AdaSwarm: Augmenting Gradient-Based Optimizers in Deep learning with Swarm Intelligence

Rohan Mohapatra, Snehanshu Saha, *Senior Member, IEEE,* Carlos A. Coello Coello, *Fellow, IEEE,* Anwesh Bhattacharya, Soma S. Dhavala, and Sriparna Saha, *Senior Member, IEEE*

***Abstract*—This paper introduces AdaSwarm, a novel gradient-free optimizer which has similar or even better performance than the Adam optimizer adopted in neural networks. In order to support our proposed AdaSwarm, a novel Exponentially weighted Momentum Particle Swarm Optimizer (EMPSO), is proposed. The ability of AdaSwarm to tackle optimization problems is attributed to its capability to perform good gradient approximations. We show that, the gradient of any function, differentiable or not, can be approximated by using the parameters of EMPSO. This is a novel technique to simulate GD which lies at the boundary between numerical methods and swarm intelligence. Mathematical proofs of the gradient approximation produced are also provided. AdaSwarm competes closely with several state-of-the-art (SOTA) optimizers. We also show that AdaSwarm is able to handle a variety of loss functions during backpropagation, including the maximum absolute error (MAE).**

***Index Terms*—EMPSO, AdaSwarm, gradient equivalence, neural loss function, backpropagation**

I. Introduction

Gradient Descent (GD) [1] is the most popularly used update rule, utilized heavily to update the weights associated with different layers when errors are back-propagated in shallow and deep neural networks (DNN). It is a numerical approximation method embraced in training neural networks (NN). GD is a dependable and potent method, primarily based on derivative-based optimization. The dependence on derivatives implies it is applicable only on functions differentiable everywhere in their domain. Computation of derivatives for functions endowed with complex behavior, often encountered in real-world scenarios, can be arduous. To overcome the drawbacks of GD, metaheuristic optimization based algorithms have been developed. Particle swarm optimization (PSO) [2], originally proposed in 1995, is a prominent member in the family of such metaheuristic algorithms. In an attempt to model the behaviour of bird flocks, PSO was first proposed by Kennedy and Eberhart. It is a gradient free optimizer in comparison to other optimization techniques such as GD.

PSO has been used in a wide variety of real-world applications, including: engineering optimization problems [3], [4], search for habitable exoplanets [5], [6], [7], power system operation [8], and generation of stable structures of carbon clusters [9], among many others. PSO has also been adopted in several NN applications [10], [11], [12], [13], in which it has been used to optimize and generate weights with high accuracy and low loss. For example, Grimaldi et al. [14] introduced techniques to train a NN with PSO. They conceptualized the particles comprising the weights of the NN and devised a way of reducing the loss in training by an equivalence conjecture between the fitness function and the loss function. While this approach is reasonable to apply in NN with limited number of weights, the particle dimension is linear in the number of layers of the network. If we consider a real-world application of a Convolutional Neural Network (CNN) [15], [16], the number of weights would be in millions. In such large-scale problems, PSO is not expected to be competitive with respect to classical optimizers. However, instead of particles being weights, a theoretical correspondence between PSO and the error gradient, supported by empirical validation, will enable us to mitigate the curse of dimensionality. This is equivalent to establishing facts mathematically that the gradient of a function can be approximated from the knowledge of the values of the hyperparameters of PSO. This will allow us to part away with actual derivative computation, especially for functions with jagged landscapes. Using the approximate gradients, we can then emulate gradient optimization techniques and apply in different types of NNs. In this regard, AdaSwarm is proposed to show that a "gradient"-free optimizer[1] can provide results that are competitive with respect to the best optimizers currently available.

Manuscript received September 27, 2020; revised February 6, 2021 and March 29, 2021; accepted April 26, 2021. Snehanshu Saha would like to thank the SERB-DST, Govt.of India [SERB-EMR/ 2016/005687] and RIG from BITS Pilani K K Birla Goa Campus for supporting this research and Carlos A.C. Coello acknowledges support from CONACyT project no. 1920 and from a 2018 SEP-Cinvestav grant (application no. 4). This work was supported by Basque Government through the BERC 2018-2021 program by the Spanish Ministry of Science. Dr. Zexuan Zhu. *(Corresponding author: Snehanshu Saha.)*

Rohan Mohapatra is with the Center for AstroInformatics (Astrirg), Bengaluru 560085, India (e-mail: rohan.ron14@gmail.com).

Snehanshu Saha is with the Department of Computer Science and Information Systems and APPCAIR, BITS Pilani, Pilani, India, and also with K K Birla Goa, Zuarinagar 403726, India (e-mail: scibase.snehanshu@gmail.com).

Carlos A. Coello Coello is with the Department of Computer Science, CINVESTAV-IPN, Mexico City 07360, Mexico, and also with the Basque Center for Applied Mathematics (BCAM), Ikerbasque, Spain (e-mail: ccoello@cs.cinvestav.mx).

Anwesh Bhattacharya is with the Department of Physics and CSIS, BITS Pilani, Pilani 333031, India (e-mail: f2016590@pilani.bits-pilani.ac.in).

Soma S. Dhavala is with ML Square, Bengaluru 560038, India (e-mail: soma. dhavala@gmail.com).

Sriparna Saha is with the Department of Computer Science and Engineering, Indian Institute of Technology Patna, Patna 800013, India (e-mail: sriparna.saha@gmail.com).

This article has supplementary downloadable material available at https://doi.org/10.1109/TETCI.2021.3083428, provided by the authors.

Digital Object Identifier 10.1109/TETCI.2021.3083428

[1]Gradient-free in this context means using the approximate gradient values instead of the analytically derived values

A. Research Background

1) PSO with Inertial Weight: The primary objective of PSO was to solve optimization problems with several objectives (*devoid of exact, analytical optima*). PSO navigates the search space by embracing a *swarm* which is nothing but a population of particles. The swarm, guided by characteristic equations, attempt to converge to an optima [2]. The movement of the particles in the search space to discover the optimal solution is governed by the velocity and the position update equations. These are represented in the following manner:

$$v_i^{(t+1)} = \phi v_i^{(t)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)}) \quad (1)$$

$$x_i^{(t+1)} = x_i^t + v_i^{(t+1)} \quad (2)$$

where $\phi, c_1, c_2 \geq 0$. Here, x_i^t suggests the position of particle i at time t, v_i^t represents the velocity of particle i at time t, p_i^{best} is the best position particle i has attained, and g_i^{best} is the best position that the swarm has ever attained. Additionally, parameters related to cognitive learning and social learning need to be defined, which regulate the position and velocity of the particles and they are c_1, c_2 respectively; r_1, r_2 are random values sampled from $U(0, 1)$ which contributes to the stochastic nature of the search process.

2) PSO with Momentum: Backpropagation algorithms often use a momentum term to steer away from stagnating local minima. This is typically done by introducing a momentum term. Inspired by this, the velocity update equation of PSO can accommodate momentum to handle the stagnation problem. The standard PSO equations are embellished with the momentum term, and the resulting algorithm is called MPSO[17].

$$v_i^{(t+1)} = \left[v_i^{(t)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)}) \right] \times (1 - \lambda) + \lambda v_i^{t-1} \quad (3)$$

Here, λ denotes the momentum factor.

B. Motivation

The sources of inspiration to understand and improve optimization techniques in general, and PSO in particular, come from several places. Said et al. [18] postulated that swarms behave similarly to classical and quantum particles. In fact, their analogy is so striking that one may think that the social and individual intelligence components in PSO are, after all, nice useful metaphors, and that there is a neat underlying dynamical system at play. This dynamical system perspective was indeed useful in unifying two almost parallel streams, namely, optimization and Markov Chain Monte Carlo sampling [19], [20]. In a seminal paper, Wellington and Teh [19], showed that a stochastic gradient descent (SGD) optimization technique can be turned into a sampling technique by just adding noise, governed by Langevin dynamics. Recently, Soma and Sato [21] provided further insights into this connection based on an underlying dynamical system governed by stochastic differential equations (SDE). While these results are new, the connections between derivative-free optimization techniques based on Stochastic Approximation and Finite Differences are well documented [22]. Such strong connections between these seemingly different subfields of optimization and sampling made us wonder: Is there a larger, more general template of which the aforementioned approaches are special cases? SGD (SGD) [23] was proposed to tackle the slow speed of convergence of the GD method and also to improve the minimization of the underlying optimization problem. However, this approach is not good enough in many cases. What are we missing and what is the role of metaheuristics in this framework? This particular question gives rise to two issues: can we use ideas developed for improving SGD (SGD) and apply them to PSO? Also, can we use the particles' history and their location awareness to offer derivative-free approximations to the gradients? In this paper, we answer these questions in the positive.

C. Summary of Contributions

Section II proposes EMPSO, a PSO variant aided with exponentially averaged momentum, contributing to reduced number of iterations required to minimize the error and reach the optima for a variety of benchmark (*differentiable and non-differentiable*) test functions (**Section III**, Tables I, II and III) and for several standard single-objective test functions used for global optimization (see the supp file[2], Section 8). EMPSO is markedly different from the existing M-PSO approach (**Section II** *where the exploration and the exploitation terms are used in an equal manner*). Section III presents the mathematical formulation of EMPSO as well as a proof of the theorems establishing mathematical equivalence between Vanilla PSO & GD (**Section III-A**) —, and between EMPSO & SGD with Momentum (**Section III-B**). Such a theoretical equivalence complements our empirical results on the application of EMPSO to both differentiable and non-differentiable test functions (**Section III, Tables I, II and III**) as well as to standard single-objective test functions (*see the supp file, Section 8*). The Theorems establish a direct gradient approximation method with precise alternatives to gradient computation by exploiting the hyperparameters of the Vanilla and EMPSO. Since our proposed approach is an approximation method, we discuss its order of accuracy in the same section briefly and present a proof for it (*see Section 1.1 in the supp file*). Next, we interpret and extend the gradient approximation approach in **Section IV** to emulate backpropagation in Vanilla feed-forward NN and CNN via EMPSO, tested on a simulated data set[3] (*see* **Section IV** *and* Fig. 1a). Results in **Sections III and IV** pave the foundation for a novel adaptive gradient-free optimizer, which we call AdaSwarm. This approach is intended for optimizing NNs (**Section V**) and we show that it produces promising results for multiple classification data sets (**Tables IV and V in the paper and Table 5** in the supp file). **Section VI** reviews differentiable and non-differentiable loss functions adopted in NNs. In order to facilitate training using high dimensional classification data, we present a rotational variant of EMPSO, called Rotation Accelerated EMPSO (REMPSO) (*see Section 5 of the supp file*) which is intended for computer vision data

[2]supp file = Supplementary Material

[3]Link for simulated data set: https://gist.github.com/rohanmohapatra/4e7bce4f0d95746d8b993437c257e99b

sets (*see Section 3 of the supp file*). **Section VI** contains the subtle technical aspects of AdaSwarm and reports the results of our experiments on several test functions commonly used in optimization [24] as well as on data sets used in complex classification tasks in deep learning. We conclude in **Section VII** by highlighting possible extensions to AdaSwarm.

II. Our Contribution: Exponentially weighted Momentum PSO (EMPSO)

We begin by proposing a novel variant, Exponentially weighted Momentum PSO [25], to momentum PSO (MPSO) by assigning greater emphasis to the exploration phase of PSO. The computed weighted average in Momentum particle swarm optimization (M-PSO) algorithm [17] contributes to exploration and exploitation simultaneously. Since finding the optima in the search space via PSO is tied to exploration mostly, the exploration part of the PSO equation would benefit if greater weights are assigned to it.

We formulate a strategy to mitigate the issues MPSO faces by presenting a variant of Particle Swarm Optimizer with momentum. The mathematical representation of EMPSO is as follows:

$$v_i^{(t+1)} = M_i^{(t+1)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)}) \quad (4)$$

where,

$$M_i^{(t+1)} = \beta M_i^t + (1-\beta) v_i^{(t)} \quad (5)$$

We use β to indicate the momentum factor, and M_i^t is the momentum of a particle. The combination of (4) and (5) yields a new representation of the mathematical formulation of EMPSO.

$$v_i^{(t+1)} = \beta M_i^t + (1-\beta) v_i^{(t)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)}) \quad (6)$$

Exploration and exploitation are two important phases for any optimization algorithm. Exploration is done by the algorithm to search for other possible solutions apart from the current solution in all of the search space of the problem (*i.e., finding diverse solutions is preferred at early stages of the search*). On the other hand, exploitation is to search for similar and better solutions around the current solution for higher precision (*i.e., preferable when the search is close to finish*). PSO, as any other metaheuristic, operates with these two phases [26]:

$$\underbrace{v_i^{(t)}}_{\text{Exploration}} + \underbrace{c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)})}_{\text{Exploitation}}$$

The exploration phase, due to the new approach, is now determined by the **exponential weighted average of the historical velocities** only. The negligible weights in MPSO do not contribute to the required acceleration [17]. The momentum in EMPSO is the exponential collection of velocities that the particles have seen so far, over time. We accumulate the velocities by multiplying by an exponential factor β, as we move ahead in time. Thus, β factor assigns more weight to the recent velocity than to older velocities.

A. Intuition behind Exponentially Weighted Average

Let us recursively expand (5) for further intuition on exponentially averaged momentum. Let's begin with $M_i^{(t)} = \beta M_i^{(t-1)} + (1-\beta) v_i^{(t-1)}$ which is expanded easily to $M_i^{(t)} = \beta^2 M_i^{t-2} + \beta(1-\beta) v_i^{t-2} + (1-\beta) v_i^{t-1}$. Generalizing —

$$M_i^{(t)} = \beta^n M_i^{(t-n)} + \beta^{(n-1)}(1-\beta) v_i^{(t-n)} + \\ \ldots + \beta(1-\beta) v_i^{(t-2)} + (1-\beta) v_i^{(t-1)} \quad (7)$$

From the above equation, we can see that the value of the t^{th} term of the momentum is dependent on all the previous values of the velocities. The $(t-i)^{th}$ time step is assigned the weight $\beta^i(1-\beta)$. Since $\beta < 1$, we have $\beta^{i+1} < \beta^i$and the associated weights decrease rapidly. Thus, older velocities get a much smaller weight and, therefore, contribute less to the overall value of the Momentum. Since the velocities are a cumulative sum, storing the velocity history of a particle is not required.

B. EMPSO Algorithm

Algorithm 1: EMPSO

```
for each particle i in swarm S do
    initialize particle with feasible random position;
    evaluate the fitness F_i of particle;
endfor
while termination condition is not fulfilled do
    for each particle i in swarm S do
        v_i = βM_i + (1−β)v_i + c1 r1 (p_i^best − x_i) +
          c2 r2 (g^best − x_i);
        x_i = x_i + v_i;
        M_i = βM_i + (1−β)v_i;
    endfor
    update p_i^best, g^best by finding the best fitness;
    choose best solution;
end
```

The parameters adopted are the following:

- S: Swarm containing a certain number of particles
- x_i: Position of Particle *i*, β: Momentum factor
- v_i, M_i: Velocity/Momentum of Particle *i*
- c_1, c_2: Cognitive/Social Learning Parameter
- r_1, r_2: Random values between [0,1]
- p_i^{best}: Personal best velocity of Particle *i*
- g^{best}: Global best velocity observed in Swarm S

Complexity Analysis of EMPSO: The EMPSO algorithm has a complexity of $\mathcal{O}(t \times n \times log(n))$. Considering the worst case of finding the maximum, EMPSO at the worst case has a computational complexity $\mathcal{O}(t \times n^2)$. This computational cost is relatively low because it is linear in t. The number of particles, n is normally a relatively small number in PSO (*See Section 7 of the supp file for detailed calculations*).

III. Approximation of Gradients: Differentiable/Non-Differentiable Functions

We present three theorems establishing approximate gradients for any arbitrary (*loss*) function, dependent on the

PSO/EMPSO parameters. The theorems thus could be applied to either differentiable or non-differentiable loss functions in training DNNs.

A. Proof of Equivalence between the GD and a Vanilla PSO to approximate gradients for Differentiable functions

Theorem - Under reasonable assumptions, for an arbitrary objective function $f(w)$ (*differentiable or not*), if PSO converges at $w = g^{best}$, the gradient approximation theorem states that $\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(w - g^{best})$

Proof - First consider a single-dimensional objective $f(x)$. One can expand it as a Taylor-series about $x = a$ as

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \\ \ldots + E_n(x;a) \quad (8)$$

where $|E_n(x;a)| = \frac{k(x-a)^{n+1}}{(n+1)!}$ is the error of the approximation. We can compute the first derivative as —

$$f'(x) \;=\; f'(a) + f''(a)(x-a) + \ldots + E_{n-1}(x;a) \quad (9)$$

Analogously, for a multidimensional objective $f(w)$, centering its Taylor-expansion about $w = \mathbf{w'}$, one can express —

$$\frac{\partial f}{\partial w} = f'(\mathbf{w'}) + f''(\mathbf{w'})(w - \mathbf{w'}) \\ + \ldots + E_{n-1}(w;\mathbf{w'}) \quad (10)$$

Strictly speaking, $f' = \nabla_w f$ and $f'' = \frac{\partial^2 f}{\partial w_i \partial w_j}$ with w_i ranging over the dimensions of the weights. The slight abuse of notation lends to better understanding of the theorem. The weight update rule of GD (*used in backpropagation to train NNs*) is[4] —

$$w^{(t+1)} = w^{(t)} + \eta \left.\frac{\partial f}{\partial w}\right|_{w^{(t)}} \quad (11)$$

At the timestep with weight $w^{(t)}$, the Taylor-expansion of the derivative (eq 10) [$w^{(t)}$ is the weight matrix of the NN, η is the learning rate and f is the loss function for the given network] could be substituted in the GD update and written as —

$$w^{(t+1)} - w^{(t)} = \eta[f'(\mathbf{w'}) + f''(\mathbf{w'})(w^{(t)} - \mathbf{w'}) \\ + \ldots + E_{n-1}(w^{(t)};\mathbf{w'})] \quad (12)$$

The purpose of GD is to find the minimum of an objective. However, the same minima could be found by PSO if we let $x \equiv w$ i.e., the particle dimensions are that of the weights of the NN. The PSO position update equation could be put into the form —

$$\begin{aligned} & x^{(t+1)} - x^{(t)} \\ &= v^{(t+1)} \\ &= \omega v^{(t)} + c_1 r_1 (p^{best} - x^{(t)}) + c_2 r_2 (g^{best} - x^{(t)}) \end{aligned} \quad (13)$$

For the corresponding PSO search process with $x^{(t)}$, the particle trajectory, prior to convergence, takes vanishingly small steps and $v^{(t)} \approx 0$. The swarm satisfies $p^{best} \approx g^{best}$ near convergence [2]. The point **w'**, about which the Taylor expansion is done, is constructed such that $g^{best} = \mathbf{w'}$ after convergence. The error term $E_{n-1}(w;\mathbf{w'})$ could be ignored for practical purposes. Moreover, if **w'** is a minima $\implies f'(\mathbf{w'}) = 0$. We make the following assumptions before asserting the equivalence —

1) $w^{(t)}$ is inside a small, $\delta-$neighborhood centred about **w'**
2) $\eta = \omega$ for mathematical convenience (*not required for the minima condition, near or far from the minima in a NN setting, $f'(\mathbf{w'}) = 0$ irrespective of this setting*)
3) $x \equiv w$ i.e., x, w are used interchangeably; x in PSO is used to update particle position while w in a neural-network setting, is used to update weights.

Under the above assumptions and settings, it is possible to equate the RHSs of eq (12) and (13) to obtain :

$$\eta f''(\mathbf{w'})(w^{(t)} - \mathbf{w'}) = c_1 r_1 (p^{best} - x^{(t)}) + c_2 r_2 (g^{best} - x^{(t)}) \\ \implies f''(\mathbf{w'})(w^{(t)} - \mathbf{w'}) = -\frac{(c_1 r_1 + c_2 r_2)(w^{(t)} - g^{best})}{\eta}$$

since $p^{best} \approx g^{best}$, $x \equiv w$ and $g^{best} \approx \mathbf{w'}$. Thus, we obtain

$$f''(\mathbf{w'}) = -\frac{(c_1 r_1 + c_2 r_2)}{\eta}; \quad (14)$$

Upon substituting eq (14) back into eq (9) along with $f'(\mathbf{w'}) = 0$ and $g^{best} \approx \mathbf{w'}$, we finally arrive at the gradient equivalence theorem —

$$\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(w - g^{best}) \quad (15)$$

Instead of the true gradient, it is now possible to use the approximated gradient evaluated at $w^{(t)}$ for running GD. This is called gradient-descent with PSO approximated gradient.

It is possible that assumption 1 above does not hold. In such cases where we are searching the search space and the points are farther away from the **optima**, the error will be larger. The p^{best} will factor in for points away from the minimum and hence the equivalence can also be written as —

- $v_i^{(t)} = f'(\mathbf{w'}) + E_{n-1}(w)$
- $f''(\mathbf{w'}) = -\frac{[c_1 r_1 (w^{(t)} - p^{best}) + c_2 r_2 (w^{(t)} - g^{best})]}{\eta (w^{(t)} - \mathbf{w'})}$
- $\frac{\partial f}{\partial w} = v_i^{(t)} - \frac{c_1 r_1 (w^{(t)} - p_i^{best}) + c_2 r_2 (w^{(t)} - g^{best})}{\eta}$

Next, we extend the equivalence to a more advanced version of the GD that provides a faster convergence and avoids stagnation at local minima.

B. Proof of Equivalence between the SGD with Momentum and EMPSO to approximate gradients for Differentiable (loss) functions

Theorem: Under reasonable assumptions in the neighborhood of the minima, the following equivalence holds: $\eta = (1 - \beta)$, $f''(\mathbf{w'}) = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}$ and $\alpha = \beta$

Proof: Please see Section 1 in the supp file for details.

[4] See section (IV-A) for a derivation

C. A Subtlety Regarding Non-Differentiable Loss Functions

Let's assume a continuous, bounded function $f(x)$ that is not differentiable. It can be expressed as $f(x+\alpha)$, where α is a shift parameter with an initial value equals to zero. Let us also define $z \equiv x+\alpha$, so that $f(z) \equiv f(x+\alpha)$. To define the Taylor series for such a function, f must be differentiable with respect to α, and consequently f must be differentiable with respect to z. When a function is shifted by a parameter α, we can see that the function becomes differentiable in that interval. Alghalith [27] introduced differentiability with respect to a shift parameter rather than the variable because a small change in the shift parameter is a constant. Since f is differentiable with respect to z, the Taylor series expansion around a can we written as:

$$f(z) = f(a) + f'(a)(z-a) + \frac{f''(a)}{2!}(z-a)^2 + ... + E_n(z,a)$$

Let's consider a non-differentiable function, $f(x)$. If it is expressed by using a shift-parameter α, then replacing $x+\alpha$ by z, we can make the function differentiable under z (*as every function, differentiable or not can be shifted*). For details, please refer to Moawia Algalith's paper [27]. The taylor series expansion, in z, remains the same as before. Then, the gradient of this non-differentiable function can be expressed with PSO parameters in the same way as for a differentiable function.

$$f'(z) = 0 + f'(a) + f''(a)(z-a) + \frac{f'''(a)}{2!}(z-a)^2 + ... + E_{n-1}(z,a)$$

The GD Weight Update with momentum is given by:

$$w^{(t+1)} = w^{(t)} + \eta V_{dw}^{t+1}$$

where,

$$V_{dw}^{t+1} = \beta V_{dw}^{t} + (1-\beta)\frac{\partial f}{\partial w} \tag{16}$$

Combining the equations and dividing equation (16) by $(1-\beta)$ we obtain: $w^{(t)} = w^{(t-1)} + \alpha V_{dw}^{t-1} + \eta \frac{\partial f}{\partial w}$

Here, η is the learning rate and V_{dw}^{t-1} is the momentum applied to the weight update, where, $\alpha = \frac{\eta\beta}{(1-\beta)}$

We apply the Taylor series expansion for non-differentiable functions to the gradient and then formulate the GD with momentum as follows:

$$w^{(t)} = w^{(t-1)} + \eta f'(\mathbf{w}') + \eta f''(\mathbf{w}')(z-\mathbf{w}') + E_{n-1}(z) + \alpha V_{dw}^{t-1} \tag{17}$$

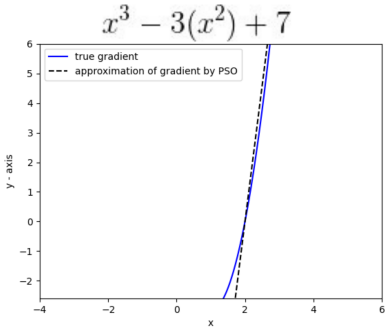
Under the same assumptions stated in the previous proofs, we define the equivalence as follows: $\eta = (1-\beta); \alpha = \frac{\eta\beta}{(1-\beta)}$

$$f''(\mathbf{w}') = -\frac{(c_1 r_1 + c_2 r_2)}{\eta} \tag{18}$$

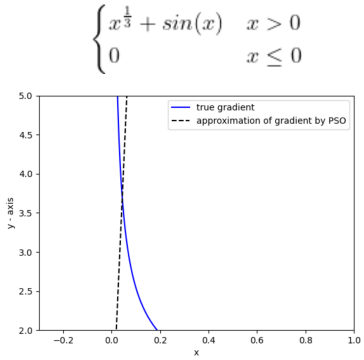
Finally, the gradient for non-difererentiable functions can be expressed as follows:

$$\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(z - g^{best})$$

Section 1.2 in the supp file for a more complete treatment of this case.



(a) True Gradient *vs.* Approximate Gradient calculated using PSO Parameters: The approximation accomplished is evidence of the efficacy of EMPSO and of the order of accuracy proof presented in Section III-D



(b) True Gradient *vs.* Approximate Gradient calculated using PSO Parameters for non-differentiable functions

Fig. 1: True Gradient *vs.* Approximate Gradient

D. Order of Accuracy: EMPSO's gradient approximation

The approximation of the exact gradient value depends on a small parameter h (*say the grid size or time step*). The order of accuracy (See the proof in Section 1.1 of the supp file) was found to be higher near the optimum. We consider two types of functions: smooth functions (*differentiable everywhere*) and non-differentiable functions (*finite number of singularities*). We compute approximate gradients and evaluate approximate optima of these functions using EMPSO. The promising results from this approximation, as theoretically proved, are experimentally validated. We observed a reasonable degree of accuracy between the optima and the approximations produced by EMPSO. The approximation holds for points away from the optima (**See Table 4 in supp file**). The graphs (see Figure 1) support our assertion. The numerical results, tabulated in Tables I, II, III reflect this observation. These experiments suggest the need for investigating the possibility of exploiting mathematical equivalence of gradients in the backpropagation algorithm in the context of DNNs (Sections IV- VII).

TABLE I: **Differentiable Functions** : GD *vs.* PSO Emulated GD using $\eta = 0.1, c_1 = 0.8, c_2 = 0.9$

Function	Global Minimum	GD Optimum	Iterations	Emulated GD	Iterations
$-\left(3x^5 - x^{10}\right)$	-2.25	2.6	31	-2.249	9
$x^3 - 3(x^2) + 7$	3	3.00	17	3.00	11
$-\exp\left(\cos\left(x^2\right)\right) + x^2$	-2.718	-2.718	349	-2.718	9
$x^{15} - \sin(x) + \exp\left(x^6\right)$	0.4747	2.6	548	0.4747	11

TABLE II: **Differentiable Functions**: SGD with Momentum *vs.* EMPSO Emulated GD $\eta = 0.1, c_1 = 0.8, c_2 = 0.9, \beta = 0.9$

Function	Global Minimum	SGD Optimum	Iterations	Emulated GD	Iterations
$-\left(3x^5 - x^{10}\right)$	-2.25	-1.45	667	-2.249	13
$x^3 - 3(x^2) + 7$	3	3.0	71	3.00	16
$-\exp\left(\cos\left(x^2\right)\right) + x^2$	-2.718	-2.718	49	-2.706	8
$x^{15} - \sin(x) + \exp\left(x^6\right)$	0.4747	0.679	66	0.4747	6

TABLE III: **Non-differentiable Functions** : SGD with Momentum *vs.* EMPSO Emulated GD using $\eta = 0.1, c_1 = 0.8, c_2 = 0.9, \beta = 0.9$

Function	Global Minimum	SGD Optimum	Iterations	Emulated GD	Iterations
$\sqrt{x}$	0	NA	NA	0.01	22
$\lvert x \rvert$	0	0.1	1001	0.03	21
$\begin{cases} x^{\frac{1}{3}} + sin(x) & x > 0 \\ 0 & x \leq 0 \end{cases}$	0	0.14	13	0.004	21
$\begin{cases} x & x > 0 \\ x^2 & x < 0 \\ 0 & x = 0 \end{cases}$	0	0.001	50	0.04	20

IV. Interpretation of Gradients and Emulating Back propagation in NNs

Let us revisit the GD rule proposed in the backpropagation algorithm for vanilla feed forward networks:

$$w^{(t+1)} = w^{(t)} + \eta \frac{\partial f}{\partial w}$$

The equivalence between GD (GD) and Particle Swarm Optimization (PSO) has been proved in the previous section. We can now substitute the gradient with PSO parameters and approximate the gradient values. Computing the gradient for some functions might be a very difficult task and some loss functions are non-differentiable. If we can get a good approximation, then we can easily emulate the GD.

Let's consider at any iteration, that the $c_1 r_1$ and $c_2 r_2$ values are computed by taking the average of n particles, and let's consider the velocity $v_i^{(t)}$ of the particle that influences the g^{best} whose position is $x^{(t)}$

Using the PSO parameters, we get the following equation:

$$w^{(t+1)} = w^{(t)} + \eta[v_i^{(t)} - \frac{c_1 r_1 (x^{(t)} - p^{best}) + c_2 r_2 (x^{(t)} - g^{best})}{\eta}] \quad (19)$$

Tables I, II, III and Fig. 1 show the emulation using approximate gradients. For training a NN, PSO has been extensively used [11], [12], since the GD has a higher chance of getting stuck in local minima. PSO's particles encode the weights of the NN and the corresponding fitness function is the loss function that has to be reduced. The caveat in this approach is that as the number of weights increase, and so will the number of dimensions of a single particle. Thus, PSO will eventually become a victim of the "curse of dimensionality", since it will fail to converge when dealing with a very large number of weight updates.

We propose a novel idea to train the NN. We use the concept of **batch** training and the dimensionality of the particles vector is now just **batch_size** $\times$ **no_of_classes**, thus significantly reducing the amount of computations and increasing the training speed with respect to previous approaches. In a NN, we essentially have a loss function that has to be minimized in order to obtain the set of weights required to classify a given instance. Traditionally, the backpropagation algorithm would do an update as follows: for different loss functions, the Error Gradient would look like (where y and $\hat{y}$ are the predicted label and the true label, respectively):

- Mean Square Error (MSE) Loss: $\frac{1}{2}(y - \hat{y})^2$

$$\frac{\partial E}{\partial w} = (y - \hat{y}) \times D(activation) \times x \quad (20)$$

- Binary Cross Entropy (BCE) Loss: $-ylog(\hat{y}) - (1 - y)log(1 - \hat{y})$

$$\frac{\partial E}{\partial w} = \left(\frac{1 - \hat{y}}{1 - y} - \frac{\hat{y}}{y}\right) \times D(activation) \times x \quad (21)$$

- Mean Absolute Error (MAE) Loss: $|y - \hat{y}|$

$$\frac{\partial E}{\partial w} = \begin{cases} +1 \times D(activation) \times x & y > \hat{y} \\ -1 \times D(activation) \times x & y < \hat{y} \end{cases} \quad (22)$$

The common equation in all of these losses, is that the neural update rule is given by:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y} \times D(activation) \times x \quad (23)$$

The equivalence between the error gradient and the EMPSO approximation is natural for any loss function but it's a non-trivial consequence of the insights gained by the theorems proved in Section III. If a derivative can be expressed mathematically using PSO and the proposed EMPSO parameters, it is expected that the error gradient whose minimum is critical to compute in the backpropagation algorithm, can be approximated using PSO's parameters. This is a clear departure from existing heuristic-based approaches.

If we can replace $\frac{\partial E}{\partial y}$ by an approximate value of the gradient, then we can use this gradient for any loss function (*differentiable/non-differentiable*). Using the theorems, we can then adopt the following weight update rule in NNs:

$$\frac{\partial E}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta} (g^{best} - y) \times D(activation) \times x \quad (24)$$

where $D(activation)$ is the derivative of the activation function adopted. Using this approximation, we are able to descend to the minimum at a faster rate. From Fig. 2 it can be clearly seen that the descent in the slope is steeper than when using the traditional gradient approach.

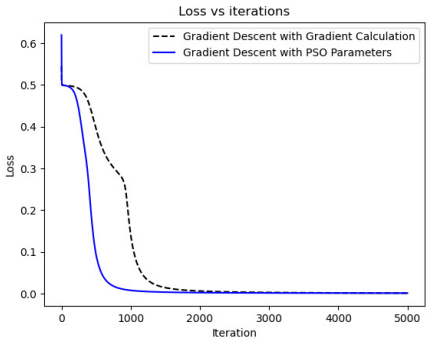


Fig. 2: Error Gradient approximation in backpropagation by EMPSO on simulated data, as proposed in Eq. (24):

For MAE, using this loss function we can clearly see that when $y = \hat{y}$, then the MAE derivative is undefined, but when we replace the gradient with the approximate gradients, we do not have to worry about $y = \hat{y}$. In this case, the approximate gradient would be zero and will not update the weight, which is expected. Therefore, we can say that the gradient can be approximated for any differentiable/non-differentiable loss function. This allows the use of this technique in each and every case.

A. Backpropagation and Error Gradient Equivalence in Vanilla Feed-Forward NNs

Consider a node in the final layer of a NN. It has inputs x_i which are the activations of the nodes of the previous layer. The term net_j is a linear sum $net_j = \sum_i w_{ji}x_i + b_j$, and y_j is the result of a univariate activation function σ, acting on net_j, leading to $y_j = \sigma(net_j)$. E is a multivariate loss function with y_j as one of its inputs. Applying the chain rule of derivatives, we get —

$$\begin{aligned}\frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_{ji}} \\ &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \\ &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j}x_i\end{aligned}$$

where y_j is the output activation. By approximating $\frac{\partial E}{\partial y_j}$ with EMPSO, where E could be a non-differentiable loss function, we have the following equivalence (*the detailed proof can be found in Section 2 the supp file*):

$$\frac{\partial E}{\partial y_j} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta} \tag{25}$$

This relation is loss function-agnostic. For background on the backpropagation equations and the notation used, please refer [28].

V. AdaSwarm

We propose a fast, gradient-free optimizer called AdaSwarm. The Adam [29] optimizer, is a very popular optimization algorithm used extensively in NNs, requires first-order derivatives and has little memory requirements. The Adam optimizer calculates adaptive learning rates for different parameters from the estimates of second and first moments of the gradients. AdaSwarm is based on the Adam optimizer but replaces the gradient with the computed, approximate gradients via the equivalence theorems, replacing these gradients in Adam's update rule. Experimental results show that AdaSwarm has a lower execution time and comparable (*and most times superior*) performance to Adam.

Algorithm 2: AdaSwarm

Require: η: Learning Rate;
Require: β_1, $\beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates;
Require: $f(\theta)$: Function with parameter θ;
Require: θ_0: Initial parameter vector;
$m_0 \leftarrow 0$ (Initialize 1st moment vector);
$v_0 \leftarrow 0$ (Initialize 2nd moment vector);
$t \leftarrow 0$ (Initialize timestep);
while θ_t *not converged* **do**
  $t \leftarrow t + 1$;
  $g_t \leftarrow$ Approximates Gradients (get gradients w.r.t. stochastic objective at timestep t);
  $m_t \leftarrow \beta_1 \cdot m_{t-1}(1 - \beta_1) \cdot g_t$;
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$;
  $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{m_t}{\sqrt{v_t}+\epsilon}$
Return θ_t

Adam is a combination of SGD with momentum and RMSProp. It leverages the momentum by using the moving average of the gradient instead of the gradient itself like in SGD (SGD) with momentum and the squared gradients are used to scale the learning rate like in RMSProp. Besides these capabilities, when we add approximate gradients calculated using EMPSO (*A fast converging Particle Swarm Optimizer*), this approach becomes a truly derivative-free optimizer. AdaSwarm is thus a combination of RMSProp, SGD with Momentum and EMPSO with the aim of providing speed and acceleration to train NNs.

A. Convergence of AdaSwarm

As there is no meaningful size of a function for the optimization tasks (*convex/non-convex*), in contrast to the GD algorithm for convex problems which finds the location of the minimum of a multi-variable function, we cannot clearly describe the computational complexity of AdaSwarm. However, there is an ample amount of literature that supports the convergence of Adam [30]. Since AdaSwarm is based on the Adam optimizer, the algorithm is similar in structure and therefore we can infer whether it converges or not, based on the upper bounds for convergence rate of Adam.

TABLE IV: AdaSwarm *vs.* other optimizers: The promising results of AdaSwarm using MSE (*Differentiable Loss Function*) could address the sensitivity (*to initialization*), and robustness (*to multiple local minima*) in the classification dataset

Dataset	**Metrics**	**Optimizer**							
		SGD	Emulation of SGD with PSO parameters	AdaGrad	AdaDelta	RMSProp	AMSGrad	Adam	AdaSwarm
Iris	Loss	0.072	0.073	0.108	0.123	0.049	0.094	0.05	**0.062**
	Accuracy	92%	96.66%	88.22%	88.22%	98.22%	88.22%	98.22%	**98.22% ±0.38%**
Ionosphere	Loss	0.238	0.5	0.064	0.072	0.012	0.0167	0.0156	**0.022**
	Accuracy	68.14%	95%	93.71%	92.57%	98.43%	98.14%	98.57%	**98.48% ±0.28%**
Wisconsin Breast Cancer	Loss	0.198	0.124	0.127	0.1234	0.116	0.116	0.121	**0.119**
	Accuracy	80.79%	85.11%	84.20%	84.53%	85.34%	85.33%	84.82%	**85.55% ±0.74%**
Sonar	Loss	0.251	0.132	0.144	0.142	0.117	0.112	0.108	**0.11**
	Accuracy	50%	81.00%	82.69%	81.97%	84.37%	86.05%	86.30%	**86.28% ±0.34%**
Wheat Seeds	Loss	0.219	0.2	0.162	0.203	0.119	0.137	0.127	**0.13**
	Accuracy	66.67%	66.67%	78.70%	66.67%	85.23%	82.38%	81.75%	**82% ±0.22%**
Bank Note Authentication	Loss	0.112	0.002	0.006	0.002	0	0	0	**0**
	Accuracy	92.71%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	**100.00% ±0.0%**
Heart Disease	Loss	0.461	0.187	0.509	0.4612	0.5312	0.48	0.18	**0.19**
	Accuracy	53.87%	72.55%	47.81%	53.87%	46.12%	51.01%	74.07%	**74.07% ±1.06%**
Haberman's Survival	Loss	0.2365	0.195	0.174	0.172	0.176	0.169	0.17	**0.169**
	Accuracy	73.70%	73.52%	75.49%	76.60%	76.79%	77.28%	76.96%	**77.10% ±1.27%**
Wine	Loss	0.4	0.445	0.445	0.4	0.219	0.333	0.226	**0.08**
	Accuracy	59.92%	55.43%	55.43%	59.92%	66.67%	66.67%	76.78%	**93.07% ±0.27%**
Car Evaluation	Loss	0.188	0.0718	0.094	0.096	0.0815	0.072	0.0918	**0.07**
	Accuracy	81.19%	91.46%	86.31%	86.15%	88.49%	90.10%	86.14%	**92% ±0.19%**
Indian Liver Patient	Loss	0.21	0.284	0.284	0.284	0.284	0.232	0.227	**0.227**
	Accuracy	71.50%	71.50%	71.50%	70.72%	71.50%	72%	72.10%	**72.63% ±0.55%**
Abalone	Loss	0.194	0.164	0.157	0.146	0.12	0.12	0.12	**0.12**
	Accuracy	77%	77%	77%	79.24%	83.44%	83.00%	83.27%	**83.54% ±0.26%**
Titanic	Loss	0.226	0.208	0.18	0.2	0.148	0.151	0.149	**0.149**
	Accuracy	67.92%	70.59%	71.57%	70.26%	79.34%	80.39%	79.06%	**80.53% ±0.41%**
Pima India Diabetes	Loss	0.307	0.2311	0.2411	0.285	0.1937	0.263	0.1908	**0.193**
	Accuracy	66.73%	66.73%	64.51%	66.99%	71.67%	69.53%	70.76%	**71.67% ±0.24%**
Agaricus Lepiota	Loss	0.24	0.16	0.08	0.03	0	0	0	**0**
	Accuracy	91.63%	97.53%	97.46%	99.16%	100%	100%	100%	**100% ±0.0%**

TABLE V: AdaSwarm *vs.* other optimizers: The promising results of AdaSwarm using MAE (*Non-Differentiable Loss Function*)

Dataset	**Metrics**	**Optimizer**							
		SGD	Emulation of SGD with PSO parameters	AdaGrad	AdaDelta	RMSProp	AMSGrad	Adam	AdaSwarm
Iris	Loss	0.22	0.177	0.319	0.195	0.138	0.226	0.224	**0.211**
	Accuracy	77.77%	95.11%	66.67%	88.22%	88.22%	77.77%	77.77%	**88.22% ±0.22%**
Ionosphere	Loss	0.468	0.34	0.295	0.32	0.216	0.218	0.217	**0.217**
	Accuracy	64%	79.57%	74.42%	70.57%	79.4%2	79.714%	79.428%	**82.714% ±0.78%**
Wisconsin Breast Cancer	Loss	0.385	0.411	0.349	0.348	0.349	0.349	0.349	**0.334**
	Accuracy	65.10%	67.96%	65.10%	65.10%	65.10%	65.10%	65.83%	**70.45% ±1.03%**
Sonar	Loss	0.498	0.245	0.304	0.275	0.21	0.199	0.1735	**0.169**
	Accuracy	50%	83.65%	77.4%	81.49%	83.89%	87.74%	86.05%	**89.53% ±0.33%**
Wheat Seeds	Loss	0.453	0.333	0.281	0.337	0.2722	0.333	0.26	**0.245**
	Accuracy	64.92%	66.67%	75.87%	67.3%	74.6%	70.79%	74.032%	**79.52% ±0.27%**
Bank Note Authentication	Loss	0.273	0.021	0.0255	0.0049	0	0.001	0.002	**0.001**
	Accuracy	79.07%	100%	100%	100%	100%	100%	100%	**100% ±0.0%**
Heart Disease	Loss	0.45	0.39	0.461	0.461	0.449	0.461	0.389	**0.378**
	Accuracy	55.21%	72.22%	53.87%	53.87%	54.88%	53.87%	62.96%	**73.23% ±0.27%**
Haberman's Survival	Loss	0.264	0.264	0.264	0.264	0.264	0.264	0.264	**0.264**
	Accuracy	73.52%	73.52%	73.52%	73.52%	73.52%	73.52%	73.52%	**73.52% ±0.0%**
Wine	Loss	0.599	0.33	274	0.315	0.333	0.4	0.247	**0.333**
	Accuracy	40.00%	66.67%	73.22%	69.10%	66.67%	59.92%	75.84%	**66.67% ±1.23%**
Car Evaluation	Loss	0.261	0.221	0.15	0.15	0.118	0.15	0.15	**0.138**
	Accuracy	85.01%	88.25%	85.01%	85.01%	88.10%	85.01%	85.01%	**92.10% ±0.31%**
Indian Liver Patient	Loss	0.5	0.43	0.285	0.285	0.285	0.285	0.285	**0.28**
	Accuracy	50%	57%	71.50%	71.50%	71.50%	71.50%	71.50%	**72.10% ±0.18%**
Abalone	Loss	0.391	0.335	0.232	0.23	0.229	0.229	0.23	**0.284**
	Accuracy	77%	77%	77%	77%	77%	77%	77%	**82.91% ±0.38%**
Titanic	Loss	0.366	0.4	0.319	0.311	0.3	0.305	0.306	**0.294**
	Accuracy	64.21%	68.20%	68.62%	69.46%	69.95%	69.74%	69.74%	**82.00% ±0.12%**
Pima India Diabetes	Loss	0.348	0.348	0.349	0.348	0.348	0.348	0.348	**0.348**
	Accuracy	65.10%	65.23%	65.10%	65.10%	65.10%	65.10%	65.10%	**73.43% ±0.36%**
Agaricus Lepiota	Loss	0.16	0.066	0.058	0.06	0.052	0.052	0.052	**0.07**
	Accuracy	85.96%	99.00%	94.53%	94.02%	94.72%	94.72%	94.72%	**99.85% ±0.04%**

The time complexity of AdaSwarm can be computed as follows: time taken to find gradient $\times$ steps required to reach the optimum. The time taken to find the gradient is the time complexity required to find the gradient. But since we approximate the gradient, for complex loss functions, the gradient calculation time taken by EMPSO is lower than mathematically computing it. Also, we note that the steps required to reach the optimum is lower for AdaSwarm. Thus, the AdaSwarm algorithm performs better/faster than Adam. The complexity analysis of EMPSO with more explanations is provided in Section 7 of the supp file.

B. REMPSO: Rotation Accelerated EMPSO for high dimensional data

PSO often fails in searching for the optima in the presence of a large number of variables mainly because of the computational cost required and the inability to escape a sub-plane of the whole search space. Hatanaka [31] proposed a Rotated Particle Swarm Optimizer to improve the performance of PSO when dealing with high-dimensional optimization problems. The motivation for this work was to tackle the computational cost of EMPSO for solving multi-class classification problems. Using Rotated PSO, we are able to overcome this problem, increasing the convergence speed of **AdaSwarm** on CV data sets (*see Section 5 of the supp file for REMPSO details and results*). REMPSO is a redefined version of EMPSO by leveraging the Rotation acceleration property to solve high-dimensional optimization problems in classification.

VI. Experiments

A. Experimental Setup

For testing AdaSwarm in NNs [5] [6], it was compared with respect to several state-of-the-art optimizers and applied on several benchmark data sets. The corresponding results are presented in **Table IV and in Table V**. In these experiments, for each of the data sets previously mentioned, we have kept the parameters of EMPSO constant ($c_1 = 0.8$**,** $c_2 = 0.9$**,** $\beta = 0.9, \eta = 0.1$). The choice of c_1, c_2 was guided by detailed stability analysis (See the supp file, Section 3). For consistency, a NN with one hidden layer with 10 neurons was used. The results shown are therefore unbiased and kept at default values to show the speed of convergence of AdaSwarm.

B. True Gradient and Approximate Gradient Comparison

The gradients were calculated for the functions mentioned in Tables I-III. We plotted the approximate gradient values computed using the PSO parameters defined in Section III-B and the true gradients. The results are shown in Fig. 1a and Fig. 1b. This graph shows that the use of this approach for computing gradients (*for differentiable and non-differentiable functions*) is indeed a viable choice and, consequently it can be embedded in the backpropagation algorithm.

[5] https://github.com/rohanmohapatra/torchswarm — EMPSO pytorch
[6] https://github.com/anuwu/PSO-Grad — EMPSO numpy

C. AdaSwarm vs. Optimizers for Classification Datasets

Using the approximated derivative, we replace the backward pass in the loss to this derivative. With such a modification, every time the algorithm backpropagates, it uses the custom approximate gradients throughout the layer to update the weights. A new loss was defined, in the forward pass. The binary cross-entropy loss was computed and in the backward pass, we replaced the gradient with our approximated gradient. The data set was divided into batches. Then, for an epoch, the batch loss and the accuracy were calculated using the optimizer. The losses were running averages in the batch. The same running averages were applied to the accuracies. The epoch loss, average accuracy with standard deviation were reported. For comparison purposes, we adopted SGD, SGD emulation with PSO parameters, AdaDelta [32], AMSGrad [33], RMSProp, Adam, and AdaSwarm on data sets from the UCI repository [34] (*Section 4 of the supp file*). AdaSwarm's iterations to converge were compared to those of Adam by keeping the accuracy as a threshold (*see Tables 6, 7 & 8 in the supp file*). AdaSwarm consistently outperformed Adam across all data sets (*sometimes AdaSwarm is twice as fast as Adam*).

D. Convolutional NN (CNN) : Architecture used for comparisons between Adam and AdaSwarm

We define a simple CNN architecture for conducting tests on benchmark data sets from computer vision. This model contains 2 convolution layers having a 3×3 kernel, a max-pooling layer and a flattened layer connected to a 128 neuron hidden layer, with an output layer of 10 neurons for 10 classes. The total weights for the MNIST model (*say*) sums up to 1,199,882 having a 28×28 image size. As the image size varies for CIFAR-10, the weights will vary, but the architecture remains the same (*See Table 10 of the supp file for results*). To further validate AdaSwarm, it was extended to deeper NNs (*considered as benchmarks in image classification*) like ResNet-18 [35]. The results are presented in Table 11 of the supp file.

Hardware Considerations: AdaSwarm requires lower number of epochs, cheaper architecture (*by comparing the hardware requirements of ResNet18 and ResNet 50*) and lower CPU/memory utilization. Even for the best hardware processing units [36], running an Epoch in ResNet50 requires almost twice the time required by ResNet18 (Section 3 of the supp file).

VII. Discussion

NN computation can be viewed as a sequence of functional compositions [37] [38]. This is one way to understand how nodes in hidden-layers progressively represent *more complex* features. However, sufficiently deep networks carry significant computation overhead in the training process. Also, lack of theoretical guarantees makes us wonder if there is some theoretical validation on how deep networks can be approximated by shallower ones. AdaSwarm constitutes an empirical validation of this hypothesis, having solved the benchmark problems under resource constraints, in a reasonably parsimonious fashion [39][40].

The proposed approach considers, at each iteration, that c_1r_1, c_2r_2, are computed by taking the average of n particles and $v^{(t-1)}$ is taken of the particle that influences the gbest. c_1 and c_2 are parameters typically used in PSO. We usually adopt the n particles of PSO to obtain the values and substitute them. That is why $c_1r_1 + c_2r_2$ is averaged over n particles whereas r_1 and r_2 change per particle, drawn from $U(0, 1)$. The global best is governed by the velocity of one particle whose velocity is taken to be $v^{(t-1)}$. No additional tuning of parameters was required on the data sets[7].

The simple yet elegant AdaSwarm model helped us to solve benchmark problems (*different papers provide different benchmarks*) quite efficiently. Since AdaSwarm is an optimizer, state-of-the-art approaches used to validate it must be optimizers rather than models. We illustrate the scalability of our proposed approach for computer vision data sets, on ResNet 18 architecture in which our approach was able to outperform other SOTA optimizers [41] tested on ResNet 18. However, we went ahead and checked the performance of these (Adam, Diffgrad etc) on ResNet 50 from[42] and our AdaSwarm on a stingier architecture ResNet 18 outperformed. It is worth noting that MAE is an unusual choice for a GD training method until recently [43] and is approximated well.

Momentum in PSO (i.e., EMPSO) is equivalent to a history of all velocities in an exponentially weighed fashion, making the search curve smoother since it contains all the momentum history. However, memory management does not have an adverse impact as we are not storing any velocities. Instead, it is a cumulative sum in the sense that when a new velocity is computed, it's accumulated over time, helping us reach the solution around the optimum. Inertia ceases to exist near the optimum. We have also used the bias correction in AdaSwarm (*default implementation*). The authors in [44] [45] claim that PSO converges near the optimum and by using initial and middle phases of the particles, we can factor in the velocity of the particles in the equivalence to go ahead with the approximation.

A subtlety to be noted in our gradient approximation formula is its applicability in the event of the underlying theoretical assumptions not being met. Say that a minority of the particles converged at g_{best}. Our gradient formulas are still applicable because of its first order form $\kappa(w - g_{best})$ — *for an appropriate constant* κ. What ultimately matters is that the best possible g_{best} be found, and not that the entire swarm converge to it. The purpose of our derivations is to *legitimize* the usage of a first-order approximation to the gradient, from PSO parameters, without making an assumption on the differentiability of the objective function. Essentially, it is a connection between swarm intelligence and gradient-based optimization (*two separate fields on their own right*). Our theoretical assumptions are not only conducive to the derivation, but also offer substantial simplification in terms of bare understanding. The *usage* of the formula does not distinguish between complete and partial convergence of the swarm so long as it converges at the correct g_{best}.

[7]Implementation: https://github.com/rohanmohapatra/pytorch-cifar

Moreover, we have adequately justified in aforementioned references that the discovery of a sub-optimal g_{best} does not make the NN weights (ultimately optimizing), via backpropagation, sub-optimal. We *do not* claim, however, that EMPSO approximated gradients be used in other contexts, where it is crucial that the swarm find the global optima of the objective function with guarantee and reliability. It is anticipated that approximated and true gradients will only deviate largely for query points w far from g_{best}. Thus a blind replacement of true gradients with EMPSO approximation gradients, in an arbitrary optimization context, for w far from g_{best}, would be unwise. The sole purpose of deriving formulas approximating gradients, and its subsequent use in AdaSwarm, is a means of overcoming the non-differentiability of loss functions in the *specific* context of NNs, which we have successfully achieved and demonstrated by experiments.

Tables I, II, III run PSO and EMPSO on sample 1-D functions and the true global minima is reliably obtained by the swarm. In some instances, despite the swarm partially converging, the correct local/global minima is obtained because it is only required that one particle find the g_{best}. Moreover table 4 in the supp file shows negligible deviation between true and approximated gradients for w far from the g_{best}.

VIII. Conclusions and Future Work

A theoretical relation directly connecting the gradient to the parameters of PSO had never been presented. The derivative of any function with countably finite singularities may now be precisely expressed in terms of the PSO and EMPSO parameters. Furthermore, the approximation of derivatives is seamlessly extended to approximate error gradients in the backpropagation algorithm used in NNs. our method surpasses standard PSO and SOTA in terms of speed of convergence and accuracy. A new optimizer, AdaSwarm, based on the widely used Adam optimizer, is introduced and tested against Adam in training NNs. The proposed incentivizes the use more natural (*like MAE*), non-differentiable loss functions in NNs.

Deep MLPs [46] learn features in the gradient space via path kernels. Therefore, if the GD does not do well, the model will have a poor performance. This leads us to ponder whether AdaSwarm learns better representations. What is the equivalent of a path kernel here? Does AdaSwarm discover better paths and also better approximations of the gradients (*for non-differentiable, rugged landscapes*)? Can we frame kernels as the integral (*over the path of gradient trajectory*) of the dot product between gradients (*of the* x *with* x_i) and argue that such kernels learn better representations of the gradients? Thus, seeing optimization via AdaSwarm just as a means to fit the model can be extended to making it integral to the learning process itself.

As an advancement to the PSO gradient equivalence theorem, it would be worthwhile to develop an analogous proof for Hessian approximations using PSO for differentiable and non-differentiable functions. These are questions worth pursuing in future research.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948, Nov 1995.

[3] H. Garg, "A hybrid pso-ga algorithm for constrained optimization problems," *Applied Mathematics and Computation*, vol. 274, pp. 292–305, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300315014630

[4] A. H. Gandomi and X.-S. Yang, *Benchmark Problems in Structural Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 259–281. [Online]. Available: https://doi.org/10.1007/978-3-642-20859-1_12

[5] A. Theophilus, S. Saha, S. Basak, and J. Murthy, "A novel exoplanetary habitability score via particle swarm optimization of CES production functions," in *IEEE Symposium Series on Computational Intelligence, SSCI 2018, Bangalore, India, November 18-21, 2018*. IEEE, 2018, pp. 2139–2147. [Online]. Available: https://doi.org/10.1109/SSCI.2018.8628669

[6] K. Bora, S. Saha, S. Agrawal, M. Safonova, S. Routh, and A. Narasimhamurthy, "Cd-hpf: New habitability score via data analytic modeling," *Astronomy and Computing*, vol. 17, pp. 129 – 143, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2213133716300865

[7] S. Basak, S. Saha, A. Mathur, K. Bora, S. Makhija, M. Safonova, and S. Agrawal, "Ceesa meets machine learning: A constant elasticity earth similarity approach to habitability and classification of exoplanets," *Astronomy and Computing*, vol. 30, p. 100335, 2020.

[8] M. Kheshti and L. Ding, "Particle swarm optimization solution for power system operation problems," in *Particle Swarm Optimization with Applications*, P. Erdoğmuş, Ed. Rijeka: IntechOpen, 2018, ch. 3. [Online]. Available: https://doi.org/10.5772/intechopen.72409

[9] G. Jana, A. Mitra, S. Pan, S. Sural, and P. K. Chattaraj, "Modified particle swarm optimization algorithms for the generation of stable structures of carbon clusters, cn (n = 3–6, 10)," *Frontiers in Chemistry*, vol. 7, p. 485, 2019.

[10] B. A. Garro and R. A. Vázquez, "Designing artificial neural networks using particle swarm optimization algorithms," *Computational Intelligence and Neuroscience*, vol. 2015, p. 369298, Jun 2015. [Online]. Available: https://doi.org/10.1155/2015/369298

[11] G. K. Jha, P. Thulasiraman, and R. K. Thulasiram, "Pso based neural network for time series forecasting," in *2009 International Joint Conference on Neural Networks*, June 2009, pp. 1422–1427.

[12] H. T. Rauf, W. H. Bangyal, J. Ahmad, and S. A. Bangyal, "Training of artificial neural network using pso with novel initialization technique," in *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, Nov 2018, pp. 1–8.

[13] H. Han, W. Lu, L. Zhang, and J. Qiao, "Adaptive gradient multiobjective particle swarm optimization," *IEEE Transactions on Cybernetics*, vol. 48, no. 11, pp. 3067–3079, 2018.

[14] E. Grimaldi, F. Grimaccia, M. Mussetta, and R. Zich, "Pso as an effective learning algorithm for neural network applications," in *Proceedings. ICCEA 2004. 2004 3rd International Conference on Computational Electromagnetics and Its Applications, 2004.*, 2004, pp. 557–560.

[15] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr 1980. [Online]. Available: https://doi.org/10.1007/BF00344251

[16] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317304120

[17] T. Xiang, J. Wang, and X. Liao, "An improved particle swarm optimizer with momentum," *2007 IEEE Congress on Evolutionary Computation*, pp. 3341–3345, 09 2007.

[18] S. M. Mikki and A. A. Kishk, *Particle Swarm Optimizaton: A Physics-Based Approach*. Morgan & Claypool, 2008.

[19] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," *In Proceedings of the 28th International Conference on Machine Learning*, p. 681–688, 2011.

[20] Y.-A. Ma, Y. Chen, C. Jin, N. Flammarion, and M. I. Jordan, "Sampling can be faster than optimization," *Proceedings of the National Academy of Sciences*, vol. 116, no. 42, pp. 20 881–20 885, 2019. [Online]. Available: https://www.pnas.org/content/116/42/20881

[21] S. Yokoi and I. Sato, "Bayesian interpretation of sgd as ito process," *ArXiv*, vol. abs/1911.09011, 2019. [Online]. Available: https://arxiv.org/abs/1911.09011

[22] J. Spall, *Introduction to stochastic search and optimization*. Wiley-Interscience, 2003.

[23] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, 09 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586

[24] M. Jamil and X. S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, p. 150, 2013. [Online]. Available: http://dx.doi.org/10.1504/IJMMNO.2013.055204

[25] R. Mohapatra, R. R. Talesara, S. Govil, S. Saha, S. S. Dhavala, and T. Sudarshan, "A new approach for momentum particle swarm optimization," in *Advances in Machine Learning and Computational Intelligence*, S. Patnaik, X.-S. Yang, and I. K. Sethi, Eds. Singapore: Springer Singapore, 2021, pp. 47–63.

[26] S. Chen and J. Montgomery, "Particle swarm optimization with thresheld convergence," *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, 06 2013.

[27] M. Alghalith, "A note on taylor expansions without the differentiability assumption," *Econometrics: Mathematical Methods Programming eJournal*, pp. 1–3, 2017. [Online]. Available: https://doi.org/10.2139/ssrn.2892746

[28] "Derivation of backpropagation," https://www.cs.swarthmore.edu/~meeden/cs81/s10/BackPropDeriv.pdf, [Online; accessed 17-March-2021].

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[30] S. Bock, J. Goppold, and M. G. Weiß, "An improvement of the convergence proof of the adam-optimizer," *CoRR*, vol. abs/1804.10587, 2018. [Online]. Available: http://arxiv.org/abs/1804.10587

[31] T. Hatanaka, T. Korenaga, N. Kondo, and K. Uosaki, "Search performance improvement for pso in high dimensional space," in *Particle Swarm Optimization*, A. Lazinica, Ed. Rijeka: IntechOpen, 2009, ch. 15. [Online]. Available: https://doi.org/10.5772/6752

[32] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[33] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *CoRR*, vol. abs/1904.09237, 2019. [Online]. Available: http://arxiv.org/abs/1904.09237

[34] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[36] C. G. Northcutt. Cnn gpu benchmarks. [Online]. Available: https://github.com/cgnorthcutt/cnn-gpu-benchmarks

[37] Z. Shen. Deep learning: Approximation of functions by composition. [Online]. Available: http://helper.ipam.ucla.edu/publications/dlt2018/dlt2018_14936.pdf

[38] Z. Shen, H. Yang, and S. Zhang, "Nonlinear approximation via compositions," *Neural Networks*, vol. 119, p. 74–84, Nov 2019. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2019.07.011

[39] R. Yedida, S. Saha, and T. Prashanth, "Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence," *Applied Intelligence*, vol. 51, no. 3, pp. 1460–1478, Mar 2021. [Online]. Available: https://doi.org/10.1007/s10489-020-01892-0

[40] S. Sridhar, S. Saha, A. Shaikh, R. Yedida, and S. Saha, "Parsimonious computing: A minority training regime for effective prediction in large microarray expression data sets," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[41] M. S. Hosseini and K. N. Plataniotis, "Adas: Adaptive scheduling of stochastic gradients," *CoRR*, vol. abs/2006.06587, 2020. [Online]. Available: https://arxiv.org/abs/2006.06587

[42] S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, and B. B. Chaudhuri, "diffgrad: An optimization method for convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, p. 1–12, 2020. [Online]. Available: http://dx.doi.org/10.1109/TNNLS.2019.2955777

[43] S. Saha, T. Prashanth, S. Aralihalli, S. Basarkod, T. Sudarshan, and S. S. Dhavala, "Lalr: Theoretical and experimental validation of lipschitz adaptive learning rate in regression and neural networks," in *2020*

International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1–8.

[44] H. Ye, W. Luo, and Z. Li, "Convergence analysis of particle swarm optimizer and its improved algorithm based on velocity differential evolution," *Computational Intelligence and Neuroscience*, vol. 2013, p. 384125, Aug 2013. [Online]. Available: https://doi.org/10.1155/2013/384125

[45] W. Qian and M. Li, "Convergence analysis of standard particle swarm optimization algorithm and its improvement," *Soft Computing*, vol. 22, no. 12, pp. 4047–4070, Jun 2018. [Online]. Available: https://doi.org/10.1007/s00500-017-2615-6

[46] P. Domingos, "Every model learned by gradient descent is approximately a kernel machine," *CoRR*, vol. abs/2012.00152, 2020. [Online]. Available: https://arxiv.org/abs/2012.00152

Rohan Mohapatra received the Bachelor's degree in Computer Science and Engineering from PES University, Bangalore, India in 2020. He is a researcher at Center for AstroInformatics (Astrirg). He has been involved in various research projects in the fields - Optimization algorithms, Neural networks and its mathematical foundations, Swarm Intelligence. He aspires to pursue his Master's majoring in the said fields.

Snehanshu Saha holds a PhD in Mathematical Sciences from the University of Texas at Arlington. He is a Senior Member of IEEE, Senior Member of ACM and a Fellow of IETE. Snehanshu is a Professor of Artificial Intelligence at BITS Pilani K K Birla Goa Campus. His current and future research interests lie in the theory of optimization, learning theory, activation functions in Deep Neural Networks and AstroInformatics.

Carlos A. Coello Coello (M'98-SM'04-F'11) received a PhD in computer science from Tulane University, USA, in 1996. He is currently Professor with Distinction (CINVESTAV-3F Researcher) at the Computer Science Department of CINVESTAV-IPN, in Mexico City, Mexico. He has authored/co-authored over 500 technical papers and book chapters. His publications currently report over 55,800 citations in Google Scholar (h-index:95). His major research interests are evolutionary multi-objective optimization and their constraint-handling techniques.

Anwesh Bhattacharya is an upcoming Physics (M.Sc.) and Computer Science (B.E.) dual-major from the Birla Institute of Technology & Science, Pilani. So far, he has been a generalist and worked in fields such as Astronomy, Swarm Intelligence, Machine Learning and his academic pursuits have been supported by the prestigious INSPIRE scholarship from DST, India. He wishes to pursue a doctorate in a field with a fine balance between theory and practice.

Soma S. Dhavala holds a PhD in Statistics from Texas A & M University. He is the founder of `mlsquare`, an open-source initiative to democratize AI. His current and future research interests lie in developing theory and tools for holistic Machine Learning.

Sriparna Saha received a PhD in computer science from Indian Statistical Institute Kolkata. She is currently Associate Professor at the Computer Science and Engineering Department of Indian Institute of Technology Patna, India. She has authored over 300 technical papers and book chapters. Her publications currently report over 4661 citations in Google Scholar (h-index:27). Her major research interests are evolutionary machine learning, deep learning, natural language processing and bioinformatics.