:
Momentum is a technique that keeps track of an exponentially weighed average of past gradients. As vanilla SGD only estimates the true gradient over a small batch of data, the estimation is often noisy and not equal to the real gradient. The momentum term mitigates this by including previous estimates in the update step, often yielding a more accurate estimate. SGD with Nesterov momentum uses the same idea, but computes the gradient at the current position plus the momentum step. This yields the following update equation [8]:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)} \quad (3)$$

where $\mathbf{v}^{(t+1)}$ is given by

$$\mathbf{v}^{(t+1)} = \mu \mathbf{v}^{(t)} + \gamma \nabla \mathbf{f}(\boldsymbol{\theta}^{(t)} + \mu \mathbf{v}^{(t)})$$

μ is the momentum parameter and γ is the step size.

Adam:
Kingman and Ba [9] introduced Adam, an adaptive learning rate optimization algorithm in 2014. Today, Adam is maybe the most utilized optimization algorithm in deep learning due to its attractive convergence properties. The algorithm adapts the learning rates based on estimates of the first and second order moments of the gradient. It achieves this by maintaining an exponential moving average of the moments [9]:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \gamma \frac{\hat{m}^{(t+1)}}{\sqrt{\hat{v}^{(t+1)}} + \epsilon} \quad (4)$$

where:

$$\hat{v}^{(t+1)} = \frac{v^{(t+1)}}{1 - \beta_2^{(t+1)}}$$

$$\hat{m}^{(t+1)} = \frac{m^{(t+1)}}{1 - \beta_1^{(t+1)}}$$

$$v^{(t+1)} = \beta_2 v^{(t)} + (1 - \beta_2) g_{t+1}^2$$

$$m^{(t+1)} = \beta_1 m^{(t)} + (1 - \beta_1) g_{t+1}$$

γ is the step size, β_1 is the decay rate for the first moment, β_2 the decay rate for the second moment, g_{t+1} is the gradient, and ϵ is a numerical stabilizer.

REFERENCES

[1] A. G. Baydin, B. A. Pearlmutter, D. Syme, F. Wood, and P. Torr, "Gradients without backpropagation," 2022. [Online]. Available: https://arxiv.org/abs/2202.08587

[2] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," 2015. [Online]. Available: https://arxiv.org/abs/1502.05767

[3] 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.

[4] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[5] R. Z. Horace He, "functorch: Jax-like composable function transforms for pytorch," https://github.com/pytorch/functorch, 2021.

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[8] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: https://proceedings.mlr.press/v28/sutskever13.html

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980