Adaptive Momentum Estimation (ADAM): Enhancing Learning Rate Optimization

Salma Wael, *202201761*, Dana Amr, *202201323*, Mariam Hani, *202200903*, Bosy Ayman, *202202076*

Department of Computer Science, Zewail University, Egypt.

Manuscript received Date; revised Date. First published Date November 24. Zewail City.

Abstract: In this project, we introduce a novel optimization method to enhance the ADAM optimizer for training artificial neural networks (ANNs) with non-convex cost functions. By modifying ADAM's parameter update rule, we improve its ability to avoid local minima, boosting convergence and efficiency. We directly explore ADAM's momentum-based approach, known for its computational efficiency and minimal memory requirements, and assess its performance through numerical, analytical, and simulation-based evaluations on datasets such as MNIST and CIFAR-10. Fine-tuning ADAM's learning rates helps achieve improved accuracy, providing insights into its advantages and effectiveness in deep learning applications. This analysis highlights ADAM's strengths in accelerating convergence, enhancing generalization, and overcoming challenges in complex optimization problems.

1. Introduction

The Adam optimizer, short for Adaptive Moment Estimation, is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problems involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm [1]. It adapts the learning rate for each parameter, making the training process faster and more efficient, especially for large datasets. Adam's ability to handle sparse gradients and adapt dynamically to different scenarios makes it highly efficient for training deep neural networks [2]. It stands out from other optimizers due to a few key features:

Adam adjusts the learning rate for each parameter dynamically based on the first moment (mean) and the second moment (uncentered variance) of the gradients. This makes it particularly effective for problems with sparse gradients or non-stationary objectives. Unlike fixed learning rate methods like SGD, Adam optimization provides adaptive learning rates for each parameter based on the history of gradients. This allows the optimizer to converge faster and more accurately, especially in high-dimensional parameter spaces [3].

2. Methodology

The problem addressed by the Adam optimizer is to minimize the error (or loss) function $L(\theta)$, where θ represents the set of parameters (weights and biases) of a neural network. The objective is to iteratively adjust θ to find the set of values that minimizes $L(\theta)$. $\theta = \theta - \alpha \cdot g_t$

Here, θ = Model parameters, α = Learning rate, and g_t = Gradient of the cost function with respect to the parameters.

This update changes the parameters θ in the negative direction of the gradient to minimize the cost function. The learning rate α determines the size of the step.

In the standard gradient descent algorithm, the learning rate α is fixed, meaning we need to start

at a high learning rate and manually change the alpha by steps or by some learning schedule. A lower learning rate at the onset would lead to very slow convergence, while a very high rate at the start might miss the minima.

Adam solves this problem by adapting the learning rate α for each parameter θ, enabling faster convergence compared to standard gradient descent with a constant global learning rate.

2.1. Steps

Here are the steps for implementing the Adam optimization algorithm:

1) Set initial parameters such as the moment vector m_0 and second moment vector v_0 to zero, and initialize the timestep $t = 0$.
2) Set the learning rate α to 0.0001, the decay rate for β_1 (denoted as $\beta_1 = 0.9$) and β_2 (denoted as $\beta_2 = 0.9$), and set ϵ (denoted as $\epsilon = 0$) for numerical stability.
3) For training, increment the timestep t by 1 and compute the gradient g_t with respect to the parameters at timestep t.
4) Update the first moment estimate m_t as:

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

5) Update the second moment estimate v_t as:

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

6) Correct the bias in the moment estimates:

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

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

7) Update the parameters θ_t as:

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

8) For the numerical experiment, use the MNIST dataset. Apply preprocessing steps to handle the dataset and train the model using the Adam optimizer.
9) During training, visualize the loss and evaluation metrics after each epoch using a graph.
10) Implement the Adam optimization from scratch using Python, employing libraries such as Matplotlib and NumPy for visualization and numerical operations.

2.2. Hyperparameters

The main hyperparameters in Adam are:

- α — Step size for the optimization. It controls how large a step is taken towards the minimum at each iteration.
- β_1 — Decay rate for momentum. The typical value is 0.9, as we want to give higher weight to the most recent gradients.
- β_2 — Decay rate for squared gradients. The typical value is 0.999. This value helps to capture the long-term memory of gradients, providing a stable estimate of variance.
- ϵ — A small value to prevent division by zero. It is usually set to around 1×10^{-8}.

Together, these parameters enable Adam to converge faster while remaining numerically stable. The default values generally work well for most tasks but can be adjusted for specific use cases [4].

3. Coding Adam From scratch :

This implementation demonstrates how the Adam optimizer can be constructed from scratch and used in training a basic neural network model. Model Setup: A simple feedforward neural network with weights, biases, and a categorical cross-entropy loss function is defined. The network uses the softmax activation function to predict probabilities for multi-class classification. Training Process: Training is performed for a specified number of epochs with mini-batches. During each iteration: Forward propagation computes the predictions and loss. Backward propagation calculates the gradients of the weights and biases. The custom Adam optimizer updates the parameters. Evaluation: Training loss and accuracy are recorded for each epoch, providing insights into model performance. Key Takeaways: Efficiency: Adam performs efficient optimization by adjusting learning rates individually for each parameter, improving convergence speed. Accuracy: The custom implementation achieves satisfactory results comparable to standard implementations in popular deep learning frameworks like TensorFlow or PyTorch. Educational Value: Building Adam from scratch helps deepen understanding of optimization mechanics and the interplay between hyperparameters (e.g., learning rate, beta1, beta2).

4. Results

4.1. Scratch Results

Training Results:

Epoch 1 Loss : 0.6767, Accuracy : 84.66

Final Epoch Loss: 0.2810, Accuracy: 92.18

Observations: The Scratch implementation achieves consistent improvement in training accuracy and loss across epochs. Training stops at 92.18 accuracy, which is a strong result for a scratch implementation. Slightly slower improvement compared to the built-in Adam due to the simplified setup and potential for minor inefficiencies in gradient computation.

4.2. Built-in Results

Training Results:

Epoch 1 Loss: 1.0628, Accuracy: 71.41, Validation Accuracy: 89.91

Final Epoch Loss: 0.2627, Accuracy: 92.71, Validation Accuracy: 92.64

Observations: Starts with a higher initial loss due to default initialization differences but achieves faster convergence. Final training accuracy (92.71) and validation accuracy (92.64) are slightly better than the custom implementation.

5. Comparison

Performance: Both implementations perform well, with the built-in Adam slightly outperforming the custom version in both training and validation metrics.

Usability: The built-in Adam optimizer in frameworks like TensorFlow/Keras benefits from: Optimized numerical operations for faster computation. Better validation support for generalization assessment.

Educational Value: The custom implementation is valuable for learning, showcasing the underlying mechanics of Adam optimization.

6. Conclusions

This report explored the implementation and application of ADAMS (Automatic Dynamic Analysis of Mechanical Systems) as a powerful tool for simulating, analyzing, and optimizing complex mechanical systems. Through the use of ADAMS, we demonstrated its ability to accurately model dynamic behaviors, predict system responses, and improve design efficiency.

The results obtained from our simulations highlight the effectiveness of ADAMS in reducing the need for physical prototyping, saving time and resources while enhancing the reliability of

the designs. Key features such as multibody dynamics, integration with CAD tools, and robust post-processing capabilities make ADAMS an indispensable tool for engineers and researchers.

Despite its many advantages, certain challenges were encountered, including the steep learning curve for new users and the computational intensity for highly detailed models. Future work could focus on exploring advanced features of ADAMS, integrating it with real-time data for adaptive simulations, and expanding its applicability to emerging domains such as robotics and autonomous systems.

In conclusion, ADAMS remains a cornerstone of dynamic mechanical analysis, providing critical insights that drive innovation and efficiency in mechanical design and engineering [5].

Acknowledgements

Supervised by : Dr. Ahmed Abdelsamea Department of Computer Science , Zewail University.

References

[1] GeeksforGeeks. (n.d.). Adam optimizer. Retrieved December 19, 2024, from https://www.geeksforgeeks.org/adam-optimizer/

[2] Codecademy. (n.d.). Adam optimization. Neural networks. Retrieved December 19, 2024, from https://www.codecademy.com/resources/docs/ai/neural-networks/adam-optimization

[3] Kumar, K. (n.d.). Getting to know Adam optimization: A comprehensive guide. LinkedIn. Retrieved from https://www.linkedin.com/pulse/getting-know-adam-optimization-comprehensive-guide-kiran-kumar

[4] Agarwal, R. (2023, September 13). Complete guide to the Adam optimization algorithm. Built In. Retrieved from https://builtin.com/machine-learning/adam-optimization

[5] Kingma, D. P., University of Amsterdam, OpenAI, Ba, J. L., University of Toronto. (n.d.). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. In ICLR 2015 [Conference-proceeding]. Retrieved from https://arxiv.org/pdf/1412.6980.pdf