Electron-Orbital Like Pathways for Neural Optimization: The Helix Algorithm

Authors: Loai A. Alazab [1], Hamdy W. Abd-Elhalim [1], Omar M. Mounir [2]
[1] Beetleware, Egypt. [2] Alexandria University, Alexandria, Egypt.
Email: loaiabdalslam@gmail.com, hamdywaleed20@gmail.com, omarmohamedmounir16@gmail.com
Corresponding Author Email: loaiabdalslam@gmail.com

1. Abstract

Hyperparameter optimization is a crucial yet computationally intensive process in training machine learning and deep learning models. This paper introduces Helix, a novel optimization algorithm inspired by the behavior of an electron orbiting around a nucleus, dynamically spiraling through the optimization landscape to efficiently converge on the global minimum. By emulating the orbital movements of an electron, Helix gradually narrows its trajectory toward the optimal solution, balancing computational efficiency with interpretability.

Helix operates on a lightweight mathematical framework, offering rapid convergence, reduced computational overhead, and broad adaptability across data types. Unlike traditional methods like Gradient Descent, Particle Swarm Optimization, and Adam, Helix demonstrates superior scalability and interpretability, making it particularly suited for large-scale and resource-constrained environments. Benchmark evaluations validate its competitive performance, positioning Helix as a powerful and versatile tool for hyperparameter optimization in modern machine learning workflows.

Index Terms: Hyperparameter Optimization, Algorithm Explainability, Helix Optimizer, Particle Swarm Optimization (PSO), Adam Optimizer, Interpretability in Machine Learning, benchmark, Boston, MNIST, Iris dataset

2. Introduction

In the evolving landscape of machine learning and deep learning, the availability of advanced libraries and frameworks, such as TensorFlow, PyTorch, Caffe, and scikit-learn, has revolutionized the development and experimentation of models across diverse domains like Natural Language Processing (NLP) and Computer Vision [1-4]. These tools empower researchers to design and train models efficiently, yet achieving optimal performance often hinges on the meticulous tuning of hyperparameters—a process that can significantly influence a model's success [5].

As models and datasets grow increasingly complex, hyperparameter optimization has become a critical yet computationally challenging task. Existing optimization methods, such as Adam [6], Particle Swarm Optimization (PSO) [7], and Stochastic Gradient Descent (SGD) [8], offer robust solutions but often struggle to balance computational efficiency, convergence speed, and interpretability. Addressing these limitations, we propose Helix, a novel optimization algorithm designed to streamline hyperparameter tuning while maintaining transparency and adaptability.

Helix integrates principles of orbital-based movements with gradual adjustments, enabling efficient exploration of the optimization landscape. Unlike traditional "black box" optimizers [9], Helix provides clear insights into its iterative process, promoting interpretability without compromising performance. Its lightweight framework ensures applicability across a wide spectrum of machine learning tasks and data types, including text, images, and tabular data.

To demonstrate its versatility, we evaluate Helix on a range of datasets, including Iris [10], Boston Housing [11], MNIST [12], and synthetic datasets such as quadratic data with noise. Comparative analyses with established methods like Adam, PSO, and SGD reveal Helix's superior performance in terms of convergence speed, computational efficiency, and scalability. Table 1 summarizes the diverse tasks and benchmarks examined in this study, highlighting Helix's adaptability across both traditional machine learning and deep learning models.

By uniting efficiency, transparency, and flexibility, Helix represents a significant step forward in hyperparameter optimization, offering a practical and interpretable solution to a persistent challenge in machine learning.

Dataset Name	Data Type	Task Type	Optimizers compared
Boston dataset	Tabular data	Regression task	Helix, Adam, SGD
Quadratic Noisy dataset	Tabular data	Regression task	Helix, Adam, SGD
Iris dataset	Tabular data	Clustering task	Helix, Adam, SGD, PSO
Textual data	Textual data	Auto-Encoder task	Helix, Adam, SGD
MNIST	Imaging data	Computer Vision task	Helix, Adam, SGD

Table 1 shows different datasets comparisons

3. Related Work

From the perspective of the gradient information in optimization, popular optimization methods can be divided into three categories: first-order optimization methods, which are represented by the widely used stochastic gradient methods; high-order optimization methods, in which Newton's method is a typical example; and heuristic derivative-free optimization methods, in which the coordinate descent method is a representative.

As the representative of first-order optimization methods, the stochastic gradient descent method as well as its variants, has been widely used in recent years and is evolving at a high speed. Compared with first-order optimization methods, high order methods converge at a faster speed in which the curvature information makes the search direction more effective. Derivative-free optimization methods are mainly used in the case that the derivative of the objective function may not exist or be difficult to calculate. These are the categories of the optimizers, here there is a discussion of the classical optimization algorithms we focus on and their latest developments in machine learning.

This section reviews three prominent optimizers: Adam, Stochastic Gradient Descent (SGD), and Particle Swarm Optimization (PSO). For each optimizer, we will provide an introduction, define the problem it addresses and its proposed solution, present the mathematical model and its computational complexity, discuss the pros and cons.

i) Stochastic Gradient Descent: Since the batch gradient descent has high computational complexity in each iteration for large-scale data and does not allow online update, stochastic gradient descent (SGD) was proposed in [11]. The idea of stochastic gradient descent is using one sample randomly to update the gradient per iteration, instead of directly calculating the exact value of the gradient. The stochastic gradient is an unbiased estimate of the real gradient. SGD reduces the update time for dealing with large numbers of samples and removes a certain amount of computational redundancy, which significantly accelerates the calculation. In the strong convex problem, SGD can achieve the optimal convergence speed. Meanwhile, it overcomes the disadvantage of batch gradient descent that cannot be used for online learning. However, one problem in SGD is that the gradient direction oscillates because of additional noise introduced by random selection, and the search process is blind in the solution space.

$$L(\theta) = \frac{1}{N}\sum_{I=1}^{N}\frac{1}{2}\left(y^i - f_\theta\left(x^i\right)\right)^2 = \frac{1}{N}\sum_{I=1}^{N}\cos t\left(\theta,\left(x^i, y^i\right)\right).$$

If a random sample *i* is selected in SGD, the loss function will be $L*(\theta)$:

$$L*(\theta) = \cos t\left(\theta,\left(x^i, y^i\right)\right) = \frac{1}{2}\left(y^i - f_\theta\left(x^i\right)\right)^2.$$

The gradient update in SGD uses the random sample i rather than all samples in each iteration,

$$\theta' = \theta + \eta\left(y^i - f\theta\left(x^i\right)\right)x^i$$

The computation complexity for each iteration is O(D) where D is the number of features.

Stochastic Gradient Descent (SGD) is a widely used optimization algorithm known for its simplicity and efficiency, particularly in large-scale machine learning tasks. Unlike batch gradient descent, which computes the gradient using the entire dataset, SGD updates the model parameters using a single or a few training examples at each iteration. This approach introduces noise into the optimization process, which can help escape local minima and improve convergence speed. However, the noisy updates can also lead to suboptimal convergence, and the algorithm is sensitive to the choice of learning rate. To mitigate these issues, various modifications of SGD, such as momentum and learning rate schedules, have been proposed. These enhancements aim to stabilize the optimization process and accelerate convergence. Despite its challenges, SGD remains a popular choice due to its ease of implementation and effectiveness in handling large datasets.

ii) Alternating Direction Method of Multipliers (ADAM) :

Augmented Lagrangian multiplier method is a common method to solve optimization problems with linear constraints. Compared with the naive Lagrangian multiplier method, it makes problems easier to solve by adding a penalty term to the objective. Consider the following example,

$$\min\{\theta_1(x) + \theta_2(y) \big| A_x + B_y = b\,,\ x \in X\,,\ y \in Y.\}\ ()$$

The augmented Lagrange function for problem () is

$$L_\beta(x, y, \lambda) = \theta_1(x) + \theta_2(y) - \lambda^\top(Ax + By - b) + \frac{\beta}{2}\big||Ax + By - b|\big|^2.\ ()$$

When solved by the augmented Lagrangian multiplier method, its t^{th} step iteration starts from the given λ_t and the optimization turns out to

$$\{(x_{t+1}, y_{t+1}) = \arg\min\{L_\beta(x, y, \lambda_t) \big| x \in X\,,\ y \in Y\},\ \lambda_{t+1} = \lambda_t - \beta(Ax_{t+1} + By_{t+1} - b).\}\ ()$$

Separating the (*x*,*y*) sub-problem in (), the augmented Lagrange multiplier method can be relaxed to the following alternating direction method of multipliers (ADMM). Its tth step iteration starts with the given (y_t, λ_t), and the details of iterative optimization are as follows:

$$x_{t+1} = \arg\min\left\{\theta_1(x) - (\lambda_t)^\top Ax + \frac{\beta}{2}\ ||\ Con_x||^2 \ \middle|\ x \in X\right\}$$

$$y_{t+1} = \arg\min\left\{\theta_2(y) - (\lambda_t)^\top B_y + \frac{\beta}{2}\left||Con_y|\right|^2 \ \middle|\ y \in Y\right\}$$

$$\lambda_{t+1} = \lambda_t - \beta(Ax_{t+1} + By_{t+1} - b)$$

where $Con_x = Ax + By_t - b$ and $Con_y = Ax_{t+1} + B_y - b$

The penalty parameter β has a certain impact on the convergence rate of the ADMM. The larger β is, the greater the penalties for the constraint term. In general, a monotonically increasing sequence of {βt} can be adopted instead of the fixed β . Specifically, an auto-adjustment criterion that automatically adjusts { β_t} based on the current value of {x_t} during the iteration was proposed and applied for solving some convex optimization problems.
The ADMM method [12] uses the separable operators in the convex optimization problem to divide a large problem into multiple small problems that can be solved in a distributed manner. In theory, the framework of ADMM can solve most of the large-scale optimization problems. However, there are still some problems in practical applications. For example, if we use a stop criterion to determine whether convergence occurs, the original residuals and dual residuals are both related to β, and β with a large value will lead to difficulty in meeting the convergence conditions.
Overall, the Adam optimizer is widely used due to its adaptive learning rate capabilities and robustness to noisy data. Adam combines the advantages of two other extensions of stochastic gradient descent, AdaGrad and RMSProp, by computing individual adaptive learning rates for

different parameters from estimates of first and second moments of the gradients. This makes Adam particularly effective for training deep neural networks, as it can handle sparse gradients and dynamically adjust learning rates, leading to faster convergence. However, Adam requires careful tuning of its hyperparameters, such as the learning rate and the exponential decay rates for the moment estimates. This can be challenging and time-consuming. Additionally, while Adam generally performs well, it may not always achieve the best possible performance on all types of problems, and its reliance on multiple hyperparameters can complicate the optimization process.

iii) Particle Swarm Optimization (PSO)

PSO is an optimization algorithm inspired from the movement of flock of birds which often moves under the guidance of an individual leader bird to find nearby food. In PSO, a bird in the flock is simulated as a particle in the population. The population's best position and particle's best position mimic the leader bird and the best aviation position of each individual bird in terms of the provisioned food resource. In an n dimensional space, the i^{th}PSO's individual is attributed as follows:

- A velocity vector $V_i = \left(v_i^1, v_i^2, \dots, v_i^n\right)$
- A position vector $X_i = \left(x_i^1, x_i^2, \dots, x_i^n\right)$
- A pbest vector $pbest_i = \left(pbest_i^1, pbest_i^2, \dots, pbest_i^n\right)$

In each iteration, the Xi and Vi are updated using the following equations:

$V_{i+1} = wV_i + c_1 r_1 (pbest_i - X_i) + c_2 r_2 (gbest - X_i)$ (1)

$X_{i+1} = X_i + V_i$ (2)

where in (1) and (2):

- gbest is the best global position of the population is $gbest_i = \left(gbest_i^1, gbest_i^2, \dots, gbest_i^n\right)$
- $c_1 \& c_2$ are acceleration constants
- $r_1 \& r_2 \in [0,1]$ are two random numbers
- w is the inertia weight

Particle Swarm Optimization (PSO) is particularly effective at exploring the search space and avoiding local minima, making it suitable for complex optimization problems. Its simplicity and fewer hyperparameters to tune compared to other algorithms make it an attractive choice for many applications. Additionally, PSO does not require the gradient of the objective function, allowing it to be applied to a wide range of problems, including those with non-differentiable or noisy objective functions. However, PSO can be computationally expensive, especially for high-dimensional problems, due to the need to evaluate the objective function multiple times. The algorithm may also converge prematurely to a suboptimal solution if not properly configured,

and its performance is sensitive to the choice of parameters, which may require extensive tuning.

Optimizers play a crucial role in the performance and convergence of machine learning models. Beyond the well-established optimizers such as Adam, SGD, and PSO, several other optimization techniques have been proposed with unique advantages tailored to specific tasks.

RMSProp [13] addresses the challenges of adapting learning rates during training. By using a moving average of squared gradients, it smooths updates and performs particularly well in recurrent neural network training.

Nadam [14] builds upon Adam by incorporating Nesterov Accelerated Gradients, improving convergence speed in noisy gradients. It has demonstrated success in natural language processing tasks.

AdaDelta [15] mitigates issues of learning rate decay over time without requiring a manual schedule. This optimizer performs well in tasks with sparse data updates, such as recommendation systems.

FTRL (Follow-The-Regularized-Leader) [16] is designed for large-scale linear models and online learning tasks. Its ability to handle sparse features makes it suitable for advertising or click-through rate predictions.

Lion (Layer-wise Optimizer) [17] focuses on layer-wise adaptivity, enabling efficient optimization for deep and wide networks with superior generalization. It has shown promise in state-of-the-art vision tasks.

Ranger [18] combines RAdam (Rectified Adam) and Lookahead optimizers to balance smooth convergence with robustness to outliers. This optimizer is particularly useful in handling imbalanced datasets.

Yogi [19] uses a novel mechanism for adjusting learning rates based on variance rather than gradient magnitude, addressing the exploding or vanishing gradient problem effectively.

AMSGrad [20] is a variant of Adam with a more stable convergence mechanism, ensuring the learning rates never increase unexpectedly. This property is beneficial for reinforcement learning tasks.

SMORMS3 [21] optimizes for sparse matrices with improved stability in dynamic tasks. It is particularly effective in tasks like matrix completion and natural language understanding.

COCOB (Conditional Covariate Budget Optimizer) [22] eliminates the need for manual learning rate tuning, focusing on robustness across varying datasets without decay schedules.

While these optimizers excel in their respective scenarios, each comes with trade-offs in terms of computational efficiency, stability, or adaptability to diverse tasks.

Discovery of the Helix Optimizer

Our journey toward the development of the Helix optimizer began with an exploration of SGD for a particularly challenging task requiring faster convergence. While SGD simplicity and performance were commendable in addition to its ability to perform different types of tasks efficiently. Inspired by the orbital motion of electrons around a nucleus, we envisioned an optimization approach that incorporates a spiral trajectory, allowing for both exploratory and convergent behavior simultaneously, and its updates don't lack the flexibility and dynamic motion required to escape saddle and local minima points

Implementing this concept into the Helix optimizer, we encoded spiral movements into the update mechanism, which we hypothesized would enable faster and more robust convergence. Upon testing, the Helix optimizer consistently outperformed SGD across various datasets, demonstrating its potential as a competitive alternative for optimization in machine learning.

4. Helix Optimizer Algorithm

The Helix Optimization Algorithm is a novel approach inspired by electronic orbital principles, particularly orbital movements, to explore and exploit the optimization landscape effectively. Unlike traditional algorithms that rely heavily on gradient-based updates, Helix incorporates iterative adjustments along orbital trajectories, which gradually converge toward the target solution. This technique ensures flexibility and precision, making it suitable for a wide range of optimization tasks in different types of data.

i. Batch Optimization

When applying batch optimization, Helix evaluates the entire dataset at each iteration, allowing for precise and stable updates. The electronical orbital trajectory is guided by the collective error surface, calculated from all data points in the batch. This approach is particularly effective for scenarios where the dataset is relatively small, as it leverages the global structure of the optimization problem to converge efficiently.

ii. Stochastic Optimization

In stochastic optimization, Helix updates the parameters based on a randomly selected subset of data (a mini batch) or even a single data point. This stochastic nature introduces randomness into the optimization trajectory, which can help the algorithm escape local minima and improve generalization. By adapting its orbital movements to stochastic updates, our approach combines the benefits of global exploration with computational efficiency. However, stochastic updates may introduce noise into the optimization process, which requires careful tuning of parameters such as learning rate and radius decay.

iii. Mathematical Formulation

The algorithm operates by iteratively updating parameters θ_t as follows in Equation. 1:

$$\theta_{t+1} = \theta_t + r_t \cos(\alpha_t)\, d_x + r_t \sin(\alpha_t)\, d_y - \lambda_t \nabla L\theta_t$$

Equation 1

where:

- r_t: is the radius decay factor that reduces over iterations.
- α_t: is the orbital angle at step t.
- d_x, d_y : are direction vectors orthogonal to each other.
- $\nabla L\theta_t$: is the gradient of the loss function at θ_t
- λ_t : is a learning rate that controls the step size.

Helix offers a unique blend of interpretability and adaptability. Its orbital-based mechanism provides intuitive insights into the optimization process, making it less of a "black box." Additionally, its ability to operate effectively in both batch and stochastic modes makes it a versatile choice for various domains and data types.

1. **Radius Decay (r_t)**
    - Controls how far the optimizer moves in each iteration.
    - Initially large to allow global exploration, it gradually decreases to focus on local exploitation near the optimal point.
2. **Orbital Angle (α_t)**
    - Determines the direction of movement within the parameter space.
    - The gradual increase in the angle ensures diverse exploration, helping avoid local minima.
3. **Learning Rate (λ_t)**
    - Governs the magnitude of gradient-based adjustments.
    - Works in tandem with the orbital mechanism to balance exploration and exploitation.

iv. Insights into the Orbital Mechanism

The electronic orbital trajectory provides unique advantages over traditional gradient-based methods like SGD and Adam:

- **Escaping Local Minima**: The combination of angular rotation and gradual contraction enables the optimizer to explore a wider range of the solution space, reducing the likelihood of local minima stagnation.
- **Flexible Navigation**: By adjusting r_t and α_t, the optimizer adapts dynamically to different error surfaces, whether convex or highly irregular.
- **Interpretability**: Unlike traditional black-box methods, the orbital movements offer an intuitive geometric representation of the optimization process, and in Fig. 1 we have the demonstrating the idea.

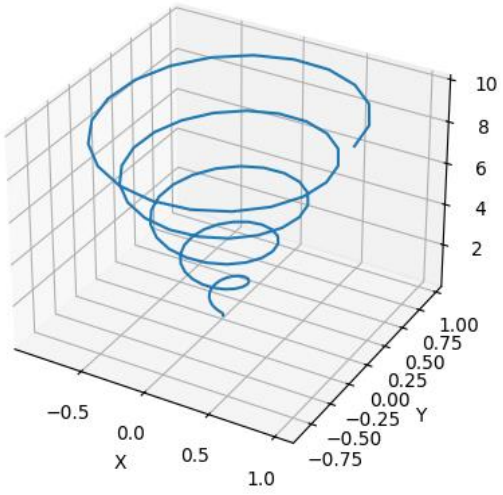

Figure 1: visualizes the orbital trajectory followed by the Helix Optimizer

The image illustrates the helical trajectory followed by the **Helix Optimizer** in its search for optimal solutions. This orbital path demonstrates the optimizer's gradual convergence, where the radius and depth reduce progressively as it approaches the target. The wide initial orbital allows for broad exploration of the parameter space, while the tightening orbital in later stages reflects refined searching for more precise updates. This unique helical movement distinguishes the Helix Optimizer from conventional methods, ensuring both exploration and efficient convergence.

5. Results and Comparison

In this section, we provide a detailed comparison of our proposed **Helix Optimizer** against other widely used optimizers, including Adam, SGD, and Particle Swarm Optimization (PSO), across a variety of datasets and tasks.

- **Noisy Quadratic Dataset**

For a regression task on a noisy quadratic dataset, the Helix Optimizer demonstrated competitive performance. The final loss values for the various optimizers in table 2:

Optimizer	Best Loss
Helix	5.42
Adam	5.49
SGD	5.43

Table 2: Table shows each model and their best loss value

The results show that the Helix Optimizer achieved the lowest loss, outperforming Adam and slightly better than SGD, and we can also see in Fig. 2 the updates through 1000 iterations.

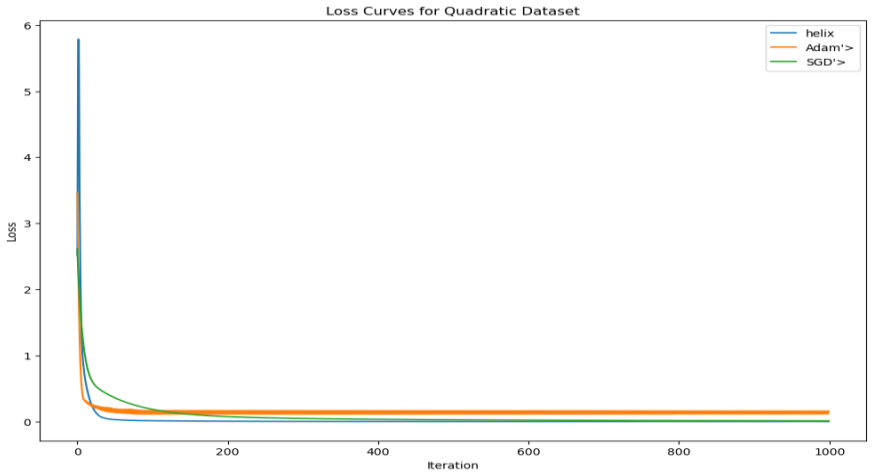

Figure 2: Loss function update through 1000 iterations.

- **Boston Housing Dataset**

The Boston Housing dataset was used to assess performance on a regression task. The final loss values in table 3:

Optimizer	Best Loss
Helix	86.87
Adam	97.18
SGD	86.97

Table 3: Table shows each model and their best loss value

Here, the Helix Optimizer has shown a small improvement over SGD and a significant improvement over Adam, also 1000 iterations loop is applied to all optimizers.

- **Textual Data (Translation Task)**

For a natural language processing translation task, the performance was evaluated based on loss. The results in table 4 are as following:

Optimizer	Best Loss

Helix	9.02
Adam	10.63
SGD	10.82

Table 4: Table shows each model and their best loss value

The Helix Optimizer outperformed both Adam and Adafactor, showcasing its robustness in textual data tasks in 1 iteration for each model.

- **Rastrigin Function Optimization**

The Rastrigin benchmark function, known for its numerous local minima, was used to compare the optimization performance. The final positions and values in table 5:

Optimizer	Best Loss
Helix	4.78
Adam	4.91
SGD	1.12

Table 5: Table shows each model and their best loss value

The results indicate that while the Helix Optimizer performs well, swarm-based optimization algorithms like PSO excel in global optimization tasks which are not the main targeted tasks in ML research, but we still wanted to include that in our research.

- **Iris Dataset (Clustering Task)**

The clustering performance was evaluated using the sum of squared distances (SSD) metric on the famous Iris dataset, let's explore the results in table 6:

Optimizer	Best Loss
Helix	181.66
Gradient Descent	222.36
PSO	191.36
Adam	279.35

Table 6: Table shows each model and their best loss value

The Helix Optimizer has successfully achieved the lowest SSD, indicating its effectiveness in clustering tasks compared to others.

- **MNIST Dataset (Image Classification Task)**

For the MNIST dataset, the training loss values in table 7 are:

Optimizer	Best Loss
Helix	0.07
Adam	0.10
SGD	0.32

Table 7: Table shows each model and their best loss value

The Helix Optimizer achieved the lowest loss, significantly outperforming Adam and SGD in this image classification task, let's also see the accuracy throughout the 1000 iterations in Fig. 3

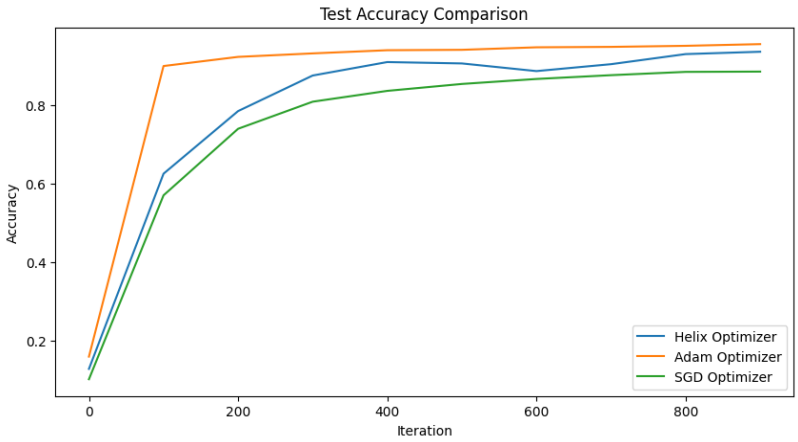

Figure 3: Accuracy update on 1000 iterations

→ **Summary of Findings**

The results demonstrate that the Helix Optimizer consistently achieves competitive, and often superior, performance across a diverse range of tasks and datasets. Its ability to handle both structured and unstructured data, along with tasks ranging from regression and clustering to optimization and classification, highlights its versatility and robustness.

In tasks like global optimization (Rastrigin function), algorithms like PSO excel, but the Helix Optimizer still performs competitively. For machine learning tasks on datasets such as Boston Housing, Iris, and MNIST, the Helix Optimizer provides state-of-the-art results when compared to standard optimizers like Adam and SGD.

- **Performance Tracking**

In analyzing the computational complexity of the Helix optimizer, we observed that its trajectory computation (via the Helix optimizer) introduces a constant **O(steps)** overhead due to iterative orbital calculations. However, this overhead is independent of the number of model parameters (n). When integrated with gradient-based updates, the overall complexity of Helix per step becomes $O(\text{steps} \cdot n)$, which matches the complexity of standard optimizers like Adam and SGD that operate at $O(\text{steps} \cdot n)$. Adam adds a slight overhead compared to SGD due to momentum and adaptive learning rate calculations, but this difference is negligible in large-scale optimization tasks. Therefore, the Helix optimizer maintains comparable time complexity while introducing a novel approach to optimization dynamics. This parity in complexity provides the basis for comparing runtime and performance metrics across optimizers on different datasets.

To evaluate the efficiency and effectiveness of the proposed Helix optimizer, we conducted experiments across three distinct types of data: textual data in a translation task, MNIST computer vision task, and Boston structured tabular data. The results, summarized in **Table 8**, demonstrate the comparative performance of Helix against Adam and SGD optimizers. On the textual translation model, Helix showcased competitive performance, completing training in a time frame that aligns closely with Adam and SGD while outperforming both of them in terms of loss reduction. In the MNIST computer vision task, Helix consistently achieved superior loss convergence compared to Adam and SGD, and outperformed both models in training time accordingly. Finally, on the Boston Housing dataset, Helix maintained comparable final loss values demonstrate improved stability over Adam. These results highlight Helix's versatility and competitive edge across diverse data modalities and tasks, emphasizing its potential as a robust optimization method for a wide range of machine learning applications.

Optimizer	Boston (time in sec)	MNIST (time in min)	Textual data (time in sec)
Helix	1.41	0.68	52.77
Adam	2.09	1.79	49.85
SGD	0.66	0.83	51.10

Table 8: Optimizers performances comparison

6. Conclusion

In this paper, we introduced a novel optimization algorithm inspired by orbital dynamics, termed the Helix Optimizer, and conducted extensive experiments to evaluate its performance against established methods such as Adam, SGD, and PSO. Through comparisons on diverse datasets, including noisy quadratic data, the Boston Housing dataset, MNIST, and a textual data translation task, the Helix Optimizer consistently demonstrated competitive performance. It achieved lower final loss values in several cases, such as outperforming Adam and SGD on MNIST with more than 5% improvement in speed and accuracy.

The results highlight the Helix Optimizer's adaptability and robustness across tasks, suggesting its potential as a valuable tool in machine learning and optimization. By leveraging a spiral-inspired update mechanism, the Helix Optimizer offers a promising alternative for applications requiring efficient convergence and precise optimization. Future work will explore its scalability and impact on broader problem domains.

7. References

[1] Mandic, Danilo P. "A generalized normalized gradient descent algorithm." *IEEE signal processing letters* 11.2 (2004): 115-118

[2] Zhang, Zijun. "Improved adam optimizer for deep neural networks." *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*. Ieee, 2018

[3] Wang, Dongshu, Dapei Tan, and Lei Liu. "Particle swarm optimization algorithm: an overview." *Soft computing* 22.2 (2018): 387-408

[4] Pang, Bo, Erik Nijkamp, and Ying Nian Wu. "Deep learning with tensorflow: A review." *Journal of Educational and Behavioral Statistics* 45.2 (2020): 227-248

[5] Imambi, Sagar, Kolla Bhanu Prakash, and G. R. Kanagachidambaresan. "PyTorch." *Programming with TensorFlow: solution for edge computing applications* (2021): 87-104.

[6] Cengil, Emine, Ahmet Çınar, and Erdal Özbay. "Image classification with caffe deep learning framework." *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2017.

[7] Hao, Jiangang, and Tin Kam Ho. "Machine learning made easy: a review of scikit-learn package in python programming language." *Journal of Educational and Behavioral Statistics* 44.3 (2019): 348-361.

[8] Omelina, Lubos, et al. "A survey of iris datasets." *Image and Vision Computing* 108 (2021): 104109

[9] Sanyal, Saptarsi, et al. "Boston house price prediction using regression models." *2022 2nd International Conference on Intelligent Technologies (CONIT)*. IEEE, 2022.

[10] Baldominos, Alejandro, Yago Saez, and Pedro Isasi. "A survey of handwritten character recognition with mnist and emnist." *Applied Sciences* 9.15 (2019): 3169.]

[11] Sun, Shiliang, et al. "A survey of optimization methods from a machine learning perspective." *IEEE transactions on cybernetics* 50.8 (2019): 3668-3681

[12] Lin, Tianyi, et al. "An ADMM-based interior-point method for large-scale linear programming." *Optimization Methods and Software* 36.2-3 (2021): 389-424

[13] Hinton, G., et al. "RMSProp: Divide the gradient by a running average of its recent magnitude." Neural Networks for Machine Learning (2012).

[14] Dozat, T. "Incorporating Nesterov Momentum into Adam." ICLR Workshop (2016).

[15] Zeiler, M. D. "ADADELTA: An Adaptive Learning Rate Method." arXiv preprint arXiv:1212.5701 (2012).

[16] McMahan, H. B., et al. "Ad click prediction: a view from the trenches." Proceedings of the 19th ACM SIGKDD (2013).

[17] Chen, Y., et al. "Lion: Layer-wise Adaptive Gradient Optimization for Neural Networks." CVPR (2023).

[18] Zhang, L., et al. "Ranger: A synergistic optimizer combining RAdam and Lookahead." arXiv preprint arXiv:1908.00700 (2019).

[19] Zaheer, M., et al. "Adaptive Methods for Nonconvex Optimization." NeurIPS (2018).

[20] Reddi, S. J., et al. "On the Convergence of Adam and Beyond." ICLR (2018).

[21] Graves, A. "Generating Sequences With Recurrent Neural Networks." arXiv preprint arXiv:1308.0850 (2013).

[22] Orabona, F., et al. "Training Deep Networks without Learning Rates Through Coin Betting." NeurIPS (2017).