

AdaGrad and Adam with gradient difference estimator

Lianjia Liu
lili00002@stud.uni-saarland.de
Universität des Saarlandes

Jaykumar Bhagiya
jabh00002@stud.uni-saarland.de
Universität des Saarlandes

Akansh Maurya
akma00001@stud.uni-saarland.de
Universität des Saarlandes

Abstract—Gradient descent optimizers such as AdaGrad[1] and Adam[2] are pivotal in machine learning. However, these methods sometimes face challenges like slow convergence and overshooting due to their reliance on gradient norms for adjusting step sizes. This study investigates an innovative approach that incorporates gradient differences into the step size calculations of AdaGrad and Adam, aiming to enhance convergence and stability across a range of machine learning tasks. We performed experiments on both convex and non-convex problems, including logistic regression and deep learning tasks, to evaluate the performance of these modified optimizers. Code of our work is available at this link.¹

I. INTRODUCTION

Optimizers are crucial for efficiently training machine learning models, and adaptive methods like AdaGrad and Adam are widely used due to their ability to adjust learning rates based on gradient information. However, these methods can encounter issues such as slow convergence and overshooting, particularly in complex tasks or datasets with varying feature distributions.

This study introduces modified versions of AdaGrad and Adam that incorporate gradient differences into their step size calculations. The primary aim is to determine whether these modifications improve optimizer performance and robustness. Specifically, the study seeks to:

- Evaluate and compare the performance of standard AdaGrad and Adam with their modified variants across different datasets and learning tasks.
- Investigate the impact of these modifications on optimizer robustness, particularly in the presence of feature shifting and batch normalization.

To evaluate the proposed optimizer variants, three distinct experiments were conducted. The first experiment tested the optimizers on logistic regression using three binary classification datasets: diabetes, credit card fraud detection, and ionosphere. This experiment included feature shifting to analyze the robustness and sensitivity of the optimizers. The second experiment involved deep learning with the CIFAR-10 dataset and the ResNet-34 model, comparing performance with and without batch normalization. The final experiment focused on image style transfer in deep learning, further assessing the new variants' performance and versatility.

II. METHODOLOGY

This study examines the standard AdaGrad and Adam optimization algorithms, along with their modified versions that incorporate gradient differences.

AdaGrad:

- Standard AdaGrad adjusts learning rates by accumulating the sum of squares of past gradients.
- Modified AdaGrad with Gradient Differences accumulates the sum of squares of differences between successive gradients, enhancing sensitivity to recent gradient changes.

Adam:

- Standard Adam adapts learning rates using the first moment (mean) and second moment (uncentered variance) of the gradients.
- Modified Adam with Gradient Differences accumulates the sum of squares of differences between successive gradients, improving robustness to recent gradient variations.

The detailed steps for these modified algorithms are provided in Algorithm 1 and Algorithm 2 in Appendix A

III. EXPERIMENT ON LOGISTIC REGRESSION:

In this experiment, we explored the performance and robustness of logistic regression models trained using various optimization algorithms. Specifically, we compared traditional optimizers such as Adam and Adagrad with their gradient difference counterparts, AdamwithDiff and AdagradwithDiff. The focus was on understanding how these optimizers perform under normal conditions and when subjected to feature shifting, where a constant value is added to each feature to simulate changes in data distributions due to sensor drift, measurement errors, or environmental changes.

The logistic regression model is represented by the following equation:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_p x_p)}} \quad (1)$$

The objective of our experiment was to compare the performance of these optimizers on three binary classification datasets—Pima Indian Diabetes, Credit Card Fraud Detection, and Ionosphere datasets—both under normal conditions and with a feature shift of 100. By analyzing the training losses and validation accuracies, we aim to understand the robustness and sensitivity of these optimizers to feature shifting and to

¹<https://github.com/akansh12/OML-2024-AdaGrad-with-grad-difference>.

identify the most effective optimization strategy for logistic regression models.

A. Datasets

For this, three binary datasets of different sizes and features were used:

1. **Pima Indian Diabetes Dataset:** This dataset consists of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, etc.

2. **Credit Card Fraud Detection Dataset:** This dataset contains only numerical input variables resulting from a PCA transformation. The 'Class' feature is the response variable, taking the value 1 in case of fraud and 0 otherwise.

3. **Ionosphere Dataset:** This dataset contains radar data gathered by a system in Goose Bay, Labrador. The targets in this dataset were free electrons in the ionosphere. "Good" radar returns indicate the presence of some structure in the ionosphere, whereas "Bad" returns do not.

Dataset	# of data	# of features
Diabetes	768	8
Credit Card	284807	29
Ionosphere	351	34

TABLE I
DATASET DETAILS

B. Results

The results are shown in the figures 2 and 3 in appendix A. Without feature shifting, AdagradwithDiff consistently outperforms Adagrad in training loss and validation accuracy. AdamwithDiff also shows similar or slightly better validation accuracy compared to Adam, with comparable loss reduction. Both AdagradwithDiff and AdamwithDiff notably enhance validation accuracy in the Diabetes dataset and achieve lower training losses and higher accuracies in the Credit Card dataset. In the Ionosphere dataset, the differences are smaller but still favor optimizers with gradient differences.

AdaGrad and AdaGradWithDiff were more sensitive to feature shifting compared to Adam and AdamWithDiff. In contrast, Adam and AdamWithDiff demonstrated greater robustness across different datasets and preprocessing techniques. This enhanced robustness made Adam and AdamWithDiff more adaptable and reliable in varied scenarios, while AdaGrad and AdaGradWithDiff struggled with changes in feature distributions.

IV. EXPERIMENT ON DEEP LEARNING: CIFAR10 IMAGE CLASSIFICATION

This experiment involved training a deep learning model on the CIFAR10[3] dataset to assess the impact of the modified optimizers on image classification tasks. We used the ResNet-34 architecture and evaluated the performance of the modified AdaGrad and Adam optimizers in terms of training and validation losses and accuracies.

A. Hyperparameters

The following table summarizes the hyperparameters used in this experiment, excluding the optimizers.

Hyperparameter	Value
Batch Size	256
Number of Epochs	100
Learning Rate	1×10^{-3}
Model Architecture	ResNet-34

TABLE II
HYPERPARAMETERS FOR CIFAR10 IMAGE CLASSIFICATION EXPERIMENT

B. Results and Discussion

- We performed multiple experiments with AdaGrad, AdaGrad with Diff, Adam, and Adam with Diff. We also conducted these experiments after removing the batch normalization (BN) layer in the ResNet-34 model. We tested with and without BN because we wanted to inspect how the use of BN affects the performance of AdaGrad in particular.
- The figure (Figure 6) shows the validation loss and accuracy on the classification task. The important curves to note are the Red (AdaGrad with BN) and Orange (AdaGrad with Diff with BN). We can see that using AdaGrad with Diff helps in faster and better convergence.
- For Adam, with and without Diff, the performance is quite comparable, as shown in Table III. We can say Adam is robust across these variations.
- However, we don't fully understand the behavior of the optimizer, particularly AdaGrad without batch norm (dashed red curve) performance improvement. This result is counter-intuitive to us.

Optimizer	With BN(%)	Without BN(%)
AdaGrad	58.25	73.3
Adam	79.87	81.39
AdaGradWithDiff	71.77	59.11
AdamWithDiff	80.36	79.42

TABLE III
VALIDATION ACCURACY

V. EXPERIMENT ON DEEP LEARNING: IMAGE STYLE TRANSFER

Image style transfer is a process that merges the content of one image with the style of another image, creating a new image that maintains the structure of the content image while adopting the visual appearance of the style image.

It's not a typical choice for experimenting optimizers however due to our previous experience such task has problem in convergence thus it is considered some bonus experiment. And surprisingly the results show the efficiency of AdaGradWithDiff and AdamWithDiff.

A. Style Transfer Model

Before jumping to the final results, some details about the architecture and loss function might help grasp the idea of

the task and understand the hyper-parameters and comparison measurements in the following sections.

The foundational model for image style transfer was proposed by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge [4]. Their approach utilizes a pre-trained convolutional neural network (VGG19 in the experiment) to extract and manipulate image features. The core idea is to separately capture the content and style representations from different layers of the CNN and then optimize a new image to match the content of the input content image and the style of the input style image. The loss function in this model is a weighted sum of the content loss and style loss.

During back-propagation, the model is freezed and the gradients of the total loss are computed with respect to the pixels of the generated image. The image starts with random noise and the whole process ends until the total loss converges to a predefined threshold or a maximum number of iterations is reached. In appendix, figure ?? gives a clearer idea of how the method works and how loss is computed.

B. Hyper-Parameter Setting

Hyper-parameters are crucial for balancing the influence of content and style in the generated image. And based on empirical experiments, the style weights are way larger in scale compared to that of the content weight. To ensure that both models could be compared effectively despite the differences in their settings, these hyper-parameters were chosen to achieve similar styled-transferred results during empirical experiments of state-of-the-art technique Adam. Table IV shows all the parameters used in the experiments.

There is a significant increase in scales in experiments with batch normalization(BN). In our understanding it's due to the relatively lower loss and gradients after applying BN.

Regrading to the learning rate, the default values are used, which are for 10^{-2} Adagrad and 10^{-8} for AdaGradWithDiff and for Adam and AdamWithDiff. For the decay rates β s, we only did the test for the default setting, $\beta_1 = 0.9$ and $\beta_2 = 0.99$

Models	Content Weight	Style Weight	Loss Threshold
BN	1e2	1e11	4
w/o BN	1	1e6	3.2

TABLE IV
HYPER-PARAMETERS FOR STYLE TRANSFER TASK

C. Analysis with Batch Norm

The style transfer experiments over only one style image and one content image. This is under the assumption that the high dimension of images as well as its feature map create a complex non-convex landscape. And the behaviors of them can still indicate some general patterns for other image pairs.

During the experiment, the epochs for convergence are considered the main measurements for comparing behaviors of optimizers. And since BN is a popular techniques in deep neural network to resolve vanishing gradients and other problems caused by the exploding depth. However the landscape of

loss would change after applying BN. And this does impact the behaviors of four optimizers. Similarly as in the experiment of Resnet, here AdaGradWithDiff outperforms Adagrad when applying BN, while the reverse without BN.

For getting some intuition behind, we also keep track of the denominators values, i.e. $\sqrt{\tilde{G}_i^{(t)}} + \epsilon$ and $\sqrt{\hat{v}_i^{(t)}} + \epsilon$ which adjusts the learning rate. We compare AdamWithDiff

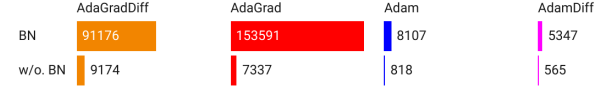


Fig. 1. Epochs to converge in image style transfer

and AdaGradWithDiff to their original versions, Adam and AdaGrad. As shown in Figure 1, AdamWithDiff converges faster, with Adam being the second fastest and showing a similar performance gap in both experiments.

However, the situation is different for AdaGrad and AdaGradWithDiff. AdaGradWithDiff requires approximately 10 times more epochs to converge compared to AdaGrad, Adam, and AdamWithDiff. AdaGrad's performance is significantly impaired, likely due to the issue of small updates being exacerbated by the increasing loss weights when the accumulated sum becomes too large. Figure 5 confirms this with small step sizes for AdaGrad. AdaGradWithDiff is more robust and performs better with batch normalization (BN).

VI. DISCUSSION

A. Performance Comparisons

Adam and AdamWithDiff perform similarly across most experiments, except in style transfer task AdamWithDiff is noticeable better. While Adagrad and AdaGradWithDiff behave differently in various tasks.

B. Robustness

In general, Adam and AdamWithDiff are both robust to batch normalization (BN) and feature shifting. In contrast, AdaGrad and AdaGradWithDiff are more sensitive to these factors, as indicated by their different performances under feature shifting. AdaGradWithDiff demonstrates better performance when BN is applied in two deep learning tasks.

Furthermore, to gain a better understanding of the motivation behind using differences and their impact, we need to explore what the difference of consecutive gradients represents. This difference roughly approximates second derivatives but is not precise since it ignores the error term and step size. Second derivatives are more sensitive to curvature than gradients. Thus, AdaGradWithDiff and AdamWithDiff may perform better in less curvy landscapes, which could be a result of applying BN. The greater improvement of AdaGradWithDiff over AdaGrad may be due to this summation effect. However, it is still unclear whether BN creates a less curvy landscape and if this is the reason for the improved performance of AdaGradWithDiff and AdamWithDiff.

REFERENCES

- [1] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: <http://jmlr.org/papers/v12/duchi11a.html>
- [2] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. [Online]. Available: <https://arxiv.org/abs/1412.6980>
- [3] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” University of Toronto, Tech. Rep., 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
- [4] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2414–2423.

APPENDIX

ALGORITHMS

Algorithm 1 Modified AdaGrad with Gradient Differences

Require: Initial parameter θ_i , learning rate η , small constant ϵ

- 1: Initialize $G_i^{(0)} = 0$
- 2: **for** each time step t **do**
- 3: Compute gradient: g_i^t
- 4: Compute gradient difference: $\tilde{g}_i^{(t)} = g_i^{(t)} - g_i^{(t-1)}$
- 5: Accumulate sum of squares of gradient differences: $\tilde{G}_i^{(t)} = \|\tilde{g}_i^{(t)}\|_2^2$
- 6: Update parameter: $\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{\tilde{G}_i^{(t)} + \epsilon}} \cdot g_i^{(t)}$
- 7: **end for**

Algorithm 2 Modified Adam with Gradient Differences

Require: Initial parameter θ_i , learning rate η , β_1 , β_2 , small constant ϵ

- 1: Initialize $\tilde{m}_i^{(0)} = 0$, $\tilde{v}_i^{(0)} = 0$
- 2: **for** each time step t **do**
- 3: Compute gradient: g_i^t
- 4: Compute gradient difference: $\tilde{g}_i^{(t)} = g_i^{(t)} - g_i^{(t-1)}$
- 5: Update biased first moment estimate: $\tilde{m}_i^{(t)} = \beta_1 \tilde{m}_i^{(t-1)} + (1 - \beta_1) g_i^{(t)}$
- 6: Update biased second moment estimate: $\tilde{v}_i^{(t)} = \beta_2 \tilde{v}_i^{(t-1)} + (1 - \beta_2) \|\tilde{g}_i^{(t)}\|_2^2$
- 7: Compute bias-corrected first moment estimate: $\hat{m}_i^{(t)} = \frac{\tilde{m}_i^{(t)}}{1 - \beta_1^t}$
- 8: Compute bias-corrected second moment estimate: $\hat{v}_i^{(t)} = \frac{\tilde{v}_i^{(t)}}{1 - \beta_2^t}$
- 9: Update parameter: $\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{\hat{v}_i^{(t)} + \epsilon}} \cdot \hat{m}_i^{(t)}$
- 10: **end for**

LOGISTIC REGRESSION

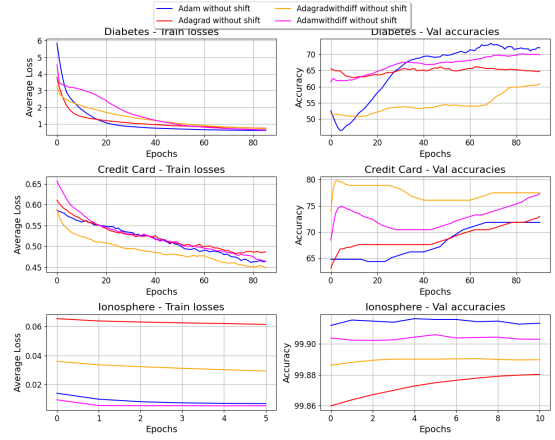


Fig. 2. Training losses and validation accuracy on the original dataset for Logistic Regression task.

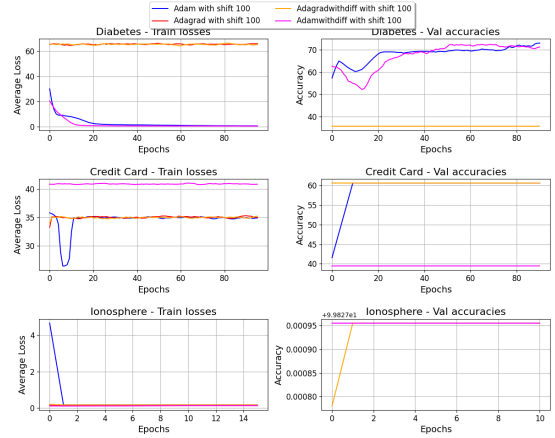


Fig. 3. Training losses and validation accuracies with feature shifting of 100 for Logistic Regression task.

IMAGE STYLE TRANSFER

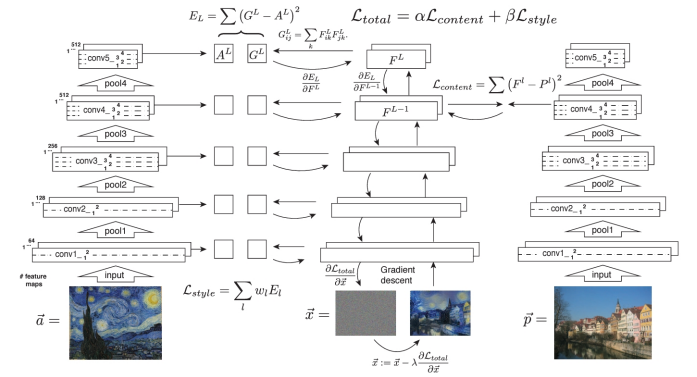


Fig. 4. Model of Image Style Transfer

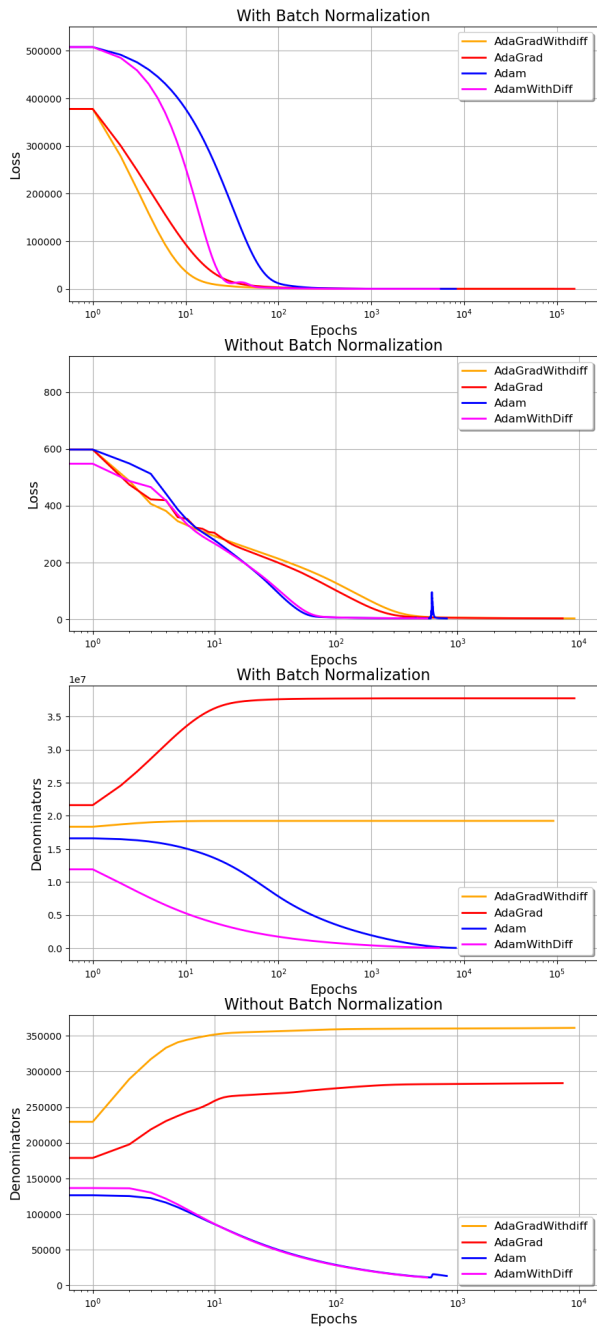


Fig. 5. Loss and denominator values for **style transfer experiment**.

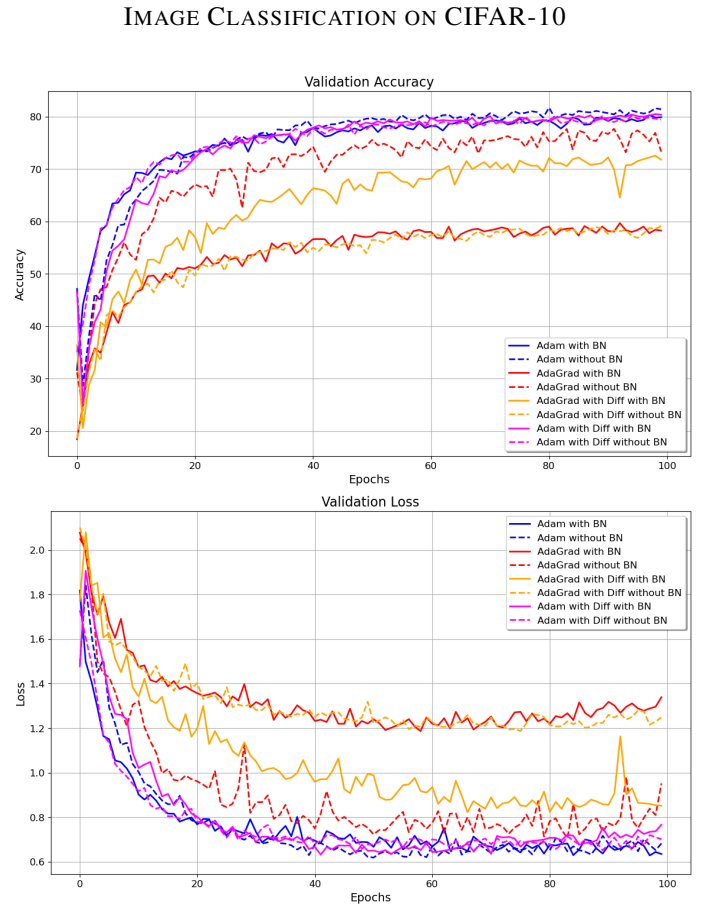


Fig. 6. Validation loss and accuracy on the image classification task with **Resnet-34 on CIFAR-10**.