A Comparative Study of Optimizers in Deep Learning: Adam, AdamW and Adamax

Anas Ahmad
Zewail City
s-anas.abdelsalam@zewailcity.edu.eg

Ibrahem Ali
Zewail City
s-ibrahem.ali@zewailcity.edu.eg

Galal Mohamed
Zewail City
s-galal.qassas@zewailcity.edu.eg

Mohamed Hassan
Zewail City
s-mohamed-hassan@zewailcity.edu.eg

Mohamed Ehab
Zewail City
s-mohamed.ehab@zewailcity.edu.eg

Abstract– Adam is one of the most used optimization algorithms in deep learning. This growing demand started from basic abstract trials to high level applications, and this came from the stability and great resources consuptions achieved by it. Still, adam optimizer has a tree of variants, in this paper we will make a performance analysis on adam. It's variants (AdamW, AdaMax), to see what is the performance differences In this report, we focused on the study of the Adam optimizer and its two most important variants, AdamW and Adamax, which supposed to be developed for overcoming the issues. We analyzed the performance of these optimizers based on a multi-class classification on the MNIST dataset and also benchmark problems. This shows how Adam, AdamW andAdamax perform with manual and built-in TensorFlow Implementations for them.

Introduction

In deep learning, the learning process depends on optimizing the step sizes to achieve the best possible updates, a principle that has evolved from basic methods to more advanced, production-level techniques. The key player in this process is optimization algorithms, which guide the training of models. This report will focus on analyzing the Adam optimizer family, one of the most used algorithms in deep learning. Adam adapts the step size for every parameter, thus allowing faster convergence and better overall performance. Being among the most popular optimization algorithms, Adam has contributed a significant deal to solving different optimization challenges. With increased popularity, several variants of Adam were developed to address such issues as stability, convergence speed, and generalization. That explains why we delegate to these different Adam variants, and especially the AdamW and Adamax where each had an objective to improve optimization different from those earlier in this area of deep learning tasks.

Problem Definition

The driving from adam came to fill the gaps of the Adam optimizer, that's how its variants, Adamax and AdamW, and more have been developed. In particular, stability, convergence speed, and generalization performance are reviewed for such optimizers. The Adam optimizer finds a wide range of applications due to certain advantages in fast convergence in optimization tasks with stochastic objectives. However, it has some defects in generalization, stability, and convergence, especially for large-scale machine learning problems where robust optimization is required [1]. The main problem of the Adam model is the use of biased estimators for the first and second moments of gradients, which may lead to slow convergence, especially in convex optimization problems where the convergence of Adam to critical points is not guaranteed [2]. Moreover, it has been found that Adam has a problem with a decreasing learning rate schedule that may lead to performance degradation in cases of noisy or sparse gradients [3].

Mathematical Formulation:

The optimization problem can be defined as minimizing an objective function [2],[4] where:

- **θ** represents the set of model parameters to be determined through optimization.
- The objective is to minimize a loss function that measures the discrepancy between predicted and actual outputs.
- **x** represents the input data, sampled from a distribution **D**, and **y** is the corresponding output label.

Objective Function:

$$J(\theta) = E\_\{(x, y) \sim D\} [\ell(f(\theta; x), y)] \textbf{ (1)[3]}$$

Where:

- **θ**: The model parameters to be optimized.
- **(x, y) ~ D**: The input data **x** and the corresponding output label **y** sampled from the distribution **D**.
- **ℓ(f(θ; x), y)**: The loss function, which measures the error between the predicted output **f(θ; x)** and the true label **y**.
- **f(θ; x)**: The model's predicted output for input **x**.

And as proposed, despite Adam's advantages, it has limitations. Sometimes non-convergence may occur from non-negative learning rates caused by using exponential moving averages of the gradients (β_0 and β_1).

We will analyze and compare Adam's stability, convergence speed, and adaptability with those of its variants, AdamW and Adamax.

Literature Review

The first version of the Adaptive Moment Estimation optimizer [2] was introduced as a solution to the challenges found in traditional stochastic optimization methods. It was referred to as a lightweight optimizing algorithm designed to overcome the limitations of AdaGrad and RMSProp by the ability to work with non-stationary objectives. The main innovation of Adam lies in computing individual learning rates for each parameter, making it particularly effective for extensive machine learning tasks requiring fast convergence with minimal memory requirements which was the main highlighted feature in this algorithm and being introduced as a solution for broader and faster ML algorithms optimizing approach

However, this initial formulation was not without limitations. The ADAM had issues in terms of its proposed convergence behavior. For example, as highlighted by Dereich and Jentzen [1], the basic version of Adam can fail to converge to the critical points of strongly convex objective functions under certain conditions. This occurs due to the algorithm's reliance on biased estimates of the gradient's first and second moments, which, while effective in many scenarios, can lead to suboptimal or divergent behavior in others especially in dealing with large datasets [2]. These issues started a family of sub-algorithms of the ADAM to refine and improve the issues addressed in each version.

In order to overcome these drawbacks, [2] investigated Adam's non-convergence characteristics, paying particular attention to the issues that arise when exponential moving averages are used. The problem of the anticipated gradient's dependence on its history is identified as critical when they present an instance of a convex optimization problem in which Adam's convergence cannot be achieved. In addition to highlighting the fundamental shortcomings of Adam's original formulation, this research led to the creation of AMSGrad, a variation that takes into account long-term memory of previous gradients. Because it strives to maintain a non-increasing learning rate during its use, AMSGrad stops the learning rate from fading. Additionally, AMSGrad ensures that all issues pertaining to divergence are adequately addressed while concurrently preserving all of Adam's benefits for computation and user ease. This refinement exemplifies the iterative process of algorithmic enhancement within the broader context of adaptive optimization methods.

Based on the main ideas of ADAM, [6] integrated it into Beetle Antennae Search (BAS), which resulted in an improved algorithm called BAS-ADAM. This paper used a hybrid approach that integrated the global search features of BAS with the adaptive learning rate feature of ADAM to improve the speed and accuracy in convergence. BAS-ADAM showed perfect performance in solving complex optimization problems with balancing the trade-offs between exploration and exploitation. The experiments showed the algorithm's adaptation capability to the change in landscapes of the problems, making it a robust choice for problems that require efficient optimization. Additionally, the research indicated the potential of hybrid algorithms in advancing optimization techniques by focusing on improving some particular components.

In optical waveguide mode solvers, Adam optimizer has proven to be quite helpful. A key application involves its incorporation into the Pseudospectral Frequency-Domain (PSFD) framework to improve its accuracy and stability. By using Adam, researchers can quickly find the penalty coefficients required to solve eigenvalue problems, such the two-dimensional Helmholtz equation, and impose boundary conditions.

Adam provides better robustness to local minima, faster convergence rates, and more computational efficiency compared to metaheuristic techniques like the Gorilla Troops Optimizer (GTO) and more conventional techniques like Strong Boundary Methods (SBM). This has been especially noticeable in intricate waveguide structures with many subdomains or sharp corners. As a result, Adam has emerged as a crucial instrument for improving the accuracy and functionality of optical component analysis and design [7].

Adam achieves faster convergence in early epochs as it has high efficiency and adoptable learning rates, but it can face overfitting issues sometimes, but Stochastic Gradient Descent (SGD) with momentum offers smoother convergence over time and better generalization. Alternatives like AdaBelief, which improves generalization by modifying step sizes according to gradients, and Padam, which incorporates partial adaptiveness to balance generalization and convergence speed, solve Adam's problems. Padam has shown that it has the ability to improve performance in some scenarios by resolving the tiny learning rate issue of the adaptive approaches [8].

METHODOLOGY

Dataset Selection

We Used the MNIST dataset for our experiments. MNIST, which was originally introduced in [9]. It consists of 70,000 grayscale images of handwritten digits (0-9) with a resolution of 28x28 pixels. We accessed the dataset through TensorFlow library [10].

Data Preprocessing

All images were normalized to have pixel values in the range [0, 1]. The dataset were split into training, validation, and test sets in an 80:10:10 ratio to ensure robust evaluation.

Task Type

The primary task is image classification. For MNIST, the objective was to accurately identify handwritten digits from 0 to 9. This task was chosen to evaluate the performance of our custom optimizers (OurAdam, OurAdamW, OurAdaMax) against standard optimizers from tensorflow.

Optimizer Selection

We used Adam (Vanilla) as our default optimizer (anchor). Apart from Adam, we studied three variants of Adam, namely, AdamW and Adamax. Each one of these optimizers was chosen since they had properties unique in some ways. Specifically,

AdamW decouples weight decay from the gradient update that helps in preventing overfitting and improving generalization.

Adamax, on the other hand, replaces the L2 norm of the infinitive norm with a scaling of the learning rate, giving it more robustness toward noisy gradients and outliers.

Experimental Setup

These experiments will serve to test and compare Adam, AdamW, and Adamax optimizers. The experiments were done on the MNIST dataset to keep the testing of different optimizers consistent. The model architecture, training, and the evaluation protocols are kept identical for a fair comparison.

Algorithms Implementations:

I. Adam (Basic Version)

Adaptive optimization algorithm designed to dynamically adjust the learning rate for each parameter using first and second moments of gradients [5].

Steps:

1. **Compute the Gradient:** At each step **t**, Adam computes the gradient of the loss function with respect to the model's parameters:

$$g_t = \nabla_0 L(\theta_t) \quad \textbf{(2) [5]}$$

L is the loss function

θ_t represents the model parameters at step **t**

g_t is the gradient at step **t**

2. **Update the First Moment:** Adam calculates the exponential moving average of the gradients (first-moment estimate):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \quad \textbf{(3) [5]}$$

β_1 is the decay rate for the first moment

m_t is the biased first-moment estimate.

3. **Update the Second Moment:** Adam calculates the exponential moving average of the squared gradients (second-moment estimate):

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2 \quad \textbf{(4) [5]}$$

β_2 is the decay rate for the second moment (default: 0.999).

v_t is the biased second moment estimate

(variance).

4. **Bias Correction:** Adam corrects the bias in both moment estimates:

$$\hat{m} = \frac{m_t}{1-\beta_1^t} \textbf{ (5) [5]}$$

$$\hat{v} = \frac{v_t}{1-\beta_2^t} \textbf{ (6) [5]}$$

5. **Update the Parameter:** The corrected moment estimates are used to update the model parameters:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \textbf{ (7) [5]}$$

is the base learning rate. ϵ is a small constant (default: 10^{-5}) to prevent division by zero.

II. AdamW (Weight Decay Adam)

AdamW introduces weight decay as a regularization term, applied directly to the parameters instead of being part of the gradient update [3].

1. **First Moment Update:**

$$m_t = \beta_1 m_t - 1 + (1 - \beta_1) g_t \textbf{ (8) [3]}$$

2. **Second Moment Update:**

$$v_t = \beta_2 v_t - 1 + (1 - \beta_2) g_t^{\,2} \textbf{ (9) [3]}$$

3. **Bias Correction for Moments:**

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

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \textbf{ (10) [3]}$$

4. **Weight Decay Update:**

$$\theta_t \leftarrow \theta_t - \eta \lambda \theta_t \textbf{ (11) [3]}$$

5. **Parameter Update Rule:**

$$\theta t + 1 = \theta t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

(12) [3]

Where:

- **η** is the learning rate
- **λ** is the weight decay coefficient
- **ε** is a small constant to prevent division by zero
- g_t is the gradient at step **t**
- β_1 and β_2 are decay rates for the first and second moments, respectively

III. Adamax (Adam with Infinity Norm)

Adamax, an extension of Adam, replaces the L2 norm in the second moment calculation with the L∞ norm, making it more stable for sparse gradients and robust to outliers [5].

1. **First Moment Update:**

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \textbf{ (13) [5]}$$

2. **Infinity Norm (u) Update:**

$$u_t = max(\beta_2 u_{t-1}, |gt|) \textbf{ (14) [5]}$$

3. **Learning Rate Adjustment:**

$$\alpha_t = \frac{\eta}{(1-\beta_1^t)} \textbf{ (15) [5]}$$

4. **Parameter Update Rule:**

$$\theta_{t+1} = \theta_t - \alpha_t \frac{m_t}{u_t + \epsilon} \textbf{ (16) [5]}$$

Where:

- **η** is the learning rate
- **ε** is a small constant to prevent division by zero
- **gt** is the gradient at step **t**
- **β1** and **β2** are decay rates for the first moment and infinity norm, respectively

---

II. Environment:

The experiments were performed in Google Colab because it offers free access to GPUs, which speed up the training, and allows execution on the cloud. This means all variants of the Adam optimizer could be tested in an environment that supports efficient deep-learning training. Software: The implementation was done in TensorFlow, version 2.16.1, on a Google Colab environment for implementing the model and optimizers.

I. Modeling approach

Model Architecture

This architecture for the proposed CNN is shown in Fig. 1, representing the flowchart of the model layers and corresponding shapes.

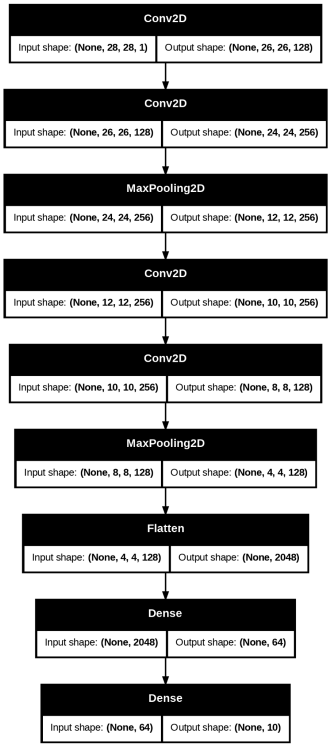

Fig. 1. *"Proposed Architecture using Convolutional Neural Network."*

Fig. 1 shows an overview over the architecture of our used model, from input through convolution blocks, max-pooling layers, fully connected layers, and finally to output. The shape of every input and output of any layer depicts how data gets processed in each step of processing. (created using TensorFlow in Google Colab) [11].

The model in this work is designed to perform image classification tasks; it has been trained for handwritten digit classification using the MNIST dataset. The architecture involves the following components:

Input Layer:

It takes grayscale images of size 28x28x1, which is appropriate for the MNIST dataset. This shape appears at the top of this architecture diagram.

Convolutional Blocks:

First, the first convolution block includes two convolutional layers: the first using 128 filters of size 3×3 while the second uses 256 filters of size 3×3; both use the ReLU activation function in order to extract the features. These capture low-level spatial features including edges and textures.

The second convolution block contains two more convolutional layers: the first with 256 filters of size 3 × 3 and the second with 128 of such size, which further detail the feature representations extracted in previous layers.

Max-Pooling Layers:

After each convolution block, it follows a max-pooling layer with a window of 2 × 2 pixels. It reduces the spatial dimensions by taking the maximum over that window. This reduces the computational load and allows the network to focus on more abstract features.

Flatten Layer:

The feature maps coming from the convolutional and pooling layers will be flattened into a 1D vector in this layer for the purpose of preparation of the output to fully connected layers.

Fully Connected Layers:

First Fully Connected Layer: There will be 64 neurons in it. An activation function ReLU has been used with an intent to model complex combinations of the extracted features.

The second fully connected layer is an output layer that consists of 10 neurons from classes 0 through 9. An output layer will apply a softmax activation function, meaning that its output will be a probability distribution across the 10 classes.

Training Process:

The model was trained on the MNIST dataset for multi-class classification, using the cross-entropy loss function for more than two classes. Training was done with three different optimizers: OurAdam, OurAdamW, and OurAdaMax. Each of them ran for the same learning rate of 0.001, and one function was created to compile and train interchangeably between all three models.

Dataset:

The MNIST dataset consists of 60,000 training images and 10,000 test images of handwritten digits. These images were normalized to lie between 0 and 1. The dataset was divided into training, validation, and test sets, and the labels were one-hot encoded for use with the categorical cross-entropy loss function.

Training Procedure:
The training procedure was carried out with the following settings:
Batch Size: A batch size of 64 images for every iteration. Epochs: The model was trained on 10 epochs, with validation at the end of every epoch. Compilation of *Optimizers:* A function was created to compile and train the model with each of the optimizers; this means all optimizers passed through the same training. Loss: The loss function applied is categorical cross-entropy, which is appropriate for multi-class classification tasks.

Metric Results for Evaluation:
Individual performances of each optimizer were analyzed with respect to the following metrics:
Accuracy: The overall number of correctly classified images, both in training and testing.
Precision, Recall, and F1-score: The scores were taken per class for detailed knowledge about the performance of the different optimizers.
Training Time: This is a total time consumed by the optimizer for completing the training and gives a hint at the efficiency of the optimizer.
Loss: The trend of the loss function during training was followed to understand how fast and effectively it was minimized.

Results:
The results of training using each of the optimizers are visualized in the following figures to compare the training accuracy over epochs:

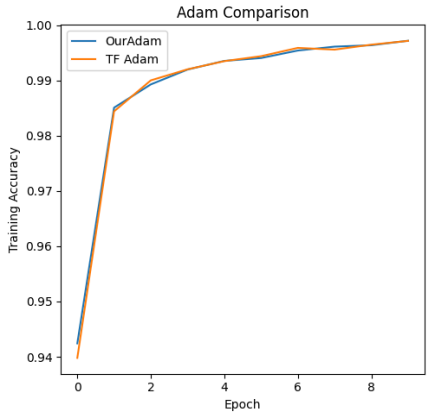

Fig. 2. *"Training accuracy comparison of OurAdam and TensorFlow Adam"* (created using TensorFlow in Google Colab) [11].
Fig. 2 presents the comparison between OurAdam and TensorFlow Adam. In general, the two optimizers have similar learning curves. However, small differences in convergence can be observed, as reflected in the figure.

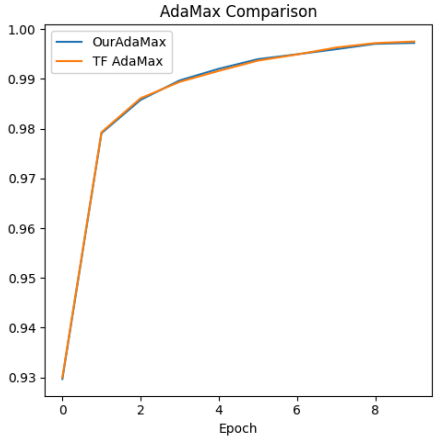

Fig. 3. **"***Training accuracy comparison of OurAdaMax and TensorFlow AdaMax."* (created using TensorFlow in Google Colab) [11].

Fig. 3 compares the performances of OurAdaMax against TensorFlow AdaMax. As can be observed, there is no significant difference between the two optimizers, and the margins are very minimal.

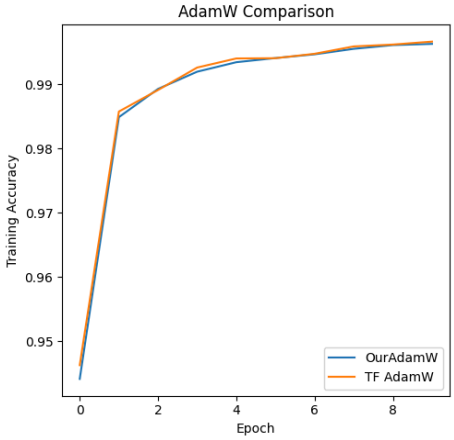

Fig. 4. **"***Training accuracy comparison of OurAdamW and TensorFlow AdamW."* (created using TensorFlow in Google Colab) [11].
Fig. 4 compares OurAdamW against TensorFlow AdamW. It can be observed that the performance of both optimizers is almost identical, though OurAdamW achieved higher accuracy in later epochs.

II. Problems (benchmarking)

We considered the Rosenbrock and Powell functions as benchmarking approaches with the modeling approach. These functions are used in optimization to test how optimizers can deal with difficult landscapes and converge accordingly.

The Rosenbrock Function: One of the popular benchmarks in optimization is Rosenbrock function, which was designed to test an optimizer for its ability in navigating through narrow, curved valleys in the landscape. It is defined as:

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \textbf{ (13)[12]}$$

It includes a narrow, curving valley with a global minimum that the optimizer should efficiently escape with local minima and converge to. The challenging landscape is actually used as a test bed for the capability of the optimizers to efficiently search the high-dimensional spaces; most of the algorithms by design would run a set of different tests to optimize in the constrained regions.

Powell Function: The Powell function is among the multi-dimensional optimization problems that would test the performance of an optimizer in non-linear, non-differentiable, and highly irregular landscapes. It is defined as:

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - x_3)^4 + 10(x_1 - x_4)^4$$
(14)[13]

This function will test the efficiency of an optimizer to go through the optimum of the high-dimensional non-convex space including some non-smooth regions. In general, the Powell function is utilized within the realm of optimizers that face more complicated and irregular landscape problems.

We proceeded mainly with our implemented versions of adam and the variants, but in first we compared the performance between both in this context

Results:

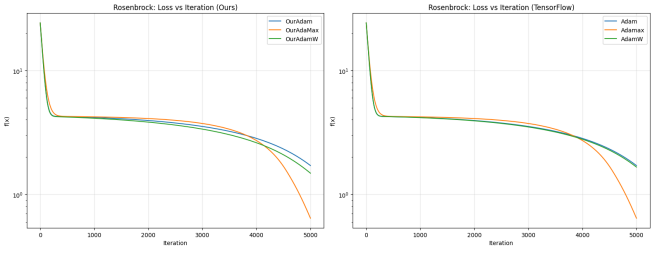

Fig. 5 *"Adam and it's variants and their loss performance with respect to iterations on Rosenbrock and Powell on our implemented vs Built-in TF versions"* [11]

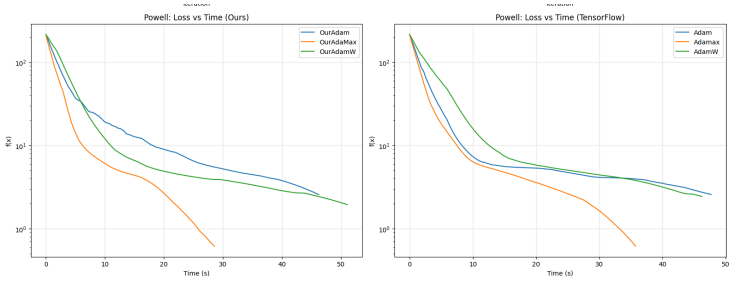

Fig 6. *"Adam and it's variants with their loss performance with respect to time on Rosenbrock and Powell on our implemented vs Built-in TF"*

From Fig 5, and Fig. 6, we e can see that our implemented and the built-in versions have very close performance in the analysis on the problems.

RESULTS

I. Modelling

Optimizer	Accuracy (%)	Precision (Avg.)	Recall (Avg.)	F1-Score (Avg.)	Training Time (s)	val_Loss
TensorFlow Adam	99.0	0.9889	0.9901	0.9900	1min 7s	0.0496
TensorFlow AdamW	98.97	0.9897	0.9897	0.9897	1min 8s	0.0303
TensorFlow AdaMax	99.0	0.9901	0.9900	0.9900	1min 8s	0.0328
OurAdam	99.08	0.9908	0.9909	0.9908	1min 6s	0.0374
OurAdamW	98.93	0.9894	0.9893	0.9893	1min 7s	0.0388
OurAdamax	98.97	0.9895	0.9897	0.9896	1min 7s	0.0405

Table 1. *"Table showing results analysis of training process using the algorithms using our model."*

Validation Accuracy:

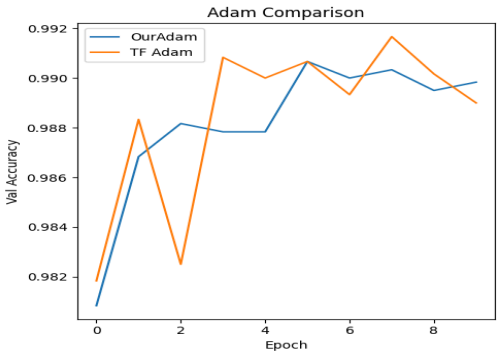

Fig. 7. "*Validation accuracy comparison of OurAdam and TensorFlow Adam*" (created using TensorFlow in Google Colab) [11].

Fig. 7 presents the comparison between OurAdam and TensorFlow Adam. Both optimizers exhibit similar learning trends, with TensorFlow Adam converging slightly faster in the initial epochs. Minor fluctuations can be observed, but both achieve nearly identical final validation accuracy.

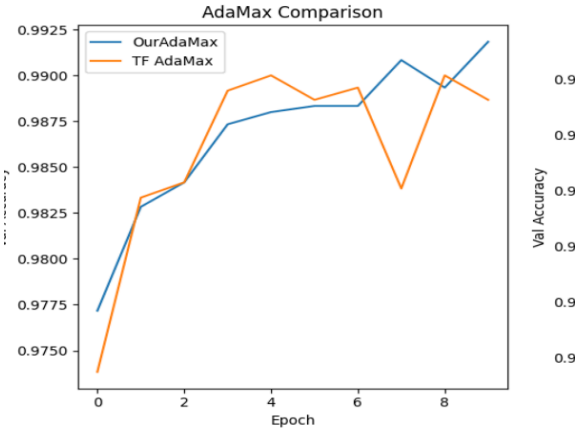

Fig.8. "*Validation accuracy comparison of our AdaMax and TensorFlow AdaMax*" (created using TensorFlow in Google Colab) [11].

Fig.8, shows the findings of our version from the AdaMax and TensorFlow AdaMax are near. In the time of the first few epochs, TensorFlow AdaMax immediately exhibits a higher growth in accuracy. On the other hand, our version of AdaMax is more consistent and stable, therefore, it conquers TensorFlow AdaMax in the later epochs. The final validation accuracy of our AdaMax is a little bit higher (~99.2%) in comparison with that of TensorFlow AdaMax (~99.0%).

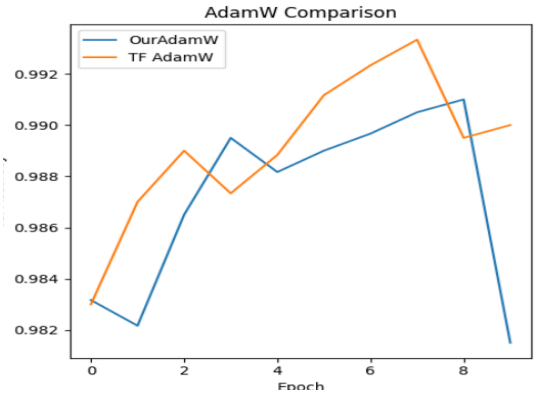

Fig.9. "*The validation accuracy comparison of our AdamW to TensorFlow AdamW*" (created using TensorFlow in Google Colab) [11] is shown in this figure.

Fig.9 illustrates the comparison of efficiency between the implemented AdamW and TensorFlow AdamW. AdamW of TensorFlow has the ability to show a faster validation accuracy growth within the initial 3 epochs. The results of the two optimizers at the start show little difference in their performance, with the TensorFlow AdamW being the best performer by the sixth epoch. As the final accuracy, TensorFlow AdamW completes at the etch point of almost 99.3%, while on the other hand, AdamW drops to 88.2% indicating instability.

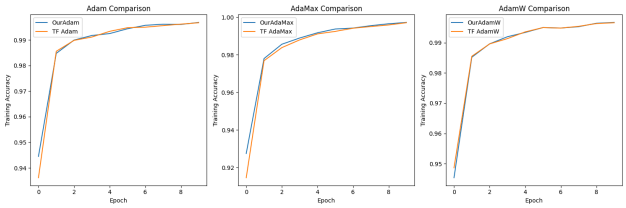

Fig 10. *"Model performance on the implemented vs Tf adam versions"* [11]

From Table 1. , Fig 10 and classification reports from the code implementation, we can conclude that the difference between the algorithms is barely seen as both have great results in early stages of training, this is due to the dataset constrained size and nature which limited the error space to test the model training process, this takes us to the results of the problem-based evaluation approach.

II. Benchmarking Results

Results on the Rosenbrock Function Optimization Using Custom Optimizers *(our implemented versions)*

Problem	Optimizer	Iters	Optimal	Optimal f	CPU Time (s)
Rosenbrock	OurAdam	5000	[-0.30707827, 0.09730258]	1.70936	30.118
	OurAdaMax	5000	[0.19911991, 0.03851836]	0.641537	24.629
	OurAdamW	5000	[-0.21763672, 0.04947444]	1.48308	34.971
Powell	OurAdam	5000	[1.7492565, -0.15232038, 0.8991471, 1.3608856]	2.56693	30.365
	OurAdaMax	5000	[1.145217, -0.10534453, 0.67196286, 0.8587022]	0.615235	24.319
	OurAdamW	5000	[1.6043972, -0.1414491, 0.8506962, 1.2386361]	1.93647	35.244

Table 2. "*Table showing benchmarking results of Adam, AdaMax, and AdamW on the Rosenbrock and Powell optimization problems, presenting the number of iterations, optimal values, and computational time across tasks.*" [11]

In Table 2, our Implementation for AdaMax showed faster convergence in terms of running time, especially in the Powell function, while Adam and AdamW display more consistent results in achieving optimal values. The data emphasizes the nuanced differences in performance among the variants, providing insights into their strengths in specific problem scenarios.

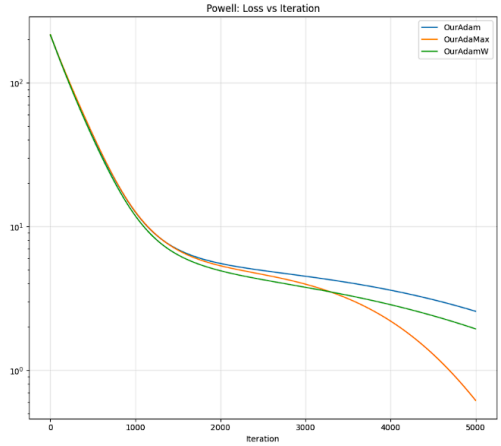

Fig.11 *"Loss vs Iteration for the Powell Optimization Problem using our implemented Adam, AdaMax, and AdamW. "* [11]

The graph in Fig 11 illustrates the convergence behavior of the three optimizers over 5000 iterations, and AdaMax demonstrates a more consistent loss reduction curve, achieving faster convergence compared to Adam and AdamW, also AdamW shows a balanced performance, while Adam converges slower in the later stages of optimization.

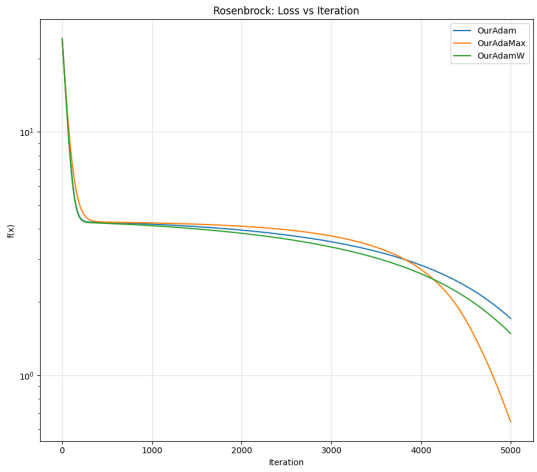

Fig. 12. *"Loss vs Iteration for the Rosenbrock Optimization Problem using Adam, AdaMax, and AdamW."* [11]

Fig 12 shows the convergence behavior of the three optimizers over 5000 iterations. AdaMax achieves the fastest and most stable convergence, followed closely by AdamW. Adam demonstrates slower convergence in the final stages, indicating a less aggressive loss reduction compared to the other two variants."

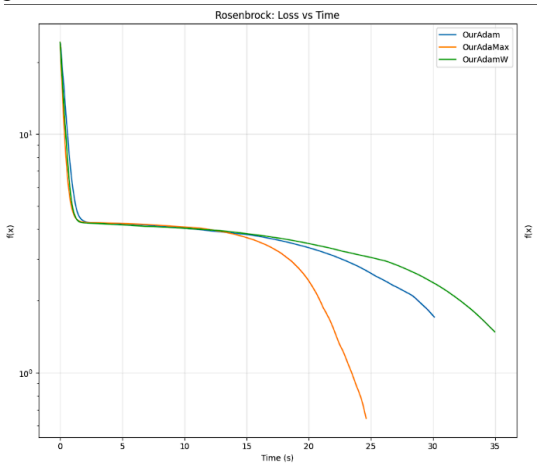

Fig. 13 *"Rosenbrock: Loss vs Time for Adam, AdaMax, and AdamW. The graph illustrates the loss reduction over*

time for each optimizer, highlighting differences in convergence speed and computational efficiency." [11]

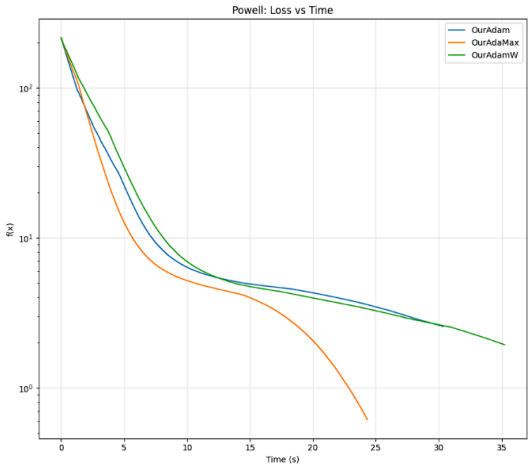

Fig. 14. *"Powell: Loss vs Time for Adam, AdaMax, and AdamW. This graph compares the performance of the three optimizers in minimizing loss over time, showcasing variations in convergence behavior."* [11]

AdaMax shows convergence in both problems, achieving significant loss reduction in less time compared to Adam and AdamW and this shows that it is good in handling noisy gradients and sparse updates, and AdamW shows a more gradual but steady decline in loss, indicating reliable performance over longer durations. and in the Adam performs well initially but exhibits slower convergence in the later stages of optimization which supports the claims we gave regarding the variants. These insights suggest that while AdaMax is better in scenarios requiring rapid convergence, AdamW has more stable long-term optimization behavior.

Visualize with function contours with the optimization OurAdam Visualization

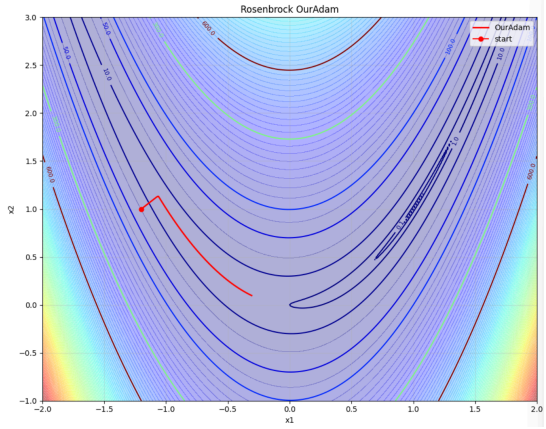

Fig. 15. "*The optimization path of OurAdam is shown, starting from an initial point (red marker) and following a path (red line) through the contours*" (created using TensorFlow in Google Colab) [11].

Fig. 15 shows the optimization path followed by OurAdaMax on the Rosenbrock function. The green path from the starting point (red marker) converges toward the global minimum along the function contours.

Initial Point and Path: The optimizer starts from point (-1.2, 1.0), and OurAdaMax is used to minimize the function. The optimal point found by the optimizer is approximately [0.19911991, 0.03851836], with a function value of 0.641537. The optimization took approximately 24.629051 seconds to reach the optimal solution.

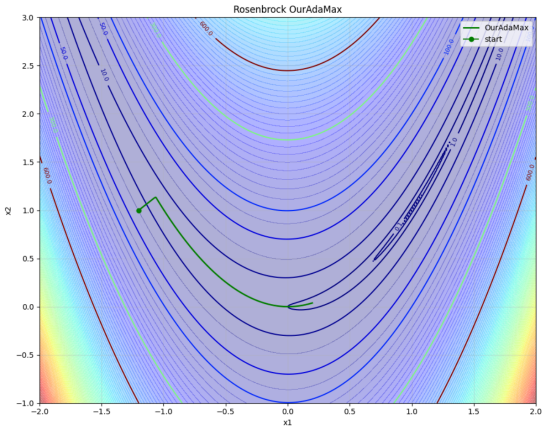

Fig. 16. "*The optimization path of OurAdaMax is shown, starting from an initial point (green marker) and following a path (green line) through the contours*" (created using TensorFlow in Google Colab) [11].

Fig. 16 shows the optimization path followed by OurAdaMax on the Rosenbrock function. The green path, starting from the initial point (green marker), efficiently converges toward the global minimum by navigating through the function's contours.

Initial Point and Path: The optimizer begins at the initial point (-1.2, 1.0) and uses OurAdaMax to minimize the Rosenbrock function. From table 2, The optimal solution found by the optimizer is approximately [0.19911991, 0.03851836], with a function value of 0.641537. The optimization process took 24.629051 seconds to complete, showing the efficient convergence of OurAdaMax in this challenging landscape.

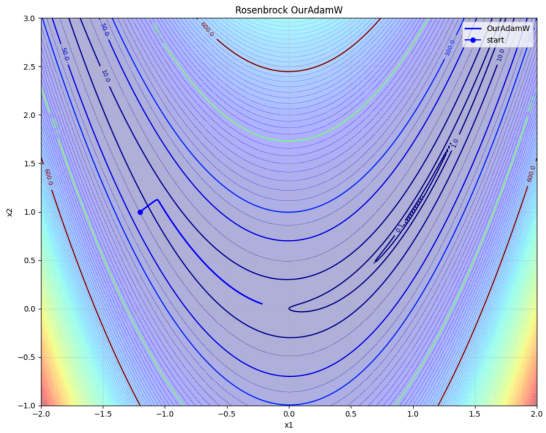

Fig. 17. "*The optimization path of OurAdamW is shown, starting from an initial point (blue marker) and following a path (blue line) through the contours*" (created using TensorFlow in Google Colab) [11].

Fig.17 represents the optimization path of OurAdamW on the Rosenbrock function. Here, it depicts the path of the optimizer in a red line starting at an initial point, blue marker, and across the contours to reach its global minimum.

The optimizer begins at the same initial point (-1.2, 1.0) and uses OurAdaMax to minimize the Rosenbrock function. From table 2, The optimal solution found by the optimizer is approximately [0.199 0.0385], with a function value of 0.64. The optimization process took 31.77 seconds to complete, showing the efficient convergence of OurAdaMax in this challenging landscape.

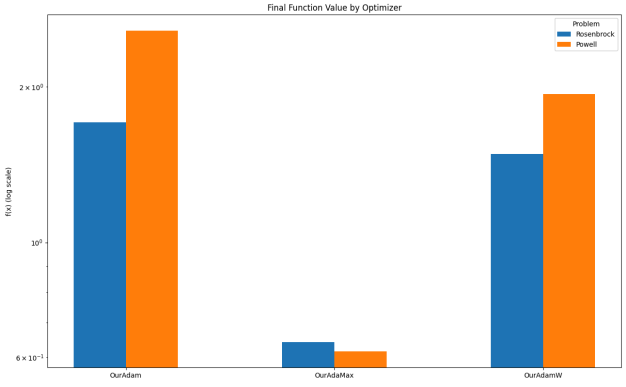

Fig. 18. *" final function values achieved by our implemented Adam, AdaMax, and AdamW across both optimization problems, highlighting differences in optimization effectiveness."*[11]

Fig. 18 represents the final function values for our implemented Adam, AdaMax, and AdamW versions across the Rosenbrock and Powell problems. AdaMax has the best performance over the other optimizers, where it achieved the lowest final function values in both tasks, indicating high convergence and optimization efficiency. AdamW performs moderately well, showing improved results compared to Adam, but still falls short of AdaMax. Adam, on the other hand, records the highest final values, suggesting slower convergence and suboptimal performance in both tasks. These results emphasize the robustness of AdaMax in handling optimization challenges across different problem types.

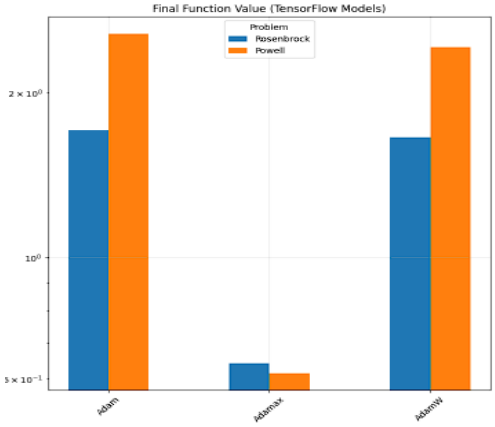

Fig 19. "*final function values achieved by Built-in Adam, AdaMax, and AdamW across both optimization problems, highlighting differences in optimization effectiveness"*

In Fig 19 we can see that the results in the built-in function don't differ significantly from the implemented version, where adamX still has the best performance, while the AdamW has slight better minimization poerformence in both of the problems

DISCUSSION

The first part of our work compared the performances of the Adam, AdamW, and AdaMax optimizers on a model-based classification approach on the MNIST dataset, for which differences in accuracy, convergence, and computational efficiency were minimal due to the constraints of the database simplicity we couldn't expand our training process to a sufficient range, so we used another problem-based approach using Rosenbrock and Powell benchmark functions, providing a harder optimization landscape. Among this,we observed that the AdaMax performed better in terms of convergence speed and final loss when gradients were noisy, while AdamW had a solid level of stability too in the updates and was able to avoid overfitting because of the decoupled weight decay mechanism, in other hand Adam showed a moderate performance and could not converge for the Powell function well.

And in computational usage, AdaMax was the fastest, however AdamW had a somewhat higher overhead because of the weight decay changes. With more distinct differences in the Powell function as compared to Rosenbrock, the final function value study confirmed AdaMax's resilience across tasks. These results highlight the need to select optimizers based on the particular requirements of the work rather than just their widespread popularity, as optimizer performance varies greatly depending on the structure and characteristics of the problem.

References

[1] S. Dereich and A. Jentzen, “Convergence rates for the Adam optimizer,” *arXiv (Cornell University),* Jul. 2024, doi: 10.48550/arxiv.2407.21078.

[2] S. J. Reddi, S. Kale, and S. Kumar, “On the Convergence of Adam and Beyond,” *arXiv (Cornell University)*, Jan. 2019, doi: 10.48550/arxiv.1904.09237.

[3] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *arXiv (Cornell University)*, Jan. 2017, doi: 10.48550/arxiv.1711.05101.

[4] I. Goodfellow, Y. Bengio, and A. Courville, "Optimization for Training Deep Models," *Deep Learning*, sanso1988.github.io. https://sanso1988.github.io/docs/deep_learning_book_chapter8.

[5] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *arXiv (Cornell University),* Jan. 2014, doi: 10.48550/arxiv.1412.6980.

[6] A. H. Khan, X. Cao, S. Li, V. N. Katsikis, and L. Liao, “BAS-ADAM: an ADAM based approach to improve the performance of beetle antennae search optimizer,” IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 2, pp. 461–471, Mar. 2020, doi: 10.1109/jas.2020.1003048.

[7] P.-J. Chiang, “Adaptive penalty method with an Adam optimizer for enhanced convergence in optical waveguide mode solvers,” Optics Express, vol. 31, no. 17, p. 28065, Jul. 2023, doi: 10.1364/oe.495855.

[8] J. Chen, D. Zhou, Y. Tang, Z. Yang, Y. Cao, and Q. Gu, “Closing the generalization gap of adaptive gradient methods in training deep neural networks,” *arXiv.org*, Jun. 18, 2018. https://arxiv.org/abs/1806.06763

[9] Y. LeCun et al., "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[10] "MNIST Dataset," TensorFlow, [Online]. Available: https://www.tensorflow.org/datasets/catalog/mnist. [Accessed: Dec. 2024]

[11] "Implementation.ipynb,"Google Colab, 2024. [Online]. Available: https://colab.research.google.com/drive/1uDj3Ifp1iNppUcp9HgvYj7RlvB9HBbRW?usp=sharing#scrollTo=4bsKdfJ0Qa53. [Accessed: Dec. 2024]

[12] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," Comput. J., vol. 3, no. 3, pp. 175–184, 1960.

[13] Surjanovic, "Powell's Method", Simon Fraser University, [Online]. Available: https://www.sfu.ca/~ssurjano/powell.html. [Accessed: Dec. 2024]

Code Implementation →
Math_Project_implementation.ipynb