UNIVERSITY AT ALBANY
STATE UNIVERSITY OF NEW YORK

Project Report

Project Name : Comparative Analysis of Ada-Hessian and First-Order Optimizers for CSI-Based Sign Language Recognition

Course Code : AMAT 591 (Spring 2025)

Course Name : Optimization Methods and Non Linear Programming

Instructor : Dr. Zi Yang

Submitted By

Name : Joy Saha

Student ID : 001657938

Submission Date : 05/14/2025

1 Introduction

With the advancement of AI and machine learning, many traditional optimization techniques have become inadequate for training or evaluating deep learning models, which often involve a massive number of parameters. To date, the most widely used optimizers for deep learning are Stochastic Gradient Descent (SGD) and its variants. However, these first-order methods rely solely on gradient information and do not utilize second-order information about the loss landscape. As a result, they can fail to capture important class-related features, potentially leading to poorer generalization.
The effectiveness of SGD and its variants heavily depends on the specific task as well as the tuning of hyperparameters such as learning rate, decay schedule, and momentum. In popular models like *AlexNet* and *GoogLeNet*, extensive trial-and-error was necessary to find the best hyperparameter configuration for each task. Achieving an optimal solution thus requires satisfying multiple conditions, and even then, there is no guarantee of good generalization.

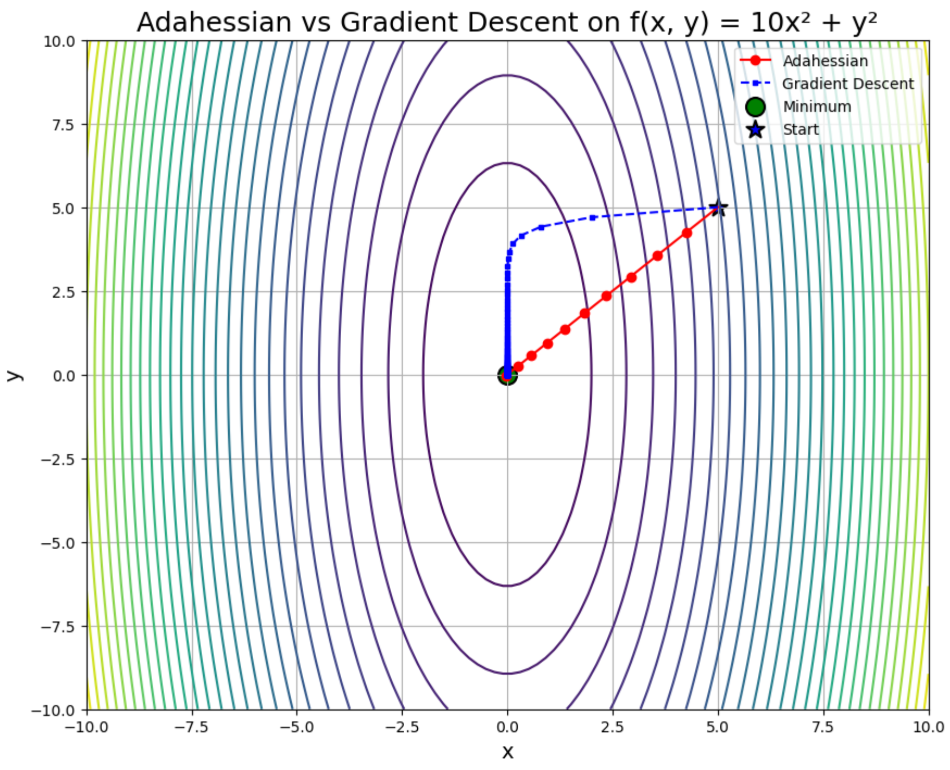


Figure 1: Trajectory for Ada-Hessian and Gradient Descent Method

In contrast, second-order optimization methods leverage curvature information from the loss landscape, making them generally more robust to hyperparameter settings. These methods adjust the gradient vector by scaling and rotating it based on the curvature, allowing for larger updates in flatter directions and smaller updates in steeper ones. As illustrated in Figure 1, the Ada-Hessian optimizer converges in just a few iterations, whereas traditional gradient descent methods require many more steps—even after tuning. However, the main drawback of second-order methods lies in their computational

cost: calculating the full Hessian matrix requires quadratic memory, i.e., $\mathcal{O}(d^2)$, making them less practical for large-scale models.
In this project, I implemented and compared the performance of the Ada-Hessian optimizer against several first-order optimization methods, including SGD, Adam, AdamW, Adamax, and Nadam, for CSI-based sign language detection. Details of the CSI dataset are provided later in the experimental setup section. To efficiently implement Ada-Hessian , I approximated the diagonal of the Hessian matrix using the Hutchinson method, avoiding the computational cost of explicitly calculating the full Hessian. To further reduce the variance introduced by stochastic noise, we applied a block RMS moving average to the Hessian diagonal estimates. Additionally, I evaluated Ada-Hessian under various hyperparameter settings, such as different learning rates and weight decay values, to assess its performance robustness.

2 Problem Formulation

Machine learning tasks are often reduced to the challenge of minimizing a non-convex loss function:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(\mathbf{x}_i, y_i; \boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ represents the model parameters, $\ell_i(\mathbf{x}_i, y_i; \boldsymbol{\theta})$ denotes the loss function for the i-th data sample, $(\mathbf{x}_i, y_i)$ is the input-label pair, and N is the total number of training examples.
To facilitate optimization, I define the gradient of the loss with respect to the model parameters as $\mathbf{g} = \frac{1}{N_B}\sum_{i=1}^{N_B} \frac{\partial \ell_i}{\partial \boldsymbol{\theta}}$, and the corresponding second-order derivative (i.e., the Hessian) as $\mathbf{H} = \frac{1}{N_B}\sum_{i=1}^{N_B} \frac{\partial^2 \ell_i}{\partial \boldsymbol{\theta}^2}$, where N_B denotes the mini-batch size used during training. Solving Equation 1 in the context of complex deep learning problems is a challenging task that requires careful selection of hyperparameters. Due to their efficiency and strong empirical performance, first-order optimization algorithms—such as SGD, Adam, Adagrad, and RMSProp—are widely adopted in machine learning tasks. The general update rule for these methods can be expressed as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\mathbf{m}_t}{\mathbf{v}_t}, \tag{2}$$

where η_t is the learning rate, and $\mathbf{m}_t$ and $\mathbf{v}_t$ represent the first- and second-order moment estimates of the gradients, respectively. Equation 2 is highly sensitive to the choice of learning rate, decay schedule, and momentum parameters. To address these challenges, various optimization techniques have been proposed that incorporate knowledge of the data, apply gradient scaling, and utilize historical gradient information. Among these, one of the most widely used methods in modern machine learning is the Adam optimizer, which updates the first and second moment estimates as follows:

$$\mathbf{m}_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i}\mathbf{g}_i}{1-\beta_1^t}, \qquad \mathbf{v}_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i}\mathbf{g}_i\mathbf{g}_i}{1-\beta_2^t}} \tag{3}$$

where $0 < \beta_1, \beta_2 < 1$ are hyperparameters representing the exponential decay rates for the first and second moment estimates, respectively. A variant of Adam that incorporates decoupled weight decay has demonstrated significant performance improvements in our CSI-based sign language detection task. To address limitations such as sensitivity to hyperparameters and task-specific tuning, second-order optimization methods have been extensively studied. Techniques such as Newton's method, BFGS, and quasi-Newton methods offer the advantage of identifying second-order critical points and selecting more informed descent directions. However, computing the full Hessian matrix requires quadratic memory complexity, making these methods computationally expensive. Some approaches attempt to approximate the Hessian using the Gauss-Newton method within a trust region framework. Despite theoretical benefits, such methods are often impractical due to the complexity of modern deep learning models and the stochastic nature of the Hessian's diagonal entries.
In contrast, stochastic gradient descent (SGD) reduces noise and improves convergence by leveraging momentum and moving averages, helping it escape suboptimal directions. Building on this idea, I demonstrate that Ada-Hessian addresses these limitations by incorporating Hutchinson's method and spatial averaging. This combination mitigates the impact of stochasticity in the Hessian and improves performance in our CSI-based sign language detection task.

3 Methodology

For a general loss function $\mathcal{L}(\theta) : \mathbb{R}^d \to \mathbb{R}$, let the gradient and Hessian at iteration t be denoted by $g_t = \nabla \mathcal{L}(\theta_t), H_t = \nabla^2 \mathcal{L}(\theta_t)$. A general Hessian-based descent direction can be expressed as:

$$\Delta\theta_t = H_t^{-k} g_t, \tag{4}$$

where H_t^{-k} denotes the matrix power of the inverse Hessian for some $0 < k < 1$. This is computed using the eigendecomposition of the Hessian: $H_t^{-k} = U_t \Lambda_t^{-k} U_t^\top$, where U_t is the matrix of eigenvectors and Λ_t is the diagonal matrix of eigenvalues. This formulation interpolates between gradient descent ($k = 0$) and Newton's method ($k = 1$). The key idea is to precondition the gradient using curvature information from the Hessian. The update direction $H_t^{-k} g_t$ adapts to the geometry of the loss surface, improving convergence by accounting for curvature variation in different directions. As illustrated in

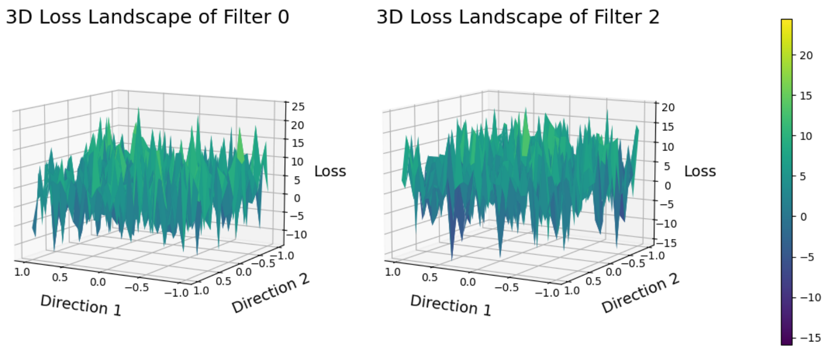


Figure 2: Example of Anisotropic Curvature in the Loss Landscape.

the Figure 2, the loss landscape can differ significantly across different channels, making curvature-aware updates essential for efficient optimization. Preconditioning the gradient with the Hessian effectively normalizes ill-conditioning by scaling different directions according to the curvature, helping the optimizer adapt to variations across dimensions. However, computing the full Hessian is computationally expensive, and relying on local Hessian information can introduce noise due to the non-smoothness of the loss landscape. These challenges can be mitigated by using only the diagonal of the Hessian, rather than the full matrix, leading to a more efficient and stable approximation. This approach forms the basis of inexact Newton methods, where the gradient is preconditioned with an approximate (diagonal) inverse Hessian at each iteration. The update direction now can be written as:

$$\Delta\theta = diag(H)^{-k} g, \tag{5}$$

Let diag(H), denoted as D, represent the diagonal of the Hessian matrix. The Hessian diagonal can be efficiently approximated using **Hutchinson's method**. Specifically, we have:

$$D = \text{diag}(H) \approx \mathbb{E}[z \odot (Hz)], \tag{6}$$

where $z \in \mathbb{R}^d$ is a random vector sampled from a *Rademacher distribution* (each element of z is independently chosen from $\{-1, +1\}$ with equal probability), and $\odot$ denotes the element-wise (Hadamard) product. To derive this approximation, consider the derivative:

$$\frac{\partial(g^\top z)}{\partial\theta} = \frac{\partial g^\top}{\partial\theta} z + g^\top \frac{\partial z}{\partial\theta} = Hz \tag{7}$$

Since z is independent of θ, the second term vanishes. Therefore, Hutchinson's method gives an unbiased estimate of the Hessian diagonal using only Hessian-vector products, which is computationally efficient for large-scale optimization problems. Another advantage of using the Hessian diagonal is that it enables computation of a moving average to reduce local noise in curvature estimates. This allows to smooth out local fluctuations and obtain a more stable and representative estimate of the global curvature. Mathematically, we apply spatial averaging to the Hessian diagonal. Let D be the original Hessian diagonal and $D^{(s)}$ its spatially averaged version. The averaging is done over non-overlapping blocks of size b, and the elements are updated as follows:

$$D^{(s)}[ib + j] = \frac{1}{b} \sum_{k=1}^{b} D[ib + k], \quad \text{for } 1 \le j \le b,\ 0 \le i \le \frac{d}{b} - 1, \tag{8}$$

where:

- $D[i]$ refers to the i-th element of the original Hessian diagonal,
- $D^{(s)}[i]$ refers to the i-th element of the spatially averaged diagonal,
- b is the spatial averaging block size,
- d is the number of model parameters, assumed to be divisible by b.

This averaging technique effectively smooths the noisy local curvature information and provides a more robust global Hessian approximation for use in optimization. Let $\tilde{D}_t$ represent the Hessian diagonal with momentum at iteration t. It is computed using an exponential moving average as follows:

$$\tilde{D}_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} D_i^{(s)} \odot D_i^{(s)}}{1-\beta_2^t}}, \tag{9}$$

The first- and second-order moment estimates in Ada-Hessian follow the same structure as those in Adam(Equation 3), with the key difference that the gradient g_i is replaced by the spatially averaged Hessian diagonal $D_i^{(s)}$, as shown below.

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} g_i}{1-\beta_1^t}, \quad v_t = (D_t)^k = \left(\sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} D_i^{(s)} D_i^{(s)}}{1-\beta_2^t}}\right)^k \tag{10}$$

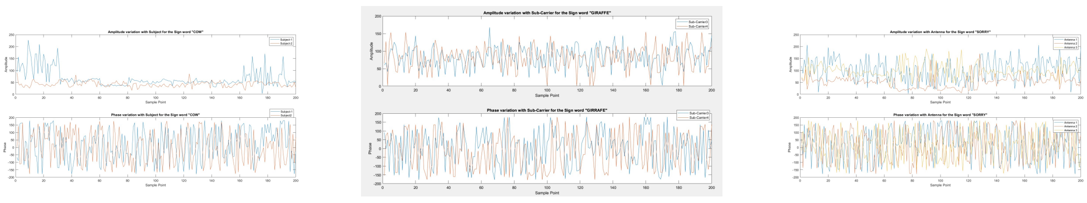
The full Algorithm for Ada-Hessian can be summarized below:

```
Algorithm 1: Ada-Hessian
Input: Initial parameter $\theta_0$, learning rate $\eta$, exponential decay rates $\beta_1, \beta_2$, block
       size $b$, Hessian power $k$
Set: $m_0 = 0$, $v_0 = 0$
1 for $t = 1, 2, \ldots$ do
2     Compute gradient: $g_t = \nabla\mathcal{L}(\theta_t)$ ;
3     Estimate Hessian diagonal: $D_t = \text{diag}(H_t)$ using Equation 6 ;
4     Compute spatially averaged Hessian diagonal $D_t^{(s)}$ (Equation 8) ;
5     Compute momentum-based smoothed Hessian diagonal $\tilde{D}_t$ (Equation 9) ;
6     Update $m_t$ and $v_t$ based on Equation 10
7     Compute parameter update: $\theta_t = \theta_{t-1} - \eta \cdot \frac{m_t}{v_t}$
```

4 Experimental Setup

4.1 CSI Data

Channel State Information (CSI) has recently been applied to various tasks such as activity recognition, localization, and sign language recognition. To evaluate the performance of the AdaHessian optimizer, we used publicly available CSI data for sign language recognition. In this setup, a user performs sign gestures in front of a Wi-Fi station, which exchanges packets with a nearby access point. The access point, equipped with three external antennas, sends packets to the Wi-Fi station, which captures the data using its internal antenna. For each subcarrier and antenna, 200 samples are collected—30 subcarriers for phase information and 30 for amplitude. Figure 3 illustrates how CSI data varies across different subjects, subcarriers, and antennas. A single CSI sample is represented as a tensor of shape $(200, 60, 3)$. We used three datasets: *Home*, *Lab*, and *Lab150*. The *Home* and *Lab* datasets each contain 276 sign gestures performed by a single subject. In



(a) Subject variation (b) Subcarrier variation (c) Antenna variation

Figure 3: Variation of CSI data with respect to subject, subcarrier, and antenna.

contrast, the *Lab150* dataset includes 150 sign gestures performed by five different users. Detailed information about these datasets is provided in Table 1.

Table 1: Details of the CSI-based sign language datasets.

Dataset	# Sign Labels	Repetitions Per Sign	# Instances
Home	276	10	2760
Lab	276	20	5520
Lab150	150	10	7500

4.2 Pre-Processing and Used Neural Network

I performed preprocessing on the CSI data before feeding it into the model. Both amplitude and phase components were processed to eliminate unwanted noise present in the raw CSI signals. The specific preprocessing steps are omitted from this report for brevity. The architecture of the neural network used for classification is illustrated in Figure 4.

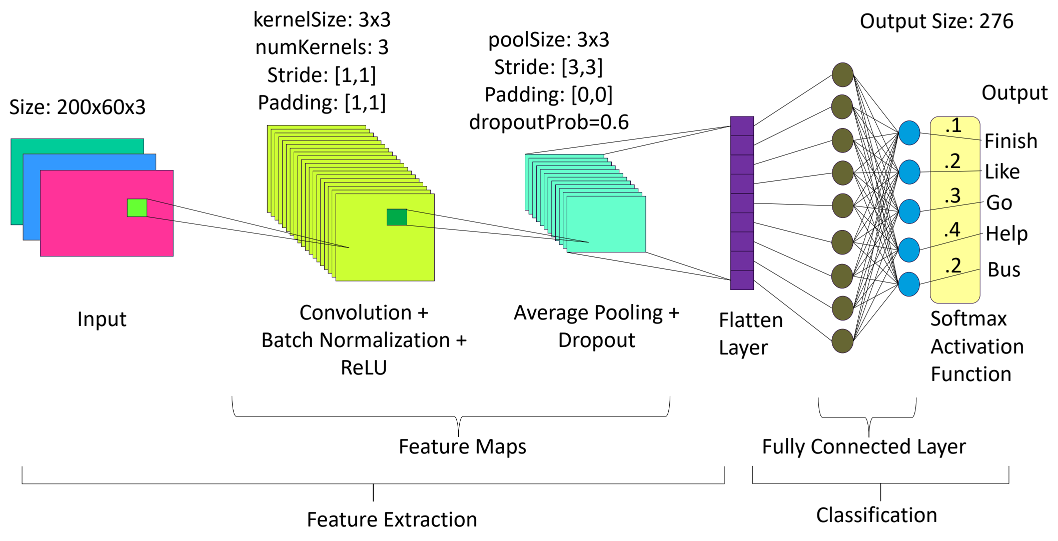


Figure 4: Neural Network Architecture Used for CSI-based Sign Language Classification.

4.3 Ada-Hessian Setup

To ensure consistency across experiments, I used the same weight decay of 5×10^{-4} and the same β_1 and β_2 values as used in Adam. All training and validation were conducted with a batch size of 256. A learning rate of 0.01 was used for all optimizers except Ada-Hessian , for which we used a learning rate of 0.15 and a block size of 9. Each experiment was run for 300 epochs, and we applied a learning rate scheduler to reduce the learning rate by a factor of 10 at epochs 80, 160, and 240.

5 Results

I conducted experiments by varying the Hessian power parameter k in Equation 5, using the values $k = 0.2, 0.4, 0.6, 0.8$, and 1.0 when computing the descent direction. My observations indicated that changing the diagonal power of the Hessian had a minimal effect on the training loss and training accuracy, as illustrated in Figure 5. However, in the validation setting, I observed a more noticeable difference. Specifically, when $k = 0.2$, the validation accuracy reached 96.6%, which is 6.9% higher than the accuracy achieved with $k = 1.0$. For the other values of k, the variation in validation accuracy was relatively small, fluctuating within a range of 2–3%. Based on these results, I chose a general value

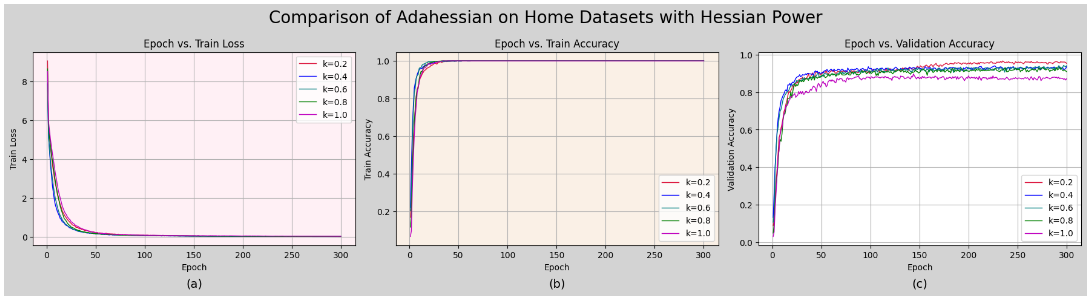


Figure 5: Effect of varying the Hessian power k on (a) training loss (b) training and (c) validation accuracy.

of $k = 0.5$ for the Hessian power in all subsequent experiments, unless otherwise stated.

Next, I evaluated the performance of various optimizers, including Adam, AdamW, RMSprop, SGD, Adamax, and Nadam, on the same dataset while tracking comparable metrics. As shown in Figure 6, the performance of most optimizers—particularly in terms of training loss, training accuracy, and validation accuracy—was highly sensitive to the choice of learning rate and its scheduling. This sensitivity is evident in the fluctuations observed across the training curves. In contrast, AdaHessian exhibited a much smoother trajectory, indicating that its performance is less affected by the choice of these hyper-parameters.

Among the optimizers tested, AdamW achieved the highest validation accuracy at 95.3%, followed by our proposed method, which attained 89.7%, outperforming SGD, which reached 88.4%. Ada-Hessian also surpassed RMSprop and Adam by 6.0% and 7.1%, respectively, in terms of validation accuracy. These results suggest that Ada-Hessian , despite leveraging second-order Hessian information, offers competitive performance compared to widely used first-order optimizers like SGD and Adam, which are often preferred

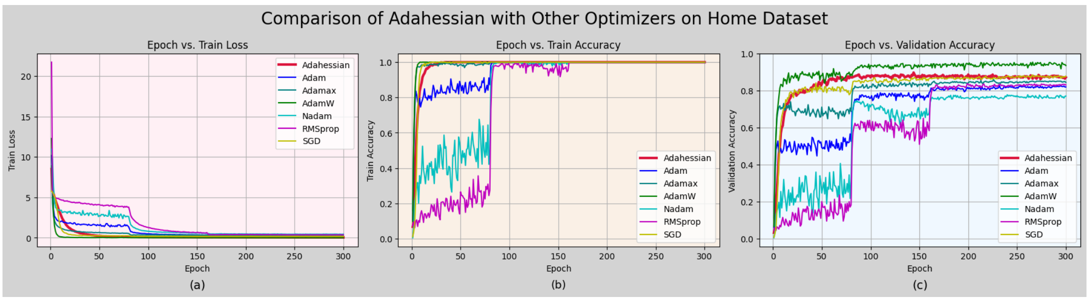


Figure 6: Performance of Various Optimizer. (a) training loss (b) training and (c) validation accuracy vs epoch.

for their lower computational cost in deep learning frameworks.

Then, I evaluated the performance of Ada-Hessian using the configuration $k = 0.5$, learning rate $= 0.15$, block size $= 9$, and weight decay $= 5 \times 10^{-4}$on CNN-based sign language classification task across all three datasets. I observed that the model converged within 50 iterations for all datasets(see Figure 7). On the Home dataset, the model achieved an accuracy of approximately 95%, while on the Lab dataset, it reached over 99% accuracy. However, for the Lab150 dataset, the validation accuracy dropped to around 75%. This reduction can be attributed to the nature of the Lab150 dataset, which involves a multi-user setup and introduces higher variability in the data for the same sign labels—a factor also noted by the dataset's providers. Despite the drop in accuracy for

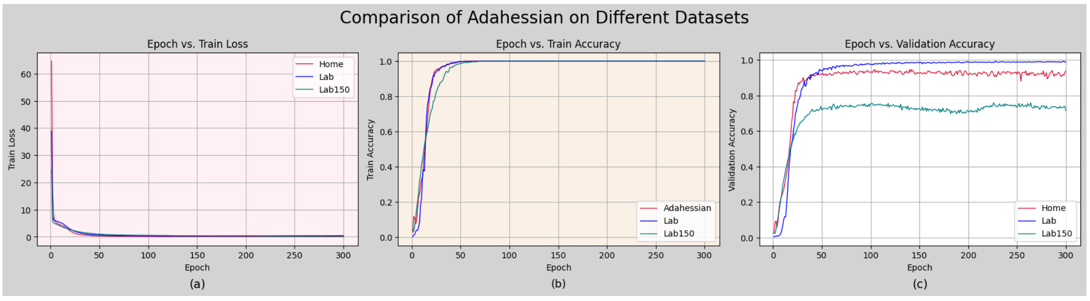


Figure 7: Performance of Ada-Hessian on all three dataset. (a) training loss (b) training and (c) validation accuracy vs epoch

Lab150, the results highlight Ada-Hessian 's ability to achieve strong performance with fewer iterations. This efficiency is due to its use of second-order gradient information, which helps shape the descent direction more effectively by adapting to the curvature of the loss surface. This curvature may vary significantly across different directions, as illustrated in Figure 2, and the Hessian-based updates help navigate it more precisely toward local minima.

In Table 2, I summarize the performance of various optimizers across all three datasets. Notably, second order method based on Ada-Hessian achieves the highest validation accuracy on the Lab dataset, clearly outperforming all other optimizers in this setting. Among the evaluated methods, only AdamW demonstrates consistently strong performance across all datasets. This observation underscores the sensitivity of many optimizers to hyperparameters such as learning rate, learning rate scheduler, and momentum. While AdamW outperforms Ada-Hessian by 0.67% on the Home dataset and by 3.91% on

Optimizer	Home	Lab	Lab150
Adam	82.61	28.71	63.67
Adamax	85.87	50.00	69.40
AdamW	**95.29**	97.01	**78.40**
Nadam	77.72	27.26	67.13
RMSprop	83.70	26.72	53.73
SGD	88.41	93.21	70.47
Ada-Hessian	94.62	**99.31**	74.51

Table 2: Comparison of optimizers Validation Accuracy (%) across different datasets: Home, Lab, and Lab150. The best result is bold+yellow-shaded and the second best is gray-shaded.

the Lab150 dataset, these gains are likely influenced by its effective use of weight decay—a factor that appears crucial for generalization in these tasks. On the other hand, most optimizers exhibit reasonable convergence during training but suffer significant performance drops during validation, particularly on the Lab dataset. For instance, optimizers like RMSprop, Nadam, and Adam achieve validation accuracies of less than 28.71%, indicating a potential overfitting or inability to generalize in scenarios with low inter-user consistency. These results highlight the advantage of using second-order information, as in Ada-Hessian , in scenarios with high variability or complex data distributions, as well as the critical role of careful hyperparameter tuning for first-order optimizers.

While Ada-Hessian is a second-order optimization method and offers improved conver-

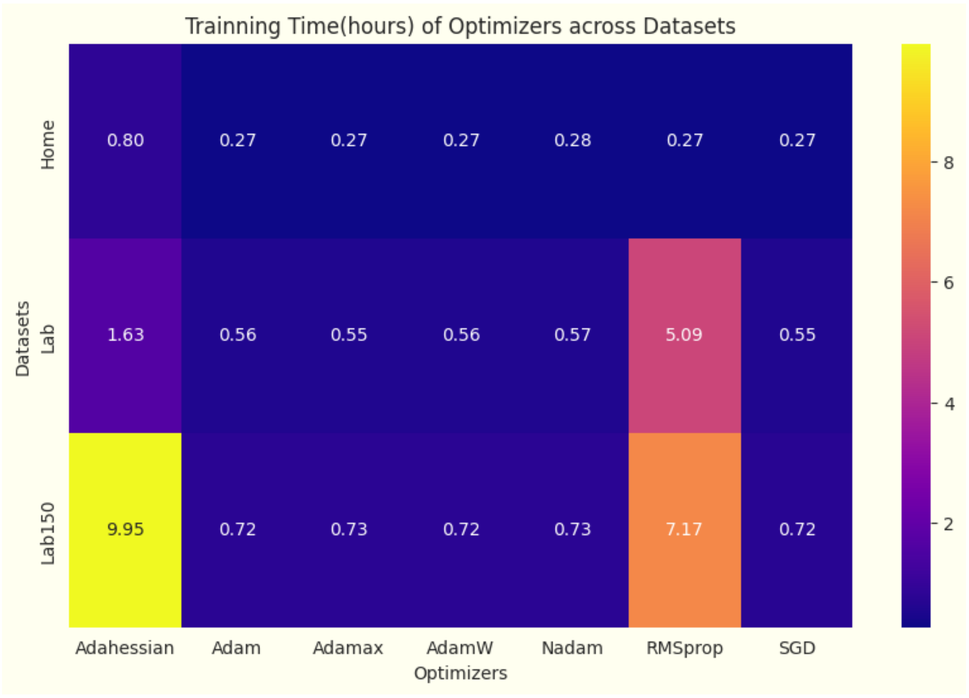


Figure 8: Comparison of Training Time

gence behavior, it comes with additional computational and time overhead due to the calculation of the Hessian-based descent direction. As shown in Figure 8, this overhead becomes evident in the training times: for the Home and Lab datasets, Ada-Hessian takes approximately 3 and 5 times longer to train, respectively, compared to first-order methods. The training time increases further for the Lab150 dataset, highlighting how the computational complexity of second-order methods scales with dataset size. This trade-off between improved convergence and increased resource requirements is an important consideration when choosing an optimization method for practical applications.

Learning Rate	**.05**	**.10**	**.15**	**.2**	**.25**
Accuracy(%)	93.30	92.75	93.32	94.24	95.29

Table 3: Effect of varying base Learning Rate

I evaluated Ada-Hessian 's performance on the `Home` dataset in Table 3 by varying the base learning rate from 0.05 to 0.25 in steps of 0.05, while keeping the weight decay, β_1, and β_2 fixed. A learning rate decay of $\frac{1}{10}$ was applied at epochs 60 and 80 over a total of 100 training epochs. The model's accuracy remained stable across different learning rates, with a variation of only 2.54%. The highest accuracy of 95.29% was achieved with a learning rate of 0.25. This indicates that Ada-Hessian is highly robust to hyperparameter selection. Furthermore, a learning rate of 0.15 achieved an accuracy within 1.32% of the maximum, while the total training time was only 0.23 hours—substantially faster than first-order optimizers(see Figure 8), which required 300 epochs to converge due to their reliance on limited curvature information. As shown in Table 4, varying the weight decay

Weight Decay	5×10^{-5}	1×10^{-4}	5×10^{-4}	1×10^{-3}	5×10^{-3}
Accuracy(%)	88.41	94.38	93.32	92.75	86.59

Table 4: Effect of varying Weight Decay

while keeping the learning rate fixed reveals that optimal performance is achieved when the weight decay lies between 1×10^{-4} and 1×10^{-3}. Outside this range, the accuracy drops by at least $\sim$4%, indicating a notable sensitivity to this hyperparameter. This emphasizes the importance of tuning weight decay in second-order methods like Ada-Hessian , even though they are generally more robust to learning rate variations. The observed performance drop suggests that too little regularization may lead to overfitting, while excessive regularization hinders effective learning.

These results highlight that Ada-Hessian is relatively insensitive to hyperparameters such as learning rate, weight decay, and learning rate decay schedule, and it converges significantly faster—often within just 50 epochs. While the choice of learning rate has minimal impact on final accuracy, the weight decay parameter shows a modest influence, with suboptimal values leading to a slight performance drop. Overall, the optimizer demonstrates strong robustness and efficiency in training.

6 Conclusions

In this project, I explored the performance of Ada-Hessian in comparison with several first-order optimizers. Ada-Hessian demonstrated results comparable to AdamW, which emerged as the best-performing first-order optimizer for our CSI-based sign language detection task. It is important to note that the β parameters and learning rates for first-order methods like Adam and AdamW have been extensively tuned over the years, while for Ada-Hessian , I used the same β values as Adam and the same weight decay as AdamW without further tuning. A more careful selection of these hyperparameters could potentially yield even better results for Ada-Hessian .

Being a second-order method, Ada-Hessian inherently has higher computational complexity compared to first-order methods such as SGD, Adam, and Adamax. However, it compensates for this with significantly faster convergence—typically within 50 epochs—while also being more robust to hyperparameter choices. Its smoother convergence behavior helps avoid getting trapped in local minima, unlike first-order methods that often require meticulous learning rate schedules and over 150 epochs to converge.

To improve computational efficiency, one could skip the computation of the Hessian diagonal for certain iterations. For higher accuracy, multiple Hutchinson trace estimations per iteration can be used to better approximate the Hessian diagonal, thereby reducing stochastic noise. Additionally, the effect of block size on both performance and computational cost remains an interesting direction for further exploration.

7 Acknowledgments

This work focuses on performance of CSI-based sign language detection, specifically using the SignFi dataset. The Ada-Hessian [2] optimizer was implemented following the methodology outlined in its original paper. The dataset used was obtained from the SignFi [1] study, and preprocessing steps are omitted in this report for brevity. During my undergraduate studies in 2023, under the supervision of Dr. Hafiz Imtiaz and Dr. Tahsina Farah Sanam, we developed a framework for CSI-based sign language recognition. Building upon that foundation, this project explores the performance of second-order optimization methods—specifically Ada-Hessian —in this domain.

References

[1] Yongsen Ma, Gang Zhou, Shuangquan Wang, Hongyang Zhao, and Woosub Jung. Signfi: Sign language recognition using wifi. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(1):Article 23, 21 pages, March 2018.

[2] Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W. Mahoney. Adahessian: An adaptive second order optimizer for machine learning. *AAAI (Accepted)*, 2021.