Effectiveness of Second-Order Optimization for Non-convex Machine Learning

Qasim Wani*
Virginia Tech
qasim@vt.edu

Minh T. Nguyen*
Virginia Tech
mnguyen0226@vt.edu

Ryan Landry*
Virginia Tech
rlandry920@vt.edu

November 28, 2021

Abstract

Classical optimization algorithms have become unfavorable in the world of large models and big data. Over the past decade, first-order optimization methods such as SGD and Adam have dominated deep learning literature. Despite proven theoretical properties, second-order optimization techniques are far less prevalent and unconventional in modern deep learning due to their prohibitive computation. We bring awareness to these second-order techniques by highlighting their advantages, such as robustness to hyperparameters, finite step convergence, and escaping saddle points.

1 Introduction

The rise of deep-learning has been fuelled by improvements in optimization techniques and accelerators [22]. The optimization algorithms chosen by a deep learning practitioner determines the training speed and the final predictive performance of their model. To date, there is no theory that adequately explains how to make this choice [7][9]. Instead, our deep learning community relies on empirical studies and bench-marking [20][23][21]. Indeed, it is the de facto standard that papers introducing new optimizers report extensive comparison across a large number of workloads. Since the birth of AlexNet in 2012, applying Graphical Processing Units (GPUs) has become second nature in deep learning researches and applications. Due to its unique features in parallel computation, GPU allows for more mathematically heavy algorithms to be apply in deep learning [17][22].

In recent years, second-order algorithms are among the most powerful optimization algorithms with superior convergence properties as compared to first-order methods such as SGD and Adam [26][28]. The main disadvantage of traditional second-order methods is their heavier periteration computation and poor accuracy as compared to first-order methods [25]. Therefore, powerful GPUs are the reliable solution for costly computation of second-order methods.

To maximize the scientific progress and introduce deep learning practitioners to more powerful optimizers, we must have confidence in our ability to make empirical comparison between first-order and second-order optimization algorithms. We will compare and contrast them in terms of four metrics: Robustness to Hyperparameters, Generalization Error, and Wall Clock Time.

1.1 Main problem with classical optimization techniques

Depending on the domain problems and datasets, the goal of a deep learning researcher is to implement a model that will produce better and faster results and minimizes the loss function. In order to do this, the researcher must tune the hyperparameters. While tuning weights are key to improving accuracy

*Equal Contribution.
Computer Vision Final Project (ECE 4554 Fall 2021), Blacksburg, Virginia, USA.

for classification models, it is heavily dependent on the setting weights with the lowest loss function (gradient descent) and the direction of change of the slop during gradient descent (back propagation).

Non-convex problems are the most common case for neural networks, thus choosing an optimization strategy that seeks to find global optima in these neural networks is occasionally challenging due to process of estimation of a very large number of parameters in high dimensional search space [28]. As an improper optimization technique may make the networks to reside in the local minima during training without any improvement. Additionally, some deep learning researches show preference to Adam optimizer for all cases, due to its robustness and performance across numerous case scenarios [9].

Moreover, in practice, the large-scale nature of many modern ML problems poses computational challenges which have rendered may classical optimization methods inefficient or inapplicable. Hence an investigation is required to analyze the performance of optimizers depending on the model and dataset employed for a better understanding of their behaviour.

1.2 Main solution to classical optimization techniques

Optimizers are algorithms or methods used to change the attributes of a neural networks such as weights and learning rate in order to reduce the losses. How one should change the weights or learning rates varies by the optimizer. First-order optimization methods, such as SGD and its variants, have been the workhorse for deep learning applications due to their simplicity and versatility [28]. However, as GPUs are more available and deep neural networks are used for more complex problems, these first-order optimizers are by no means the ideal solution for training neural networks models.

There are often a lot of ad-hoc rules, specifically when choosing first-order optimizers, that deep learning researchers follow to get neural networks to converge to a point with good generalization properties. For instance, SGD with momentum is typically used in Computer Vision; Adam is used for training transformer models for Natural Language Processing; and Adaptive Gradient (AdaGrad) is used for Recommendation Systems [9]. Using the wrong first-order optimizer could lead to a neural networks stuck in the local minima. We observe this scenario when testing RMSprop with CIFAR-10 as depicted by Figure 4b.

1.3 Proposed solution

The deficiencies are particularly problematic in highly non-convex machine learning problems such as those that arise in neural networks applications. By incorporating curvature information, second-order optimization methods hold the promise to solve these well-known deficiencies. When implemented naively, second-order methods are clearly not computationally competitive with first-order methods [26]. However, the availability of GPUs will ease this computational problem of second-order optimization.

In this paper, we demonstrate two important points about empirical comparisons of neural networks optimizers. First, we show why first-order optimizers are more practical than second-order optimizers. Second, we explain why second-order optimizer should be wildly implemented, practiced, scaled in deep learning models.

The remainder of this paper is structured as follows. In Section 2, we provide an overview on neural networks, optimization techniques, brief mathematical review of frequently used first-order optimizers, and a detailed explanation of second-order optimization technique called Ada-Hessian. We present our experimental results and discussion in Section 3. We will show the effective of an efficient implementation to approximating Hessian with $O(d)$ memory complexity by making use of the Hutchinson's method along with a momentum based approach borrowed from first order method greatly improves prospective second order methods with favorable iteration cost over its advesaries.

2 Backgrounds

This section contains the mathematical overview of neural networks optimization techniques. Figure 1 provide a simple comparative visualization between first-order and second-order methods.

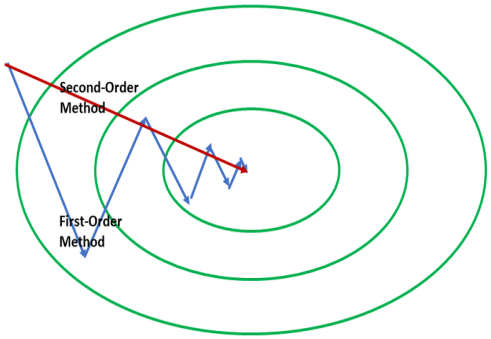


Figure 1: First-order gradient descent V.S. second-order gradient descent.

2.1 First-order optimization

First-order optimization algorithms explicitly involve using the first derivative (gradient) to choose the direction to move in the search space. Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function [12]. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of that function; the procedure is then known as gradient ascent. There are five first-order optimizers that we experiment with: SGD, RMSprop, AdaGrad, Adam, AdamW.

2.1.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an iterative optimization technique that uses mini batches of data to form an expectation of gradient, rather than the full gradient using all available data [15]. That is for weight w, a loss function L, and the learning rate η, we have the following equation:

$$w_{t+1} = w_t - \eta \nabla_w L(w_t)$$

SGD reduces redundancy compared to batch gradient, which recomputes gradients for similar examples before each parameter update, so it is faster to train on a machine learning model.

2.1.2 Root Mean Square Propagation

Root Mean Square Propagation (RMSprop) is an unpublished, adaptive learning rate method proposed by Geoffrey Hinton [13]. The motivation is that the magnitude of gradients can differ from different weights, and can change during learning, making it hard to choose a single global learning rate. RMSprop tackles this by keeping a moving average of the squared gradient and adjusting the weight updates by this magnitude [13]. The gradient updates are performed as:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} . g_t$$

RMSprop divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggested γ to be set to 0.9, while a good default value for the learning rate η is 0.001 [13].

2.1.3 Adaptive Gradient

Adaptive Gradient (AdaGrad) is the gradient-based optimizer that adapts the learning rate to the parameters, performs smaller learning rate for parameters associates with frequently occurring features, and executes larger learning rate for parameters associated with infrequent features [10]. In its update rule, AdaGrad modifies the general learning rate η at each time step t for every parameter w_i based on the past gradients for w_i:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{E[g^2]_{t,ii} + \epsilon}} . g_{t,i}$$

The benefit of AdaGrad is that it eliminates the need to manually tune the learning rate; mostly leaving it at a default value of 0.01 [10]. However, its main weakness is the accumulation of squared gradient in the denominator [10]. Since every added term is positive, the accumulated sum keeps growing during training which causes the learning rate to continue to shrink and become infinitely small. At this point, the algorithm is no longer able to acquire any additional knowledge.

2.1.4 Adam

Adaptive Moment Estimation (Adam) is an adaptive learning rate optimization algorithm that utilizes both momentum and scaling, combining the benefits of RMSprop and SGD [16]. The weight updates rule are performed as follow

$$w_t = w_{t-1} - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

with

$$m_t = \frac{m_t}{1 - \beta_1^t}$$
$$v_t = \frac{v_t}{1 - \beta_2^t}$$
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_1 v_{t-1} + (1 - \beta_2) g_t^2$$

Note that η is the learning rate (preferred 10^{-3}), ϵ is preferred at 10^{-8} or 10^{-10} to prevent dividing by zero. β_1 and β_2 are forgetting parameters, with typical values of 0.9 and 0.999, respectively [16]. In recent years, Adam has proven to work well in practice and is favorable to other adaptive learning-method algorithms [9].

2.1.5 AdamW

Adam with decoupled weight decay (AdamW) is a stochastic optimization method that modifies the typical implementation of weight decay in Adam, by decoupling weight decay from the gradient update [14]. To see this, L_2 regularization in Adam is implemented with the equation below where w_t is the rate of the weight decay at time t:

$$g_t = \nabla(w_t) \ + w_t w_t$$

while AdamW adjusts the weight decay term to appear in the gradient update:

$$w_{t+1,i} = w_{t,i} - \eta(\frac{1}{v_t + \epsilon}.m_t + w_{t,i} w_{t,i}), \forall t$$

2.2 Second-order optimization and Ada-Hessian

Second-order optimization algorithms explicitly involve using the second derivative (Hessian) to choose the direction to move in the search space. These algorithms are only appropriate for those objective functions where the Hessian matrix can be calculated or approximated. While first-order optimization methods only consider the gradient of the loss function, second-order methods also take the curvature of the loss function into account [27]. Thus, this makes second-order optimizations more powerful (but computationally cost) compared to first-order ones. There are a handful number of second-order optimization methods; however, we are mostly interested in Ada-Hessian.

2.2.1 Ada-Hessian

AdaHessian, a second order stochastic optimization algorithm dynamically incorporates the curvature of the loss function via adaptive estimates of the Hessian. AdaHessian incorporates the following approaches:

- fast Hutchinson based method to approximate the curvature matrix with low computational overhead.
- Root-mean-square exponential moving average smoothing function to minimize variations of the Hessian diagonal across different iterations.

- Block diagonal averaging function to reduce variance of the Hessian diagonal elements.

AdaHessian, like other second order techniques are resilient to *ill-conditioned* loss landscapes. An ill-conditioned problem is one where, for a small change in the inputs there exists a large change in the output. This makes the correct/optimal solution much harder to find. It is due to this property that learning rate tuning in first order methods is essentially a *babysit* process. Figure 5 demonstrates this effect. AdaHessian has the smallest variance among all methods of about 0.0003. Second order optimization utilizing the concept of gradient vector *preconditioning* before updating the weight vector. For a high dimensional problem, different parameters exhibit different behaviors. Based on the conditions, one parameter might change much smoother than another parameter. As a result, an optimizer should be capable of taking different steps per parameter, i.e., bigger steps for the flatter directions and relatively smaller steps for sharper directions. This can be mathematically represented as the following gradient step update:

$$w_{t+1} = w_t - \eta \boldsymbol{P}(t) \nabla_{w_t} L(f w_t), \quad t = 0, 1, \cdots \tag{1}$$

Here L refers to a loss function, f refers to a model we wish to learn with network parameters w, and $\mathbf{P}$ refers to a preconditioner matrix. Note that setting $\boldsymbol{P} = \boldsymbol{I}$ recovers original gradient descent. Extensive work has been done on finding an optimal class of preconditioner matrix. Some techniques make use of Fisher Information matrix [3], while others make use of past gradient information for adaptive gradient methods [4, 19, 11].

AdaHessian, like other second order optimization techniques, addresses the ill-conditioned problem by utilizing the curvature of the loss function which automatically rescales and rotates the gradient. This not only allows us to choose a better gradient direction, but also automatically adjust hyperparameters such as the learning rate for each parameter. Similar to Adam, each parameter can have its own learning rate. Second order optimization finds the optimal set of learning rate to solve the stochastic optimization problem. This results in better convergence as shown extensively by [1, 2, 6].

AdaHessian doesn't naively utilize Hessian information. Classical ways of computing the Hessian is to solve the linear system of equation using Newton method which generates optimal set of parameters at every local iteration. The main challenge with this approach is its high cost per iteration which is roughly cubic to the number of variables. The real effect of using this approach is seen when computing the direction of gradient, i.e. $\mathbf{H}^{-1} \nabla_w$ to update for parameter w. Several techniques leverage matrix free approaches which don't explicitly call the Hessian inverse and is only implicitly approximated, and thus, addressing the cubic computational complexity.

Just like Adam makes use of momentum for a smoother approximation of the gradient, adaptive second method make smoother approximations of the Hessian. Ideally, this helps smooth out local curvature noise to better approximate non-noisy curvature of the global loss landscape. However, such an approximation cannot be explicitly formed to be averaged, unlike the case for first order method which is essentially just a vector of information. Hutchinson's method solves this problem by allowing us to compute a stochastic, unbiased estimator of the trace of the Hessian matrix using a gradient-free method. Mathematically, this can be represented as:

$$Tr(\mathbf{A}) = \mathbb{E}[z^T \mathbf{A} z]$$

where z is a random Gaussian (in our case, Rademacher) distributed variable and A is the Hessian matrix. Hutchinson's method makes use of Monte Carlo approach to be more memory efficient and instead of using the full matrix to compute gradient at each step, we only make use of the Hessian diagonal, $\mathbf{D}$ which can be calculated using the above procedure via the trace function. AdaHessian uses a randomized numerical linear algebra method RandNLA [5] which applies a small number of matrix-vector products using a select few vectors using Chebychev expansion [24].

Another contribution AdaHessian makes in improving the approximation of Hessian diagonal is by reducing the variance of individual parameters when computing the Hessian diagonal. AdaHessian performs a spatial averaging function of the Hessian diagonal to smooth out spatial variations. This can be mathematically represented as:

$$\mathbf{D}^{(s)}[ib + j] = \frac{\sum_{k=1}^{b} \mathbf{D}[ib + k]}{b}, \quad for \quad 1 \leq j \leq b, \quad 0 \leq i \leq \frac{d}{b} - 1,$$

where $\mathbf{D}$ is the Hessian diagonal, $\mathbf{D}^{(s)}$ is the spatially averaged Hessian diagonal, b is the spatial average block size, and d is the number of model parameters divisible by b. More details are provided

in the AdaHessian paper [27]. Ultimately, AdaHessian uses this spatially averaging function along with Hessian momentum inspired from Adam to smooth out local variations in the Hessian diagonal using Hutchinson's method. This leads to the first and second order moments (m_t and v_t) for AdaHessian, defined as follows:

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\nabla_i}{1-\beta_1^t},$$

$$v_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\mathbf{D}_i^{(s)}\mathbf{D}_i^{(s)}}{1-\beta_2^t}}^{k}$$

where $0 < \beta_1, \beta_2 < 1$ are the first and second moment parameters, respectively.

3 Experiment Setup

We implemented five first-order methods: SGD [15], Adam [16], AdamW [14], AdaGrad [10], and RMSprop [13] to compare with a second-order method of choice: AdaHessian [27]. We experimented our work on two computer vision datasets, MNIST and CIFAR-10. AdaHessian has shown great resiliency to learning rate. We also showcase the effective of using Hutchinson method by measuring relative wall-clock time between all these methods. Had it not been for such efficient approximationo of the Hessian diagonal, the cost per iteration would've been much higher. All our experiments and models are open-sourced for reproducibility.[1]

This section will provide details about the chosen datasets and the Convolutional Neural Network models.

3.1 Datasets

We use supervised learning with MNIST and CIFAR-10 datasets to benchmark the optimization techniques explained above.

3.1.1 MNIST

The MNIST (Modified National Institute of Standards and Technology) dataset is a large collection of handwritten digits [8]. MNIST is primarily used to experiment with different machine learning algorithms and to compare their relative strengths. MNIST has been considered as the simple, light-weight but well-known test bench amongst the machine learning community, thus it is sensible to us to experiment on the optimization techniques using MNIST and compare their performances.

Figure 2: Sample images from MNIST test dataset.

MNIST has a training set of 60,000 examples, and a test set of 10,000 examples [8]. It is a subset of a larger NIST Special Database 3 (digits written by employees of the United States Census Bureau)

[1] **GitHub:** github.com/mnguyen0226/soo_non_convex_ml

and Special Database 1 (digits written by high school students) which contain monochrome images of handwritten digits [8]. The digits have been size-normalized and centered in a fixed-size image. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm which provides the 28x28 number image. The image labels values are 0 to 9.

3.1.2 CIFAR-10

In addition to MNIST dataset, we also use CIFAR-10 (Canadian Institude for Advanced Research, 10 classes) dataset as a second test bench to compare the optimization methods.

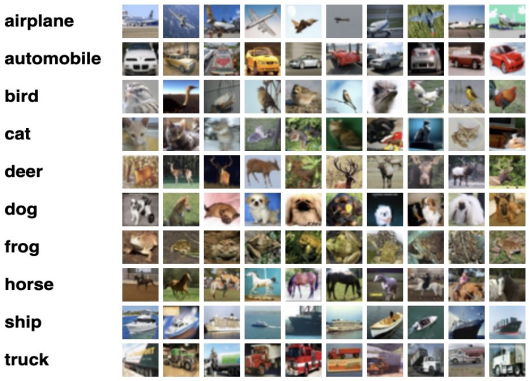


Figure 3: Sample images from CIFAR-10 test dataset.

It is a subset of the Tiny Images dataset and consists of 60,000 32x32 color images [18]. The images are labelled with one of 10 mutually exclusive classes: airplane, pickup truck, bird, cat, deer, dog, frog, horse, ship, and truck. There are 6000 images per classes with 5000 training and 1000 testing images per class [18].

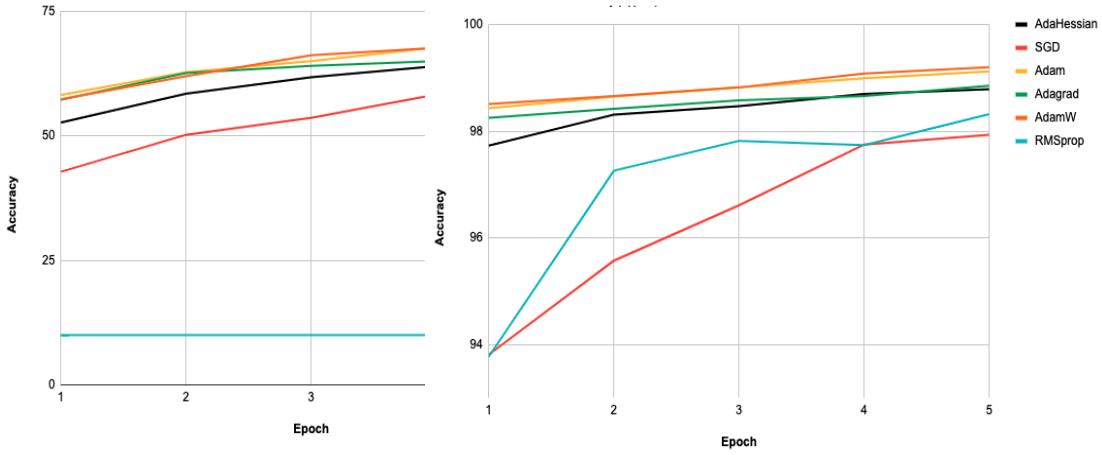
3.2 Convolutional neural network models

Convolutional neural networks (CNNs) are powerful models with non-convex objective function. Unlike most fully connected neural networks, weight sharing in CNNs results in vastly different gradients in different layers. In our experiments, we made model choices that are consistent with previous publications in the area. We used two different CNNs models for MNIST and CIFAR-10 datasets. One the one hand, a CNNsarchitecture with six alternating stages of two 3x3 convolution filters, two dropout layers, and two fully connected layer with rectified linear hidden units (ReLU) activation are used for MNIST dataset. On the other hand, a CNNs architecture with seven alternating stages of three 3x3 convolution filters, one 2x2 max-pooling filters, two fully connected layers, and a dropout layer with exponential linear unit (eLU) activation are used for CIFAR-10 dataset.

4 Results

We compared AdaHessian with five other first order methods on MNIST and CIFAR-10 datasets. We found that AdaHessian performs near optimally in these settings as shown in figure 4. It was quite interesting to observe clear lack of any generalization of RMSprop on CIFAR-10. This is perhaps due to an untuned learning rate which would require extra fine-tuning, furthering our case on the pitfalls of first-order optimization methods. Due to constraints on time and resources, we tested our algorithms for five epochs each which still proved to generate good enough results, mostly due to the lack of complexity in the underlying datasets.

The major advantage of first-order methods is observed when comparing their wall-clock time with AdaHessian. Because we're implicitly computing the Hessian diagonal, we expend exponentially more time computing the direction of gradient for individual parameters as compared to first order methods. While this seems very high as compared to first-order methods, AdaHessian with its utilization of trace matrix has made a significant improvement on estimate of Hessian information.



(a) Performance on CIFAR-10 with batch size 20

(b) Performance on MNIST with batch size 64

Figure 4: Performance (test accuracy) of various optimization algorithms on vision datasets.

Wall Clock Time Comparison (in seconds)		
Optimizers	**MNIST**	**CIFAR-10**
AdaHessian	26.9	104.3
SGD	8.1	34.5
Adam	8.5	32.6
AdaGrad	8.1	37.7
AdanW	8.3	33.2
RMSprop	7.9	47.6

Table 1: Cost per iteration of various optimization methods.

Perhaps the most noticable advantage of using AdaHessian is its robustness to learning rate, λ. AdaHessian bypasses the ill-conditioned matrix problem by making use of the curvature information. We see this in Figure 5 where with different learning rates, we observe relatively same generalization accuracy while as first-order methods in general clearly aren't robust to the choice of alpha.

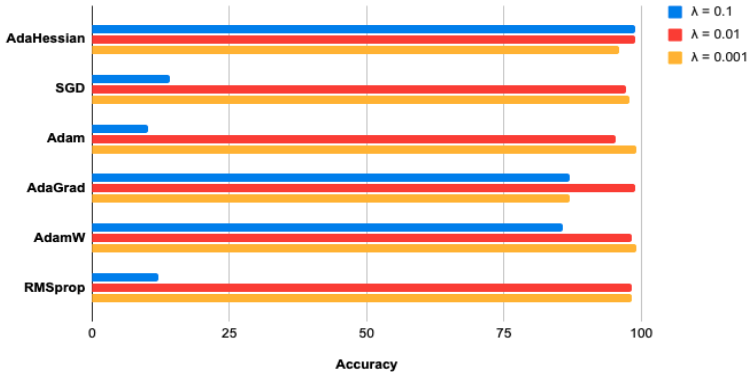


Figure 5: Robustness to learning rate

5 Conclusion

In this project our team has provided mathematical backgrounds of a second-order optimizer technique and compared it to several first-order techniques using Convergence Time, Robustness to Hyperparameters, Generalization Error, and Wall Clock Time in hopes to highlight its advantages. In order to improve this experiment, more second-order techniques could be explore and also be tested on different datasets to create a variety of different trials. In the future project, we suggest to take in to account other metrics besides the ones we focused on to get a more holistic view of the optimizers' performances. Ada-Hessian is truly a powerful method for image recognition compared to first-order methods. There is still a lot of room for experimentation and investigation with other second-order optimizers in different deep learning models to solve complex tasks in computer vision such as Object-tracking or multi-object recognition.

6 Contributions

All team members were involved in researching different Computer Vision topics and creating a proposal for our specific project. Once the topic was decided on, all members focused on different algorithms to learn about. Ryan focused on SGD and ADAM. Minh focused on RMS Prop, Adagrads, and AdaDelta. Qasim focused on Ada-Hessian and wrote a script to take in different hyper-parameters. Once we gathered information on all of the different algorithms the team created experiments and then Qasim ran them all. After all the experiments were run, all team members helped write the final report with Ryan and Minh focusing on the Introduction, Background and Conclusion sections and Qasim focusing on the Experiment and Results sections. All team members contributed equally to the project and were able to collaborate well together.

References

[1] Naman Agarwal, Brian Bullins, and Elad Hazan. "Second-order stochastic optimization in linear time". In: *stat* 1050 (2016), p. 15.

[2] Naman Agarwal et al. "Finding approximate local minima faster than gradient descent". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 2017, pp. 1195–1199.

[3] Shun-Ichi Amari. "Natural gradient works efficiently in learning". In: *Neural computation* 10.2 (1998), pp. 251–276.

[4] Shun-ichi Amari et al. "When Does Preconditioning Help or Hurt Generalization?" In: *arXiv preprint arXiv:2006.10732* (2020).

[5] Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. "An estimator for the diagonal of a matrix". In: *Applied numerical mathematics* 57.11-12 (2007), pp. 1214–1229.

[6] Raghu Bollapragada, Richard H Byrd, and Jorge Nocedal. "Exact and inexact subsampled Newton methods for optimization". In: *IMA Journal of Numerical Analysis* 39.2 (2019), pp. 545–578.

[7] Dami Choi et al. *On Empirical Comparison of Optimizers of Deep Learning*. 2020. URL: `https://arxiv.org/pdf/1910.05446.pdf`.

[8] Dan C. Ciresan et al. *High-Performance Neural Networks for Visual Object Classification*. 2011. URL: `https://arxiv.org/pdf/1102.0183.pdf`.

[9] E. M. Dogo et al. "A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks". In: *International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)* (2018).

[10] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* (2011).

[11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[12] *Gradient Descent*. URL: `https://en.wikipedia.org/wiki/Gradient_descent`.

[13] G. Hinton, N. Srivastava, and K. Swersky. "rmsprop: Divide the gradient by a running average of its recent magnitude". In: (2012).

[14] Loshchilov Ilya and Hutter Frank. *Decoupled Weight Decay Regularization*. 2019. URL: https://arxiv.org/pdf/1711.05101.pdf.

[15] Sutskever Ilya et al. *On the importance of initialization and momentum in deep learning*. 2013. URL: https://www.cs.toronto.edu/~hinton/absps/momentum.pdf.

[16] Diederik P. Kingma and Jimmy Lei Ba. *Adam: A Method For Stochastic Optimization*. 2017. URL: https://arxiv.org/pdf/1412.6980.pdf.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in neural information processing systems* (2012).

[18] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[19] Yann A LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[20] F. Schneider, L. Balles, and P. Hennig. *DeepOBS: A Deep Learning Optimizer Benchmark Suite*. 2019. URL: https://arxiv.org/abs/1903.05499.

[21] S. Schneider et al. *Past, present and future approaches using computer vision for animal reidentification from camera trap data*. 2019.

[22] Mittal Sparsh and Vaishay Shraiysh. "A Survey of Techniques for Optimizing Deep Learning on GPUs". In: *Journal of Systems Architecture* (2019).

[23] A Wilson et al. "The marginal value of adaptive gradient methods in machine learning". In: *Advances in Neural Information Processing Systems* (2017).

[24] Robert E Wyatt. "Matrix spectroscopy: Computation of interior eigenstates of large matrices using layered iteration". In: *Physical Review E* 51.4 (1995), p. 3643.

[25] Peng Xu, Roosta Fred, and W. Mahoney Michael. "Newton-type methods for non-convex optimization under inexact hessian information." In: *Mathematical Programming 184.1* (2020).

[26] Peng Xu, Roosta Fred, and W. Mahoney Michael. "Second-order optimization for non-convex machine learning: An empirical study". In: *Society for Industrial and Applied Mathematics (SIAM)* (2020).

[27] Zhewei Yao et al. *ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning*. 2021. URL: https://arxiv.org/abs/2006.00719.

[28] Chengxi Ye et al. *On the Importance of Consistency in Training Deep Neural Networks*. 2017. URL: https://www.researchgate.net/publication/318868196_On_the_Importance_of_Consistency_in_Training_Deep_Neural_Networks.