Final Project Report: Analysis between First Order Gradient Decent and Second Gradient Decent

Haoran Pu(hp2565), Yucen Sun(ys3393)

April 2021

1 Introduction

The second-order optimization methods, such as Newton's method, have theoretical guarantee of performance. However, as known the performance in practice is not good and the second order optimizers are not widely used in applications. In this project, we did a thorough analysis and comparison of first and second order optimizations with a main focus on Second order GD, AdaHessian. We did both theoretical analysis of the potentials and issues and practical experiments to verify and illustrate the results.

2 Problem Motivation

Regardless of the deep learning architectures, all of them share the common objective: for some loss function f, based on Taylor Expansion, we have, for any $x, \delta \in \mathbb{R}^d$:

$$f(x+\delta) = f(x) + \nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(y)\delta$$

where $y \in \mathbb{R}^d$, ∇f is the gradient, and $\nabla^2 f$ is the second order derivative (Hessian). And the objective is to find the choice of δ such that, given a starting point x_0, it converges to the optimal $f(x^*)$ with reasonable speed. (Faster is better) Here the concept of convergence is similar with the convergence in real analysis: $(1 \pm \epsilon)f(x^*)$ estimate, where smaller ϵ means closer estimate (but in most cases requires more computation).

Given the Taylor expansion, various approaches were proposed for finding the optimal. Some have been vetted and integrated into well-knoen DL architectures, while other is still under close examination because of the theoretical potential. For this project, we will provide a deep examination in terms of theoretical analysis and experimental analysis to show their strengths and weaknesses.

3 Related Works

First/Second Order GD Analysis, and Randomized Algorithm in Advanced Algorithm taught by Professor Alexandr Andoni, is a reference of the theoretical analysis. The Space Complexity of Approximating the Frequency Moments is the base of Randomized Algorithms (AMS Sketch) analysis[1].

AdaHessian is a recent breakthrough in second order optimizer in June 2020. AdaHessian: An Adaptive Second Order Optimizer for Machine Learning presents the all-round second-order optimization that performs great on a variety of tasks [4]. The implementation of AdaHessian part we used pytorch implementation of AdaHessian on github by @davda54 [3].

4 Theoretical Analysis

4.1 Frist Order Gradient Descent

The most intuitive (and also the most well-recognized) way to optimize is via gradient ∇f, because of the reasonable step bound under mild assumption.

Definition 4.1. *β-smooth: f is β-smooth if $\lambda_{max}(\nabla^2 f) \leq \beta$ for some $\beta > 0$, where λ refers to eigenvalue*

Theorem 4.2. *if f is convex and β-smooth, then for $x_0 \in \mathbb{R}^d$ and $x_0 = argmin_{x\in\mathbb{R}^d}\{f(x)\}$, first order gradient can achieve $(1 \pm \epsilon)f(x^*)$ approximation with at most $\frac{16\beta||x_0 - x^*||_2^2}{\epsilon}$ steps, by choosing $\delta = -\beta\nabla f(x)$*

Please note that **1)**: β-smooth can be easily achieved by real world dataset: for any dataset with large eigenvalue (elongated in an axis), normalization can reduce β for faster convergence. And **2)**: smallest possible β has a strong connection with the optimal learning rate η^* for first order gradient descent, as it is the theoretical best result.

4.2 Second Order Gradient Descent: Newton Method

Although first order gradient descent has shown promising result in practical deep learning application, it still remains shortcomings in theoretical perspective. We leave the hessian matrix $\nabla^2 f(y)$ which may have the potential for faster optimization. However, applying Hessian matrix for optimization is tricky, because we don't know the exact y, from Taylor expansion theorem we only know y is close to x. Due to the unknown property of y, the second moment optimization requires to take more assumptions compared with first order gradient descent, which lead to the general idea of **Newton Method**: assume $\nabla^2 f(x) \approx \nabla^2 f(y)$ and proceed the optimization. Under such assumption, indeed the step convergence is much faster.

Definition 4.3. *α-strong convexity: f has α-strong convexity if $\lambda_{min}(\nabla^2 f) \geq \alpha$*

Note: higher α means f is better behaved: if the smallest eigenvalue is 0, then $\nabla^2 f$ is not invertible.

Theorem 4.4. *if f has α-strong convexity and $||\nabla^2 f(x) - \nabla^2 f(y)||_{op} \leq L||x-y||_2$ ($||A||_{op}$ is spectral norm and $L \in \mathbb{R}$), then for any x_0 such that $||x_0 - x^*|| \leq \frac{18\alpha}{10L}$, Newton method achieves $(1 \pm \epsilon) f(x^*)$ approximation with at most $O(\ln(\ln(\frac{1}{\epsilon})))$ steps by the update algorithm $x_{t+1} = x_t + [\nabla^2 f(x)]^{-1} \nabla f(x)$*

4.3 Superiority of Newton Method in a theoretical setting

Suppose we are optimizing the following loss function $f : \mathbb{R}^d \to \mathbb{R}$ and $f(x) = ||A(x-p)||_2^2$ (optimal is at p), where all eigenvalues of A are positive. Given matrix A has similar behavior of rotation and elongation, first order gradient descent suffers the β-smooth problem if β is too large because of the largest eigenvalue of A. And this reasoning leads to the optimal step bound for first order gradient descent to be $\frac{\ln(\frac{||Ap||_2^2}{\epsilon})}{\ln(\frac{\lambda_{max}^2}{\lambda_{max}^2 - \lambda_{min}^2})}$ for $(1 \pm \epsilon)$ approximation.

On the other hand, since Hessian for quadratic function is constant matrix, $\nabla^2 f(x) = \nabla^2 f(y)$ for arbitrary choice of x, y. This means, by using **Newton Method**, we achieve the **exact optimal** by just **1 step**.

4.4 Inferiority of Newton Method in a practical setting

However, such advantage does not place **Newton Method** into practical DL application, for the following reasons.

- Unlike β-smooth which is achieve-able via normalization, in most cases there are column dependency in real world dataset, which will lead small α-strong convexity which attract **Newton Method** into saddle points.
- Even α-strong convexity is hold, computing and storing Hessian inverse is too expensive to be considered practical. The space complexity is $\Omega(n^2)$ and computation complexity is $\Omega(n^3)$ for inverse computation such as **Gaussian Elimination**.

Although there were proposals for **Quasi-Newton** method where the algorithm apples efficient algorithm for Hessian inverse estimation, the storage requirement($\Omega(n^2)$) is still too large to be considered practical.

5 Recent Breakthrough: AdaHessian[4]

The key breakthrough turns out to use less information from Hessian matrix for similar performance. The parameter reduction is to only utilize Hessian diagonal (second derivative of each parameter) via randomized estimation algorithm, which has shown space complexity $O(n)$ and similar computational efficiency with **Quasi-Newton Method**.

And for weak α-strong convexity of the real dataset, **AdaHessian** utilize the momentum approach for the second order optimizer: jiggling when current position is trapped in saddle point so that the algorithm still optimizes to the optimal. Combining all, **AdaHessian** has the update equation similar with **Adam**, while using Hessian diagonal instead of gradient.

$$m_t = \frac{(1 - \beta_1 \sum_{i=1}^{t} \beta_1^{t-i} g_i)}{1 - \beta_1^t}; v_t = \sqrt{\frac{(1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} D_{ii}^2}{1 - \beta_2^t}}$$

5.1 Randomized Algorithm Analysis

Since Diagonal (second derivative) has similar behavior with $l2$ norm, the randomized scheme is similar with the commonly used norm estimation scheme **AMS sketch**[1]:

Algorithm 1 (**AMS sketch**) Estimate $||x||_2^2$ by $O(1)$ space

```
Require: a stream of $X_i$ (coordinate)
1: Initialize result=0
2: for $x_i \in X_i$ do
3:     Assign each $x_i$ with an independent Bernoulli (+1/-1) with probability $p$
4:     multiply $x_i$ with the corresponding Bernoulli
5:     add value to result
6: end for
       return $result$ square.
```

This algorithm can achieve $(1 \pm \epsilon)||x||_2^2$ approximation with $O(\frac{||x||_2}{\epsilon})$ such independent algorithm copies because the expectation of `result` square is $||x||_2^2$. Given Bernoulli variable is bounded, Chebychev inequality provides a good probability concentration around expectation.

For **Hessian Diagonal** estimation, we will take the similar concept but changing single Bernoulli random variable to Bernoulli random vector(Rademacher).(z) Then we have $diag(H) = E(z \odot (Hz))$ by arguing the probability concentration via matrix Chernoff bound with sub-Gaussian distribution. ($\odot$ refers to point-wise multiplication)

6 Practical Experiments

6.1 Experiment Setup

The overall object is to compare the application of first-order optimizers and second-order optimizers.

We designed two tasks in the experiment. The first is a classification on Cifar-10 dataset with Resnet-18 model, and the second is a regression task on the eggholder function with a fully-connected deep neural network. The fully connected network is composed of five fully connected layers with Relu activation. For each task, we verify or fine-tune to find the best hyper-parameters, and train the model for a fixed number of epochs to collect data. The first experiment is to make sure our result aligns with the paper as well (ResNet-20) and the second experiment is for testing AdaHessian performance where the paper hasn't tested yet. And for regression, the reason for eggholder dataset is to see how different optimizers behave under functions which are hard to optimize.

We select the following optimizers for comparison:
First-order optimizers: SGD, SGD w momentum, Adam, Adamw
Second-order optimizer: Adahessian, Quasi-Newton (deprecated due to huge loss - nan)

The evaluation metrics for classification task is based on four metrics.

(1) Maximum Test Accuracy (160 epochs training)

(2) Minimum Test Loss

(3) Average Optimizer Step Time

(4) Convergence observed

The evaluation metrics for the regression task are:

(1) Minimum Test Loss - RMSE (2000 epochs training)

(2) Average Optimizer Step Time

6.2 Implementation

Resnet-18 model code modified from github repo[2], due to the official pytorch resnet-18 not adapt well to small images like cifar-10.

Adahessian Optimizer code modified from github repo [3].

First-order optimizers: package in Pytorch torch.optim.

Quasi-newton optimizer: using torch.optim.lbfgs, etc.

6.3 Technical Challenges

(1) **Analysis of algorithms**: To fully understand and illustrate, we need to do in-depth analysis and comparison of second order optimization algorithms, the theoretical guarantee and reasons of practical issue or potentials.

(2) **Problem of Pytorch implementation of Resnet**: During the project experiment, we realized the official pytorch impl of resnet doesn't apply well on datasets with small image size such as Cifar-10.

(3) **Long Experiment Time**: Pytorch has no official implementation of second-order optimizers, and it has almost no optimization for Hessian Diagonal Estimation. Therefore, the training on AdaHessian takes a long training time.

7 Experiment Results

7.1 Classification Task

The optimizers we tested for this task, the hyper-parameters, and the result and performance are all included in Table 1. The highest accuracy and lowest lost are marked in bold.

Dataset: Cifar-10	Optimizer	Hyperparameters			Result		Performance
Model: Resnet 18		learning_rate	momentum	weight decay	Cross-entropy Loss	Accuracy	Avg step time (s)
First-order	SGD	1.00e-5	0	5.00e-4	0.295	0.9095	0.001
	SGD w momentum	1.00e-5	9.00e-1	5.00e-4	**0.2889**	**0.9178**	0.039
	Adam	1.00e-3	-	5.00e-4	0.3849	0.8804	0.0044
	Adamw	1.00e-3	-	5.00e-4	0.3198	0.9053	0.0043
Second-order	AdaHessian	1.00e-1	-	5.00e-4	0.2894	0.9122	1.1811

Table 1: Classification of Resnet-18 on Cifar-10

The validation accuracy after each training epoch is shown in figure 1, and the validation loss (cross-entropy) is shown in figure 2. From the table and the plots, we can see the following results. With Resnet-18 on on Cifar-10, the validation accuracy 0.9122 with AdaHessian outperforms Adam, Adamw and SGD, but SGD w Momentum gives the best accuracy. The best validation accuracy with AdaHessian is very close to (0.056 lower than) the highest accuracy 0.9178 by SGD w Momentum.

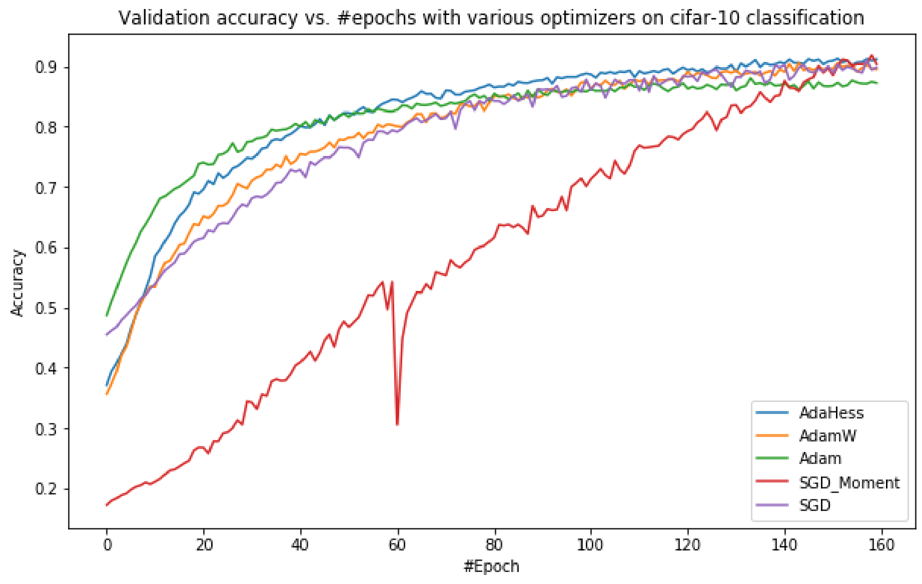

Figure 1: Validation accuracy vs. number of epochs with various optimizers on cifar-10 classification

From the bar plot of time we can see that AdaHessian optimization takes much longer time than the first-order optimizations (30x SGD w momentum, 100× SGD).

AdaHessian optimization slightly overfit at the end of training. Since it's second order optimization, it only slightly overfit after 160 training epochs with learning rate 0.1.

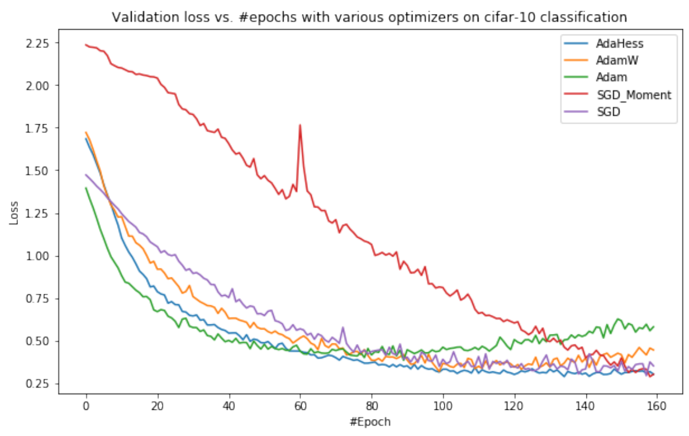

Figure 2: Validation loss vs. number of epochs with various optimizers on cifar-10 classification

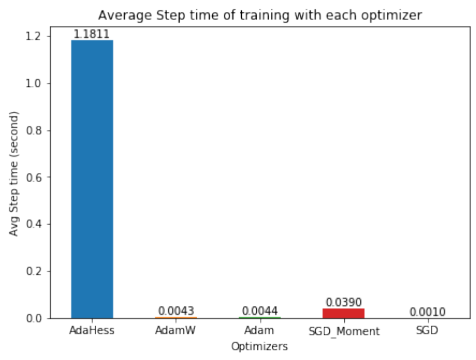

Figure 3: Average optimizer step time of optimizers w Resnet-18 on Cifar-10 dataset

AdamW optimization obviously overfit, as the validation loss starts to increase near the end in the plot.

SGD w Momentum=0.9 doesn't overfit during the 160 epochs of training, which can be a combined result of Stochastic method, the momentum involved and learning rate 1e-5.

Please note that Adahession uses a much higher learning rate compared with Adam and AdamW but still results in less overfitting. It may be due to the fact that the second order moment optimizes advantage of adjusting learning rate based on Hessian matrix diagonal, which takes convexity into account as well.

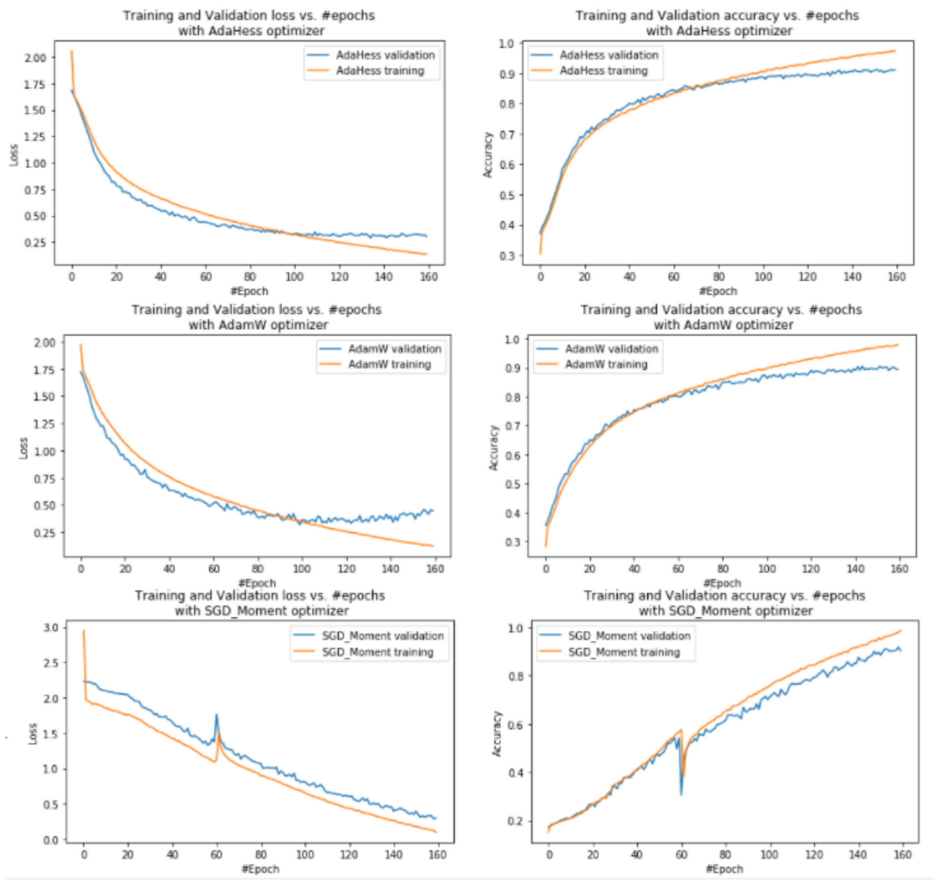

Figure 4: Training and validation accuracy and loss plot v.s. number of epochs in classification task

7.2 Regression Task

The optimizers we tested for the regression task, the hyper-parameters, and the result and performance are all included in Table 2. The lowest lost is marked in bold.

Table 2: Generated by Spread-LaTeX

Eggholder function	Optimizer	Hyperparameters			Result	Performance
A FC DNN		learning_rate	momentum	weight decay	Loss (RMSE)	Avg step time (s)
First-order	SGD	1.00e-5	0	5.00e-4	290.16	1.75e-4
	SGD w momentum	1.00e-5	1.00e-2	5.00e-4	288.36	2.82e-4
	Adam	1.00e-3	-	5.00e-4	286.59	6.33e-4
	Adamw	1.00e-3	-	5.00e-4	286.58	6.11e-4
Second-order	AdaHessian	1.00e-1	-	5.00e-4	**284.38**	2.51e-3

From figure 4 we can see that the second-order optimizer AdaHessian clearly outperforms the first-order optimizers on the regression task. The eggholder function is complicated and by nature hard to estimate, and the fully connected deep neural network we use has a large number of parameters. Given these properties, we can see that AdaHessian optimizer demonstrates great performance in complicated task.

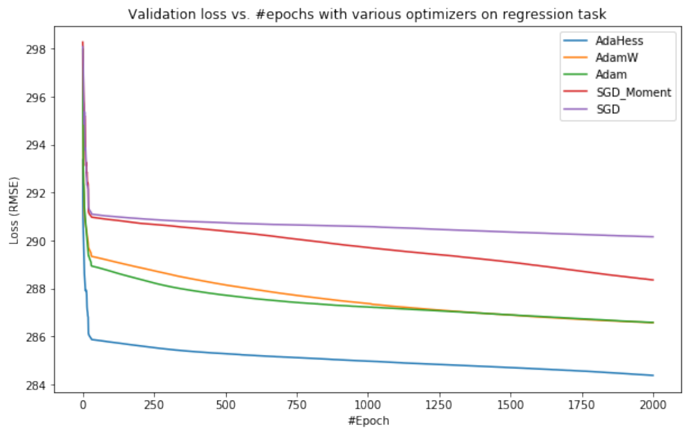

Figure 5: Validation loss vs. number of epochs with various optimizers on eggholder function regression

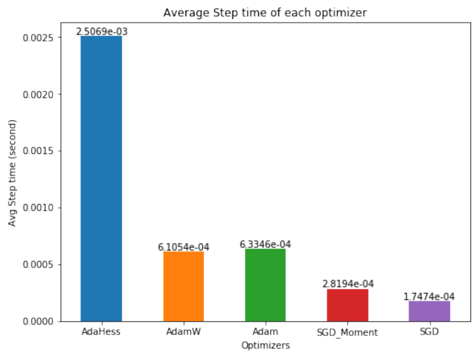

Figure 6: Average optimizer step time of optimizers in regression task

From the bar plot of time we can see that for this network, the optimization time is much lower than for Resnet-18 model, but still AdaHessian optimization takes longer time than the first-order optimizations (4x SGD w momentum, 14× SGD).

8 Conclusion

In this project we did in-depth analysis for first order and second order optimizers. We incorporate theoretical analysis and experiments in practice to achieve a thorough comparison.

Second order optimizers such as Newton Method are theoretically able to yield better results. However, Naive Newton Method is not practical, while efficient Newton Estimate (Quasi-Newton) still performs poorly on the Cifar-10 dataset with Cross Entropy loss.

The recent second-order optimizer AdaHessian demonstrates great performance on a variety of tasks, including CV tasks and the DNN regression tasks. In the classification task on Cifar-10, its accuracy is just a tiny lower than the highest(best performance by SGD with Momentum). In the regression experiment with our self-defined DNN and tuned hyper-parameters, AdaHessian outperforms all other optimizers while using longer but acceptable step time.

Our experiment results on the classification task align with the AdaHessian Paper's conclusion that SGD w Momentum gives the best maximum accuracy on cifar-10 and the Adahessian gets the very close, second highest score[4]. Our experiments are able to get close validation accuracy values to the paper's as well.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *Journal of Computer and System Sciences* 58.1 (1999), pp. 137–147. ISSN: 0022-0000. DOI: `https://doi.org/10.1006/jcss.1997.1545`. URL: `https://www.sciencedirect.com/science/article/pii/S0022000097915452`.

[2] Kuang Liu. *Train CIFAR10 with PyTorch*. `https://github.com/kuangliu/pytorch-cifar`. 2021.

[3] David Samuel. *AdaHessian*. `https://github.com/davda54/ada-hessian`. 2020.

[4] Zhewei Yao et al. *ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning*. 2020. arXiv: `2006.00719 [cs.LG]`.