Gradients with or without Backpropagation

Xiang Gao[*] **Yao Kuang**[*] **Hongcheng Wei**[*]
University of Toronto
{xiangt.gao, yao.kuang, homelet.wei}@mail.utoronto.ca

Abstract

Gradient-based optimization plays a central role in the modern day of machine learning. Backpropagation, or reverse-mode automatic differentiation, is the most widely accepted methodology for computing the gradients during the optimization stage of learning algorithms. Recently, an formulation called **forward gradient** [1] overcame the drawback of forward-mode automatic differentiation, providing an efficient way to obtain an unbiased estimate of the true gradient through one forward pass of a given neural network model. In this project, we implemented the proposed formulation and showed the convergence of forward gradient under SGD. In addition to the original work and experiments under SGD, we utilize the Adam optimizer [7] with forward gradient, and the training dynamics under various learning rates suggested that backpropagation performs significantly better than forward gradients with Adam.

1 Introduction

The recent success of deep learning heavily relies on the capability to perform automatic differentiation in a scaled and effective manner. Unlike reverse-mode automatic differentiation, forward-mode AD has received little attention due to its inability to scale for common machine learning functions taking the form $f : \mathbb{R}^n \to \mathbb{R}$. To obtain the exact gradient of the model, the number of forward passes is proportional to the number of parameters in the model. Since most of the successful deep neural networks like GPT-3 [4] contain billions of parameters, it would be unfeasible to do such computations. Nevertheless, by initializing a standard Gaussian perturbation, Baydin et al. [1] proposed an unbiased gradient estimator that only requires a single forward pass.

Using this estimator with SGD, Baydin et al. [1] has shown that it has substantial savings in computation and other advantages compared to backpropagation. We extended such estimator with Adam optimizer and reproduced their results with SGD. Our contributions are threefold: 1. We formalize the convergence results of gradients obtained from forward gradient formulation under SGD. 2. We provide an implementation of forward gradient in JAX [3]. 3. We discover and characterize the failure modes of forward gradient with an Adam optimizer by comparing the loss and runtime performance.

2 Related Works

Many current automatic differentiation frameworks provide comprehensive support and a common workflow for reverse-mode automatic differentiation. Among them, JAX has easy composable transformations for computing derivatives and provides a native implementation of Jacobian-Vector products that can be easily utilized for implementing forward gradients. PyTorch provides a forward-mode AD API; nevertheless, at the time of experiments, this module is still in beta testing with very limited support, and in particular, the support for ReLU activation is missing. Moreover, we did not find this framework could easily adapt to the forward gradient formulation. On the other hand, JAX

[*]Equal contributions.

Course Project (CSC413 Winter 2022).

contains easily modified neural networks library `flax` [5] and a library called `optax` [6] supporting widely used optimizers. Moreover, researchers have proposed many alternatives to backpropagation in the past. Notably, a very recent work from DeepMind [8] also uses directional derivatives to improve gradient estimators, which is advantageous to sequential learning settings like RNNs and reinforcement learning. If interested, we refer the readers to the more comprehensive related works section presented in [1, 8].

3 Method and Algorithm

3.1 Forward Gradients

Definition 1 (Baydin et al. (2022, Definition 1))**.** *Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the "forward gradient" of f with respect to $\boldsymbol{v} \in \mathbb{R}^n$ is $g : \mathbb{R}^n \to \mathbb{R}$ defined as*

$$g(\boldsymbol{\theta}) = (\nabla f(\boldsymbol{\theta}) \cdot \boldsymbol{v})\, \boldsymbol{v}$$

The forward gradient formulation does not require the exact $\nabla f(\boldsymbol{\theta})$. Instead, it only requires $\nabla f(\boldsymbol{\theta}) \cdot \boldsymbol{v} = D_{\boldsymbol{v}} f(\boldsymbol{\theta})$, which is a directional gradient. This is the reason why it is possible to compute the forward gradient without backpropagation.

The following theorem, gives conditions under which $g(\boldsymbol{\theta})$ become an unbiased estimator of the true gradient $\nabla f(\boldsymbol{\theta})$. The unbiasedness of forward gradient gives the basic theoretical evidence to use it to replace gradient. We adapt the proof from Baydin et al. [1] in Appenix A.

Theorem 1. *For a multivariate random variable $\boldsymbol{v} \in \mathbb{R}^n$, the forward gradient of f with respect to $\boldsymbol{v}$ $g(\boldsymbol{\theta})$ is an unbiased estimator of the gradient $\nabla f(\boldsymbol{\theta})$ if*

$$\mathbb{E}[\boldsymbol{v}] = \mathbf{0}_n \text{ and } \operatorname{Cov}[\boldsymbol{v}] = \boldsymbol{I}_n$$

Baydin et al. [1] used forward gradient with stochastic gradient descent, and empirically showed that this formulation converges nicely during training compared to backpropagation. Additionally, we gave a theoretical proof of **convergence** of forward SGD in terms of error $\|\nabla f(\theta_k)\|^2$ in the following theorem.

Theorem 2. *Under some assumptions (shown in Appendix B), in forward SGD with constant learning rate α, for any $\epsilon > 0$, we can choose small α accordingly, such that*

$$\min_{k=0,1,\ldots,t-1} \mathbb{E}\left[\|\nabla f(\theta_k)\|^2\right] \leq \epsilon$$

for large enough t.

The detail of learning schedule and proof is shown in Appendix B.

3.2 Optimization using Forward Gradients

Forward gradient, $g(\boldsymbol{\theta})$, is an unbiased estimator of the true gradient $\nabla f(\boldsymbol{\theta})$. Therefore, any first-order gradient-based optimizer can utilize the forward gradient formulation. In this project, we reproduced the results under SGD and further experimented such formulation with the most widely used adaptive learning rate optimization algorithm, Adam [7]; we analyzed its performance through direct comparison to the original optimizer based on backpropagation. `Algorithm 1` demonstrates the optimization flow when using forward gradient. The algorithm starts by obtaining a random perturbation vector v that follows a standard Gaussian distribution, and then it runs f via forward-mode AD that evaluates $f(\theta)$ and $\nabla f(\theta) \cdot v$ simultaneously in a single forward pass of the model. To obtain $g(\theta)$, we can simply multiply the directional derivative $\nabla f(\theta) \cdot v$ with the perturbation vector v.

Algorithm 1 Generic Forward Gradient Optimizer

Require: $f : \mathbb{R}^n \to \mathbb{R}$ objective function
Require: $\mathcal{M}$: a first-order gradient-based optimizer
Require: $\boldsymbol{\theta}_0$: initializer
  Initialize $\mathcal{M}$ at $\boldsymbol{\theta}_0$
  $t \leftarrow 0$
  while $\boldsymbol{\theta}_t$ not converged **do**
    $t \leftarrow t + 1$
    $\boldsymbol{v}_t \sim \mathcal{N}(\mathbf{0}_n, \boldsymbol{I}_n)$ ▷ Sampling
    $g_t \leftarrow (\nabla f(\boldsymbol{\theta}_t) \cdot \boldsymbol{v}_t)\, \boldsymbol{v}_t$ ▷ Forward gradient
    $\boldsymbol{\theta}_{t+1} \leftarrow \mathcal{M}(\boldsymbol{\theta}_t, g_t)$
    $t \leftarrow t + 1$
  end while

3.3 JAX Implementation

We have chosen JAX [3] along with the support libraries `flax` [5] and `optax` [6] for building the workflow of our neural network optimizations. To implement our version of forward gradient and perform experiments, JAX are equipped with the capability of calculating `jvp` - Jacobian-Vector product and has the runtime advantage with Just-In-Time (JIT) compilation.

We rewrite the `jax.value_and_grad` function as `value_and_forward_grad` using `jax.jvp`. The `jax.jvp` function takes a "primal" and a "tangent" to evaluate a function at "primal" and calculate the directional derivative in the direction of "tangent" simultaneously.

```
1 keys_tree = random_split_like_tree(key, params)
2 v = jax.tree_map(lambda w, key: jax.random.normal(key, w.shape), params, keys_tree)
3 f, d = jax.jvp(loss_func, (params,), (v,))
4 fwd_grad = jax.tree_map(lambda v_i: d * v_i, v)
```

Since JAX uses stateful computation, `jax.random.normal` is a pure function, which means we need a different key to generate a different value. The helper function `random_split_like_tree` on line 1 would generate multiple subkeys, one for each parameter.

On line 2, we use the corresponding subkeys of a parameter to sample a perturbation $\boldsymbol{v}$ from standard normal. Then, in `jax.jvp` we can pass the perturbation as the "tangent" to get $\nabla f(\boldsymbol{\theta}) \cdot \boldsymbol{v}$ as $\boldsymbol{d}$. Finally, we multiply the scalar $\boldsymbol{d}$ back to the perturbation v_i for each parameter to get the forward gradient $g(\boldsymbol{\theta})$.

4 Experiments

In all experiments, we run the code on Colab connected to a local runtime with access to a NVIDIA RTX 2080 Ti GPU. Our code is made public and reproducible on Google Colab.

4.1 Implementation Verification

First, since the forward gradient is an unbiased estimate of the true gradient, we will verify our forward gradient implementations by reproducing the figure that shows the expected forward gradients move closer to the true gradient as the number of samples increase.

To give a clear visualization on the property of forward gradient, we illustrate the results on the Beale function, $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$ at $x = 1.5, y = -0.1$. The red arrow indicates the Monte Carlo estimates by averaging distinct number of forward gradient samples, i.e., $\frac{1}{K}\sum_{k=1}^{K}(\nabla f \cdot v_k)v_k \approx \mathbb{E}[(\nabla f \cdot v)v]$, and the blue arrow shows the true gradient at this point which points to the opposite direction as the global minimum shown by a star. From figure 1, it showed empirical evidence that our implemented forward gradient moves to the true gradient in expectation.

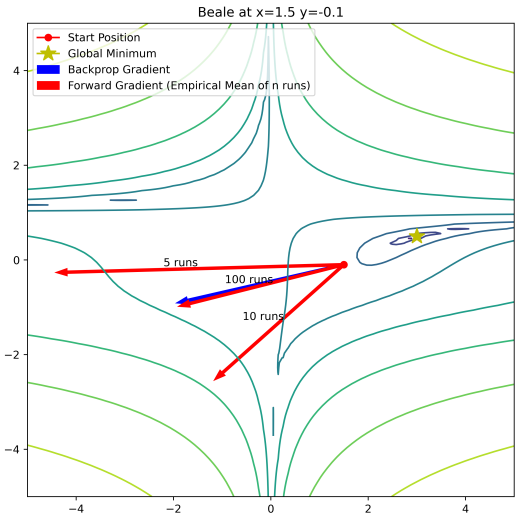
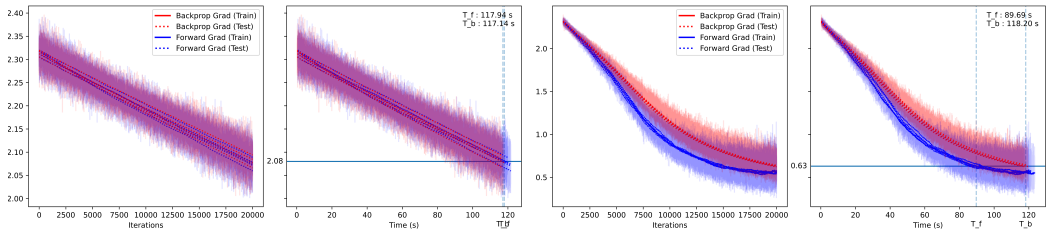


Figure 1: The empirical mean of 100, 10, 5 samples of forward gradient, and the true gradient for the Beale function at $x = 1.5, y = -0.1$.

4.2 Training Dynamics with Multilayer Perceptron

To empirically measure the convergence and complexity of the forward gradient formulation, we employed a simple multilayer perceptron that contains two hidden layers of size 1024 with ReLU activations and used a minibatch size of 64. The model performed handwritten digits classification on the MNIST dataset. Similar to [1], we run all experiments with equal number of iterations and plot

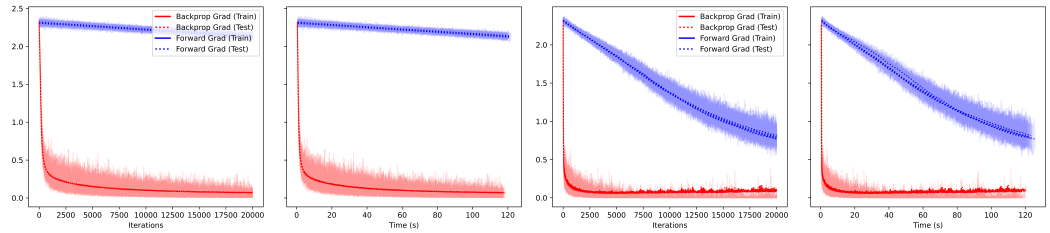
their training dynamics; also, we define T_b as the time at when the lowest validation loss is achieved by backpropagation. T_f is the time when the same loss is achieved.



(a) learning rate: 2×10^{-5} (b) learning rate: 2×10^{-4}

Figure 2: Comparison of forward gradient and backpropagation with Adam, showing two learning rates. Five independent runs per experiment.

Figure 2 shows the experiments comparing the two gradient formulations in SGD. The loss per iteration plots largely aligns with the results from Baydin et al. [1]; however, in terms of time complexity, we do not observe the same conclusions as in [1]. Although the forward gradients spent similar amount of time to run 20000 iterations, we noticed that our backpropagation runs much faster (approximately 120 seconds) comparing to 300 seconds from [1], which possibly explains that we have lower ratio of $\frac{T_f}{T_b}$. However, we must recognize that the relative runtime is highly dependent on the hardware as well as implementation details between frameworks.



(a) learning rate: 2×10^{-5} (b) learning rate: 2×10^{-4}

Figure 3: Comparison of forward gradient and backpropagation in SGD, showing two learning rates. Five independent runs per experiment.

On the other hand, figure 3 illustrates that Adam does not work well with forward gradient whereas backpropagation was able to converge very early. We suspect that the default Adam hyperparameter settings favors past gradients information, but the forward gradient is merely an approximation every iteration where it could have high variance from the past. This may slows down the learning process. It is worth noting that with a larger learning rate than the above settings, then the forward gradient training becomes unstable and diverges quickly; the relevant plots are provided in the appendix.

5 Conclusions

In this project, we give a proof that shows the convergence of forward gradient under the setting of SGD. Moreover, we provide a JAX-based implementation for the forward gradient formulation, and it was shown no slower than the original authors' PyTorch implementation that is not public as promised currently. With stochastic gradient descent, the performance of forward gradient is only superior to backpropagation under strict hyperparameter settings, and we found that the gain from forward gradient was not nearly as the Baydin et al. [1] claimed.

Furthermore, we experimented forward gradient with the most established optimizer in the deep learning community - Adam [7]. Nevertheless, forward gradients encountered a much slower optimization process whereas the usual backpropagation enjoys the advantage that Adam provided. Adam relies heavily on the past gradient information; however, forward gradient is only approximations and may not be as always reliable. This insight could help to develop a specific version of Adam that has adaptive weight on past gradient information to specifically aid the learning process of forward gradient.

References

[1] Atılım Güneş Baydin, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation, 2022. URL https://arxiv.org/abs/2202.08587.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004. ISBN 0521833787. URL https://stanford.edu/~boyd/cvxbook/.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[5] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL http://github.com/google/flax.

[6] Matteo Hessel, David Budden, Fabio Viola, Mihaela Rosca, Eren Sezener, and Tom Hennigan. Optax: composable gradient transformation and optimisation, in jax!, 2020. URL http://github.com/deepmind/optax.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[8] David Silver, Anirudh Goyal, Ivo Danihelka, Matteo Hessel, and Hado van Hasselt. Learning by directional gradient descent. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=5i7lJLuhTm.

Contributions

All of us discussed and planned out the project idea.

Xiang Gao designed the experimental setups, analyzed the results and wrote most of the manuscript.

Yao Kuang formalized Generic Forward Gradient Optimizer, adapted definition and proof of unbaisedness from Baydin et al., and gave a statement with proof about convergence.

Hongcheng Wei implemented forward gradient in JAX and wrote code to conduct experiments and generate graphs.

A Appendix (Proof of Unbiasedness of Forward Gradient)

Theorem 3 (Baydin et al. (2022, Theorem 1))**.** *For a multivariate random variable $\boldsymbol{v} \in \mathbb{R}^n$, the forward gradient of f with respect to $\boldsymbol{v}$ $g(\boldsymbol{\theta})$ is an **unbiased** estimator of the gradient $\nabla f(\boldsymbol{\theta})$ if*

$$\mathbb{E}[\boldsymbol{v}] = \mathbf{0}_n \text{ and } \operatorname{Cov}[\boldsymbol{v}] = \boldsymbol{I}_n$$

Proof. Let $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$, the conditions $\mathbb{E}[\boldsymbol{v}] = \mathbf{0}_n$ and $\operatorname{Cov}[\boldsymbol{v}] = \boldsymbol{I}_n$ implies, for $i \neq j$,

$$\mathbb{E}[v_i] = 0, \operatorname{Cov}[v_i] = 1, \text{ and } v_i \perp v_j$$

Then

$$\begin{aligned}
\mathbb{E}[v_i^2] &= \mathbb{E}[(v_i - \mathbb{E}[v_i])^2] = \operatorname{Cov}[v_i] = 1 \\
\mathbb{E}[v_i v_j] &= \mathbb{E}[v_i]\mathbb{E}[v_j] = 0
\end{aligned}$$

Using it we can compute $\mathbb{E}[g(\boldsymbol{\theta})]$. At first, simplify $g(\boldsymbol{\theta})$

$$\begin{aligned}
g(\boldsymbol{\theta}) &= (\nabla f(\boldsymbol{\theta}) \cdot \boldsymbol{v})\, \boldsymbol{v} \\
&= \left(\sum_{i=1}^{n} \frac{\partial f}{\partial \theta_i} v_i\right)\left(\sum_{i=1}^{n} v_i\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial f}{\partial \theta_i} v_i v_j \\
&= \sum_{i=1}^{n} \frac{\partial f}{\partial \theta_i} v_i^2 + \sum_{i=1}^{n}\sum_{j\neq i} \frac{\partial f}{\partial \theta_i} v_i v_j
\end{aligned}$$

Then apply linearity of expectation,

$$\begin{aligned}
\mathbb{E}[g(\boldsymbol{\theta})] &= \sum_{i=1}^{n} \frac{\partial f}{\partial \theta_i} \mathbb{E}[v_i^2] + \sum_{i=1}^{n}\sum_{j\neq i} \frac{\partial f}{\partial \theta_i} \mathbb{E}[v_i v_j] \\
&= \sum_{i=1}^{n} \frac{\partial f}{\partial \theta_i} = \nabla f(\boldsymbol{\theta})
\end{aligned}$$

□

B Appendix (Convergence of Forward SGD)

For SGD, we consider the optimization problem:

$$\text{minimizes } f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x)$$

where $f : \mathbb{R}^n \to \mathbb{R}$. The forward stochastic gradient descent with constant step α is a iterative method that in each iteration:

1. Select $i_t \in \{1, 2, \ldots, m\}$ randomly and uniformly, and sample $v_t \sim \mathcal{N}(\mathbf{0}_n, \mathbf{I}_n)$.
2. Get forward gradient $g_t = (\nabla f_{i_t}(\theta_t) \cdot v_t)\, v_t$
3. Gradient descent $\theta_{t+1} = \theta_t - \alpha g_t$

We assume f satisfies the following assumptions:

- f is bounded below by $f^* \in \mathbb{R}$.
- ∇f is L-Lipschitz continuous.
- There exists $\sigma \in \mathbb{R}$ such that $\mathbb{E}_i\left[\|\nabla f_i(x) - \nabla f(x)\|^2\right] \leq \sigma^2$ for all $x \in \mathbb{R}^n$ (variance of gradient is bounded).

Theorem 4. *Under these assumptions, in the iteration of forward SGD with small enough constant learning rate α,*

$$\min_{k=0,1,\dots,t-1} \mathbb{E}_{i,v}\left[\|\nabla f(\theta_k)\|^2\right] = O(1/(\alpha t)) + O(\alpha)$$

More precisely, when $\alpha < \epsilon$ for some $\epsilon > 0$, there exists N_1, N_2, independent with α and t, such that

$$\min_{k=0,1,\dots,t-1} \mathbb{E}_{i,v}\left[\|\nabla f(\theta_k)\|^2\right] \leq N_1/(\alpha t) + N_2\alpha$$

Proof. At first, we show an upper bound for $\mathbb{E}\left[\|g_k\|^2\right]$.

$$\begin{aligned}
\|g_k\|^2 &= \|(\nabla f_{i_k}(\theta_k)\cdot v_k)\, v_k\|^2 \\
&= |\nabla f_{i_k}(\theta_k)\cdot v_k|^2 \|v_k\|^2 \\
&\leq \|\nabla f_{i_k}(\theta_k)\|^2 \|v_k\|^2 \|v_k\|^2 && \text{by Cauchy–Schwarz inequality}
\end{aligned}$$

Take expectation, and let $C_1 = \mathbb{E}_{v\sim\mathcal{N}(\mathbf{0}_n,\mathbf{I}_n)}\left[\|v\|^4\right]$,

$$\begin{aligned}
\mathbb{E}_{i_k,v_k}\left[\|g_k\|^2\right] &\leq \mathbb{E}_{i_k,v_k}\left[\|\nabla f_{i_k}(\theta_k)\|^2 \|v_k\|^4\right] \\
&= \mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\theta_k)\|^2\right]\mathbb{E}_{v_k}\left[\|v_k\|^4\right] && \text{by } i_k \perp v_k \\
&= C_1\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\theta_k)\|^2\right] && (1)
\end{aligned}$$

Since

$$\begin{aligned}
\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\theta_k)\|^2\right] &= \mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\theta_k) - \nabla f(\theta_k) + \nabla f(\theta_k)\|^2\right] \\
&\leq \mathbb{E}_{i_k}\left[\left(\|\nabla f_{i_k}(\theta_k) - \nabla f(\theta_k)\| + \|\nabla f(\theta_k)\|\right)^2\right] \\
&\leq 2\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\theta_k) - \nabla f(\theta_k)\|^2\right] + 2\|\nabla f(\theta_k)\|^2 \text{ by } (a+b)^2 \leq 2(a^2+b^2) \\
&\leq 2\sigma^2 + 2\|\nabla f(\theta_k)\|^2 && (2)
\end{aligned}$$

So, putting (2) into (1), we have an upper bound for $\mathbb{E}\left[\|g_k\|^2\right]$:

$$\mathbb{E}\left[\|g_k\|^2\right] \leq 2C_1\sigma^2 + 2C_1\|\nabla f(\theta_k)\|^2 \tag{3}$$

Then go back to the original theorem. Since ∇f is L-Lipschitz continuous, in each iteration, we have

$$f(\theta_{k+1}) \leq f(\theta_k) + \nabla f(\theta_k)^\top(\theta_{k+1} - \theta_k) + \frac{L}{2}\|\theta_{k+1} - \theta_k\|^2$$

Put $\theta_{k+1} = \theta_k - \alpha g_k$ into it,

$$f(\theta_{k+1}) \leq f(\theta_k) - \alpha\nabla f(\theta_k)^\top g_k + \frac{L}{2}\alpha^2\|g_k\|^2$$

Take expectation over i_k, v_k,

$$\begin{aligned}
\mathbb{E}_{i_k,v_k}\left[f(\theta_{k+1})\right] &\leq f(\theta_k) - \alpha\nabla f(\theta_k)^\top\mathbb{E}_{i_k,v_k}\left[g_k\right] + \frac{L}{2}\alpha^2\mathbb{E}_{i_k,v_k}\left[\|g_k\|^2\right] \\
&= f(\theta_k) - \alpha\nabla f(\theta_k)^\top\nabla f(\theta_k) + \frac{L}{2}\alpha^2\mathbb{E}_{i_k,v_k}\left[\|g_k\|^2\right] \\
&\qquad \text{by } \mathbb{E}_{i_k,v_k}\left[g_k\right] = \mathbb{E}_{i_k}\left[\nabla f_{i_k}(\theta_k)\right] = \nabla f(\theta_k) \\
&\leq f(\theta_k) - \alpha\|\nabla f(\theta_k)\|^2 + \frac{L}{2}\alpha^2\left(2C_1\sigma^2 + 2C_1\|\nabla f(\theta_k)\|^2\right) && \text{by (3)} \\
&= f(\theta_k) - \alpha(1 - LC_1\alpha)\|\nabla f(\theta_k)\|^2 + LC_1\sigma^2\alpha^2 && (4)
\end{aligned}$$

Since α is small enough, we can assume $\alpha \leq \frac{1}{2LC_1}$, which is equivalent to $1 - LC_1\alpha \geq \frac{1}{2}$. Then (4) implies

$$\mathbb{E}_{i_k, v_k}\left[f(\theta_{k+1})\right] \leq f(\theta_k) - \frac{1}{2}\alpha \left\|\nabla f(\theta_k)\right\|^2 + LC_1\sigma^2\alpha^2$$

Take expectation over i, v,

$$\mathbb{E}\left[\left\|\nabla f(\theta_k)\right\|^2\right] \leq \frac{2}{\alpha}\left(\mathbb{E}\left[f(\theta_k)\right] - \mathbb{E}\left[f(\theta_{k+1})\right]\right) + 2LC_1\sigma^2\alpha \tag{5}$$

Sum (5) over $k = 0, 1, \ldots, t-1$

$$\sum_{i=0}^{t-1} \mathbb{E}\left[\left\|\nabla f(\theta_k)\right\|^2\right] \leq \frac{2}{\alpha}\left(f(\theta_0) - \mathbb{E}\left[f(\theta_t)\right]\right) + 2tLC_1\sigma^2\alpha \tag{6}$$

Therefore,

$$\begin{aligned}
\min_{k=0,1,\ldots,t-1} \mathbb{E}\left[\left\|\nabla f(\theta_k)\right\|^2\right] &\leq \frac{1}{t}\sum_{i=0}^{t-1} \mathbb{E}\left[\left\|\nabla f(\theta_k)\right\|^2\right] \\
&\leq \frac{2}{\alpha t}\left(f(\theta_0) - \mathbb{E}\left[f(\theta_t)\right]\right) + 2LC_1\sigma^2\alpha \qquad \text{by (6)} \\
&\leq \frac{1}{\alpha t}\left(2f(\theta_0) - 2f^*\right) + 2LC_1\sigma^2\alpha
\end{aligned}$$

□

This theorem gives a theoretical evidence of convergence of forward SGD. In the constant learning rate setting, forward SGD will converge to a circle with radius $\propto$ learning rate.

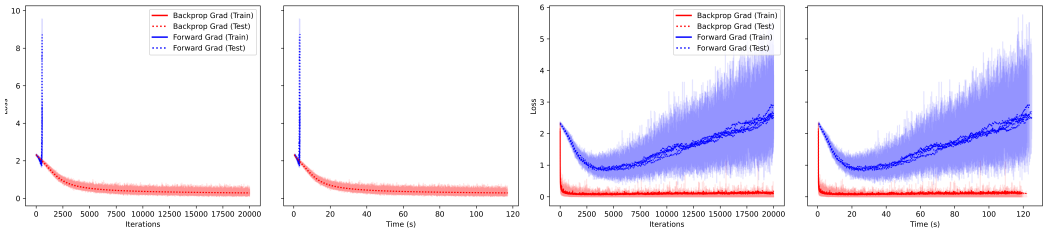
Corollary 4.1. *Under the same assumptions, for any $\epsilon > 0$, we can choose small α accordingly, such that*

$$\min_{k=0,1,\ldots,t-1} \mathbb{E}\left[\left\|\nabla f(\theta_k)\right\|^2\right] \leq \epsilon$$

for large enough t.

Proof. This is a direct result from last theorem. □

C Appendix (Figures)



(a) learning rate: 0.001; SGD (b) learning rate: 0.001; Adam

Figure 4: Comparison of forward gradient and backpropgation in SGD and Adam, showing a larger learning rate. Five independent runs per experiment.