AdaSwarm : Supplementary Material

Rohan Mohapatra, Snehanshu Saha, Carlos A. Coello Coello, Anwesh Bhattacharya, Soma S. Dhavala and Sriparna Saha

April 30, 2021

1 Equivalence between SGD with Momentum and EMPSO

This is a detailed proof of finding the error gradient equivalence, and is referenced in Section III.B of the paper.
Theorem: Under reasonable assumptions on the global minimum, the following equivalence holds: $\eta = (1-\beta)$, $f''(\mathbf{w'}) = \frac{-(c_1r_1+c_2r_2)}{\eta}$ and $\alpha = \beta$

Proof: The Gradient Descent Weight Update with momentum is given by,

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

where,

$$V_{dw}^{t} = \beta V_{dw}^{t-1} + (1-\beta)\frac{\partial f}{\partial w} \tag{2}$$

Combining the equations and dividing (2) by $(1-\beta)$ we get,

$$w^{(t)} = w^{(t-1)} + \alpha V_{dw}^{t-1} + \eta\frac{\partial f}{\partial w} \tag{3}$$

Here, η is the learning rate and V_{dw}^{t-1} is the momentum applied to the weight update, where,

$$\alpha = \frac{\eta\beta}{(1-\beta)} \tag{4}$$

The Taylor approximation of the gradient can be written as,

$$\frac{\partial f}{\partial w} = f'(\mathbf{w'}) + f''(\mathbf{w'})(w - \mathbf{w'}) + ... + E_{n-1}(w) \tag{5}$$

We apply the Taylor series expansion to the gradient as defined in equation (5) and then formulate the gradient descent with momentum as follows,

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

The proposed EMPSO, is defined above and can be written as,

$$x_i^t = x_i^{t-1} + \beta M_i^{t-1} + (1-\beta)v_i^{t-1} + c_1r_1(p_i^{best} - x_i^{t-1}) + c_2r_2(g^{best} - x_i^{t-1}) \tag{6}$$

Under the same assumptions stated in the previous proof, we define the equivalence as follows:

$$\eta = (1-\beta) \tag{7}$$

$$f''(\mathbf{w}') = \frac{-(c_1r_1 + c_2r_2)}{\eta} \tag{8}$$

$$\alpha = \frac{\eta\beta}{(1-\beta)} \tag{9}$$

We find that $\mathbf{M}_i^{t-1}$ works the same way as the momentum term in equation (6), i.e., $\mathbf{V}_{dw}^{t-1}$ which helps in smoothing and speeding up convergence to the minimum. When we substitute the value of $f''(\mathbf{w'})$ in (5), the gradient of a differentiable function would be,

$$\frac{\partial f}{\partial w} = \frac{-(c_1r_1 + c_2r_2)}{\eta}(w - g^{best}) \tag{10}$$

1.1 Approximation of gradients by EMPSO for benchmark functions

This section accompanies Section III.D. in the paper.

We must find the numerical approximation of an exact value of the gradient. The approximation depends on a small parameter h, which can be, for instance, the grid size or time step in a numerical method. We denote the approximation of the gradient as $\tilde{u_h}$. The order of accuracy is shown in Sec. 1.2 of the Supplementary Material. The order of accuracy was found to be higher when we are closer to the optimum than when lying farther away from it. It is worth noting, however, that this is an empirical observation. We consider two types of functions: smooth functions (*differentiable everywhere*) and functions differentiable everywhere except for a finite number of singularities.

We compare approximate gradients and evaluate approximate optima of these functions using EMPSO, which bypasses explicit analytical gradient computations. The promising results from this approximation, as theoretically proved, are experimentally validated. We observed a reasonable degree of accuracy between the global optima and the approximations produced by EMPSO. The graphs (Fig. 1a) support our assertion. The numerical results, tabulated in Table 1, Table 2 and Table 3 reflect this visual observation. These experiments form the basis of further interpretation for emulating gradients and also suggests the need for investigating the possibility of exploiting this mathematical equivalence in the back propagation algorithm in the context of Deep Neural Networks. Sections V-VII in the main text corroborate that our intuition is indeed correct. Let's find the order of accuracy here.

Using the theorems proved, the gradient can be approximated as follows for a function $f(x)$,

$$\tilde{u_h} = \frac{-(c_1r_1 + c_2r_2)}{\eta}(x - g^{best}) \tag{11}$$

Considering the gradient as a true value, we have,

$$u = f'(x) \tag{12}$$

Since the value $\frac{-(c_1r_1+c_2r_2)}{\eta}$ is constant, we can replace it by M. Then we have,

$$|\tilde{u_h} - u| = M(x - g^{best}) - f'(x) \tag{13}$$

After the Taylor expansion we get,

$$|\tilde{u_h} - u| = M(f(x) + hf'(x) + \frac{h^2 f''(\xi)}{2} - g^{best}) - f'(x) \tag{14}$$

for some $\xi \in [0, h]$.

$$|\tilde{u_h} - u| = Mf(x) + Mhf'(x) + \frac{Mh^2 f''(\xi)}{2} - Mg^{best} - f'(x) \tag{15}$$

$$|\tilde{u_h} - u| = Mf(x) + (Mh - 1)f'(x) + \frac{Mh^2 f''(\xi)}{2} - Mg^{best} \tag{16}$$

Based on our theory, g^{best} is the solution for $f(x)$. After running PSO for a fixed number of iterations, we get the optimum. Hence, we can assume that $g^{best} \approx f(x)$ and if h is significantly smaller,

$$|\tilde{u_h} - u| = (M - 1)hf'(x) + \frac{Mh^2 f''(\xi)}{2} \tag{17}$$

Often the error $\tilde{u_h} - u$ depends smoothly on h. Then, there is an error coefficient D such that $\tilde{u_h} - u = Dh^p + O(h^{p+1})$. The order of accuracy was found to be higher when we are closer to the optimum than when being farther away from it. This observation is empirical. The following tables are shown in support of assertions made in section III.D. in the paper.

Table 1: Gradient Descent vs Emulated Gradient Descent using PSO; Parameter Set : η=0.1, c1=0.8, c2=0.9.

Function	**Global Minimum**	**Gradient Descent Optimum**	**Iterations**	**Emulated Gradient Descent with PSO Parameters Optimum**	**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 2: Stochastic Gradient Descent with Momentum vs Emulated Gradient Descent using EMPSO Parameter Set: η=0.1, c1=0.8, c2=0.9, β=0.9

Function	**Global Minimum**	**Stochastic Gradient Descent Optimum**	**Iterations**	**Emulated Gradient Descent with EM-PSO Paramters Optimum**	**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 3: Stochastic Gradient Descent with Momentum vs Emulated Gradient Descent using EMPSO for Non-Differentiable Functions (NA - Not available) Parameter Set : η=0.1, c1=0.8, c2=0.9, β=0.9

Function	**Global Minimum**	**Stochastic Gradient Descent Optimum**	**Iterations**	**Emulated Gradient Descent with EMPSO Parameters Optimum**	**Iterations**
$\sqrt{x}$	0	NA	NA	0.01	22
$\|x\|$	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

Table 4: Analytical gradient vs Approximate gradient for points away from global minima

	x	**Analytical gradient**	**Approximate gradient**
$x^3 + e^{-2x} - 5$ (*Differentiable*)	1.63**	7.894	7.53
Global minimum at 0.497	-0.73**	-7.013	-7.32
	-0.281**	-3.27	-3.38
	0.623*	0.589	0.58
	0.13**	-1.5	-1.53
	0.5*	0.02	0.021
$e^{\sin(x)} + x^4$ (*Differentiable*)	0.87**	4	4.3
Global minimum at -0.511	0.52**	2	2.9
	-0.38*	0.711	0.722
	-0.511*	0	0.1
	-0.706**	-1	-1.12
$\sqrt{x}$ (*Non-Differentiable*)	0.5**	0.707	0.667
Global minimum at 0	0*	inf	78556.23
	0.1*	1.581	1.583
	2**	0.355	0.312
	0.25**	1	1.23
	0.063*	2	2.1
$\|x\|$ (*Non-Differentiable*)	0*	inf	0
Global minimum at 0	-3**	-1	-0.89
	-2**	-1	-0.82
	0.5**	1	0.67
	1**	1	-0.75

$*\frac{\partial f}{\partial x} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(x^{(t-1)} - g^{best}) \rightarrow$ near minima
$** \ \frac{\partial f}{\partial x} = \frac{\phi}{\eta} v_i^{(t-1)} + \frac{-c_1 r_1 (x^{(t-1)} - p_i^{best}) - c_2 r_2 (x^{(t-1)} - g^{best})}{\eta} \rightarrow$ away from minima

1.2 Note on Non-differentiable loss functions

As claimed, we could establish the gradient equivalence for non-differentiable loss functions in training neural networks. It was not a common practice to use MAE until a recent paper by Saha et. al [1]. However, in order to establish such equivalence, we need to expand the scope of the Taylor Series approximation. Let us consider a continuous, bounded function $f(x)$ that is not differentiable with respect to x. Using a shift parameter, we can express $f(x)$ as $f(x + \xi)$, where ξ is a shift parameter.

Define $z \equiv x + \xi$, then we have $f(z) \equiv f(x + \xi)$. Differentiability with respect to the shift parameter (as opposed to a variable) is derived from the fact that the small change in the shift parameter is a constant. We have, $d\xi = \epsilon - 0 = \epsilon$ and thus $dz = dx + d\xi = dx + \epsilon$.

$$d\xi = \epsilon - 0 = \epsilon$$

Thus,

$$dz = dx + d\xi = dx + \epsilon$$

Using the forward difference rule, we obtain:

$$\frac{df(x)}{dx}|_{\Delta x=0} = \lim_{\Delta z \to 0} \frac{f(x + \Delta z) - f(x)}{\Delta z}|_{\Delta x=0}$$
$$= \lim_{\Delta z \to 0} \frac{f(x + \Delta x + \epsilon) - f(x)}{\Delta x + \epsilon}|_{\Delta x=0}$$
$$= \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

Using the central difference rule, we obtain:.

$$\frac{df(x)}{dx}|_{\Delta x=0} = \lim_{\Delta z \to 0} \frac{f(x+\Delta z) - f(x-\Delta z)}{2\Delta z}|_{\Delta x=0}$$
$$= \lim_{\Delta z \to 0} \frac{f(x+\Delta x+\epsilon) - f(x-\Delta x-\epsilon)}{2\Delta x+2\epsilon}|_{\Delta x=0}$$
$$= \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

This is a more precise approximation of $\frac{df}{dx}$. By the continuity of f and the fact that $\epsilon \neq 0$, the derivative exists. Since we know that the derivative exists, we obtain the following Taylor expansion around a:

$$f(z) = f(a) + f'(a)(z-a) + R(z,a) \tag{18}$$

where R is the remainder. For details, please refer to Moawia Algalith's paper [2]. Additionally, table 4 has experimental results on the gradient equivalence for both differentiable and non-differentiable loss functions, near and far away from the point of convergence.

2 Error Gradient Equivalence in Feed-Forward Neural Networks

This is a detailed proof of finding the error gradient equivalence, and is referenced in Section IV.A. of the paper. For background on the notation used in this section, refer to [3].

The back propagation weight-tuning rule uses the error-gradient $\frac{\partial E}{\partial w_{ji}}$. The observation is that the weight w_{ji} can influence the rest of the networks through net output coming from a neuron, net_j. Let us consider j to be an output unit. Then, We can use the chain rule to write,

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \tag{19}$$

Calculating the gradient $\frac{\partial E}{\partial y_j}$ would mean choosing a loss function that is differentiable, and that would eliminate a lot of loss functions. In order to tackle this problem, we propose a novel idea to approximate the gradient using PSO parameters. Using the theorem, we will prove that

$$\frac{\partial E}{\partial y_j} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(g^{best} - y) \tag{20}$$

Using the Taylor series expansion cetnred at y_{opt}, the gradient is approximated as follows,

$$\frac{\partial E}{\partial y_j} = E'(y_{opt}) + E''(y_{opt})(y_j - y_{opt}) + ... + R_{n-1}(y_j) \tag{21}$$

Here $E(y)$ is the loss with respect to which the error is minimized so as to get good predictions, R_{n-1} is the remainder and y_{opt} is the optimal value of the loss at which the error is the lowest. If we apply the PSO parameter approximation proved in the theorem mentioned in Section 1, we get

$$E''(y_{opt}) = \frac{-(c_1 r_1 + c_2 r_2)}{\eta} \tag{22}$$

substituting this value in (21), we will get

$$\frac{\partial E}{\partial y_j} = E'(y_{opt}) - \frac{(c_1 r_1 + c_2 r_2)}{\eta}(y_j - y_{opt}) + ... + R_{n-1}(y) \tag{23}$$

Let's state a few assumptions that are likely to be true in most cases,

- Around the optimum value (*global minimum*), the loss gradient at the optimum value is 0, hence $E'(y_{opt}) = 0$.
- We can consider the Remainder term of the Taylor series expansion as it doesn't have significance in most cases, hence $R_{n-1}(y) = 0$
- In Particle Swarm Optimization, it has been found that the g^{best} is the optimum value of any function, so if $E(y)$ is a loss function, when using PSO, g^{best} can be substituted by y_{opt}.

Then, we get a rather simple equation for the Error gradient,

$$\frac{\partial E}{\partial y_j} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(y_j - g^{best}) \tag{24}$$

Substituting yields

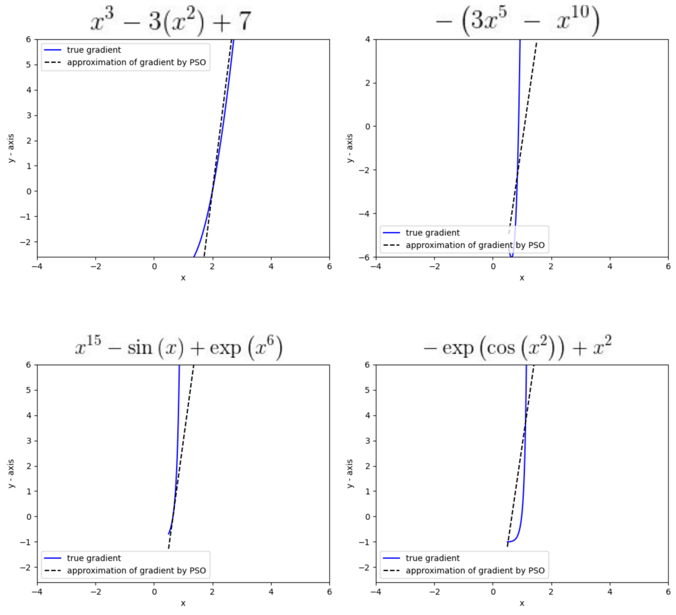
$$\frac{\partial E}{\partial w_{ji}} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(y_j - g^{best})\frac{\partial y_j}{\partial net_j} x_i \tag{25}$$

where $\frac{\partial y_j}{\partial net_j}$ is the value of the gradient with respect to the activation function used and $\frac{\partial net_j}{\partial w_{ji}} = x_i$. **This relation is loss function-agnostic**.

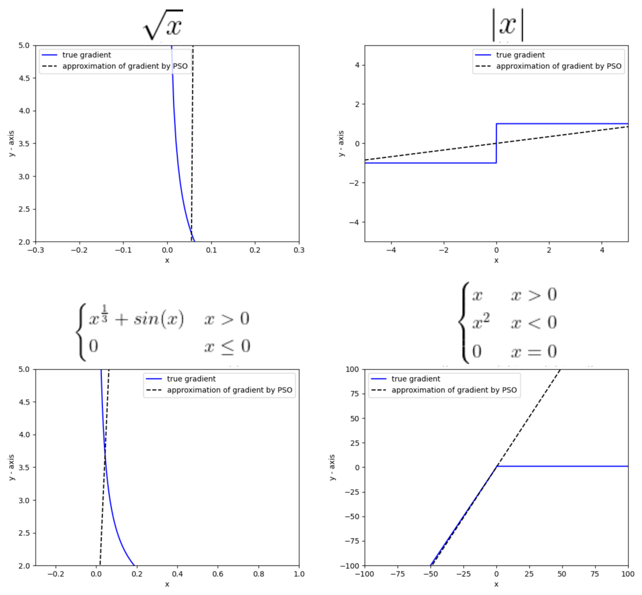
3 AdaSwarm vs different Optimizers on various Datasets

Table 5: AdaSwarm vs other optimizers: The promising results of AdaSwarm could address the sensitivity (to initialization), and robustness (to multiple local minima) in classification datasets using Binary Cross Entropy (BCE)/Cross Entropy Loss (Differentiable Loss) (cf. sec. V-VI in the paper)

Dataset	**Metrics**	**Optimizer**							
		SGD	Emulation of SGD with PSO parameters	AdaGrad	AdaDelta	RMSProp	AMSGrad	Adam	AdaSwarm
Iris	Loss	0.184	0.133	0.232	0.272	0.199	0.222	0.219	**0.188**
	Accuracy	96.223%	98.223%	88.444%	86%	92.222%	89.222%	90.667%	**97.97% ±0.28%**
Ionosphere	Loss	0.665	0.37	0.564	0.545	0.243	0.3978	0.259	**0.243**
	Accuracy	52.429%	88.857%	73.571%	76.142%	92.143%	83.425%	90.857%	**91.82% ±0.1%**
Wisconsin Breast Cancer	Loss	0.560	0.414	0.436	0.422	0.414	0.383	0.405	**0.399**
	Accuracy	81.231%	84.384%	83.284%	83.431%	84.384%	84.970%	83.357%	**84.680% ±0.12%**
Sonar	Loss	0.69	0.441	0.439	0.428	0.374	0.3530	0.33	**0.358**
	Accuracy	58.173%	80.769%	81.492%	81.971%	83.413%	85.099%	86.298%	**88.635% ±0.18%**
Wheat Seeds	Loss	0.612	0.638	0.565	0.586	0.43	0.434	0.434	**0.433**
	Accuracy	66.984%	66.667%	66.667%	73.334%	80.317%	78.889%	81.905%	**81.958% ±0.09%**
Bank note Authentication	Loss	0.228	0.001	0.0300	0.004	0.0005	0.002	0.004	**0.0001**
	Accuracy	97.255%	100%	100%	100%	100%	100%	100%	**100% ±0.0%**
Heart Disease	Loss	0.494	0.550	0.557	0.517	0.499	0.398	0.564	**0.562**
	Accuracy	77.94%	71.17%	70.34%	74.580%	76.43%	82.15%	72.27%	**79.16% ±0.78%**
Haberman's Survival	Loss	0.5766	0.5766	0.534	0.556	0.5266	0.533	0.5242	**0.5267**
	Accuracy	73.856%	73.529%	75.163%	75.490%	76.143%	76.470%	77.00%	**77.68% ±0.06%**
Wine	Loss	0.663	0.652	0.385	0.555	0.631	0.603	0.400	**0.399**
	Accuracy	66.667%	66.667%	80.337%	69.101%	66.667%	71.535%	83.142%	**83.075% ±0.02%**
Car Evaluation	Loss	0.375	0.268	0.282	0.304	0.286	0.273	0.260	**0.273**
	Accuracy	85.011%	87.355%	85.894%	86.038%	85.156%	87.340%	86.850%	**87.141% ±0.27%**
India Liver Patient	Loss	0.540	0.531	0.530	0.538	0.507	0.597	0.510	**0.509**
	Accuracy	71.58%	71.50%	71.58%	71.15%	72.10%	71.50%	71.58%	**72.68% ±0.04%**
Abalone	Loss	0.552	0.556	0.475	0.449	0.382	0.400	0.389	**0.388**
	Accuracy	77.00%	77.00%	78.12%	79.11%	82.925%	82.098%	82.84%	**82.85% ±0.03%**
Titanic	Loss	0.6	0.6	0.55	0.529	0.478	0.47	0.432	**0.431**
	Accuracy	70.02%	68.62%	72.05%	73.73%	78.99%	78.50%	81.02%	**81.33% ±0.22%**
Pima Indian Diabetes	Loss	0.6	0.6	0.65	0.6	**0.55**	0.55	0.574	0.58
	Accuracy	69.92%	65.10%	66.67%	68.75%	**73.95%**	72.52%	71.35%	**71.87% ±0.45%**
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%**



(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

Table 6: AdaSwarm vs other optimizers: AdaSwarm epochs required to reach Adam threshold accuracy in BCE (Differentiable) Loss

Dataset	**Adam Accuracy Threshold**	**Adam epochs till saturation**	**AdaSwarm epochs to reach threshold**
Iris	90.66%	24	10
Ionosphere	90.86%	40	28
Wisconsin Breast Cancer	83.36%	36	11
Sonar	86.30%	39	23
Wheat Seeds	81.91%	37	30
Bank Note Authetication	100%	17	7
Heart Disease	72.27%	38	29
Haberman's Survival	77.00%	32	31
Wine	83.14%	39	22
Car Evaluation	86.85%	34	21
Indian Liver Patient	71.58%	40	25
Abalone	82.84%	38	31
Titanic	81.02%	38	21
Pima Indian Diabetes	71.35%	40	26
Agaricus Lepiota	100%	36	17

Table 7: AdaSwarm vs other optimizers: AdaSwarm epochs required to reach Adam threshold accuracy in MSE (Differentiable) Loss

Dataset	**Adam Accuracy Threshold**	**Adam epochs till saturation**	**AdaSwarm epochs to reach threshold**
Iris	98.22%	49	45
Ionosphere	98.57%	41	39
Wisconsin Breast Cancer	84.82%	44	32
Sonar	86.30%	42	38
Wheat Seeds	81.75%	47	33
Bank Note Authetication	100%	47	26
Heart Disease	74.07%	43	42
Haberman's Survival	76.96%	43	31
Wine	76.28%	46	26
Car Evaluation	86.14%	42	30
Indian Liver Patient	72.10%	50	28
Abalone	83.27%	46	21
Titanic	79.06%	46	24
Pima India Diabetes	70.76%	46	22
Agaricus Lepiota	100%	49	15

Table 8: AdaSwarm vs other optimizers: AdaSwarm epochs required to reach Adam threshold accuracy in MAE (Non-Differentiable) Loss

Dataset	**Adam Accuracy Threshold**	**Adam epochs till saturation**	**AdaSwarm epochs to reach threshold**
Iris	77.77%	95	49
Ionosphere	79.43%	93	42
Wisconsin Breast Cancer	65.83%	93	44
Sonar	86.05%	93	59
Wheat Seeds	74.03%	92	48
Bank Note Authetication	100%	34	8
Heart Disease	62.96%	100	46
Haberman's Survival	73.52%	98	59
Wine	75.84%	95	43
Car Evaluation	85.01%	92	41
Indian Liver Patient	71.50%	92	56
Abalone	77%	95	40
Titanic	69.74%	100	44
Pima Indian Diabetes	65.10%	98	40
Agaricus Lepiota	94.72%	100	57

Table 9: Number of times AdaSwarm beat Adam for various loss functions

	Times AdaSwarm beat Adam (out of 10, greater or equal)				
Dataset	**BCE**	**MSE**	**MAE**	**Adam Avg. Accuracy in** % (BCE, MSE, MAE)	**AdaSwarm Avg. Accuracy in** % (BCE, MSE, MAE)
Iris	10	7	10	[90.67±0.22, 98.22±0.45, 77.7±0.28]	[97.97±0.28, 98.22±0.38, 88.22±0.22]
Ionosphere	8	5	10	[90.857±0.26, 98.57±0.36, 79.428±1.08]	[91.82±0.1, 98.48±0.28, 82.714±0.78]
Wisconsin Breast Cancer	9	10	10	[83.357±0.11, 84.82±0.13, 65.83±2.04]	[84.680±0.12, 85.55±0.74, 70.45±1.03]
Sonar	8	7	10	[86.3±0.16, 86.30±0.89, 86.05±0.31]	[88.635±0.18, 86.30±0.34, 89.53±0.33]
Wheat Seeds	7	8	10	[81.90±0.16, 81.75±0.22, 74.032±0.88]	[81.96±0.09, 82±0.22, 79.52±0.27]
Bank Note Authentication	10	10	10	[100±0.0 ,100±0.0, 100±0.0]	[100±0.0, 100±0.0, 100±0.0]
Heart Disease	10	5	10	[72.27±0.63, 74.07±0.03, 62.69±0.12]	[79.16±0.75, 74.07±0.06, 73.23±0.27]
Haberman's Survival	5	8	8	[77±0.01, 76.96±0.03, 73.52±0.05]	[77.68±0.06, 77.910±1.27, 73.52±0.0]
Wine	4	10	3	[83.142±0.21,76.78±1.56, 75.84±0.95]	[83.075±0.02, 93.07±0.27,66.67±1.23]
Car Evaluation	8	10	10	[86.85±0.26, 86.14±0.33, 85.01±0.08]	[87.14±0.27, 92±0.19, 92.10±0.31]
Indian Liver Patient	9	10	8	[71.6±0.02, 72.10±1.56, 71.5±0.69]	[72.68±0.04, 72.63±0.55, 72.10±0.18]
Abalone	6	7	10	[82.84±0.01, 83.27±0.07, 77±1.52]	[82.85±0.03, 83.54±0.26, 82.91±0.38]
Titanic	10	9	10	[81.02±0.22, 79.06±0.39, 69.74±1.07]	[81.33±0.22, 80.53±0.41, 82±0.12]
Pima Indian Diabetes	6	9	10	[71.35±0.12, 70.76±0.33, 65.10±0.46]	[71.87±0.45, 71.67±0.24, 73.43±0.36]
Agaricus Lepiota	10	10	10	[100±0.0, 100±0.0, 100±0.0]	[100±0.0, 100±0.0, 100±0.04]

Table 10: AdaSwarm vs Adam for Computer Vision Datasets: The promising results of AdaSwarm could address the sensitivity (to initialization), and robustness (to multiple local minima) This table is referenced in the Section VI.D, Adaswarm and other optimizers were run on Computer Vision datasets which used the ResNet18 Model.

Dataset	**Best Loss**		**Best Training Accuracy**		**Testing Accuracy**		**Total Execution Time**	
	Adam	AdaSwarm	Adam	AdaSwarm	Adam	AdaSwarm	Adam	AdaSwarm
MNIST	0.073	0.0727	97.09%	97.3%	97.2%	97.3%	~1180 s	~1046 s
Fashion MNIST	0.027	0.017	99%	99.5%	99.32%	99.48%	~990 s	~900 s
CIFAR 10	0.234	0.223	91.058%	91.6%	91.074%	91.447%	~1110 s	~994 s
CIFAR 100	0.036	0.038	99.122%	99.100%	99.007%	99.039%	~2760 s	~3312 s
iBeans [4]	0.631	0.524	66.7%	71.3%	68.182%	72.56%	~876 s	~860 s

Table 11: AdaSwarm vs different Optimizers on Computer Vision Datasets: The optimizers were applied for the following datasets on the ResNet 18 Network. It is also referenced in the Section VI.D, Adaswarm and other optimizers were run on Computer Vision datasets which used the ResNet18 Model. NA :Values are not available; MNIST: MN, Ada: AdaShift[5], WNG: WN-Grad[6], FMNIST: Fashion MNIST, AMSG: AMSGrad, DG: DiffGrad [7] along with CPU Usage and Memory Usage

Opt	**Dataset**			
	MN	FMNIST	CIFAR10	CIFAR100
Ada	NA	NA	91.25%	NA
WNG	NA	NA	92.00%	NA
AMSG	NA	NA	90%	NA
Adam	99.20%	94.81%	92.50%	63.82%
DG	99.20%	94.93%	92.25%	65.54%
AdaSwarm	**99.36%**	**95.01%**	**92.59%**	**70.79%**

Dataset	**Total CPU Usage**			**Total Memory Usage**		
	Adam	DG	AdaSwarm	Adam	DG	AdaSwarm
MN	19%	21%	21%	31%	31%	32%
FMNIST	18%	20.36%	20.21%	31%	31%	32%
CIFAR10	18.32%	20.36%	20.21%	31%	31%	32%
CIFAR100	36.00%	37.30%	38.19%	31%	31%	32%

3.1 Hardware Requirements: ResNet18 vs ResNet50

A comparison of hardware requirements between ResNet18 vs ResNet 50. This comparison is referenced in Section VI.D. of the paper.

GPU	ResNet18 Time/Per Epoch(ms)	ResNet50 Time/Per Epoch(ms)
GTX 1080 Ti	31.54	51.31
Pascal Titan X	32.78	51.59
Pascal Titan X	33.13	55.58
GTX 1080	43.94	72.09
GTX 1080	47.52	79.76
Maxwell Titan X	51.42	80.23
Maxwell Titan X	54.87	91.98
Maxwell Titan X	63.8	116.11
Pascal Titan X	96.4	172.98
GTX 1080 Ti	116.03	195.73
GTX 1080	122.1	220.13
Maxwell Titan X	151.82	275.13
CPU: Dual Xeon E5-2630 v3	2195.78	3965.21

4 Datasets Specification on which AdaSwarm was tested on (Ref: Section VI. in the paper)

Dataset	Attributes	Classes
Iris	4	3
Ionosphere	34	2
Wisconsin Breast Cancer	9	2
Sonar	60	2
Wheat Seeds	7	3
Bank Note Authentication	4	2
Heart Disease	13	2
Haberman's Survival	3	2
Wine	13	3
Car Evaluation	6	4
Indian Liver Patient	10	2
Abalone	8	2
Titanic	7	2
Pima Indians Diabetes	8	2
Agaricus Lepiota	22	2

5 REM-PSO: Rotation Accelerated EMPSO for high dimensional data

This section is in reference to Section V.B in the paper.

There has been extensive research in swarm optimization to show the efficacy of Particle Swarm Optimization in optimizing various discrete/continuous problems. Particle swarm optimization has been used in plenty of applications to optimize successfully the underlying problem in Neural networks training, PID controller tuning and many such applications. Despite that, PSO fails in searching the global optimal solution in the presence of many decision variables, which increases the computation cost, but also because there is a high probability of getting stuck in a sub-plane of the entire search space.

Hatanaka et al. [8] propose a variant of Particle Swarm Optimizer known as Rotational PSO, which improves the performance of PSO in high-dimensional space optimization problems. The reason for this improvement was to tackle the computation lag of EMPSO for multi-class classification problems. Using the previously indicated Rotated PSO technique, we overcome the lag and increase the speed of **AdaSwarm**.

We can re-define the EMPSO equation as follows/ Let's assume,

$$\Phi_1 = diag(c_{1,1} * r_{1,1}, c_{1,2} * r_{1,2}, c_{1,3} * r_{1,3}...., c_{1,d} * r_{1,d}) \tag{26}$$

$$\Phi_2 = diag(c_{2,1} * r_{2,1}, c_{2,2} * r_{2,2}, c_{2,3} * r_{2,3}...., c_{2,d} * r_{2,d}) \tag{27}$$

The EMPSO equation can be written as,

$$v_i^{t+1} = \beta M_i^t + (1-\beta)v_i^t + \Phi_1(p_i^{best} - x_i^t) + \Phi_2(g^{best} - x_i^t) \tag{28}$$

The EMPSO algorithm imitates a flock of birds seeking food, but here the emphasis is to increase the seeking ability (fast convergence rate). Birds most likely never change the seeking strategy according to if food exists in the true direction or maybe at an angle from the true direction. We infer from this that even particles can search for optima even if the axes are rotated by and angle θ. We use a coordinate transformation technique on the velocity update by using a matrix A. A is a $D \times D$ matrix where a certain constant factor is used to control the number of axes to rotate. If we consider a point in the original space x, y, in the transformed rotated coordinate space, we would have x', y', and we can write them as

$$x' = xcos(\theta) + ysin(\theta) \tag{29}$$

$$y' = -xsin(\theta) + ycos(\theta) \tag{30}$$

A has an orthonormal basis, i.e. $A^{-1} = A^T$. Consider an arbitrary matrix with orthonormal basis vectors P, then

$$T_A : e_1, e_2, ..., e_N \rightarrow e'_1, e'_2, ..., e'_N \tag{31}$$

can be expressed as the following transformation matrix,

$$A = [e'_1, e'_2, ..., e'_N] \forall e_i, e_j \in R^N, j = 1, ..., N \tag{32}$$

A matrix rotating an N-dimensional solution space θ degrees is expressed as:

$$M(\theta, N) = \prod_{i=1}^{N-1} \prod_{j=i+1}^{N} M^{i,j}(\theta) \tag{33}$$

and each element $p_{q,l}^{i,j}(\theta)$ of $M(\theta, N)$ is expressed as follows:

$$p_{q,l}^{i,j} = \begin{cases} \cos(\theta), & \text{if } q = \text{i}, l = \text{i} \\ \text{-}\sin(\theta), & \text{if } q = \text{i}, l = \text{j} \\ \sin(\theta), & \text{if } q = \text{j}, l = \text{i} \\ \cos(\theta), & \text{if } q = \text{j}, l = \text{j} \\ 1, & \text{if } q = l \neq \text{i}, \text{if } q = l \neq \text{j} \\ 0, & \text{otherwise.} \end{cases}$$

Let's define a matrix with $i = 5, j = 5$, and the matrix M (or A) is

$$\begin{bmatrix} cos(\theta) & sin(\theta) & 0 & 0 & 0 \\ -sin(\theta) & cos(\theta) & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & cos(\theta) & sin(\theta) \\ 0 & 0 & 0 & -sin(\theta) & cos(\theta) \end{bmatrix}$$

5.1 Transformation In variance

EMPSO optimization has transformation in variance demonstrated below,

- $f \circ T_{r_s} \circ update = f \circ update \circ T_{r_s}$ (transformation in variance for solution space)
- $T_{r_f} \circ f \circ update = f \circ update \circ T_{r_f}$ (transformation in variance for objective function)

T_{r_s}: Scale Transformation/Parallel Shift/Rotation of the solution space
T_{r_f}: Scale Transformation/Parallel Shift/Rotation of the objective function

In variance for rotation of solution space: $T_{r_s}x \rightarrow G_x, G \in R^N$, where G satisfies $G^{-1} = G^T$, rotating a vector in solution space.

Using the R-PSO equation and combining with the EMPSO equation, we get

$$v_i^{t+1} = \beta M_i^t + (1-\beta)v_i^t + A^T\phi_1 A(p_i^{best} - x_i^t) + A^T\phi_2 A(g^{best} - x_i^t) \tag{34}$$

where,

$$A = [e'_1, e'_2, ..., e'_N]\forall \in R^N, j = 1, ..., N \tag{35}$$

is a coordinate transformation matrix. $e'_1, e'_2, ..., e'_N$, is the normal basis of the eigen vectors of the covariance matrix, $\Sigma(Z) \in R^N$, of solution set Z.

6 Stability Analysis of EMPSO

The stability analysis of the EMPSO algorithm is presented here, and is referenced in Section VII of the paper. As stated in the paper, EMPSO is based on Particle Swarm Optimization, making it an iterative procedure to solve real world optimization problems. All nature-inspired algorithms are stochastic in nature whereas Stability Analysis is a more deterministic approach. EMPSO proceeds to find near optimal solution for problems, and there is always an error that is generated as the iterations proceed. So, it is important to find conditions for which the error remains bounded- This is done by finding the suitable hyper-parameters through stability analysis and this mechanism plays a vital role in making the algorithm efficient. To undergo the stability analysis, we leverage the von Neumann stability for EMPSO. Gopal *et al.* [9] describe the von Neumann Stability criterion [10] in detail.

Theorem: Exponentially weighted Particle Swarm Optimization with the momentum factor β is said to be stable iff the acceleration coefficients and β satisfy the conditions:

- $0 < \beta < 1$
- $0 \leq (c_1 + c_2) \leq 2$

Proof: In Section III, we define the velocity update rule as follows,

$$v_{d,t+1} = M_{d,t+1} + c_1(p_1 - x_{d,t}) + c_2(p_2 - x_{d,t}) \tag{36}$$

where, $c_1 =$*weight of the local information* $\times r_1$, $c_2 =$*weight of global information*$\times r_2$. Combining with the position update rule, we have,

$$x_{d,t+1} = x_{d,t} + M_{d,t+1} + c_1(p_1 - x_{d,t}) + c_2(p_2 - x_{d,t}) \tag{37}$$

Expanding $M_{d,t+1}$, we get

$$\begin{aligned} x_{d,t+1} &= x_{d,t} + \beta M_{d,t} + (1-\beta)v_{d,t} + c_1(p_1 - x_{d,t}) + c_2(p_2 - x_{d,t}) \\ &= x_{d,t} + \beta(M_{d,t-1} + (1-\beta)v_{d,t-1}) + (1-\beta)v_{d,t} + c_1(p_1 - x_{d,t}) + c_2(p_2 - x_{d,t}) \\ &= x_{d,t} + \beta(1-\beta)v_{d,t-1} + (1-\beta)v_{d,t} + c_1(p_1 - x_{d,t}) + c_2(p_2 - x_{d,t}) \\ &= x_{d,t} + \beta(1-\beta)(x_{d,t-1} - x_{d,t-2}) + (1-\beta)(x_{d,t} - x_{d,t-1}) + c_1(p_1 - x_{d,t}) + c_2(p_2 - x_{d,t}) \\ &= (2 - \beta - c_1 - c_2)x_{d,t} - (\beta^2 - 2\beta + 1)x_{d,t-1} - \beta(1-\beta)x_{d,t-2} + c_1p_1 + c_2p_2 \end{aligned} \tag{38}$$

For the stability analysis of the EMPSO algorithm, let's consider the update equation (38) as a finite differences scheme,

$$x_{d,t+1} - (2 - \beta - c_1 - c_2)x_{d,t} + (\beta^2 - 2\beta + 1)x_{d,t-1} - \beta(\beta - 1)x_{d,t-2} = c_1p_1 + c_2p_2 \tag{39}$$

Equation (38) is a non-homogeneous finite difference scheme with $A_{-2} = -\beta(\beta-1)$, $A_{-1} = (\beta^2 - 2\beta + 1)$, $A_0 = -(2 - \beta - c_1 - c_2)$, $A_1 = 1$, $B_{-2} = B_{-1} = B_0 = B_1 = 0$ and $C = c_1p_1 + c_2p_2$. So, in order to find the stability condition of the PSO algorithm, for further analysis we will consider the following associated homogeneous difference scheme which is obtained by setting $c_1p_1 + c_2p_2 = 0$.

$$x_{d,t+1} - (2 - \beta - c_1 - c_2)x_{d,t} + (\beta^2 - 2\beta + 1)x_{d,t-1} - \beta(\beta - 1)x_{d,t-2} = 0 \tag{40}$$

We apply the transformation $t \rightarrow t + 2$, and we get

$$x_{d,t+3} - \lambda_1 x_{d,t+2} + \lambda_2 x_{d,t+1} - \lambda_3 x_{d,t} = 0 \tag{41}$$

where,

- $\lambda_1 = 2 - \beta - c_1 - c_2$
- $\lambda_2 = \beta^2 - 2\beta + 1$
- $\lambda_3 = \beta(1 - \beta)$

If the exact solution in the $d - t$ computational domain is taken as $x = x(d,t)$, the approximate solution at the nodes of the grid is given by $x(d_i, t_j)$, where $i \in 1, 2, 3..., b1$, $j \in 1, 2, 3..., b2$. Therefore, for the stability analysis of the PSO algorithm we consider a finite differences scheme given by equation (41) instead of equation (38). The von Neumann stability criterion for a finite differences scheme is used for deriving the stability condition for the update equation of the EMPSO algorithm.

Let the n^{th} component of the complex Fourier series solution to the given equation be given by:

$$x_{d_i,t_j} = B_n e^{\iota(\sigma_n i\Delta d - \beta_n j\Delta t)} \tag{42}$$

where $\iota = \sqrt{1}$, B_n represents the amplitude of the n^{th} component, β_n is the angular frequency and σ_n is the wave number of the n^{th} component. In terms of grid points, (41) can we written as,

$$x_{d_i,t_{j+3}} - \lambda_1 x_{d_i,t_{j+2}} + \lambda_2 x_{d_i,t_{j+1}} - \lambda_3 x_{d_i,t_j} = 0 \tag{43}$$

Substituting the value for the grid, we get

$$B_n e^{\iota(\sigma_n i\Delta d - \beta_n j\Delta t)}(e^{-\iota\beta_n 3\Delta t}) - \lambda_1(e^{-\iota\beta_n 2\Delta t}) + \lambda_2(e^{-\iota\beta_n \Delta t}) - \lambda_3) = 0 \tag{44}$$

Since, $B_n \neq 0$ until the algorithm terminates, therefore

$$e^{-\iota\beta_n 3\Delta t}) - \lambda_1(e^{-\iota\beta_n 2\Delta t}) + \lambda_2(e^{-\iota\beta_n \Delta t}) - \lambda_3 = 0 \tag{45}$$

or

$$A^3 - \lambda_1 A^2 + \lambda_2 A - \lambda_3 = 0 \tag{46}$$

where, $A =$ Amplification Factor.

Now, according to von Neumann's stability criterion, the finite differences scheme (41) is stable iff for the amplification factor (A), $|A| \leq 1$ [11]. Therefore, the finite differences scheme given by equation (41) and so the finite differences scheme given by equation (38) and hence the EMPSO algorithm are stable iff $|A| \leq 1$.

On finding the root of the cubic equation, we conclusively say that EMPSO is stable iff,

- $0 < \beta < 1$
- $0 \leq (c_1 + c_2) \leq 2$

7 Complexity Analysis of EMPSO

This is in reference to section II.A in the paper.

7.1 EMPSO Algorithm

Algorithm 1: EMPSO

```
for each particle p in swarm S do
    initialize the particle with a feasible random position;
    evaluate the fitness $F_i$ of the particle;
endfor
while termination condition is not fulfilled do
    for each particle p in swarm S do
        $v_i = \beta M_i + (1-\beta)v_i + c_1 r_1(p_i^{best} - x_i) + c_2 r_2(g^{best} - x_i)$;
        $x_i = x_i + v_i$;
        $M_i = \beta M_i + (1-\beta)v_i$;
    endfor
    update $p_i^{best}$, $g^{best}$ by finding the best fitness;
    choose best solution;
end
```

7.2 Primitive Operations

We must count the number of primitive operations to determine the complexity of EMPSO,

- Steps 2 & 3 contribute one operation for n times
- Step 6 contributes for n loops with 16 operations
- Step 11 performs n iterations and $n\log(n)$ operations to find the p_i^{best}
- Step 11 performs $\log(n)$ to find g^{best}

The complexity is

$$T(n) = 2n + 16n + nlog(n) + nlog(n) \tag{47}$$

$$T(n) = 18n + nlog(n) + nlog(n) \tag{48}$$

This is done for t times until the termination condition is not met. Hence,

$$T(n) = t(18n + (n+1)log(n)) \tag{49}$$

In summary, the EMPSO algorithm has a complexity of $\mathcal{O}(tnlog(n))$. Considering the worst case of finding the maximum, EMPSO has a complexity $\mathcal{O}(tn^2)$. The computational cost is relatively inexpensive because it is linear in t. The number of particles, n is actually kept low for such meta-heuristic algorithms.

8 EMPSO for single-objective optimization problems

The EMPSO algorithm[12] was validated using several single-objective optimization problems which are presented here, and are referenced in Section VI of the paper. The proposed EMPSO is applied on benchmark problems [13], habitability optimization problems in Astronomy and in engineering optimization problems. Usually a problem may be constrained or unconstrained. An unconstrained problem's search space allows algorithms to explore the full search space using the particles, whilst constraining the search space makes it more difficult to explore the search space. Theophilus *et al.* [14] propose a technique to handle constrained optimization problems. Here, we use the same technique to optimize the test functions and a few standard optimization problems.

8.1 Standard Test Optimization Functions

In this section, we present few benchmarks optimization problems with which we have assessed our proposed algorithm and compared it with Momentum PSO mention in Section I.C in the Main text. The following single-objective unconstrained optimization problems [13] are chosen: Ackley, Rosenbrock-2D, Beale, Goldstein, Booth, Bulkin, Matyas, Levi, Himmelblau, Threehumpcamel, Eason, Crossintray. For each problem, we have tracked the iterations and minima achieved for 100 instances on both Vanilla and EMPSO algorithms and reported their mean values. Statistical significance tests (p-values) have also been reported. The stopping criteria was — each particle being inside a 10^{-2} hypercube centred at every other particle, or the swarm reaching 10,000 iterations, whichever event occured earlier. Refer to table (12) for the results and p-values (*in subscript*).

Discussion - As demonstrated by the statistical testing, exponentially averaged momentum clearly has a significant role in decreasing the iterations taken to convergence. Over all functions, a p-value of < 0.05 is achieved, most notably in Rosenbrock-2D and Himmelblau's function. In the case of iterations, there is a large tradeoff in the minima obtained for rosenbrock-2D, however in most functions, the minima achieved is slightly better, if not significantly better.

The following constrained optimization problems were tested : Rosenbrock constrained with a cubic line, Rosenbrock constrained to a disc and Mishra's bird.

Differentiable test optimization problems: Jamil[13] states the algebraic description of the test functions in his work in terms of characterizing each objective function as (non)-differentiable, (non)-separable, scalable, unimodal or multimodal, or a combination thereof. For example, Ackley Function is Ccontinuous, Differentiable, Non-separable, Scalable, Multimodal/unimodal. Beale Function is Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal. Bird Function is Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal. Booth Function is Continuous, Differentiable, Non-separable, Non-Scalable, Unimodal. Camel Function–Three Hump is Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal. Jamil [13] documents more details on test optimization problems in his work. The results for the above mentioned benchmark optimization problems are summarized in 17**Table 10 & 12** and 13**Table 11 & 13**.

8.2 Engineering Optimization Problems

In this section we briefly describe the benchmark engineering optimization problems [15] chosen to evaluate our proposed algorithm and compare its performance to that of several other algorithms. These problems have been formulated based on real-world scenarios.

8.3 Constrained Single-Objective Optimization Problems

Here, we present two constrained single-objective from exoplanetary habitability score computation [16, 17]. We solve these problems with our proposed algorithm and compare with the scores computed earlier through gradient ascent/descent type approaches.

8.3.1 Representing Habitability score (CDHS)

The CDH score can be constructed to form a constrained optimization problem where the objective function is represented as follows:

$$Y = R^{\alpha}.D^{\beta}.V_e^{\gamma}.T_s^{\delta} \tag{50}$$

where,

- R, D, V_e and T_s are density, radius, escape velocity and surface temperature for a particular exoplanet respectively.
- α, β, γ and δ are elasticity coefficients.

$$\begin{aligned} \underset{\alpha,\beta,\gamma,\delta}{\text{maximize}} \quad & Y \\ \text{subject to} \quad & 0 < \phi < 1, \forall \phi \in \{\alpha, \beta, \gamma, \delta\}, \\ & \alpha + \beta - 1 - \tau \leq 0, \\ & 1 - \alpha - \beta - \tau \leq 0, \\ & \gamma + \delta - 1 - \tau \leq 0, \\ & 1 - \gamma - \delta - \tau \leq 0 \end{aligned} \tag{51}$$

It can be subject to two scales of production: CRS (Constant Return to Scale) and DRS (Decreasing Return [18]. Equation (50) is concave under constant returns to scale (CRS), when $\alpha + \beta = 1$ and $\gamma + \delta = 1$, and also under decreasing returns to scale (DRS), $\alpha + \beta < 1$ and $\gamma + \delta < 1$.

8.3.2 Constant Elasticity Earth Similarity CEESA

The CEESA score to estimate habitability of an exoplanet is:

$$\begin{aligned} \underset{r,d,t,v,e,\rho,\eta}{\text{maximize}} \quad & Y = (r.R^{\rho} + d.D^{\rho} + t.T^{\rho} + v.V^{\rho} + e.E^{\rho})^{\frac{\eta}{\rho}} \\ \text{subject to} \quad & 0 < \phi < 1, \forall \phi \in \{r, d, t, v, e\}, \\ & 0 < \rho \leq 1, \\ & 0 < \eta < 1, \\ & (r + d + t + v + e) - 1 - \tau \leq 0, \\ & 1 - (r + d + t + v + e) - \tau \leq 0 \end{aligned} \tag{52}$$

where E represents Orbital eccentricity, τ is tolerance. Two scales of production are used: CRS (Constant Return to Scale) and DRS (Decreasing Return to Scale).

The Confirmed Exoplanets Catalog maintained by the Planetary Habitability Laboratory (PHL) [19] is used to perform experiments on the proposed algorithm . We use the parameters described in Table 16. The surface temperature and the eccentricity are not recorded in Earth Units. These values are normalized by dividing with Earth's surface temperature (288° K) and eccentricity (0.017). The PHL-EC records are empty for exoplanets with unknown surface temperature. These records are dropped for the experiment.

We tested the **proposed swarm algorithm** on the test optimization problems, CDHS and CEESA objective functions, Engineering Optimization problems mentioned below. We used $n = 1000$ with a *target error* $= 1 \times 10^{-6}$ and 50 particles for all the experiments performed in this section.

Table 12: Unconstrained Test Problems [13]

Name	***Global Minimum***	***MPSO Optimized Value***	**Iterations**	***EMPSO Optimized Value***	**Iterations**
Ackley	0	0.001	59	0.001 $_{1.726E-02}$	47 $_{0.00}$
Rosenbrock2D	0	$2 * 10^{-8}$	215	67.14 $_{7.770E-03}$	124 $_{7.368E-13}$
Beale	0	$4.38 * 10^{-7}$	68	0 $_{1.214E-02}$	136 $_{1.339E-03}$
Goldstein–Price	3	3	61	3	53 $_{0.00}$
Booth	0.0	$1.07 * 10^{-7}$	90	$4.09 * 10^{-7}$ $_{1.391E-02}$	49 $_{0.00}$
Bukin N.6	0	0.05	503	0.047 $_{8.344E-13}$	191 $_{3.979E-02}$
Matyas	0	0	55	0 $_{3.915E-02}$	30 $_{0.00}$
Lévi N.13	0	0	83	0 $_{7.165E-02}$	56 $_{2.292E-03}$
Himmelblau's	0	$2.51 * 10^{-7}$	95	0 $_{2.307E-02}$	48 $_{1.054E-06}$
Three-hump camel	0.0	$2 * 10^{-8}$	59	0.0 $_{3.511E-02}$	36 $_{0.00}$
Easom	-1	$-3 * 10^{-10}$	45	-1 $_{3.218E-02}$	40 $_{3.689E-02}$
Cross-in-tray	-2.06261	-2.06	43	-2.06	30 $_{3.796E-02}$

Table 13: Constrained Test Problems [13]

Name	**Global Minimum**	**M-PSO Optimized Value**	**Iterations**	**EM-PSO Optimized Value**	**Iterations**
Mishra Bird	-106.76	-106.76	121	-106.76	55
Rosenbrock constrained with a cubic and a line function	0	0.99	109	0.99	64
Rosenbrock constrained to a disk	0	0	69	0	39

Table 14: Unconstrained Test Problems with fixed iterations [13]

Name	**Iterations**	**MPSO Optimized Value**	**EMPSO Optimized Value**
Ackley	40	0.009	0.001
Rosenbrock2D	100	0.003	5.2×10^{-7}
Beale	60	1.13×10^{-6}	1.04×10^{-6}
Goldstein–Price	50	3.012	3
Booth	50	2.37×10^{-6}	2.38×10^{-6}
Bukin	200	0.064	0.01
Matyas	30	0.003	0
Lévi	60	0.004	0.0001
Himmelblau's	50	0.0007	0
Three-hump camel	40	0.00068	0
Easom	40	-2.004	-1
Cross-in-tray	30	-2.13	-2.067

Table 15: Constrained Test Problems with fixed iterations [13]

Name	**Iterations**	**MPSO Optimized Value**	**EMPSO Optimized Value**
Mishra Bird	50	-107.23	-106.81
Rosenbrock constrained with a cubic and a line function	70	1.03	0.99
Rosenbrock constrained to a disk	40	0.00112	0.0004

Table 16: CDHS and CEESA scores computed by EMPSO are close to the scores computed by [16, 17] with a high accuracy

Name	Algorithm	[α, β, γ, δ]	[Y_i, Y_s]	CDHS Scores	Iterations	[r, d, t, v, e, ρ]	CEESA Scores	Iterations
TRAPPIST-1 b	M-PSO	[0.99, 0.01, 0.01, 0.99]	[1.09, 1.38]	1.234	130	[0.556, 0, 0.398, 0.045, 0, 0.629]	1.193	76
	EM-PSO	[0.99, 0.01, 0.01, 0.99]	[1.09, 1.38]	1.234	75	[0.107, 0.314, 0.578, 0.001, 3.704, 0.999]	1.126	92
TRAPPIST-1 c	M-PSO	[0.99, 0.01, 0.01, 0.99]	[1.17, 1.21]	1.19	65	[0.117, 0.384, 0.273, 0.225, 0, 0.999]	1.161	54
	EM-PSO	[0.99, 0.01, 0.01, 0.99]	[1.17, 1.21]	1.19	80	[0.053, 0.348, 0.212, 0.386, 0, 0.999]	1.161	60
TRAPPIST-1 e	M-PSO	[0.99, 0.01, 0.01, 0.99]	[0.92, 0.88]	0.9096	30	[0.455, 0.486, 0.033, 0.027, 5.182, 0.504]	0.868	82
	EM-PSO	[0.99, 0.01, 0.2, 0.8]	[0.92, 0.88]	0.9096	69	[0.264, 0.004, 0.626, 0.105, 0, 0.936]	0.897	7
TRAPPIST-1 f	M-PSO	[0.99, 0.01, 0.95, 0.05]	[1.04, 0.8]	0.92	93	[0.718, 0, 0.276, 0.006, 0, 0.969]	0.972	77
	EM-PSO	[0.99, 0.01, 0.7, 0.3]	[1.04, 0.8]	0.92	59	[0.382, 0.24, 0.093, 0.284, 5.392, 0.719]	0.836	46

Table 17: Engineering Optimization Problems (NA - number of iterations is not available)

Optimization Problem	**Method (minima value, Number of iterations)**								
	Previously Best known soln.	**W-PSO**	**M-PSO**	**EM-PSO**	**H. Garg**	**SMPSO**	**NSGA-2**	**NSGA-3**	**PSO with Momentum [20]**
Himmelblau's nonlinear Problem [15]	$(-31025.574717,$ NA)	$(-30018.053335,$ 34)	$(-30439.2877572,$ 412)	$(-30469.5964172,$ 217)	$(-31025.574717,$ NA)	$(-30661.083090,$ 10000)	$(-30552.749592,$ 10000)	$(-30655.685504,$ 10000)	$(-29673.453273,$ 1000)
Welded Beam Problem [15]	(1.695247, NA)	(1.611633, 512)	(1.611633, 504)	(1.619996, 155)	(1.695247, NA)	(1.632858, 10000)	(1.643901, 10000)	(1.956199, 10000)	(1.781361, 1000)
Speed Reducer Problem[21]	(2, 994.337292, NA)	(3009.825014, 843)	(3041.737500, 576)	(3070.474141, 743)	(NA, NA)	(3007.668781, 10000)	(2996.452023, 10000)	(2996.554276, 10000)	(3161.473142, 1000)
Pressure Vessel Problem [15]	(5885.332773, NA)	(7570.7047916, 53)	(5921.9787155, 703)	(5986.3712241, 204)	(5885.332773, NA)	(6386.675189, 10000)	(6340.756917, 10000)	(6021.229240, 10000)	(9561.108353, 1000)
Compression Spring Problem [21]	(0.0126652, NA)	(0.012665, 496)	(0.0126675, 382)	(0.0126662, 871)	(NA, NA)	(0.013659, 10000)	(0.0130492, 10000)	(0.013804, 10000)	(0.013170, 1000)

9 Additional Notes on Adaswarm Improvements

In this section, we deliberate on subtle improvements on AdaSwarm, strictly from the point of view of improving function approximation capabilities of AdaSwarm and more importantly, integrate the mathematical sophistication in Deep Neural Nets to improve training error in classification problems. The latter aspect involves investigating activation functions that are traditionally used in AdaSwarm and tuning one hyper-parameter, η originating from gradient descent update rule. Let us recall the fundamental relations that helped us reach gradient and loss gradient approximations.

- $f''(\mathbf{w}') = \frac{-(c_1r_1+c_2r_2)}{\eta}$
- $\frac{\partial f}{\partial w} = \frac{-(c_1r_1+c_2r_2)}{\eta}(w - g^{best})$
- $\frac{\partial E}{\partial w_{ji}} = \frac{-(c_1r_1+c_2r_2)}{\eta}(g^{best} - y_j)\frac{\partial y_j}{\partial net_j}x_{ji}$; where $\frac{\partial y_j}{\partial net_j}$ is the value of the gradient with respect to the activation function used.

The above implies that there are possibilities for further improvement. This could be accomplished by

- Improving derivative approximation capabilities (Section 1.2 & 9.1)
- Discovering better learning rate, η adaptive in nature (to loss functions) (Section 9.2)
- Exploiting the activation function (ReLU/AReLU) in an overparametrized setting (Section 9.2)

9.1 Note on Approximation

From the theorems mentioned in main text, the approximation can be formulated as,

$$\tilde{u_h} = \frac{-(c_1r_1+c_2r_2)}{\eta}(x - g^{best})$$

Considering the gradient as a true value, we have,

$$u = f'(x)$$

Since the value $\frac{-(c_1r_1+c_2r_2)}{\eta}$ is constant, we can replace it by M. We will use a central difference error to check the approximation,

$$|\tilde{u_h} + u| - |\tilde{u_h} - u| = M(x - g^{best}) + f'(x) - (M(x - g^{best}) - f'(x))$$

$$|\tilde{u_h} + u| - |\tilde{u_h} - u| = 2f'(x)$$

$$\frac{|\tilde{u_h} + u| - |\tilde{u_h} - u|}{2} = f'(x)$$

This gives us more confidence that the approximation is more precise when central difference is used.

9.2 Note on tuning AdaSwarm

Let us revisit $\frac{\partial f}{\partial w}\Big|_{w=w'} = \frac{-(c_1r_1+c_2r_2)}{\eta}(w - g^{best})$, the equivalence relation which drives AdaSwarm approximation of Error gradients in Deep and Shallow Neural Nets. Typically, η, the learning rate is fixed in experiments and empirically found out from data and experiments. Lipschitz Adaptive Learning Rates have shown evidence of effectiveness in reducing computational infrastructure [22] and accelerating speed of convergence in [1].

9.2.1 Learning Rates

Learning rates in gradient descent based optimization employed in regression and classification problems are manually chosen, traditionally. This is driven by the common wisdom that such choices help control the rate of convergence. There are recent evidences against such conventional choices where decay-based and non-monotonic learning rates have been proposed. Significant advances have been been made in the direction of making learning rate choices adaptive in optimization including Adam. The first and second moments of gradients are leveraged by a combination of RMSProp and Momentum based Gradient Descent.

Methods using historical gradients for updating parameters and thus making effective learning rate adaptive are being utilized. LipschitzLR [23] computes adaptive learning rates based on the principal that learning rates can be computed by taking the reciprocal of the Lipschitz constant for smooth loss functions $\in C^1$. These losses, MSE and BCE/CE need to be Lipschitz continuous and not $\beta-$smooth. In the context of AdaSwarm, we can formulate and illustrate methods which converge faster with minimal compromise on the prediction error. We can also derive LALR for tractable loss such as the Quantile Loss and use it in regression and classification tasks. It is also possible to choose a loss function, robust and reliable such as the MAE, and use Lipschitz Adaptive learning rates for regression and classification tasks.

Local Lipschitz continuity may be resistant to adversarial deception aided by Lipschitz-constrained networks to possess better generalization ability. The ability of the network to generalize well does not depend only on the smoothness of the function to be learnt by the network. Smoothness of the loss function is critical in characterizing the learnabiity of the task in terms of sample size, architecture complexity, and convergence rates. Since we have used smooth loss functions such as MSE and BCE/CE, we can attempt to provide non-asymptotic error rates for flexible feedforward networks, when the loss functions are smooth. Additionally, loss function such as Mean Absolute Error (MAE) is robust to additive noise and conformable to L^1 Lipschitz continuity and therefore adaptive learning rates can be used in conjunction with MAE, instead of a fixed η.

9.2.2 Local Smoothness and Activation Functions

We have used RELU in AdaSwarm driven neural nets. Pre-trained models such as Resnet (used in our experiments on Computer Vision data sets) use ReLU activation by default. ReLU suffers from singularity at $x = 0$. Even though RELU works well in practice, singularity and the dying RELU problem are sufficient grounds for exploring continuous approximates of RELU. Neural networks can accommodate a variety of activation functions based on their ability to generalize, given these activations satisfy Universal Approximation theorems (UAT). Since RELU is strictly monotone in the positive half of the input domain and does not suffer from the vanishing gradient problem, a continuous variant of RELU, free from the singularity problem, should do well if these nice properties of RELU are imbibed by it. Most of the neural network architecture nowadays heavily use the ReLU activation function due to its faster training capabilities. ReLU is a linear function, expressed as $f(x) = max(0, x)$. It can be easily seen that ReLU is non-smooth at $x = 0$, and hence, suffers from the "dying" problem.

To overcome this problem, Approximate Rectified Linear Unit (ARELU) activation function of the form: $f(x) = max(0, kx^n)$ has been proposed. AReLU smoothly approximating ReLU, even at $x = 0$. Apart from this, it can be easily shown that ARELU does not have a saddle point for the chosen parameters (For example, if $n = 3$, ARELU indeed has one but ARELU is defined for $1 < n < 2$) and also, does not suffer from the vanishing gradient problem if one initializes $1 < n < 2, k \in (0.5, 0.54)$, which also interestingly ensures that the gradients explode with a very less probability. It is also possible to arrive at optimum values of the hyperparameters k and n based on least square optimization. However, the Universal Approximation Theorem (UAT) was not shown to hold for ARELU.

9.2.3 Universal Approximation Theorem

Finite sums $S(x)$ approximating $1-$ hidden layer DNN are learnt by the activation function f i.e. for $\epsilon > 0$, there is a sum, $S(x)$, of the above form, for which $|S(x) - f(x)| < \epsilon, \ \forall x \in I_n$, where I_n is locally compact.

We can we show that ARELU is a valid activation by proving that it satisfies UAT and thus use it in AdaSwarm.

9.2.4 Overparameterized Networks and Activation Functions

While Neural Nets have been shown to demonstrate very good results across many domains, their working still remains largely a black-box. Today, such models boast of millions of trainable parameters and are able to generalize well even on a relatively small data set, and hence, they defy the prior belief of overfitting on such data sets. ([24]) was one of the first seminal works that demonstrated the generalization capabilities of such overparameterized networks via thorough empirical analysis and laid a foundation towards understanding such wide, black-box algorithms. Recently, many works have started exploring theoretic grounds for such overparamatrized settings, especially for Stochastic Gradient Descent-based optimization in Rectified Linear Unit (ReLU) setting ([25]; [26]; [27]). However, ReLU suffers from non-differentiability at 0. The components of tuning an efficient Neural Network that achieves zero training are bridged by adopting adaptive learning learning rates as discussed and exploiting inherent properties of activation functions (ReLU/ARelU) in an overparameterized setting (OVAL). The pertinent aspects of such tasks involve proving the following:

- OVAL achieves zero training error
- Relevant theoretical results in overparameterized ReLU/A-ReLU activation
- Apply LALR [1] for established loss functions and MAE in overparameterized ReLU/A-ReLU network
- Compute Lipschitz Learning rates for Quadratic Loss and provide theoretical and empirical evidence for improved convergence speed.

9.2.5 Overparameterized Learning Model

Definition: An overparameterized Neural network is the one which has more parameters to learn than the size of the training data. The model achieves zero training error by interpolating training data and is capable of excellent memorization.

For example, it is possible to train such model on small subsamples from the MNIST or Fashion MNIST data set. Overparameterized Learning Models thrive on the evidence that neural wider networks generalize better. Belkin et al [28] have shown that learning models tend to overfit when the bias-variance curve hits certain threshold. Then, as the model grows in complexity, the overparameterization i.e. as the number of parameters grow larger than the data points, the test error starts diminishing.

Li and Liang [26] studied the problem of learning a two-layer overparameterized neural network using SGD for classification. The data in each class is a mixture of several components, and components from different classes are well separated in distance, but the components in each class can be close to each other. When the network is sufficiently overparameterized, SGD provably learns a network close to the random initialization and with a small generalization error. Thus, SGD with random initialization introduces a strong inductive bias and leads to good generalization.

A two-layer neural network with ReLU activation for k-classes classification is given by $f = (f_1, f_2, \cdots, f_k)$ such that for each $i \in [k]$:

$$f_i(x) = \sum_{r=1}^{m} a_{i,r} ReLU(\langle w_r, x \rangle)$$

where $\{w_r \in \mathbb{R}^d\}$ are the weights for the m neurons in the hidden layer, $\{a_{i,r} \in \mathbb{R}\}$ are the weights of the top layer, and $ReLU(z) = \max\{0, z\}$.

Assumptions about the data

There are $k \times l$ unknown distributions $\{\mathcal{D}_{i,j}\}_{i\in[k],j\in[l]}$ over $\mathbb{R}^d$ and probabilities $p_{i,j} \geq 0$ such that $\sum_{i\in[k],j\in[l]} p_{i,j} = 1$. Each data point (x, y) is i.i.d. generated by: (1) Sample $z \in [k] \times [l]$ such that $\Pr[z = (i, j)] = p_{i,j}$; (2) Set label $y = z[0]$, and sample x from $\mathcal{D}_z$. Assume we sample N points $\{(x_i, y_i)\}_{i=1}^N$.

(A1) (Separability) There exists $\delta > 0$ such that for every $i_1 \neq i_2 \in [k]$ and every $j_1, j_2 \in [l]$, minimum distance between $\mathcal{D}_{i_1,j_1}$ and $\mathcal{D}_{i_2,j_2}$ is at least δ. Moreover, for every $i \in [k], j \in [l]$, diameter of $\mathcal{D}_{i,j}$ is linearly bounded by δ.

(A2) (Normalization) Any x from the distribution has $\|x\|_2 = 1$.

(A3) (Random initialization) $w_r^{(0)} \sim \mathcal{N}(0, \sigma^2)$, $a_{i,r} \sim \mathcal{N}(0, 1)$, with $\sigma = \frac{1}{m^{1/2}}$.

Loss function and its Gradient

The learning process minimizes the cross entropy loss over the softmax, defined as:

$$L(w) = -\frac{1}{N}\sum_{s=1}^{N} \log o_{y_s}(x_s, w), \text{ where } o_y(x, w) = \frac{e^{f_y(x,w)}}{\sum_{i=1}^k e^{f_i(x,w)}}.$$

Let B be the batch size, $T = N/B$ be the number of iterations and η be the learning rate for the minibatch SGD. At each iteration, the update is:

$$w_r^{(t+1)} = w_r^{(t)} - \eta\frac{1}{B}\sum_{s\in B_t} \frac{\partial L(w^{(t)}, x_s, y_s)}{\partial w_r^{(t)}}, \forall r \in [m], \qquad \text{where}$$

$$\frac{\partial L(w, x_s, y_s)}{\partial w_r} = (\sum_{i\neq y_s} a_{i,r} o_i(x_s, w) - \sum_{i\neq y_s} a_{y_s,r} o_i(x_s, w))1_{\langle w_r, x_s\rangle \geq 0} x_s. \tag{53}$$

Main Result

Theorem: Suppose the assumptions **(A1)(A2)(A3)** are satisfied. Then for every $\epsilon > 0$, there is $M = \text{poly}(k, l, 1/\delta, 1/\epsilon)$ such that for every $m \geq M$, after doing a minibatch SGD with batch size $B = \text{poly}(k, l, 1/\delta, 1/\epsilon, \log m)$ and learning rate $\eta = \frac{1}{m\cdot\text{poly}(k,l,1/\delta,1/\epsilon,\log m)}$ for $T = \text{poly}(k, l, 1/\delta, 1/\epsilon, \log m)$ iterations, with high probability:

$$\Pr_{(x,y)\sim\mathcal{D}}\left[\forall j \in [k], j \neq y, f_y(x, w^{(T)}) > f_j(x, w^{(T)})\right] \geq 1 - \epsilon.$$

This theorem says that given that data satisfies the above assumptions, there exist parameters for the network such that only polynomial in $k, l, 1/\delta$ many samples are needed to achieve an arbitrarily less test sample error. Moreover, the number of training iterations needed increases by the factor of only $\log m$.

Furthermore, if we treat each data sample as an individual distribution, we can see that $l = N = BT$. So by the theorem if $m = poly(N, 1/\delta')$, where δ' is the minimal distance between each example, we can fit arbitrary labels of the input data. However, since T only depends on $\log m$, but the input data is actually structured (well separated), then SGD can achieve a small generalization error, even when the network has enough capacity to fit arbitrary labels.

9.2.6 Proof sketch

The following are important questions to address:

1. Why can SGD optimize training loss?

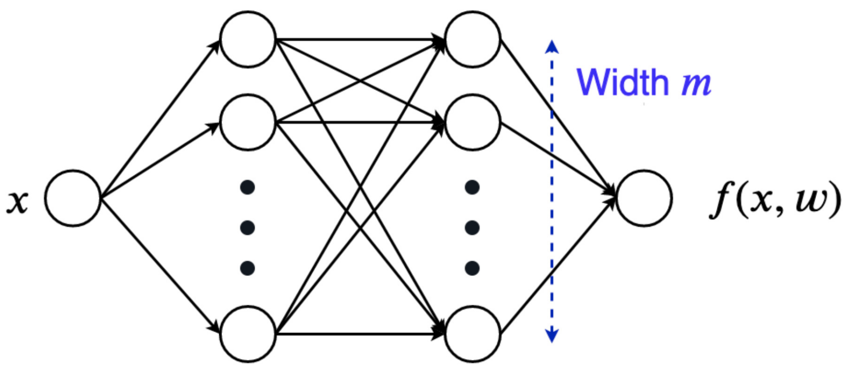


Figure 2: Two-hidden layer neural network having A-ReLU activation functions only, with 1-D input and output. Source: [31]

2. Why does the trained network generalize well to test samples?

Li and Liang showed that more the network is overparameterized, the less likely it is that the activation (sign of input to ReLU) pattern of one neuron for each data point will change in a fixed number of iterations. This allows us to couple the gradient of the true neural network with a "pseudo gradient" where the activation pattern of each neuron for each data point is fixed for some number of iterations (lemma B.1 in paper). Moreover, it can be shown that if the generalization is large, then the "pseudo gradient" will also be large and will result in convergence to a minimum with low generalization error (final theorem stated above).

9.2.7 Polyak-Lojasiewicz (PL) condition

We can approach the same problem from the perspective of Polyak-Lojasiewicz (PL) condition, and its effect on the condition number of the neural tangent kernel[29].

Output layer	Tangent kernel	Hessian norm	PL condition & cond. number	Convergence GD/SGD
linear	nearly constant	$\Theta(1/\sqrt{m})$	$\Theta(1)$	Yes
non-linear	non-constant	$\Theta(1)$	$\Theta(1)$	Yes

9.2.8 Mathematical Setup

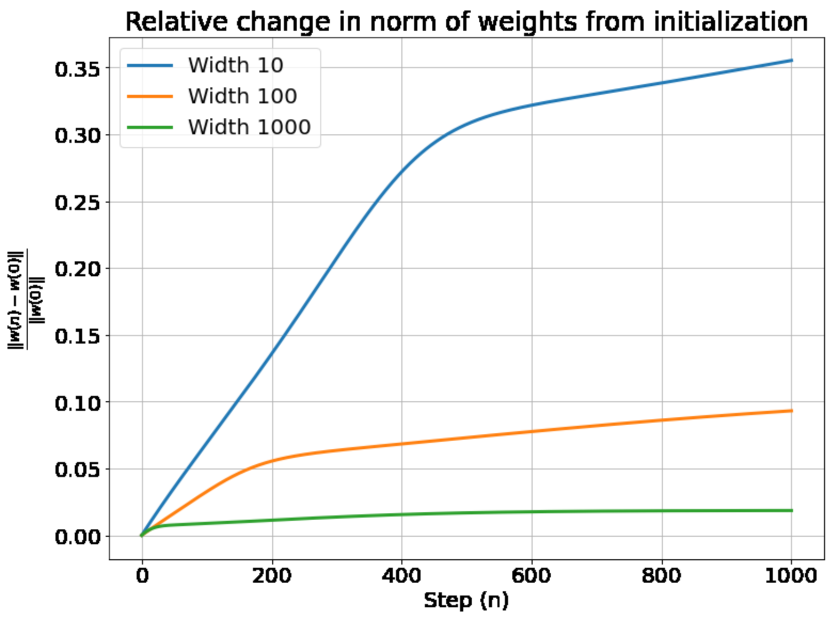
For simplicity, we assume a 1-D input and a 1-D output. We initialise a 2-hidden layer A-ReLU neural network (see Fig. 2), having width m. We use $k = 0.5$ and $n = 1.3$, as per the constraints defined in ([30]).

9.2.9 Relative change in weight-norm

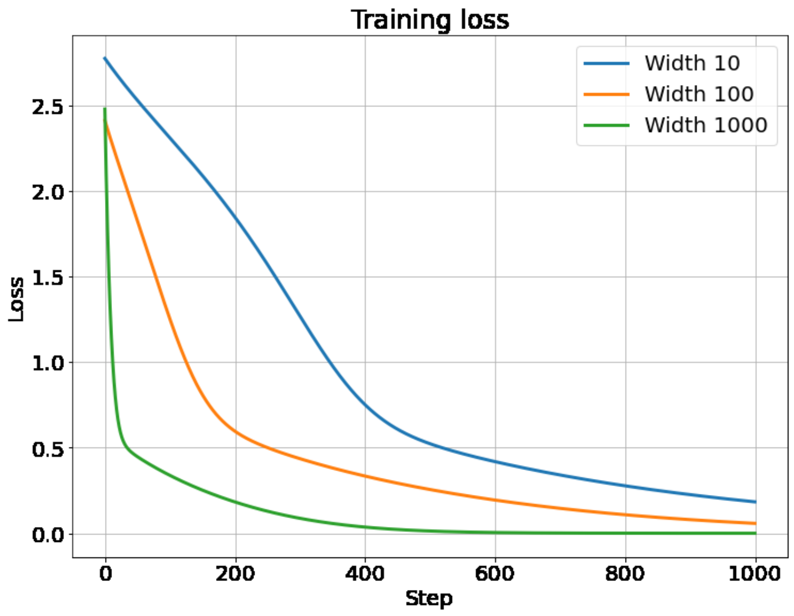
[26] had provided a theoretical proof about majority of weights being constant in a two-hidden layer ReLU network. It's straightforward to run three A-ReLU models with $m = 10, 100$ and 1000 for 1000 iterations and plot $\frac{\|\boldsymbol{w}(n)-\boldsymbol{w}_0\|_2}{\|\boldsymbol{w}_0\|_2}$, which is the relative change in the norm of the weight vector from initialization. As it can be seen, that weights don't change much at all for larger hidden widths.

9.2.10 Convergence of training loss

For the same three models, we also plotted the training loss dynamics in Fig. 3, wherein, it can be clearly seen that overparameterization leads to faster convergence of zero training loss, thus corroborating with most of the studies on overparameterization.

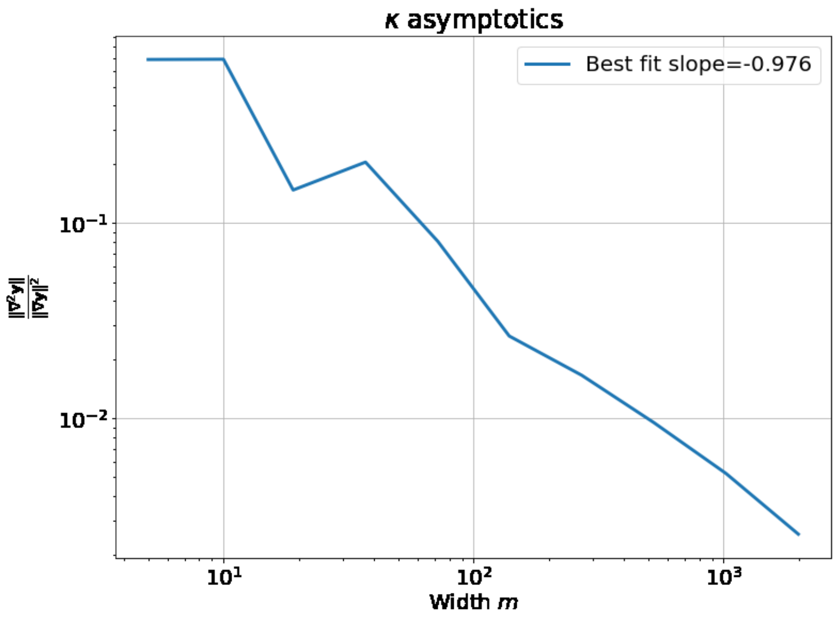


(a) Relative change in the norm of the weight vector from initialization

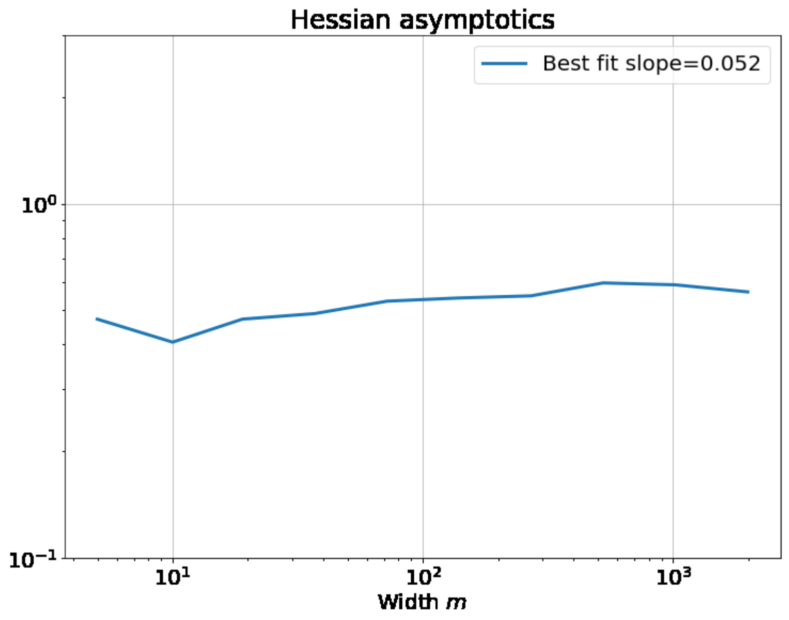


(b) Training loss dynamics. More the overparametrization, faster the convergence.

Figure 3: Three A-ReLU models with widths = $10, 100, 1000$. Rest all of the hyperparameters have been kept fixed.



(a) Change in Tangent kernel with increasing width $\sim \mathcal{O}\left(\frac{1}{m}\right)$



(b) Dynamics of norm of the Hessian $\sim \mathcal{O}(1)$

Figure 4: 1-hidden layer A-ReLU model with varying widths. Rest all of the hyperparameters have been kept fixed. Note that the final output is linear (no activation function used in the output layer)

9.2.11 Proof of Lemma A.1 and A.2

1. **(Lemma A.2)** Say $w_r^{(0)} \sim N(0, \sigma^2)$ and $x_{i,j} \in \mathrm{Re}^d$ such that $\|x_{i,j}\| = 1$ and we can decompose w_r^0 as

$$w_r^0 = \alpha x_{1,1} + \beta$$

where $\beta \perp x_{1,1}$. Then,

$$\alpha \sim N(0, \sigma^2)$$
$$\langle \beta, x_{a,b} \rangle \sim N(0, (1 - \langle x_{a,b}, x_{1,1} \rangle^2)\sigma^2)$$

for any a, b.

Proof: Let $w := w_r^{(0)}$, $x := x_{1,1}$ and $x' := x_{a,b}$ Now

$$w = \alpha x + \beta$$

or

$$\langle w, x \rangle = \alpha \langle x, x \rangle + \langle \beta, x \rangle$$

or

$$\langle w, x \rangle = \alpha$$

or

$$\alpha = \sum_{i=1}^{d} w_i x_i$$

Since $w_i \sim N(0, \sigma^2) \implies w_i x_i \sim N(0, x_i^2 \sigma^2)$. Also, w_i's are i.i.d. random variables. Thus,

$$\alpha = \sum_{i=1}^{d} w_i x_i \sim N(0, \|x\|^2 \sigma^2)$$

or

$$\alpha \sim N(0, \sigma^2)$$

Now,

$$w = \alpha x + \beta$$

or

$$\langle w, x' \rangle = \alpha \langle x, x' \rangle + \langle \beta, x' \rangle$$

or

$$\langle \beta, x' \rangle = \langle w, x' \rangle - \langle w, x \rangle \langle x, x' \rangle$$

or

$$\langle \beta, x' \rangle = \langle w, x' - x\langle x, x' \rangle \rangle$$

Now

$$\|x' - x\langle x, x' \rangle\|^2 = 1 - \langle x, x' \rangle^2$$

which implies

$$\langle \beta, x_{a,b} \rangle = \langle w, x' - \langle x, x' \rangle x \rangle \sim N(0, (1 - \langle x, x' \rangle^2)\sigma^2)$$

2. **(Lemma A.1)** Given $\langle w_r^{(0)}, x_{a,b}\rangle \sim N(0,\sigma^2)$ and
$$\mathcal{H} = \Big\{ r \in [m] | \forall a \in [k], b \in [l] : | \Big\langle w_r^{(0)}, x_{a,b} \Big\rangle | \geq \tau \Big\}$$
, then, $|\mathcal{H}| \geq 1 - \frac{e\tau kl}{\sigma}$.
Proof: By anti-concentration inequality bounds, if $X \sim N(0,\sigma^2)$, then,
$$\Pr(|X| \leq \epsilon) \leq \sup_{x\in\mathcal{R}} \Pr(|X - x| \leq \epsilon)$$
where,
$$\mathcal{L}(X,\epsilon) = \sup_{x\in\mathcal{R}} \Pr(|X - x| \leq \epsilon)$$
Therefore,
$$\mathcal{L}(X,\epsilon) \leq \frac{4\epsilon(a_p + 1)}{\sigma}$$
$$\implies \Pr(|X| \leq \epsilon) \leq \frac{4\epsilon(a_p + 1)}{\sigma}$$
Now, in our case, since $\forall a \in [k], b \in [l] : | \Big\langle w_r^{(0)}, x_{a,b} \Big\rangle | \sim N(0,\sigma^2)$, therefore,
$$a_p = \mathbf{E}[\max_j \frac{X_j}{\sigma_j}] = 0$$
So,
$$\Pr(|X| \leq \epsilon) \leq \frac{4\epsilon}{\sigma}$$
Using this result, we will be deriving the bounds for $\mathcal{H}$ and later on for β too.
Since, $\langle w_r^{(0)}, x_{a,b}\rangle \sim N(0,\sigma^2)$, therefore,
$$\Pr\Big(\langle w_r^{(0)}, x_{a,b}\rangle \leq \tau\Big) \leq \frac{4\tau}{\sigma}$$
By using union-over-bound (Boole's inequality),
$$\Pr\Big(\forall a \in [k], b \in [l] : | \Big\langle w_r^{(0)}, x_{a,b} \Big\rangle | \leq \tau\Big) \leq \frac{4kl\tau}{\sigma}$$
Therefore (since $e < 4$) ($\mathcal{H}$ has m random binomial random variables (corresponding to m neurons), where the $\Pr(X_m) = 1 - \Pr\Big(\forall a \in [k], b \in [l] : | \Big\langle w_r^{(0)}, x_{a,b} \Big\rangle | \leq \tau\Big)$),
$$|\mathcal{H}| \geq 1 - \frac{ekl\tau}{\sigma}$$

3. **(Lemma A.3)** Given $\alpha \sim N(0,\sigma^2)$, show that
$$\Pr[|\alpha| \leq \tau] \geq \frac{\tau}{e\sigma}$$
Proof: By Gauss's inequality,
$$\Pr(|\alpha| > \tau) \leq 1 - \frac{\tau}{\sqrt{3}\sigma}$$
Note: this is valid only if $0 \leq \tau \leq \frac{2\sigma}{\sqrt{3}}$. Since, we are placing a bound over Pr, therefore $0 \leq \frac{\tau}{e\sigma} \leq 1$ and hence, $0 \leq \tau \leq \frac{2\sigma}{\sqrt{3}} \leq e\sigma$
$$\implies \Pr(|\alpha| \leq \tau) \geq \frac{\tau}{\sqrt{3}\sigma}$$
and, $\frac{1}{e} < \frac{1}{\sqrt{3}}$ Therefore,
$$\Pr[|\alpha| \leq \tau] \geq \frac{\tau}{e\sigma}$$

Proposition:

$$\text{If function } F \text{ is } L_F \text{ -Lipschitz, then } \|K(\mathbf{w})\|_2 \leq L_F^2$$

Proof: Since, $K(\mathbf{w}) := \nabla F(\mathbf{w})\nabla F(\mathbf{w})^T$, therefore, $||K(\mathbf{w})|| := ||\nabla F(\mathbf{w})\nabla F(\mathbf{w})^T|| \leq ||\nabla F(\mathbf{w})|| \cdot ||\nabla F(\mathbf{w})^T||$ And, $||\nabla F(\mathbf{w})|| = ||\nabla F(\mathbf{w})^T||$, therefore,

$$||K(\mathbf{w})|| \leq ||\nabla F(\mathbf{w})|| \cdot ||\nabla F(\mathbf{w})||$$

If F is Lipschitz continuous, then $\|\nabla F(\mathbf{w})\|_2 \leq L_F$ for all $\mathbf{w} \in \mathbb{R}^m$

$$\implies ||K(\mathbf{w})|| \leq L_F^2$$

Hessian matrix of the loss function:

$$H_{\mathcal{L}}(\mathbf{w}) = \underbrace{\nabla F(\mathbf{w})^T \frac{\partial^2 \mathcal{L}}{\partial F^2}(\mathbf{w}) \nabla F(\mathbf{w})}_{A(\mathbf{w})} + \underbrace{\sum_{i=1}^{n} \left(\frac{\partial \mathcal{L}}{\partial F}(\mathbf{w}) \right)_i H_i(\mathbf{w})}_{B(\mathbf{w})}$$

References

[1] S. Saha, T. Prashanth, S. Aralihalli, S. Basarkod, T. S. B. Sudarshan, S. S. Dhavala, LALR: theoretical and experimental validation of lipschitz adaptive learning rate in regression and neural networks, CoRR abs/2006.13307 (2020). arXiv:2006.13307.
URL https://arxiv.org/abs/2006.13307

[2] M. Alghalith, A note on taylor expansions without the differentiability assumption, Australian Journal of Mathematical Analysis and Applications (02 2019).

[3] Derivation of backpropagation, https://www.cs.swarthmore.edu/~meeden/cs81/s10/BackPropDeriv.pdf, [Online; accessed 17-May-2021] (2010).

[4] M. A. Lab, Bean disease dataset (January 2020).
URL https://github.com/AI-Lab-Makerere/ibean/

[5] Z. Zhou, Q. Zhang, G. Lu, H. Wang, W. Zhang, Y. Yu, Adashift: Decorrelation and convergence of adaptive learning rate methods, CoRR abs/1810.00143 (2018). arXiv:1810.00143.
URL http://arxiv.org/abs/1810.00143

[6] X. Wu, R. Ward, L. Bottou, Wngrad: Learn the learning rate in gradient descent (2018). arXiv:1803.02865.

[7] S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, B. B. Chaudhuri, diffgrad: An optimization method for convolutional neural networks, IEEE Transactions on Neural Networks and Learning Systems (2020) 1–12doi:10.1109/tnnls.2019.2955777.
URL http://dx.doi.org/10.1109/TNNLS.2019.2955777

[8] T. Hatanaka, T. Korenaga, N. Kondo, K. Uosaki, Search Performance Improvement for PSO in High Dimensional Space, 2009. doi:10.5772/6752.

[9] A. Gopal, M. M. Sultani, J. C. Bansal, On stability analysis of particle swarm optimization algorithm, Arabian Journal for Science and Engineering 45 (4) (2020) 2385–2394. doi:10.1007/s13369-019-03991-8.
URL https://doi.org/10.1007/s13369-019-03991-8

[10] J. G. Charney, R. FjÖrtoft, J. V. Neumann, Numerical integration of the barotropic vorticity equation, Tellus 2 (4) (1950) 237–254. arXiv:https://doi.org/10.3402/tellusa.v2i4.8607, doi:10.3402/tellusa.v2i4.8607.
URL https://doi.org/10.3402/tellusa.v2i4.8607

[11] R. Warming, B. Hyett, The modified equation approach to the stability and accuracy analysis of finite-difference methods, Journal of Computational Physics 14 (2) (1974) 159 – 179. doi:https://doi.org/10.1016/0021-9991(74)90011-4.
URL http://www.sciencedirect.com/science/article/pii/0021999174900114

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

[13] M. Jamil, X. S. Yang, A literature survey of benchmark functions for global optimisation problems, International Journal of Mathematical Modelling and Numerical Optimisation 4 (2) (2013) 150. doi:10.1504/ijmmno.2013.055204.
URL http://dx.doi.org/10.1504/IJMMNO.2013.055204

[14] A. Theophilus, S. Saha, S. Basak, 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. doi:10.1109/SSCI.2018.8628669.
URL https://doi.org/10.1109/SSCI.2018.8628669

[15] H. Garg, A hybrid pso-ga algorithm for constrained optimization problems, Applied mathematics and conputation 274 (2015) 292–305.

[16] K. Bora, S. Saha, S. Agrawal, M. Safonova, S. Routh, A. Narasimhamurthy, Cd-hpf: New habitability score via data analytic modeling, submitted to Astronomy and Computing 17 (04 2016). doi:10.1016/j.ascom.2016.08.001.

[17] S. Saha, S. Basak, M. Safonova, K. Bora, S. Agrawal, P. Sarkar, J. Murthy, Theoretical validation of potential habitability via analytical and boosted tree methods: An optimistic study on recently discovered exoplanets, Astronomy and Computing 23 (2018) 141–150. doi:10.1016/j.ascom.2018.03.003.
URL http://dx.doi.org/10.1016/j.ascom.2018.03.003

[18] C. W. Cobb, P. H. Douglas, A theory of production, The American Economic Review, Vol. 18, No. 1, Supplement, Papers and Proceedings of the Fortieth Annual Meeting of the American Economic Association (Mar., 1928), pp. 139- 165.

[19] A. Méndez, A thermal planetary habitability classification for exoplanets.planetary habitability laboratory @ upr arecibo (2011).
URL http://phl.upr.edu/library/notes/athermalplanetaryhabitabilityclassificationforexoplanets

[20] J. Ren, S. Yang, A particle swarm optimization algorithm with momentum factor 1 (2011) 19–21. doi:10.1109/ISCID.2011.13.

[21] Y. X. Gandomi A.H., Benchmark problems in structural optimization 356 (2011). doi:https://doi.org/10.1007/978-3-642-20859-1_12.

[22] S. Sridhar, S. Saha, A. Shaikh, R. Yedida, S. Saha, Parsimonious computing: A minority training regime for effective prediction in large microarray expression data sets, CoRR abs/2005.08442 (2020). arXiv:2005.08442.
URL https://arxiv.org/abs/2005.08442

[23] R. Yedida, S. Saha, A novel adaptive learning rate scheduler for deep neural networks, CoRR abs/1902.07399 (2019). `arXiv:1902.07399`.
URL `http://arxiv.org/abs/1902.07399`

[24] C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, Understanding deep learning requires rethinking generalization, arXiv preprint arXiv:1611.03530 (2016).

[25] D. Zou, Y. Cao, D. Zhou, Q. Gu, Stochastic gradient descent optimizes over-parameterized deep relu networks. arxiv e-prints, art, arXiv preprint arXiv:1811.08888 (2018).

[26] Y. Li, Y. Liang, Learning overparameterized neural networks via stochastic gradient descent on structured data, in: Advances in Neural Information Processing Systems, 2018, pp. 8157–8166.

[27] S. Arora, S. S. Du, W. Hu, Z. Li, R. Wang, Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks, arXiv preprint arXiv:1901.08584 (2019).

[28] M. Belkin, D. Hsu, J. Xu, Two models of double descent for weak features, CoRR abs/1903.07571 (2019). `arXiv:1903.07571`.
URL `http://arxiv.org/abs/1903.07571`

[29] C. Liu, L. Zhu, M. Belkin, Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning, arXiv preprint arXiv:2003.00307 (2020).

[30] S. Saha, Anonymous, Diffact: Leveraging differential equations in activation functions for computer vision data sets, In WACV review (2021).

[31] R. V. Dwaraknath, Understanding the neural tangent kernel (Nov 2019).
URL `https://rajatvd.github.io/NTK/`