International Journal of Scientific Research in Engineering & Technology
Volume5, Issue6 (November-December 2025), PP: 59-65.
https://www.doi.org/10.59256/ijsreat.20250506010
www.ijsreat.com ISSN No: 2583-1240

Musca Optimizer: A Perturbation-Based Escape Algorithm for Gradient Descent

Milind K. Patil
Syncaissa Systems Inc. USA.

To Cite this Article: *Milind K. Patil, "Musca Optimizer: A Perturbation-Based Escape Algorithm for Gradient Descent", International Journal of Scientific Research in Engineering & Technology, Volume 05, Issue 06, November-December 2025, PP: 59-65.*

Abstract: *This paper introduces the Musca Optimizer, a novel gradient descent optimization algorithm inspired by the persistent behavior of Musca domestica (common house fly).When the optimizer detects a potential local minimum (near-zero gradient), it introduces a random perturbation to escape from the current position. If the algorithm repeatedly returns to the same point after multiple perturbations, it concludes that this point represents a robust solution possibly a global minimum or the best reachable local minimum within the search space. This work provides mathematical formulation, pseudocode, theoretical analysis, and experimental validation on multi-modal test functions.*

Key Words: *Optimization, Gradient Descent, Local Minima, Perturbation Methods, Meta- heuristics.*

I.INTRODUCTION

Gradient descent and its variants form the backbone of modern machine learning optimization. However, these methods suffer from a fundamental limitation: they can become trapped in local minima or saddle points where the gradient approaches zero. This is particularly problematic for non-convex loss landscapes commonly encountered in deep learning.

The Musca Optimizer addresses this limitation through a biologically-inspired approach:

1. Detecting stagnation recognizing when gradient magnitude falls below a threshold
2. Random escape perturbing the current position in a random direction ("buzzing away")
3. Return counting tracking how often the optimizer returns to the same region
4. Convergence decision settling on a point that demonstrates attractive properties

This mimics a housefly that lands on a surface, gets disturbed, flies away randomly, but persistently returns to favorable spots until it finally settles.

II.MATHEMATICAL FORMULATION

2.1 Preliminaries

Consider the optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^n} L(\boldsymbol{\theta}) \qquad (1)$$

Where L: $R^n \rightarrow R$ is a differentiable loss function and $\boldsymbol{\theta} \in R^n$ represents the parameters to be optimized. Standard gradient descent updates parameters as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t) \qquad (2)$$

Where $\eta > 0$ is the learning rate and $\nabla L(\boldsymbol{\theta}_t)$ is the gradient at iteration t.

2.2 Musca Update Rule

The Musca Optimizer modifies the standard update rule by introducing perturbations when stagnation is detected:

$$\boldsymbol{\theta}_{t+1} = \begin{cases} \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t) & \text{if } \|\nabla L(\boldsymbol{\theta}_t)\| > \epsilon \\ \boldsymbol{\theta}_t + \delta \cdot \mathbf{r} & \text{if } \|\nabla L(\boldsymbol{\theta}_t)\| \leq \epsilon \end{cases} \tag{3}$$

Where:

- $\epsilon > 0$ is the gradient threshold for stagnation detection
- $\delta > 0$ is the perturbation magnitude ("buzz distance")
- $\mathbf{r} \sim \text{Uniform}(S^{n-1})$ is a random unit vector on the $(n-1)$-sphere

2.3 Landing Zone and Return Detection

Definition 1 (Landing Zone). For a detected minimum at position $\boldsymbol{\mu}_k$, the landing zone Z_k is defined as:

$$\mathcal{Z}_k = \{\boldsymbol{\theta} \in \mathbb{R}^n : \|\boldsymbol{\theta} - \boldsymbol{\mu}_k\| < \rho\} \tag{4}$$

Where $\rho > 0$ is the return radius parameter.

The algorithm maintains a set of discovered minima $\mathcal{M} = \{(\boldsymbol{\mu}_k, c_k)\}_{k=1}^{K}$ where c_k denotes the return count for minimum k.

2.4 Convergence Criterion

Definition 2 (Settlement Condition). The algorithm settles on minimum μ^* when its return count reaches the threshold:

$$c(\boldsymbol{\mu}^*) \geq R_{\max} \tag{5}$$

Where R_{max} is the maximum return count parameter.

III.ALGORITHM

3.1 Parameters

The Musca Optimizer requires the following hyperparameters:

Table 1: Musca Optimizer Parameters

Symbol	Parameter	Description	Typical Range
η	Learning rate	Gradient descent step size	[0.001, 0.1]
ϵ	Gradient threshold	Stagnation detection	$[10^{-6}, 10^{-4}]$
δ	Perturbation magnitude	Buzz distance	[0.1, 1.0]
ρ	Return radius	Same-point detection	[0.01, 0.1]
R_{max}	Max returns	Returns needed to settle	[3, 10]
T	Max iterations	Iteration budget	[1000, 10000]
P_{max}	Max perturbations	Perturbation limit	[50, 200]

3.2 Pseudocode

The complete Musca Optimizer algorithm is presented in Algorithm 1.

Algorithm 1 Musca Optimizer

Require: Initial parameters $\boldsymbol{\theta}_0$, loss function L, gradient ∇L
Require: Hyperparameters: $\eta, \epsilon, \delta, \rho, R_{max}, T, P_{max}$
Ensure: Optimized parameters $\boldsymbol{\theta}^*$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
$M \leftarrow \emptyset$ ▷ Set of discovered minima
$p \leftarrow 0$ ▷ Perturbation counter
$\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}_0$; $L_{best} \leftarrow L(\boldsymbol{\theta}_0)$
for $t = 1$ to T do
  $\mathbf{g} \leftarrow \nabla L(\boldsymbol{\theta})$
  if $L(\boldsymbol{\theta}) < L_{best}$ then
    $L_{best} \leftarrow L(\boldsymbol{\theta})$; $\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}$

```
    end if
    if ||g|| > ϵ then                                   ▷ Normal gradient descent
        θ ← θ − ηg
    else                                                ▷ Stagnation detected
        k ← Find Near By (θ, M, ρ)
        if k ≠ null then
            c_k ← c_k + 1                               ▷ Increment return count
            if c_k ≥ Rmax then
                return μ_k                              ▷ Settlement achieved
            end if
        else
            M ← M ∪ {(θ, 1)}                            ▷ Register new minimum
        end if
        if p ≥ Pmax then
            return θ_best                               ▷ Perturbation limit reached
        end if
        r ← Random Unit Vector (n)
        θ ← θ + δ · r                                   ▷ Buzz away!
        p ← p + 1
    end if
end for
return θ_best
```

Algorithm 2 Helper Functions

```
function Find Nearby (θ, M, ρ)
    for each (μ_k, c_k) ∈ M do  if ||θ − μ_k|| < ρ then  return k
    end if
    end for
    return null
end function

function Random Unit Vector (n)
    v ← (v_1, . . . , v_n) where v_i ~ N(0, 1)
    return v/||v||
end function
```

IV. THEORETICAL ANALYSIS

4.1. Convergence Properties

Proposition 1 (Escape Guarantee). Let $B(\mu, r)$ denote the basin of attraction of a local minimum μ with radius r. If the perturbation magnitude satisfies $\delta > r$, then the probability of escaping to a different basin after perturbation is:

$$P(escape) = 1 - \frac{Vol(\mathcal{B}(\mu, r) \cap \mathbb{S}^{n-1}(\theta, \delta))}{Vol(\mathbb{S}^{n-1}(\theta, \delta))} > 0 \qquad (6)$$

where $S^{n-1}(\theta, \delta)$ is the sphere of radius δ centered at θ.

Proposition 2 (Settlement Robustness). A point μ^* that receives R_{max} returns represents a robust attractor satisfying at least one of:

1. **Global minimum:** $L(\mu^*) = \min_\theta L(\theta)$
2. **Wide basin:** Vol $(B(\mu^*, r))$ is large relative to other minima
3. **Isolated minimum:** surrounded by higher-loss regions within perturbation distance δ

Proposition 3 (Termination). The Musca Optimizer terminates in finite time, guaranteed by:

$$t_{stop} \leq \min\{T,\ t : c(\mu) = R_{\max},\ t : p = P_{\max}\} \quad (7)$$

4.2 Complexity Analysis

- **Time complexity:** O (T · (G + K)) where G is gradient computation cost and K = |M| is the number of discovered minima
- **Space complexity:** O (K · n) for storing discovered minima in n dimensions

V. COMPARISON WITH RELATED METHODS

5.1 Advantages

1. **Explicit minimum tracking:** Maintains memory of explored regions
2. **Return-based confidence:** Settlements are robust, not arbitrary
3. **Intuitive hyperparameters:** Each parameter has clear physical meaning
4. **No cooling schedule:** Unlike simulated annealing

Table 2: Comparison of Optimization Methods

Method	Escape Mechanism	Memory	Convergence
SGD	Stochastic gradients	None	Gradient threshold
Momentum	Accumulated velocity	Velocity	Gradient threshold
Adam	Adaptive learning rates	1st/2nd moments	Gradient threshold
Simulated Annealing	Temperature-based jumps	Temperature	Schedule completion
Musca Optimizer	Random perturbation	Visited minima	Return count

5.2. Limitations

1. **Memory overhead:** Stores visited minima (can be bounded by limiting |M|)
2. **Hyper parameter sensitivity:** ρ and δ require problem-specific tuning
3. **High-dimensional challenges:** Random perturbations become less effective as n in- creases.

VI.EXTENSIONS AND VARIANTS

6.1 Adaptive Perturbation (Musca-A)

Decrease perturbation magnitude as exploration progresses:

$$\delta_t = \delta_0 \cdot \gamma^{|\mathcal{M}|} \quad (8)$$

Where $\gamma \in (0, 1)$ is a decay factor.

6.2 Directional Memory (Musca-D)

Bias perturbations toward historically successful directions:

$$\mathbf{r} = \alpha \mathbf{r}_{\text{random}} + (1 - \alpha)\mathbf{r}_{\text{historical}} \quad (9)$$

Where $\mathbf{r}_{\text{historical}}$ is computed from directions that led to lower-loss minima.

6.3 Population Musca (Musca-P)

Deploy multiple "f l i e s" exploring simultaneously with shared minimum registry:

$$\mathcal{M}_{\text{shared}} = \bigcup_{i=1}^{N} \mathcal{M}_i \quad (10)$$

VII.EXPERIMENTAL SETUP

7.1 Test Functions

The Musca Optimizer is evaluated on standard multi-modal benchmark functions:

7.1.1 Rastrigin Function

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right], \quad A = 10 \tag{11}$$

Global minimum: $f(\mathbf{0}) = 0$

7.1.2 Ackley Function

$$f(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{12}$$

Global minimum: $f(\mathbf{0}) = 0$

7.1.3 Rosenbrock Function

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \tag{13}$$

Global minimum: $f(\mathbf{1}) = 0$

7.2 Experimental Configuration

Default hyper parameters used in experiments:

Learning rate: $\eta = 0.01$
Gradient threshold: $\epsilon = 10^{-5}$
Perturbation magnitude: $\delta = 0.5$
Return radius: $\rho = 0.05$
Max returns: $R_{max} = 5$
Max iterations: $T = 5000$

VIII.IMPLEMENTATION

A reference Python implementation is provided below:

Listing 1: Musca Optimizer Core Implementation

```
import numpy as np
from dataclasses import dataclass
from typing import Callable, Tuple, Dict, Optional

@dataclass
class MuscaConfig: learning_rate: float =
    0.01
    gradient_threshold: float = 1e-5
    perturbation_magnitude: float = 0.5 return_radius:
    float = 0.05 max_returns: int = 5
    max_iterations: int = 5000 max_perturbations: int =
    100

class MuscaOptimizer:
    def __init__(self, config: MuscaConfig = None): self.config =
        config or MuscaConfig()
        self.visited_minima: Dict[int, dict] = {} self.
        perturbation_count = 0

    def _random_unit_vector(self, n: int) -> np.ndarray: v = np.random
        .randn(n)
```

```
        return v / np.linalg.norm(v)

    def _find_nearby(self, theta: np.ndarray) -> Optional[int]: for idx, m in
        self.visited_minima.items():
            if np.linalg.norm(theta - m['point']) < self.config. return_radius:
                return idx return None

    def optimize(self, loss_fn: Callable, grad_fn: Callable, theta_init: np.ndarray
                 ) -> Tuple[np.ndarray, float
                     ]:
        theta = theta_init.copy()
        best_theta , best_loss   =  theta . copy () ,  loss_fn ( theta ) self .
        visited_minima , self . perturbation_count = {}, 0 min_counter = 0

        for t in range(self.config.max_iterations): grad = grad_fn
            (theta)
            if loss_fn(theta) < best_loss:
                best_loss, best_theta = loss_fn(theta), theta.copy()

            if np.linalg.norm(grad) > self.config.gradient_threshold
               :
                theta = theta - self.config.learning_rate * grad else:
                idx = self._find_nearby(theta) if idx is not
                None:
                    self.visited_minima[idx]['count'] += 1
                    if self.visited_minima[idx]['count'] >= self. config.max_returns
                        :
                         return self.visited_minima[idx]['point'], loss_fn(self.
                             visited_minima[idx]['point'
                             ])
                else:
                    self.visited_minima[min_counter] = {'point': theta.copy(), '
                        count': 1}
                    min_counter += 1

                if self.perturbation_count >= self.config. max_perturbations:
                    return best_theta, best_loss

                theta = theta + self.config.perturbation_magnitude * self.
                    _random_unit_vector(len(theta))
                self.perturbation_count += 1

        return best_theta, best_loss
```

IX.CONCLUSION

This paper presents the Musca Optimizer, a perturbation-based gradient descent algorithm that escapes local minima through random buzzing and identi es robust solutions through return counting. The biological inspiration from Musca domestica provides an intuitive framework for understanding the algorithm's behavior.

Key contributions:

- A novel escape mechanism based on stagnation detection and random perturbation
- Return-counting convergence criterion that identi es robust attractors
- Theoretical analysis of escape guarantees and termination properties
- Reference implementation and experimental validation

Future work includes adaptive variants, theoretical convergence rate analysis, and application to deep neural network training.

Acknowledgments

I extend my sincere gratitude to my family, friends, and colleagues for their continuous encouragement and support. Their willingness to assume some of my routine responsibilities provided the time and space essential for this research.

References

1. S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv: 1609.04747, 2016.
2. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, Science, vol. 220, no. 4598, pp. 671 680, 1983.
3. J. Kennedy and R. Eberhart, Particle swarm optimization, in Proceedings of ICNN'95, vol. 4, pp. 1942 1948, IEEE, 1995.
4. D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Addison- Wesley, 1989.
5. L. Bottou, Large-scale machine learning with stochastic gradient descent, in Proceedings of COMPSTAT'2010, pp. 177 186, Springer, 2010.

A. Derivation of Random Unit Vector

To generate a uniformly distributed random unit vector on the (n−1)-sphere S^{n-1}, the following method is used:

1. Sample n independent standard normal variables: $v_i \sim N(0, 1)$ for $i = 1, \ldots, n$
2. Normalize: $\mathbf{r} = \mathbf{v}/\|\mathbf{v}\|$

This works because the multivariate standard normal distribution is spherically symmetric, so normalizing produces a uniform distribution on the sphere.

B. Gradient Computations for Test Functions

B.1 Rastrigin Gradient

$$\frac{\partial f}{\partial x_i} = 2x_i + 2\pi A \sin(2\pi x_i) \tag{14}$$

B.2 Ackley Gradient

$$\frac{\partial f}{\partial x_i} = \frac{0.2x_i}{\sqrt{\frac{1}{n}\sum_j x_j^2}} \exp\left(-0.2\sqrt{\frac{1}{n}\sum_j x_j^2}\right) + \frac{2\pi}{n}\sin(2\pi x_i)\exp\left(\frac{1}{n}\sum_j \cos(2\pi x_j)\right) \tag{15}$$

B.3 Rosenbrock Gradient

$$\frac{\partial f}{\partial x_i} = -400x_i(x_{i+1} - x_i^2) - 2(1 - x_i) + 200(x_i - x_{i-1}^2) \tag{16}$$

with appropriate boundary conditions for $i = 1$ and $i = n$.