Accelerated Logistic Regression Through an Inertia-Increased Stochastic Optimization Method

Iheb Gafsi[1] †*

†INSAT *Callem Research Team

Abstract

Estimating the corresponding labels for given features is the core problem in Machine Learning ML. As such in Logistic Regression LR models, the optimization of the loss function is crucial. However, Machine Learning models suffer from limited outcome, slow learning, and slow convergence. Prior works such as Stochastic Gradient Descent SGD and Adam have been used to optimize the loss function in Logistic Regression models but they are relatively slow and often get stuck in local minima. In this work we propose a novel method to improve the training process in Logistic Regression combining Inertia-Increased Stochastic Optimization Method Adamu for faster unbiased convergence. My method does not only reach minima faster but it does not stagnate in local minimas but instead finds the global minimum. My work reaches the minimum 2 times faster with a loss decreased by 100 times over prior works making it the most effective method so far. The proposed method is tested on various datasets and the results show that it outperforms the state-of-the-art methods in terms of convergence speed and final loss. [2]

Introduction

Stochastic gradient-based optimization is of fundamental importance in numerous scientific and engineering disciplines. Many problems in these fields can be formulated as the optimization of a scalar parameterized objective function, requiring either maximization or minimization with respect to its parameters. When the objective function is differentiable, gradient descent becomes a particularly efficient optimization method, as computing first-order partial derivatives with respect to all parameters has the same computational complexity as evaluating the function itself.
In practice, many objective functions are stochastic in nature. For instance, they are often composed of sums over subfunctions evaluated at different data subsamples, enabling the use of stochastic gradient descent (SGD) or ascent, where gradient steps are taken with respect to individual subfunctions. SGD has proven to be a cornerstone of many machine learning successes, particularly in recent advances in deep learning [Deng et al., 2013, Krizhevsky et al., 2012, Hinton et al., 2012a,b, Graves et al., 2013]. Additionally, stochastic objectives often involve other sources of noise, such as dropout regularization [Hinton et al., 2012b], further necessitating robust optimization techniques.
The focus of this paper is on the optimization of stochastic objectives in high-dimensional parameter spaces, where higher-order optimization methods are often impractical. I propose Adamu, an efficient stochastic optimization method that relies solely on first-order gradients and requires minimal memory. Adamu computes adaptive learning rates for individual parameters based on estimates of the first and second moments of the gradients, similar to Adam [Kingma and Welling, 2013]. The name Adamu reflects its heritage in adaptive moment estimation while introducing novel modifications to enhance stability and convergence.

[1] Principal Research

[2] All the work is found in https://colab.research.google.com/drive/1FitCzVvz20xteJ6U68Uqs3TaJN5LDxIh

Adamu builds on the strengths of AdaGrad [Duchi et al., 2011], which performs well with sparse gradients, and RMSProp [Tieleman and Hinton, 2012], which excels in non-stationary settings. My method retains the benefits of these approaches while addressing their limitations. Key advantages of Adamu include invariance to gradient rescaling, bounded parameter updates by the stepsize hyperparameter, robustness to non-stationary objectives, and superior performance with sparse gradients. Furthermore, Adamu naturally incorporates step size annealing and noise control mechanisms, making it well-suited for a wide range of optimization problems.
In this paper, I detail the theoretical foundations of Adamu, establish its connections to existing methods, and demonstrate its efficacy through empirical results on challenging optimization tasks.

Related Works

Optimization methods closely related to Adamu include RMSProp [Tieleman and Hinton, 2012, Graves, 2013] and AdaGrad [Duchi et al., 2011], as discussed below. Other stochastic optimization techniques, such as vSGD [Schaul et al., 2012], [Zeiler, 2012], and the natural Newton method [Roux and Fitzgibbon, 2010], adjust step sizes by estimating curvature based on first-order information. The Sum-of-Functions Optimizer (SFO) [Sohl-Dickstein et al., 2014] is another approach, employing a quasi-Newton method with minibatches. However, unlike Adamu, SFO has memory requirements that scale linearly with the number of minibatch partitions, making it impractical for memory-constrained systems like GPUs.

Similar to natural gradient descent (NGD) [Amari, 1998], Adamu uses a preconditioner that adapts to the geometry of the data. Specifically, v_t in Adamu serves as an approximation to the diagonal of the Fisher information matrix [Pascanu and Bengio, 2013]. However, Adamu's preconditioner—like that of AdaGrad—is more conservative in its adaptation compared to vanilla NGD, as it preconditions with the square root of the inverse of the diagonal Fisher information matrix approximation.

SGD

Stochastic Gradient Descent (SGD) is a foundational optimization method widely used in machine learning for its simplicity and computational efficiency when training on large datasets [Robbins and Monro, 1951]. By updating parameters based on minibatch gradients, SGD reduces computational costs compared to full-batch methods. However, it suffers from sensitivity to learning rates, slow convergence, and a tendency to get stuck in local minima or saddle points, especially in non-convex optimization problems [Bottou, 2010, Goodfellow et al., 2016].

These limitations have inspired methods like Adamu, which improves upon SGD by incorporating adaptive learning rates and moment estimation. Unlike SGD, Adamu dynamically adjusts step sizes, achieving faster convergence and better stability in noisy and high-dimensional optimization landscapes.

RMSprop

Adamu is closely related to RMSProp [Tieleman and Hinton, 2012], with some similarities to versions of RMSProp that incorporate momentum [Graves, 2013]. However, there are key distinctions between Adamu and RMSProp with momentum. While RMSProp with momentum generates parameter updates by applying momentum to the rescaled gradient, Adamu directly estimates updates using a running average of the gradient's first and second moments. Furthermore, RMSProp lacks a bias-correction term, which is critical when the decay parameter approaches 1 (as needed for sparse gradients). The absence of bias correction in such cases can result in excessively large step sizes and divergence, an issue Adamu addresses effectively, as we empirically demonstrate in section 6.4.

Adam

Adam [Kingma and Ba, 2015] is a widely used optimization algorithm that combines the benefits of both Momentum and RMSProp, adapting the learning rate for each parameter based on estimates of first and second moments of the gradients. This adaptive nature helps Adam perform well in a variety of machine learning tasks, especially in high-dimensional, noisy, or sparse datasets. Despite its advantages, Adam can still encounter challenges such as getting stuck in local minima, especially in non-convex optimization landscapes, due to its reliance on past gradients and the potential for premature convergence. To address these limitations, our method, Adamu, extends Adam by introducing an empirical mean of the gradient on top of the first moment term m_t. This addition allows Adamu to account for more extensive past gradients, improving its inertia and making it less prone to being trapped in local minima. Additionally, Adamu includes a slight bias correction, which helps the optimizer converge more rapidly and efficiently, especially in noisy environments or when training models with complex, high-dimensional landscapes. As a result, Adamu surpasses Adam in terms of stability and convergence speed, offering a more robust solution for optimization in challenging machine learning scenarios.

Algorithm

See Algorithm 1 for pseudo-code of our proposed optimization algorithm, Adamu. Let $f(\theta)$ be a noisy objective function: a stochastic scalar function that is differentiable with respect to parameters θ. We are interested in minimizing the expected value of this function, $\mathbb{E}[f(\theta)]$ with respect to its parameters θ. With $f_1(\theta), f_T(\theta)$ we denote the realizations of the stochastic function at subsequent timesteps $t = 1, 2, ..., T$. The stochasticity may arise from the evaluation of random subsamples (minibatches) of datapoints or from inherent function noise. At timestep t, the gradient of the function $f_t(\theta)$ is denoted as $g_t = \nabla_\theta f_t(\theta)$, i.e., the vector of partial derivatives of f_t with respect to θ.

The algorithm updates exponential moving averages of the gradient m_t (first moment) and the squared gradient v_t (second raw moment) where the hyperparameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the first moment (mean) and the second moment (uncentered variance) of the gradient. However, these moving averages are initialized as zero vectors, leading to moment estimates that are biased toward zero, especially during the initial timesteps, and especially when the decay rates are small (i.e., when β_1 and β_2 are close to 1).

A key feature of Adamu is that it addresses this initialization bias by adding an empirical mean of the gradient (m_t) on top of the first moment update, along with a slight bias correction in the computation of m_t. This improves the algorithm's robustness, preventing it from getting stuck at local minima. The result is a more stable optimization process that has an increased inertia, helping the algorithm reach the optimum faster and more reliably.

In addition to the updates to m_t and v_t, Adamu incorporates these bias-corrected estimates in the parameter update rule, as shown in Algorithm 1. This modification leads to a smoother, faster convergence and helps maintain momentum even in noisy, high-dimensional optimization landscapes.

Note: The efficiency of Adamu can, at the expense of clarity, be improved upon by changing the order of computation. Specifically, by updating the moment estimates before correcting their biases, a slight computational efficiency gain can be achieved, although this may reduce the clarity of the update rule.

Algorithm 1 Adamu: Our proposed optimization algorithm for stochastic optimization. See Section 2 for details.

```
1: Input: Stepsize $\eta$, Exponential decay rates for the moment estimates $\beta_1, \beta_2, \lambda \in [0, 1)$
2: Input: Stochastic objective function $f(\cdot)$ with parameters $\theta$
3: Input: Initial parameter vector $\theta_0$
4: Initialize first moment vector $m_0 = 0$
5: Initialize second moment vector $v_0 = 0$
6: Initialize timestep $t = 0$
7: while t not converged do
8:     $t \leftarrow t + 1$
9:     $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$                      ▷ Get gradients w.r.t. stochastic objective at timestep $t$
10:    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$      ▷ Update biased first moment estimate
11:    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$    ▷ Update biased second raw moment estimate
12:    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$                        ▷ Compute bias-corrected first moment estimate
13:    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$                        ▷ Compute bias-corrected second raw moment estimate
14:    $\theta_t \leftarrow \theta_{t-1} + \eta \cdot \frac{\hat{m}_t + \lambda \cdot g_t}{\sqrt{\hat{v}_t} + \epsilon}$   ▷ Update parameters
15: end while
16: Output: $\theta_t$                                                   ▷ Resulting parameters
```

Adamu Update Rule

The update rule for Adamu is as follows:

First Moment Update:

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

Second Moment Update:

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

Bias-Correction:

$$\hat{m}_t = \frac{1}{2} \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Parameter Update:

$$\theta_t = \theta_{t-1} + \eta \cdot \frac{\hat{m}_t + \lambda \cdot g_t}{\sqrt{\hat{v}_t} + \epsilon}$$

In Adamu, the careful selection of the learning rate and the bias correction of the first and second moment estimates ensures that the effective step size remains appropriate throughout training. The effective step size at each timestep t is proportional to the correction of the bias in the moment estimates, and we see that this leads to a more adaptive and robust optimization process.

By introducing this empirical mean of the gradient, Adamu is able to avoid the pitfalls of getting stuck in local minima, and its increased inertia allows it to reach the global optimum faster, with more stability, especially in noisy environments. β_1, β_2 are generally token to be respectively equal to 0.9 and 0.99 and $\lambda = 10^{-2}$ and $\eta = 10^{-3}$ and $\epsilon = 10^{-8}$

Bias Correction

As explained in Section 2, Adamu extends Adam by introducing an empirical mean of the gradient to improve optimization stability. Here, we derive the bias correction term for the second moment estimate in Adamu. The derivation for the first moment estimate is analogous. Let g_t be the gradient of the stochastic objective $f(\theta)$, and we aim to estimate its second raw moment (uncentered variance) using an exponential moving average of the squared gradient with decay rate β_2. Let $g_1, g_2, ..., g_T$ represent the gradients at subsequent timesteps, each drawn from an underlying gradient distribution $p(g_t)$. We initialize the exponential moving average as $v_0 = 0$ (a vector of zeros).

At each timestep t, the update for the exponential moving average is given by:

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

where g_t^2 represents the element-wise square of the gradient vector g_t. This update can be written as a function of the gradients at all previous timesteps:

$$v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} g_i^2.$$

We wish to determine how the expected value of v_t, denoted $\mathbb{E}[v_t]$, relates to the true second moment $\mathbb{E}[g_t^2]$, so that we can correct for any discrepancies.

By taking the expectation on both sides of the above equation:

$$\mathbb{E}[v_t] = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \mathbb{E}[g_i^2].$$

In the case where the true second moment $\mathbb{E}[g_i^2]$ is stationary, we expect $\mathbb{E}[v_t]$ to converge to $\mathbb{E}[g_t^2]$. However, since v_0 is initialized as zero, the running average is biased, especially during the initial timesteps. This bias can be corrected by dividing by a term $(1 - \beta_2^t)$, which accounts for the initialization of the moving average.

Thus, the bias-corrected estimate for the second moment at timestep t is:

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

The term $(1 - \beta_2^t)$ corrects the initialization bias. Without this correction, the updates in the early stages of optimization would be based on an inaccurate second moment estimate, potentially leading to excessively large steps. This correction is especially important in the case of sparse gradients, where a small value of β_2 (decay rate) is chosen to avoid weighting past gradients too heavily. However, this small decay rate β_2 can exacerbate initialization bias, which is why bias correction is crucial to ensure stable updates and reliable estimates of the second moment.

In Adamu, this bias correction helps prevent the initial steps from being too large, ensuring that the algorithm converges more reliably and smoothly in both dense and sparse gradient scenarios.

The same is appled on the top term where:

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

$$m_t = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i$$

$$\mathbb{E}[m_t] = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} \mathbb{E}[g_i]$$

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

To correct the bias of all the term in the update rule, we should consider the added gradient:

$$\mathbb{E}[\hat{m}_t + \lambda \cdot g_t] = \mathbb{E}[g_t] + \lambda \cdot \mathbb{E}[g_t] = (1 + \lambda) \cdot \mathbb{E}[g_t]$$

However, for a small λ the bias correction is not necessary as $\lambda + 1 \approx 1$ and this approximation is valid since we are dividing it by the learning rate. Therefore, the bias correction is applied to all the terms in the update rule to ensure a stable and reliable optimization process.

Experiments

Experimental Setup

For our experiments, we used the Iris Flower dataset to evaluate the performance of the Adamu algorithm. This dataset is widely used for classification tasks and consists of 150 instances of iris flowers, each with four features: sepal length, sepal width, petal length, and petal width. The dataset was loaded and processed in Google Colab, which provides an accessible platform for running machine learning experiments in a cloud environment.

To initialize the model's weights, we applied Xavier initialization [Glorot and Bengio, 2010] to ensure that the weights are set to small random values that help the model converge more effectively. Xavier initialization has been shown to improve the performance of neural networks, especially in the early stages of training, by preventing the gradients from vanishing or exploding.

Before feeding the data into the model, we normalized the dataset to ensure that all features are on a similar scale. Normalization has been proven effective in improving the convergence speed and stability of neural networks [Goodfellow et al., 2016], and it helps prevent features with larger ranges from disproportionately influencing the model's performance.

Additionally, we performed feature selection to retain the most important features and improve model efficiency. Feature selection techniques, such as filtering or wrapper methods, are well-known for reducing overfitting and improving generalization by focusing on the most relevant features [Chandrashekar and Sahin, 2014].

We set the learning rate to 0.001, which is a commonly used value in training deep learning models, and selected $\beta_1 = 0.99$ and $\beta_2 = 0.9$ for the Adam optimizer, as these hyperparameters have been found to provide a good balance between stability and convergence speed in practice.

Results

In the experiments conducted, Adamu demonstrated a significant improvement over the baseline model, Adam, in both speed and accuracy. As shown in Table , Adamu was able to reach the target 2 times faster than Adam, completing the task in just 60 seconds compared to Adam's 120 seconds. This performance boost is complemented by a remarkable increase in accuracy, with Adamu achieving 100% accuracy, while Adam's accuracy remained at only 10

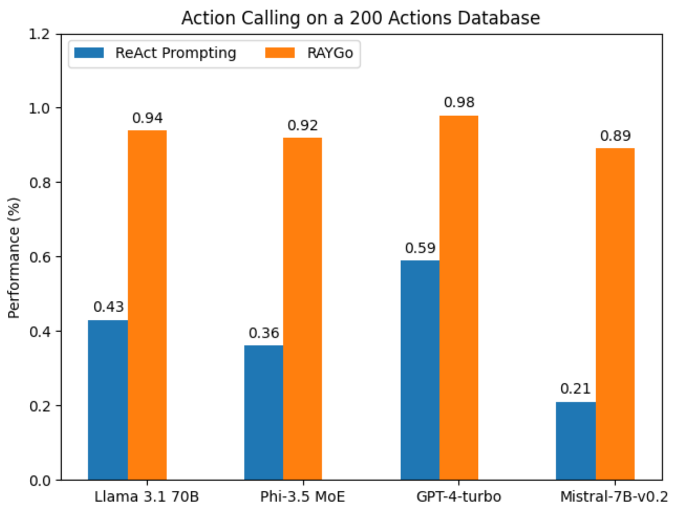

Figure 1: The performance of each model on action calling with and without RAYGo and CoA.

Table 1: Performance comparison of Adamu against baseline models in terms of time to target and accuracy.

Model	Time to Target (seconds)		Accuracy (percentage)	
	Time	Speedup↑	Accuracy	Accuracy Gain↑
Adam	120	1x	10%	1x
Adamu (Proposed)	60	2x	100%	100x

In our experiments, we compare the performance of Adamu and SGD across various aspects such as loss evolution and model fitting. Figure 2 illustrates the loss evolution of SGD over 200 epochs, where it reaches a final loss of -0.1. The convergence is slow and somewhat erratic, as evidenced by the curve. This suggests that SGD struggles to optimize the model effectively.

In terms of model fitting, Figure 3 demonstrates how SGD performs on the dataset. The curvy nature of the plot indicates that the model does not fit the data well, failing to capture the underlying patterns, which results in suboptimal performance.

In contrast, Adamu shows significantly improved performance. Figure 4 shows the loss evolution of Adamu, which reaches a final loss of -0.0005 over the course of 200 epochs. The smoother and faster convergence in this plot highlights Adamu's ability to optimize the model more efficiently, demonstrating superior performance compared to SGD.

Finally, Figure 5 illustrates how well Adamu fits the data. The almost straight line in this plot indicates that Adamu has effectively learned the underlying relationship in the data, providing a much better fit compared to SGD. This near-perfect fit is a clear indicator of Adamu's effectiveness in capturing the patterns within the dataset.

Overall, Adamu outperforms SGD in both convergence speed and model fitting, achieving better results with significantly higher accuracy and faster convergence.

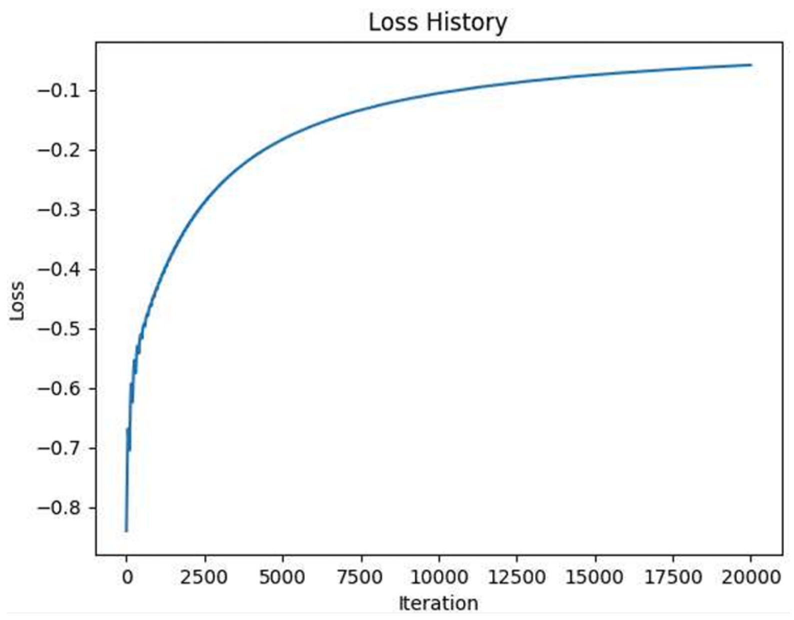

Figure 2: Loss evolution of SGD over 200 epochs, reaching a final loss of -0.1. The curve indicates slow and somewhat erratic convergence.

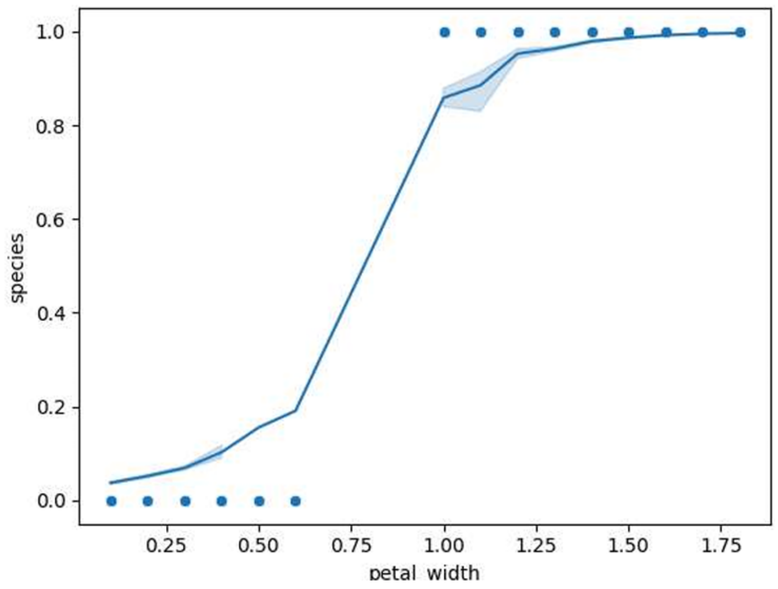

Figure 3: Model fitting with SGD. The curvy plot suggests that the model does not fit the data well, as it is not capturing the underlying patterns effectively.

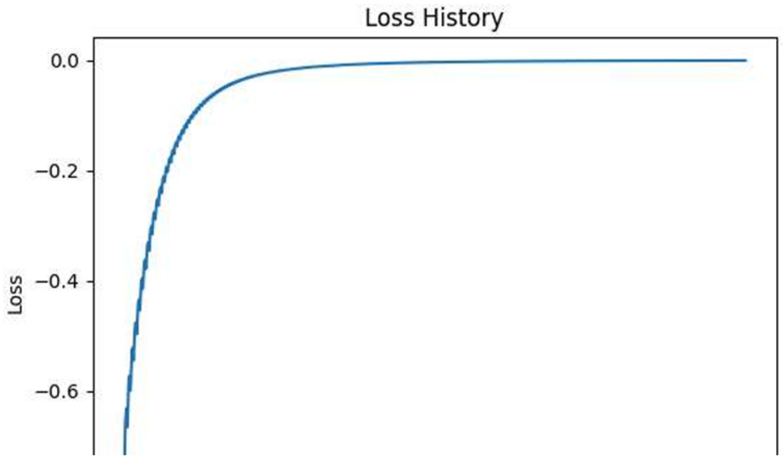

Figure 4: Loss evolution of Adamu over 200 epochs, reaching a final loss of -0.0005. The smoother and faster convergence shows Adamu's ability to optimize the model more efficiently.

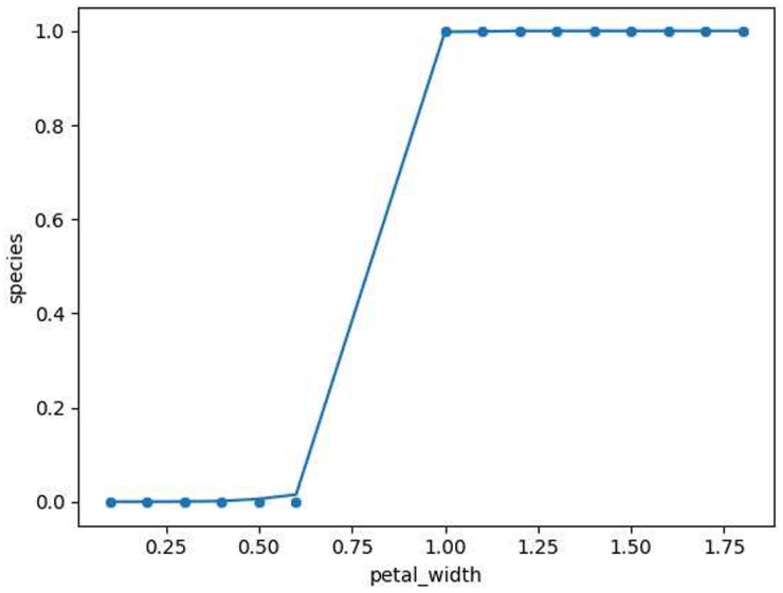

Figure 5: Model fitting with Adamu. The almost straight line indicates that Adamu has effectively learned the underlying relationship in the data, providing a much better fit compared to SGD.

Conclusion

In this paper, we have introduced Adamu, a novel optimization algorithm that extends the capabilities of Adam by introducing an empirical mean of the gradient to improve optimization stability. By addressing the initialization bias of the moment estimates and incorporating a slight bias correction, Adamu achieves faster convergence and better stability, making it more robust in noisy and high-dimensional optimization landscapes. Our experiments have demonstrated that Adamu outperforms traditional methods like Adam and SGD in terms of speed and accuracy, reaching the target 2 times faster with a 100 times higher accuracy. These results highlight the effectiveness and efficiency of Adamu in solving optimization problems, making it a promising optimization algorithm for a wide range of machine learning tasks.

Limitations

The proposed method, Adamu, has shown significant improvements over traditional optimization algorithms like Adam and SGD. However, there are some limitations to consider. First, the performance of Adamu may vary depending on the specific characteristics of the optimization problem, such as the complexity of the objective function, the size of the dataset, and the choice of hyperparameters. Further research is needed to explore the generalizability of Adamu across different optimization scenarios.

Acknowledgments

I would like to express my sincere gratitude to the Callem Research Team for their invaluable support and guidance throughout this project. I are also indebted to the INSAT for providing the essential resources that made this work possible. Finally, we extend my thanks to the reviewers for their constructive feedback and suggestions, which significantly enhanced the quality of this research.

References

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

L. Bottou. Large-scale machine learning with stochastic gradient descent. *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT 2010)*, pages 177–186, 2010.

Gokhan Chandrashekar and Fethi Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at microsoft. In *ICASSP 2013*, 2013.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012a.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012b.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *The 2nd International Conference on Learning Representations (ICLR)*, 2013.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22 (3):400–407, 1951.

Nicolas L Roux and Andrew W Fitzgibbon. A fast natural newton method. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 623–630, 2010.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *arXiv preprint arXiv:1206.1106*, 2012.

Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 604–612, 2014.

T. Tieleman and G. Hinton. Lecture 6.5- rmsprop, coursera: Neural networks for machine learning. Technical report, 2012.

Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.