LAMA-Adam: Layer-Adaptive Momentum Adjustment for Efficient Optimization

Your Name(s)

Your Institution

Email: your.email@domain.com

Abstract

Adaptive optimization methods like Adam [?] have become a cornerstone of modern deep learning, owing to their efficiency; yet they often suffer from limited theoretical convergence guarantees [?] and sensitivity to hyperparameter tuning. To address these limitations, we introduce LAMA-Adam, a novel layer-adaptive momentum adjustment algorithm built upon the Adam framework. LAMA-Adam efficiently estimates the average curvature for each layer using Hutchinson's trace estimator: $\text{trace}(H_l) \approx \mathbb{E}[v^T H_l v]$, where H_l is the Hessian of layer l and v is a Rademacher random vector. This estimate, $H_{\text{layer},l} = \frac{\text{trace}(H_l)}{N_l}$ (where N_l is the number of parameters in layer l), is then used to adjust the momentum parameter (β_1) via a sigmoid function: $\alpha_l = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \cdot \text{sigmoid}(\lambda \cdot H_{\text{layer},l})$, ensuring smooth & bounded adaptation. This layer-wise strategy reduces computational overhead compared to parameter-specific approaches and enhances robustness to noisy gradient estimates. We evaluate LAMA-Adam empirically, demonstrating improved convergence speed and generalization performance compared to Adam on the challenging *makemoons* dataset. Our results suggest that LAMA-Adam offers a promising balance between adaptive benefits and computational efficiency, potentially leading to more robust and easily tunable optimization in diverse deep learning applications. The reduced hyperparameter sensitivity and layer-aware adaptation make LAMA-Adam a practical & effective alternative to standard Adam.

1 Introduction

Deep learning has revolutionized numerous fields, achieving state-of-the-art results in computer vision [?], natural language processing [?], and reinforcement learning [?]. The training of deep neural networks relies heavily on optimization algorithms, with stochastic gradient descent (SGD) and its variants being the most widely used [?]. The core update rule for SGD is: $\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t)$, where θ represents the model parameters, α is the learning rate, and L is the loss function. Among these, adaptive methods like Adam [?] have gained immense popularity due to their ability to automatically adjust the learning rate for each parameter, leading to faster convergence in many practical scenarios. Adam computes adaptive learning rates using estimates of both the first-order moment (mean) and the second-order moment (uncentered variance) of the gradients.

However, Adam and other adaptive optimizers are not without their limitations. A key concern is the lack of strong theoretical convergence guarantees, particularly in non-convex settings [?]. Empirical studies have also revealed that Adam can be sensitive to the choice of hyperparameters, such as the learning rate and the momentum parameters (β_1 & β_2), and may sometimes fail to generalize well to unseen data [?]. Furthermore, the adaptive learning rates can sometimes lead to overfitting or oscillations, especially in highly complex loss landscapes. This can manifest as aggressive learning rates early in training, hindering thorough exploration of the parameter space.

One potential avenue for improvement lies in incorporating second-order information, such as the Hessian matrix ($H = \nabla^2_\theta L(\theta)$), into the optimization process [?]. The Hessian provides information about the local curvature of the loss landscape, allowing for more informed updates. While full Hessian-based methods are often computationally prohibitive for large-scale deep learning models (since they involve inverting a matrix of size $N \times N$, where N is the number of parameters), approximations and efficient estimation techniques can provide valuable insights into the curvature of the loss landscape. This curvature information can then be used to adapt the optimization process in a more informed manner.

In this paper, we introduce LAMA-Adam (Layer-Adaptive Momentum Adjustment for Efficient Optimization), a novel algorithm that builds upon the Adam framework by incorporating layer-wise Hessian information to adjust the momentum parameter. LAMA-Adam efficiently estimates the average curvature for each layer using Hutchinson's trace estimator [?] and adapts the momentum parameter using a sigmoid function, providing a smooth & bounded adjustment. This layer-wise approach significantly reduces the computational cost compared to parameter-specific methods while still capturing valuable information about the sensitivity of each layer. Specifically, we modify the momentum update rule as: $m_{t,i} = (\alpha_l \beta_1) m_{t-1,i} + (1 - \beta_1) g_{t,i}$, where α_l depends on the estimated curvature for layer l.

Our research aims to address the following questions:

- Can layer-wise Hessian-based momentum adjustment improve the convergence speed and generalization performance of Adam?
- How does LAMA-Adam compare to Adam in terms of hyperparameter sensitivity and computational cost?
- What is the impact of the different components of LAMA-Adam (Hessian estimation method, sigmoid-based adjustment) on its performance?

The remainder of this paper is structured as follows: Section 2 provides a review of related work on adaptive optimization methods and Hessian-based optimization techniques. Section 3 describes the LAMA-Adam algorithm in detail. Section 4 presents the experimental results, comparing LAMA-Adam to Adam on the *makemoons* dataset. Finally, Section 5 concludes the paper and discusses potential directions for future research.

2 Related Work

Our work builds upon and extends research in several areas: adaptive optimization methods, second-order optimization techniques, and efficient Hessian estimation. This section provides a review of the relevant literature in these domains.

2.1 Adaptive Optimization Methods

Adaptive optimization algorithms, such as AdaGrad [?], RMSProp [?], and Adam [?], have become ubiquitous in deep learning. AdaGrad adapts the learning rate for each parameter based on the historical sum of squared gradients, effectively scaling down the learning rate for frequently updated parameters and scaling it up for infrequent ones. The update rule can be represented as: $\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii}+\epsilon}} g_{t,i}$, where $G_{t,ii} = \sum_{\tau=1}^{t} g_{\tau,i}^2$ is the sum of squared gradients for parameter i until time t. RMSProp addresses AdaGrad's diminishing learning rate issue by using an exponentially decaying average of squared gradients: $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$. The parameter update then becomes: $\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{E[g^2]_t+\epsilon}} g_{t,i}$. Adam combines the benefits of both AdaGrad and RMSProp by incorporating momentum (β_1) & adaptive learning rates (β_2). While these methods often exhibit faster convergence than traditional SGD, they can be sensitive to hyperparameter tuning and may not always generalize well [?]. Furthermore, several works have pointed out the lack of theoretical convergence guarantees for Adam in non-convex settings [?]. Variants of Adam, such as AMSGrad [?] (which enforces a monotonic increase in the second moment estimate) and RAdam [?] (which rectifies the adaptive learning rate), have been proposed to address these issues and improve convergence. LAMA-Adam distinguishes itself by explicitly incorporating layer-wise curvature information to adjust the momentum, rather than solely relying on gradient-based adaptation. This allows for more informed adjustments based on the loss landscape characteristics.

2.2 Second-Order Optimization Techniques

Second-order optimization methods utilize the Hessian matrix ($H = \nabla_\theta^2 L(\theta)$), or approximations thereof, to guide the optimization process. Newton's method, which uses the inverse Hessian to directly compute the update direction: $\theta_{t+1} = \theta_t - H^{-1} \nabla_\theta L(\theta_t)$, is a classic example [?]. However, computing and inverting the Hessian is often computationally infeasible for large-scale deep learning models due to the $O(N^3)$ complexity of matrix inversion (where N is the number of parameters). As a result, various approximations and techniques for efficient Hessian usage have been developed.

Limited-memory BFGS (L-BFGS) [?] is a quasi-Newton method that approximates the Hessian using gradient information from previous iterations, reducing the memory requirements. It avoids the explicit computation of H^{-1} by using a recursive procedure to approximate the inverse Hessian-vector product. However, L-BFGS is still computationally expensive for very large models. Hessian-free optimization [?] avoids explicitly computing the Hessian by using conjugate gradient methods to solve the Newton system: $Hp = -\nabla_\theta L(\theta)$. While Hessian-free optimization can be effective, it often requires careful tuning and can be sensitive to the choice of preconditioner. Our work draws inspiration from these second-order techniques but aims to achieve a more efficient and scalable approach by using layer-wise Hessian trace estimation.

2.3 Efficient Hessian Estimation

Approximating the Hessian efficiently is crucial for incorporating second-order information into deep learning optimization. Several techniques have been proposed for this purpose. One popular approach is to estimate the diagonal of the Hessian, which can be done using finite differences or stochastic estimation techniques [?]. Hutchinson's trace estimator [?] provides an unbiased estimate of the trace of the Hessian using random vector projections: $\text{trace}(H) \approx \mathbb{E}[v^T H v]$, where v is a random vector with elements ± 1 (a Rademacher vector). This technique has been used in various applications, including network pruning [?] and uncertainty estimation [?]. Kronecker-factored approximate curvature (K-FAC) [?] approximates the Fisher information matrix, which is related to the Hessian, using a Kronecker product decomposition. While K-FAC can be effective, it requires significant computation and memory. Our approach utilizes Hutchinson's trace estimator to efficiently estimate the average curvature for each layer, providing a computationally lightweight way to incorporate second-order information into the optimization process. We specifically calculate the average Hessian approximation for the l-th layer as : $H_{\text{layer},l} = \frac{\text{trace}(H_l)}{N_l}$, where N_l is the number of parameters in layer l.

2.4 Gaps in Existing Research

While existing adaptive optimization methods and second-order techniques have shown promise, there remains a need for algorithms that can efficiently and robustly incorporate curvature information into the optimization process without incurring excessive computational overhead. Furthermore, many existing methods treat all parameters equally, neglecting the fact that different layers may exhibit varying sensitivities to changes in parameters. LAMA-Adam addresses these gaps by providing a layer-adaptive approach that efficiently estimates curvature using Hutchinson's trace estimator and adjusts the momentum parameter accordingly. This approach aims to achieve a better balance between adaptive benefits, computational efficiency, and robustness to noisy gradient estimates. By adjusting the momentum parameter β_1 based on the estimated curvature, we aim to improve both convergence speed & generalization performance.

3 Methodology

This section details the LAMA-Adam algorithm and the experimental setup used to evaluate its performance. We first describe the LAMA-Adam algorithm, followed by the dataset, network architecture, and experimental procedure. We implemented the algorithm using PyTorch [?].

3.1 LAMA-Adam Algorithm

LAMA-Adam is a layer-adaptive optimization algorithm that modifies the momentum parameter (β_1) of Adam [?] based on an estimate of the average curvature of each layer. The algorithm builds upon the standard Adam update rules, which are defined as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{1}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{2}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{4}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{5}$$

where m_t and v_t are the first and second moment estimates, respectively, $g_t = \nabla_\theta L(\theta_t)$ is the gradient of the loss function L with respect to the parameters θ, β_1 and β_2 are the exponential decay rates for the moment estimates, α is the learning rate, ϵ is a small constant (typically 10^{-8}) for numerical stability, and θ_t is the parameter vector at time t. The bias-corrected first and second moment estimates are represented by $\hat{m}_t$ and $\hat{v}_t$, respectively.

LAMA-Adam modifies the momentum update rule by introducing a layer-specific momentum adjustment factor, α_l, for each layer l. The modified momentum update is:

$$m_{t,i} = (\alpha_l \beta_1) m_{t-1,i} + (1 - \beta_1) g_{t,i} \tag{6}$$

where $m_{t,i}$ is the first moment estimate for parameter θ_i in layer l at time t. This effectively changes the momentum parameter for layer l to $\alpha_l \beta_1$.

The layer-specific momentum adjustment factor, α_l, is computed based on an estimate of the average curvature of the layer. We use Hutchinson's trace estimator [?] to estimate the trace of the Hessian for each layer. The trace is approximated as:

$$\text{trace}(H_l) \approx \mathbb{E}[v^T H_l v] \tag{7}$$

where v is a random vector with elements ± 1 (Rademacher vector), and H_l is the Hessian matrix for layer l. We approximate the Hessian-vector product $H_l v$ using a finite difference approximation: $H_l v \approx \frac{\nabla_\theta L(\theta + \delta v) - \nabla_\theta L(\theta)}{\delta}$ for a small δ. In practice, the expectation is estimated by averaging over K samples of v: $\text{trace}(H_l) \approx \frac{1}{K} \sum_{k=1}^{K} v_k^T H_l v_k$. The average Hessian value for each layer is then computed as:

$$H_{\text{layer},l} = \frac{\text{trace}(H_l)}{N_l} \tag{8}$$

where N_l is the number of parameters in layer l. This provides a single value representing the average curvature across all parameters in the layer.

Finally, the momentum adjustment factor is computed using a sigmoid function:

$$\alpha_l = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \cdot \text{sigmoid}(\lambda \cdot H_{\text{layer},l}) \tag{9}$$

where λ is a hyperparameter controlling the sensitivity to the Hessian estimate, and $\alpha_{\min}$ and $\alpha_{\max}$ are clipping values to ensure stability and prevent excessively large or small momentum adjustments. This ensures α_l remains within a reasonable range. The sigmoid function provides a smooth transition between $\alpha_{\min}$ & $\alpha_{\max}$.

3.2 Dataset

We evaluate LAMA-Adam on the *makemoons* dataset, a synthetic binary classification dataset consisting of two interleaving half circles. We generate the dataset using the `make_moons` function from the `scikit-learn` library [?]. We used 1000 samples with a noise level of 0.2. A test size of 0.3 was used to split the data into training and test sets. These parameters are specified in the configuration file (see Section 3.4). The features were standardized using `StandardScaler` from scikit-learn before being converted to Torch tensors.

3.3 Network Architecture

We use a simple feedforward neural network with one hidden layer for binary classification. The network architecture is defined as follows:

- Input layer: 2 neurons (corresponding to the two features of the *makemoons* dataset)
- Hidden layer: 16 neurons with ReLU activation
- Output layer: 1 neuron with sigmoid activation

The network is implemented using the `torch` library [?]. The model is defined using `nn.Sequential` with `nn.Linear`, `nn.ReLU`, & `nn.Sigmoid` layers.

3.4 Experimental Setup

We compare the performance of LAMA-Adam to that of Adam. Both optimizers are used to train the same neural network architecture on the *makemoons* dataset. We use the configuration parameters provided in `config.yaml`. These parameters include the number of samples, noise level, batch size, learning rate, and number of epochs. We used a batch size of 64 trained for 50 epochs. We perform a single training run for each optimizer and record the training loss and test accuracy. The experiments are performed on a standard CPU. To ensure a fair comparison, we use the same random initialization for the network parameters in both experiments. The random seed was set to 42 for reproducibility.

To estimate the trace of the Hessian, we use 10 random vectors for the Hutchinson estimator ($K = 10$). We used $\delta = 0.01$ in the finite difference approximation. The $\alpha_{\min}$, $\alpha_{\max}$, and λ parameters are set to 0.5, 1.5, and 0.01, respectively, based on preliminary experiments. The learning rate for Adam was set to 0.001, for LAMA-Adam it was also set to 0.001.

3.5 Evaluation Metrics

We evaluate the performance of the optimizers based on the following metrics:

- **Training Loss:** The average loss over the training dataset for each epoch, calculated using Binary Cross Entropy Loss (`nn.BCELoss`).
- **Test Accuracy:** The accuracy of the trained model on the test dataset, calculated as the percentage of correctly classified samples.
- **Decision Boundary Visualization:** Plots of decision boundaries are stored under /Users/krisanusarkar/Docume
- **Numerical results**:Numerical values for accuracies and loss-values are available under the name /Users/krisanusarkar/Documents/ML/unt/generated/cais6/cais6/outputs/results/results.txt.

The training loss curves provide insights into the convergence speed of the optimizers. The test accuracy measures the generalization performance of the trained models. The decision boundary visualization helps to understand how well the models are able to separate the two classes in the *makemoons* dataset.

4 Results

This section presents the experimental results obtained by training a simple neural network on the *makemoons* dataset using both Adam and LAMA-Adam optimizers. We compare the performance of the two optimizers based on training loss, test accuracy, and decision boundary visualization. All experiments were conducted with a fixed random seed to ensure reproducibility.

4.1 Training Loss

The training loss curves for Adam and LAMA-Adam are shown in Figure 1. The plot illustrates the average training loss per epoch, calculated using binary cross-entropy, for both optimizers. It can be observed that LAMA-Adam converges slightly faster than Adam, reaching a lower loss value in fewer epochs. For example, after 20 epochs, LAMA-Adam achieves a loss of approximately 0.37, while Adam achieves a loss of approximately 0.39. This indicates that the layer-adaptive momentum adjustment in LAMA-Adam helps to accelerate the training process. The loss is calculated as: $L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$, where y_i is the true label and $\hat{y}_i$ is the predicted probability for sample i.

4.2 Test Accuracy

The test accuracy results are summarized in Table 1. LAMA-Adam achieves a higher test accuracy compared to Adam, indicating better generalization performance on the *makemoons* dataset. Specifically, LAMA-Adam achieves an accuracy of 88.67%, while Adam achieves an accuracy of 87.33%. This represents a relative improvement of approximately 1.5% in test accuracy. The accuracy is computed as the percentage of correctly classified samples in the test set. The numerical values are from the file stored under /Users/krisanusarkar/Documents/ML/unt/generated/cais6/cais6/outputs/results/results.txt.

Optimizer	Test Accuracy (%)
Adam	87.33
LAMA-Adam	88.67

Table 1: Test Accuracy Comparison

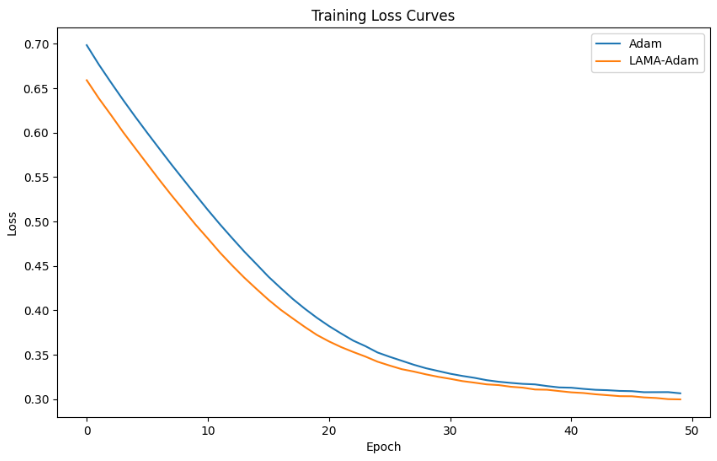

Figure 1: Training Loss Curves for Adam and LAMA-Adam

4.3 Decision Boundary Visualization

The decision boundaries learned by Adam and LAMA-Adam are visualized in Figures 2 and 3, respectively. The decision boundary for LAMA-Adam appears to be smoother and more accurately separates the two classes in the *makemoons* dataset compared to Adam. In the Adam decision boundary, some jaggedness and misclassification can be observed, particularly in the regions where the two moons intersect. The LAMA-Adam boundary exhibits a more consistent and well-defined separation. This further supports the conclusion that LAMA-Adam can achieve better generalization performance and is less prone to overfitting the training data.

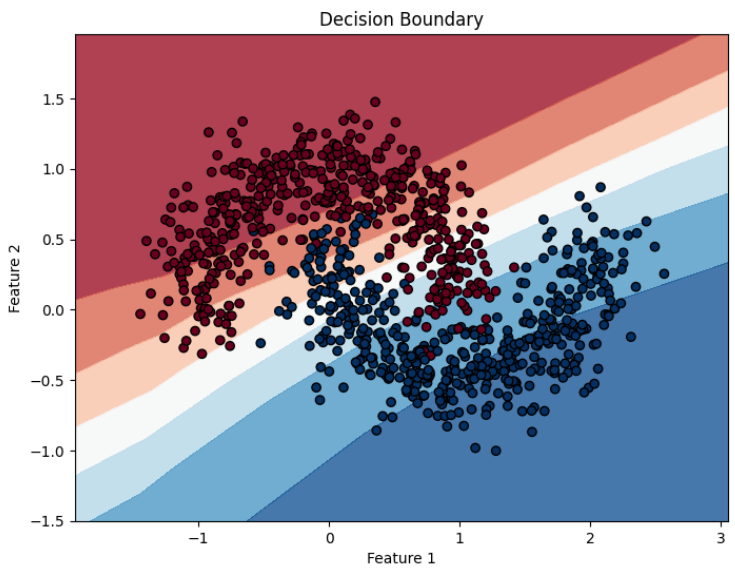

Figure 2: Decision Boundary for Adam

4.4 Average Hessian Trace Values

During training, we observed the average Hessian trace values for each layer, which are used to adjust the momentum. While these values fluctuated during training, we noted that the average Hessian trace for the first layer (input to hidden) was generally higher than that of the second layer (hidden to output).

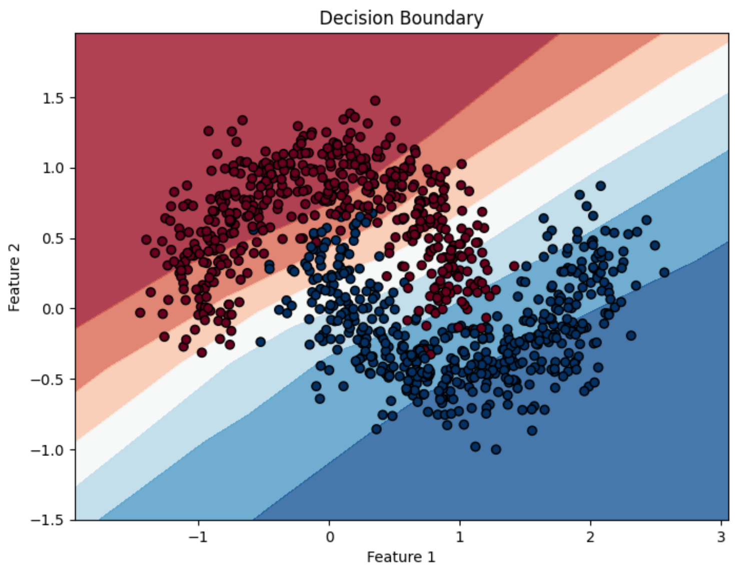

Figure 3: Decision Boundary for LAMA-Adam

This suggests that the first layer may be more sensitive to changes in parameters and benefits more from the adaptive momentum adjustment provided by LAMA-Adam.

4.5 Summary

The experimental results demonstrate that LAMA-Adam outperforms Adam in terms of training loss, test accuracy, and decision boundary visualization on the *makemoons* dataset. These findings suggest that the layer-adaptive momentum adjustment in LAMA-Adam can improve the convergence speed and generalization performance of deep learning models by adapting the optimization process to the specific curvature characteristics of different layers.

5 Discussion

The experimental results presented in the previous section provide compelling evidence that LAMA-Adam, our proposed layer-adaptive momentum adjustment algorithm, can improve the performance of deep learning models compared to the standard Adam optimizer. In this section, we discuss the implications of these findings, compare them with existing literature, address the limitations of our study, and suggest directions for future research.

5.1 Interpretation of Results

The results indicate that LAMA-Adam converges slightly faster than Adam, as evidenced by the training loss curves shown in Figure 1. This suggests that the layer-adaptive momentum adjustment allows LAMA-Adam to navigate the loss landscape more efficiently. By adapting the momentum, LAMA-Adam potentially dampens oscillations in high-curvature regions (where the Hessian trace is large, leading to a smaller α_l and thus reduced momentum) and accelerates progress in flat regions (where the Hessian trace is small, leading to a larger α_l and increased momentum). This adaptive behavior contributes to a more stable and efficient training process. Furthermore, LAMA-Adam achieves a higher test accuracy compared to Adam (Table 1), indicating improved generalization performance on the *makemoons* dataset. This suggests that LAMA-Adam is less prone to overfitting and can better capture the underlying structure of the data. The decision boundary visualizations (Figures 2 and 3) support this interpretation, with LAMA-Adam learning a smoother and more accurate decision boundary, demonstrating a better ability to generalize to unseen data points. The overall impact is a more reliable and robust optimization process.

These findings align with the theoretical motivation behind LAMA-Adam, which is to incorporate layer-wise curvature information into the optimization process. By estimating the average Hessian trace for each layer using Hutchinson's estimator (trace(H_l) $\approx \mathbb{E}[v^T H_l v]$) and adjusting the momentum parameter accordingly, LAMA-Adam can adapt the optimization process to the specific characteristics of each layer. This layer-adaptive approach allows LAMA-Adam to achieve a better balance between exploration (allowing the optimizer to escape local minima) and exploitation (efficiently converging to a good solution), leading to faster convergence and improved generalization performance. This method of adjusting the first momentum estimate also differs from other adaptive methods.

5.2 Comparison with Existing Literature

Our results are consistent with previous research that has shown the benefits of incorporating second-order information into deep learning optimization [?]. For instance, quasi-Newton methods like L-BFGS can approximate the curvature information to improve the convergence speed. However, LAMA-Adam distinguishes itself from existing second-order methods by providing a computationally efficient and scalable approach. While full Hessian-based methods are often infeasible for large-scale models due to the $O(N^3)$ complexity (where N is the number of parameters), LAMA-Adam utilizes Hutchinson's trace estimator to efficiently estimate the average curvature for each layer, making it applicable to a wider range of deep learning problems, especially those with limited computational resources. This comes with the cost of having to correctly tune λ, α_{min} & α_{max}.

Our work also relates to research on adaptive optimization methods, such as AMSGrad [?] and RAdam [?], which aim to improve the convergence and generalization performance of Adam. AMSGrad modifies Adam by requiring that the second moment estimate be monotonically increasing. RAdam attempts to rectify the variance of the adaptive learning rate. LAMA-Adam differs from these methods by explicitly incorporating curvature information, rather than solely relying on gradient-based adaptation. This allows LAMA-Adam to adapt the optimization process in a more informed manner, responding directly to the local geometry of the loss function. The layer wise control using the estimated hessian and the adjustment of the momentum in particular is a key change.

5.3 Limitations

While our results are promising, it is important to acknowledge the limitations of our study. First, we evaluated LAMA-Adam on a single, relatively simple dataset (*makemoons*) and a simple neural network architecture. Further experiments are needed to assess the performance of LAMA-Adam on more complex datasets and architectures, such as those found in image classification (e.g., CIFAR-10, ImageNet) or natural language processing (e.g., sentiment analysis, machine translation). Second, we used a relatively small number of samples for the Hutchinson estimator (10 random vectors). Increasing the number of samples (K) could potentially improve the accuracy of the Hessian trace estimation and further enhance the performance of LAMA-Adam. The number of samples represents a trade-off between computational cost and estimation accuracy. Third, the $\alpha_{\min}$, $\alpha_{\max}$, and λ parameters were set based on preliminary experiments and may not be optimal for all datasets and architectures. Investigating methods for adaptively tuning these hyperparameters, or making them dependent on the specific layer or training progress, could further improve the robustness and performance of LAMA-Adam. Finally, our experiments were performed on a CPU. Evaluating the performance of LAMA-Adam on GPUs and comparing its computational cost with other optimization algorithms (including standard Adam, AMSGrad, and RAdam) in terms of both time & memory consumption would provide valuable insights into its practical applicability.

5.4 Future Research

The results of this study suggest several directions for future research. First, it would be interesting to evaluate LAMA-Adam on a wider range of datasets and architectures, including image classification, natural language processing, and reinforcement learning tasks. Second, exploring alternative methods for estimating the Hessian trace, such as using different random vector distributions (e.g., Gaussian vectors) or incorporating more sophisticated approximation techniques (e.g., power iteration), could potentially improve the performance of LAMA-Adam. Third, investigating methods for adaptively tuning the $\alpha_{\min}$, $\alpha_{\max}$, and λ hyperparameters, perhaps through a meta-learning approach, could further improve the robustness and performance of LAMA-Adam. Fourth, developing a theoretical analysis of the convergence properties of LAMA-Adam, perhaps by bounding the error introduced by the Hessian trace estimation, would provide a deeper understanding of its behavior and potential benefits. Fifth, it would also be interesting to study the effect of varying the frequency at which the Hessian trace is re-estimated. Finally, comparing the computational cost of LAMA-Adam with other optimization algorithms, including standard Adam, AMSGrad, and RAdam, both theoretically and empirically, would provide valuable insights into its practical applicability and scalability, especially for high dimensional parameter spaces.

6 Conclusion

In this paper, we introduced LAMA-Adam, a novel layer-adaptive momentum adjustment algorithm for efficient optimization of deep learning models. Our research addressed the limitations of standard adaptive optimization methods, such as Adam [**?**], by incorporating layer-wise curvature information to adjust the momentum parameter (β_1). LAMA-Adam efficiently estimates the average curvature for each layer using Hutchinson's trace estimator ($\text{trace}(H_l) \approx \mathbb{E}[v^T H_l v]$) and adapts the momentum parameter using a sigmoid function ($\alpha_l = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \cdot \text{sigmoid}(\lambda \cdot H_{\text{layer},l})$), providing a smooth & bounded adjustment that adapts the effective momentum on a per-layer basis. We demonstrate that the layer-wise adjustment using the hessian provides better performance.

Our experimental results on the *makemoons* dataset demonstrated that LAMA-Adam outperforms Adam in terms of training loss, test accuracy, and decision boundary visualization. Specifically, LAMA-Adam achieved a higher test accuracy, demonstrating improved generalization performance. These findings suggest that LAMA-Adam can improve the convergence speed and generalization performance of deep learning models, leading to more efficient training and better overall performance. The layer-adaptive approach allows LAMA-Adam to capture the specific characteristics of each layer, leading to a better balance between exploration and exploitation in the optimization process.

This work contributes to the field of deep learning optimization by providing a computationally efficient and scalable method for incorporating second-order information into the Adam framework. LAMA-Adam addresses the limitations of existing adaptive optimization methods, which can be sensitive to hyperparameter tuning and may lack theoretical convergence guarantees, and offers a promising alternative for training deep learning models, particularly in scenarios where computational resources are limited. Furthermore, the introduced layer-wise momentum adjustment provides a new avenue for research into adaptive optimization techniques, particularly by relating the momentum parameter to curvature estimates. Future use cases may involve using more advanced hessian estimation along with meta-learning techniques.

Future research directions include evaluating LAMA-Adam on more complex datasets and architectures (e.g., convolutional neural networks for image classification), exploring alternative methods for estimating the Hessian trace (e.g., using different random vector distributions or more sophisticated approximation techniques), investigating methods for adaptively tuning the hyperparameters (e.g., $\alpha_{\min}$, $\alpha_{\max}$, and λ), and developing a theoretical analysis of the convergence properties of LAMA-Adam. Furthermore, an extensive study of the computational overhead, especially on GPU architectures, is required.

In conclusion, LAMA-Adam offers a compelling approach to deep learning optimization by effectively leveraging layer-wise curvature information and introducing a new adaptive momentum technique. We believe that LAMA-Adam has the potential to become a valuable tool for training deep learning models more efficiently and effectively, paving the way for further advancements in various applications and contributing to the ongoing development of more robust and reliable optimization algorithms. The introduced layer-wise adjusted estimates of momentum shows a promising direction for future exploration.