BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 71(6), 2023, Article number: e147924
DOI: 10.24425/bpasts.2023.147924

A hybrid model of heuristic algorithm and gradient descent to optimize neural networks

Amer MIRKHAN[1]* and **Numan ÇELEBI**[2]

[1] Sakarya University, Computer Engineering Department
[2] Sakarya University, Information Systems Engineering Department

Abstract. Training a neural network can be a challenging task, particularly when working with complex models and large amounts of training data, as it consumes significant time and resources. This research proposes a hybrid model that combines population-based heuristic algorithms with traditional gradient-based techniques to enhance the training process. The proposed approach involves using a dynamic population-based heuristic algorithm to identify good initial values for the neural network weight vector. This is done as an alternative to the traditional technique of starting with random weights. After several cycles of distributing search agents across the search domain, the training process continues using a gradient-based technique that starts with the best initial weight vector identified by the heuristic algorithm. Experimental analysis confirms that exploring the search domain during the training process decreases the number of cycles needed for gradient descent to train a neural network. Furthermore, a dynamic population strategy is applied during the heuristic search, with objects added and removed dynamically based on their progress. This approach yields better results compared to traditional heuristic algorithms that use the same population members throughout the search process.

Key words: optimization; heuristic algorithms; neural networks; dynamic population.

1. INTRODUCTION

Neural networks (NN) can be considered the best method for solving artificial intelligence problems including image classification, object detection, voice recognition, etc. However, training an NN is a long process and requires a considerable amount of time and computing resources. During the training process, which is an iterative process, an optimizer function should be used in each training cycle to update the model parameters, which can be called a weight vector. In the literature, the most common way to train neural networks is using gradient-based optimizers. Those optimizers simply perform steps within the solution space in order to improve the current parameters. This process of updating the weights is critical for NN training and keeps a considerable place in the literature. The main challenge of traditional optimization techniques is time complexity because, in each iteration, the algorithm computes gradients (derivatives) of the loss (objective) function with respect to each model parameter.

Heuristic algorithms [1, 2] are used to find an approximate solution when traditional algorithms fail to find the exact one within time and computing resource constraints. Swarm-based optimization, also called swarm intelligence or population-based techniques, is the most popular heuristic algorithm. The behavior of those algorithms is inspired by nature where elements of the swarm, which can also be called objects or search agents, are first distributed randomly in the search space, then iteratively will be moving trying to find the optimal solution inside the domain of solutions, which is usually very wide or even infinite. The population-based heuristic algorithms are static in terms of individuals of the population which means equal opportunity will be given to each member to find the solution regardless of its performance during the search process. In other words, objects that perform well and are close to a good solution will be given the same chance as the ones searching in areas far from a good solution.

Swarm-based heuristic algorithms usually have three components in common (local search, global search, and objective/fitness function). The implementation of each component varies from one algorithm to another. The local search is the effort of each element to find a better solution in its neighborhood while the global search is a kind of communication among the swarm members after each cycle where all members have completed their own local search, then members will be moving toward the best solution obtained so far with different strategies to avoid trapping in local optimum. An objective function is a function that can evaluate a solution and return a value to be minimized or maximized. For instance, when training a neural network, the fitness function could be the training accuracy that we need to maximize, or could be the loss function that should be minimized.

2. BACKGROUND AND RELATED WORK

In this section, we give details about the gradient descent-based learning algorithms since we compare the performance of our algorithm with them in addition to some alternatives to the gra-

*e-mail: amermir@gmail.com

Manuscript submitted 2023-04-27, revised 2023-10-09, initially accepted for publication 2023-10-16, published in December 2023.

dient descent such as random and grid search. Additionally, we give place to heuristic algorithms, which are derivative-free techniques as we are proposing to combine both techniques in this research.

2.1. Gradient descent

Gradient descent [3] is the most common way of training NNs, which simply aims to minimize the objective (loss) function by updating the parameters of an NN in the opposite direction of the objective function, through making steps -which is the learning rate – in order to reach the (local) minimum. Choosing a small learning rate will result in obtaining accurate results but learning time will be longer due to small steps, especially when dealing with complicated models and a high amount of training data. On the other hand, increasing the learning rate will lead to faster learning but might cause a loss of the optimal result due to bigger jumps. In the literature, several papers were written in this regard to specify the optimum learning rate. Kingma *et al*. [4], introduced Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Adam became the most popular gradient-based optimizer for training NNs, and in our experiments, we will be using Adam to represent the gradient-based optimizers. Nesterov [5] proposed the Nesterov accelerated gradient, which is also known as NAG, as a way to give the momentum term a kind of prescience. Dozat [6] combined Adam and NAG within a new technique called Nesterov-accelerated Adaptive Moment Estimation (Nadam). Duchi [7] proposed Adagrad, another stochastic gradient descent (SGD) based algorithm by adapting the learning rates to the parameters which showed good performance with sparse data. Adagrad was extended by Adadelta [8] searching to update the learning rate more smoothly than Adagrad.

To overcome the limitations of gradient optimizations, mainly slow convergence and the need for long training cycles, a considerable amount of literature has explored other techniques to replace gradient descent or even hybrid models along with gradient descent. Liashchynskyi *et al.* [9], compared optimizing the parameters using three techniques, depending on the number of parameters to be optimized, for small NNs grid search (brute force) can be used where the entire search space will be scanned. However, this might not be possible for bigger networks, where random search can perform better but without a guarantee of finding the optimal solution, and for even bigger networks, where random algorithms are not able to find (guess) the solution, genetic algorithms were evaluated as we can control the number of generations and the population length. So according to the given time/computing constraints, those parameters can be set. Similar research [10,11] also proved that random search can perform better than grid and sequential search while the performance is not affected much by increasing the number of NN layers/parameters. The restart technique was evaluated by [12–15] using similar algorithms, with minor differences, by evaluating the performance of the NN after a few gradient steps and terminating the process when poor performance is detected and starting over. This technique showed six to seven times better performance than grid/random search. In [16] a new derivative-free parallel optimization of hyperparameters was proposed and showed satisfactory results in terms of the needed computational cost. Similarly, [17] using an NN with 5 million parameters could train the FashionMNIST dataset using a derivative-free optimizer and obtained 100% training accuracy and 84% validation accuracy which outperformed SGD.

2.2. Population-based heuristic algorithms

Population-based heuristic algorithms are biologically inspired and make no assumptions on the optimization landscape. These methods can be also called black-box optimization methods and can also be considered as derivative-free optimization [18]. Additionally, they are powerful techniques to be used in solving complex and non-linear optimization problems. Because many starting points (candidate solutions) are used, the chances of converging on the global minima are significantly increased [3]. Furthermore, there are several studies, which used heuristic algorithms to train NNs and possibly outperform gradient-based optimizers in several cases. Results were summarized by [19] where the most common heuristic algorithms were evaluated in training NN with a summary of the advantages and drawbacks of each used algorithm. Authors of [20–23] proposed using the particle swarm optimization (PSO) algorithm as an alternative for gradient optimization in solving real engineering applications and got promising results when hardware resources are limited. Similar research [24] showed consistent improvement in accuracy, training time, and stability when using population-based training techniques to solve several classifications and reinforcement learning problems. Evolutionary algorithms were evaluated in [25] for the same purpose which utilized the past results abilities of genetic algorithms in narrowing the search space in future cycles. The researcher in [26] proposed a method using the ant colony algorithm to train feed-forward NNs to avoid trapping in local optima with gradient descent. Moreover, [27] combines traditional SGD with a population-based evolutionary strategy in a framework named ESGD. Additionally, [28] proved that simple, gradient-free, population-based genetic algorithms (GA) can perform well on hard deep-learning problems. In [29], using evolutionary algorithms instead of gradient descent to train neural networks is criticized. Hybrid models combining two heuristic algorithms were also evaluated; [30] HACPSO proposed a combination of the cuckoo search (CS) algorithm with PSO which resulted in improving the convergence rate.

In contrast to most of the population-based heuristic algorithms, like particle swarm optimization (PSO) [31], ant colony optimization [32], cuckoo search algorithm (CSA) [33], firefly algorithm (FA) [34], and artificial bee colony (ABC) [35], dynamic population strategy is used, inspired by the one proposed by polar bear optimization (PBO) [36], where population members are not fixed. Instead, according to some parameters iteratively death and re-birth techniques are applied to allow better performance and faster convergence. The dynamic population strategy proposed by [36] was utilized in several optimization research. [38] proposed using the PBO for feature selection of the rough set. [39] also used the PBO for loading pattern optimization.

3. THE PROPOSED MODEL

To find the optimal values for the model parameters, a hybrid model of a derivative-free technique based on dynamic population optimization (DPO) is proposed in addition to gradient-based traditional techniques. Those gradient-based techniques update the weights based on the calculated error, trying to minimize this error iteratively. The error is simply the difference (distance) between the desired output $\hat{y}$ and the model output y. Several cost functions can be used to compute the error, for instance, if the mean square function [40] is used then the cost function will look as follows:

$$c = \frac{1}{n}\sum_{p}^{n}\sum_{k}^{m}\left(\hat{y}_{pk} - y_{pk}\right)^2, \tag{1}$$

where n is the number of training samples and m the number of output nodes. After calculating the error, weights will be updated by computing the derivative for all network's parameters:

$$\Delta_w = -\eta \frac{\mathrm{d}C}{\mathrm{d}w}, \tag{2}$$

where η is the learning rate and w represents the weight vector. Gradient-based techniques are powerful in exploring the neighborhood, but they tend to fall in local optima, according to [40], local optima can be formally defined as:

- Let $c_f : S \to R \geq 0$, where $S \subset R^n$ is nonempty and [41], c_f is the cost function and S represents a list of possible solutions.
- A point $w^* \in S$ is called global minima if $c_f(w^*) \leq c_f(w)$ for any $w \in S$ holds.
- A point $w^* \in S$ is called local minima if there exists $\varepsilon > 0$ and ε-neighborhood $B_\varepsilon(w^*, \varepsilon)$ around w^* such that $c_f(w^*) \leq c_f(w)$ for any $w \in S \cap B_\varepsilon(w^*, \varepsilon)$ holds.

To enhance the use of conventional methods by adding exploratory capability, our proposal is to train the neural network in two phases: the initialization phase, and the training phase. The initialization phase can be simply summarized by quickly exploring the search space using a heuristic algorithm in order to find a good starting point for the next training phase. The gradient-based optimization techniques are sequential based, searching for the optimal solution iteratively usually starting from a random point in the search space. The selection of a starting point might seriously impact the entire performance and can lead to a trap in a local optimum. To overcome these challenges, we first explore the search domain trying to find a good starting point. Each member of the population P will represent the weight vector of an NN:

$$P = \{p1, p2, p3, \ldots, p_n\}, \tag{3}$$

$$w = (w_1, w_2, \ldots, w_m), \tag{4}$$

where n and m are the population size and the number of weights to be tuned, respectively. The initialization phase in our model starts by randomly distributing the solution set P in the search space, then each object will have a chance to explore its neighborhood looking for a better solution. Solutions will be evaluated according to equation (5)

$$\text{Fitness} = \frac{1}{1+c}. \tag{5}$$

After completing each cycle, where each object has made or tried a step toward the target, the dynamic population strategy mentioned in [36] is applied. The entire population is evaluated after each cycle and based on a random parameter. An object is either removed or a new one is created. If the decision was to remove, then the worst solution is pulled out from the population to avoid investigating more in a hopeless piece of the search domain. Figure 1 shows an example of solutions, white dots in the search space looking for the optimal solution (dark blue). According to our dynamic strategy, the solution on the most left should be dismissed as it is very far from the target while the one close to the optimal solution will be duplicated. The need for this dynamic behavior is that we are dealing with a high-dimensional search space, and it is nearly impossible to have objects covering the entire space. So, this strategy allows covering more areas with fewer search agents, this will be discussed in detail within the experimental analysis.

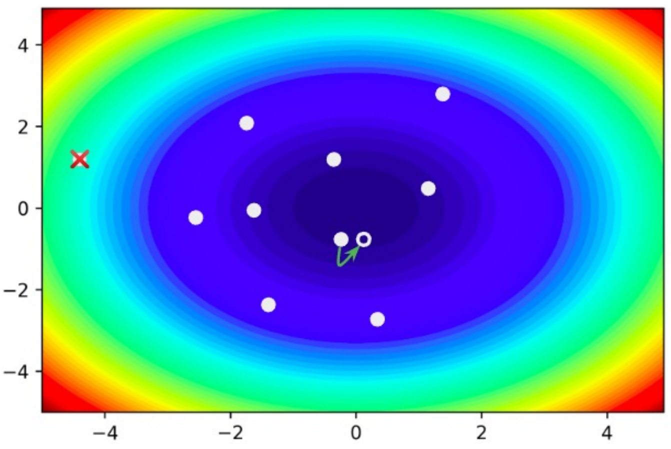

Fig. 1. Dynamic population strategy

To avoid trapping in the local optimum the best solution is not duplicated and the worst at each cycle is not removed. Instead, a random variable $k \in \{0, 1\}$ is generated after each cycle. If the parameter value is less than 0.75 then the decision will be to duplicate otherwise to remove.

$$\begin{aligned} &\text{Reproduction} && \text{if } 0 \leq k \leq 0.75, && \text{(6a)}\\ &\text{Remove} && \text{if } 0.75 < k \leq 1. && \text{(6b)} \end{aligned}$$

Once the initialization phase finishes, the best solution (highest accuracy) found as a starting point for the training process will use the traditional gradient-based techniques. Accuracy (same as fitness) will be calculated according to equation (5). Due to very high dimensional search space, the heuristic algorithm is not expected to find the optimal values for the weight vector like several studies that tried to fully replace the gradient-based optimizers with heuristic ones. Instead, we propose that the heuristic algorithm will explore the search space at all speeds and will deliver a good starting point for the gradient-based optimizer to perform the actual NN training. We also believe that (and that will be supported later by experiments) the dynamic population strategy will facilitate exploring a big search domain using a limited number of search agents.

The algorithm expects three main input parameters, population size, cycle count (epochs), and heuristic ratio that will regulate the distributions of training cycles between heuristic and traditional techniques. More objects used increase the likelihood of finding the optimal values for the weight vector but might negatively impact the performance. After initialization, for each potential solution, the algorithm loops through a certain number of loops. The number of iterations is another input parameter and should also be selected carefully. The higher number of iterations more likely increases the probability of moving closer to the optimal solution, i.e. local search. However, it also increases the runtime of the algorithm. In each cycle, each object explores to make a step by looking at the right and left sides, which are represented by positive values in equation (8) on the right, while negative values are on the left. Then, both generated possible solutions will be evaluated and will only move if a better solution is produced. Algorithm-1 and Fig. 2 summarize our dynamic population optimization approach.

Algorithm 1. Dynamic Population Optimization – DPO

```
Requirement: Model: Neural Network Model to
Optimize. S, N: Population Size and Max Iterations.
h: heuristic ratio
 1:  randomly initialize S objects
 2:  calculate heuristic cycles H according to (9)
 3:  while Loop Counter < H do
 4:     for all Population Members do
 5:        Generate new solutions according to
           equations (7), (8)
 6:        Evaluate a new solution according to
           equation (5) and move if better
 7:        Generate random value $\phi$
 8:        If $\phi$ < 0.75
 9:               Duplicate the best solution
10:        else
11:               Remove the worst Solution
12:        select the best model M
13:        calculate gradient cycles G according to
     (10)
14:  End While
15:  continue the training using gradient descent
     with G cycles starting with initial weights M
```

In each iteration, values of the weight vector are updated according to equation (7) and equation (8) as proposed by [36] a kind of local search starting from the current position exploring the neighborhood.

$$r = 4a\cos\theta_0 \sin\theta_0, \tag{7}$$

$$w_{\text{new}} = w_{\text{actual}} \pm \left[r\sin(\phi_k) + r\cos(\phi_j)\right], \tag{8}$$

where $a \in \{0, 0.3\}$ is a random value that regulates the distance in which an object can see, and $\theta \in \left\{0, \frac{\pi}{2}\right\}$ the angle of the movement. k and j are the angular values selected at random for each point $\in \{0, 2\pi\}$. In this context, the word "Point" represents the current solution, and moving to a new point means the trial to generate a better solution, for more details about the moving function [36] was used.

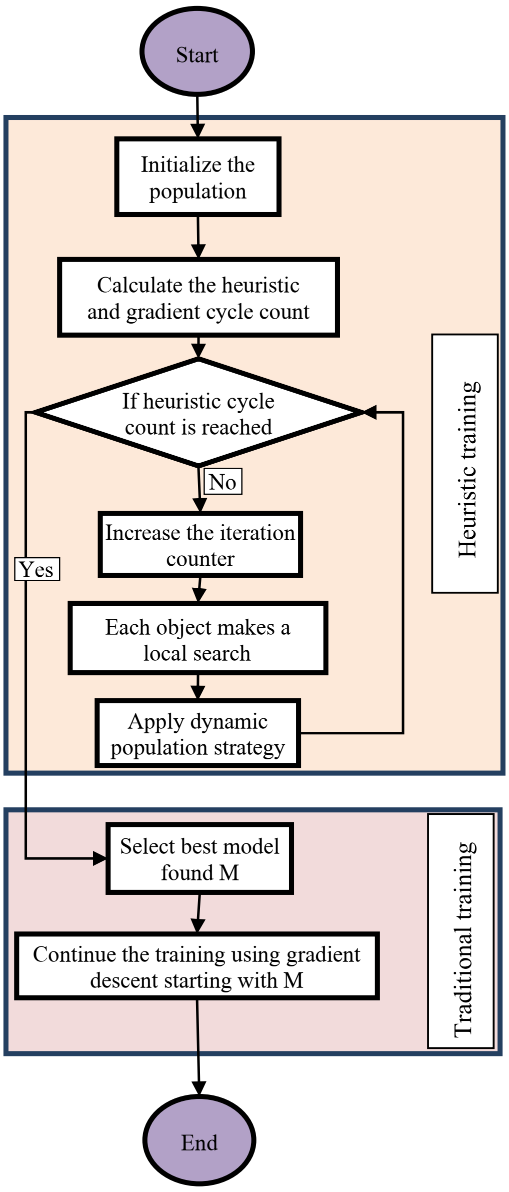
Start
Initialize the population
Calculate the heuristic and gradient cycle count
If heuristic cycle count is reached
No
Yes
Increase the iteration counter
Each object makes a local search
Apply dynamic population strategy
Heuristic training
Select best model found M
Continue the training using gradient descent starting with M
Traditional training
End

Fig. 2. Procedure of DPO

After completing a cycle where all members find the chance to enhance their current positions, the dynamic population strategy is applied. In lines 7–11 of Algorithm 1, based on a randomly generated variable, the decision will be made whether to reproduce or to remove. If the value of the random variable is less than 0.75, the decision is reproduction or otherwise to remove. Here, the value 0.75 means randomly the reproduction rate to be three times of removing decision, this ratio is fixed in this study but can be an input parameter optionally.

The last parameter is the heuristic ratio, this parameter will distribute the training cycles between the heuristic algorithm H and the traditional training G:

$$H = \frac{h_{\text{ratio}}}{100} * N, \tag{9}$$

$$G = N - H. \tag{10}$$

In other words, this parameter decides how much time will be spent on initialization, so setting this parameter to 0 means no initialization and only gradient descent will be used while setting it to 100 means fully heuristic training without traditional training.

4. EXPERIMENTAL ANALYSIS

To evaluate the proposed algorithm, five benchmark datasets listed in Table 1 were selected. For each dataset, the number of layers and the number of trainable parameters can be seen. The number of trainable parameters is very important from the performance point of view as it represents the length of the weight vector and determines the size of the solution space. For each dataset, a model performing well after several experiments was selected. However, our approach is not limited to any architecture, as mentioned in Algorithm-1; the model to be optimized is an input for the algorithm without any limitation regarding the number of layers or the number of trainable parameters. The datasets were split into training and test sets, number of samples allocated for testing is mentioned also in Table 1. These split ratios (train/test) were selected as they are widely accepted in the machine-learning community and have been used in numerous studies. They are considered reasonable because they strike a balance between having a substantial training dataset and a representative test dataset.

Because outcomes can potentially be influenced by various random factors, including initial weight settings and the generation of random variables during solution creation, to ensure a fair comparison, the same initial weights were used for all experiments on the same model, and the same seed when generating random values.

The training process was executed for all datasets using 20, 30, and 40 epochs, and measured the obtained accuracy. The balance between whether to use traditional gradient descent, heuristic approach, or hybrid model can be adjusted by setting the parameter heuristic ratio. In the experimental analysis, the values (0, 20, 50, 70, and 100) were evaluated, where 0 means no heuristic initialization but only gradient optimization, while setting to 100 will only use a heuristic approach for optimization. Other values (20, 50, and 70) will be hybrid models by using a heuristic algorithm for initialization and to continue the training process using gradient descent. For instance, when running 40 epochs and the heuristic ratio is 20, it means that 8 cycles will be used by the heuristic algorithm for initialization while the remaining 32 cycles will be used by gradient descent. The optimizer Adam [4] was used to represent the gradient-based optimization approach as it outperforms [37] and [4] similar gradient-based optimizers in terms of performance. However, in the following analysis, SGD will be mentioned as the base of the gradient-based because any other gradient-based optimizer can also replace Adam in our model.

Table 1
List of used datasets

Dataset	Layers	Parameters	Features	Total instances	Test instances	Output
Iris	3	55	5	150	30	3
Mnist	7	1 256 080	784	60 000	10 000	10
Cifar-10	7	1 632 080	1024	60 000	10 000	10
Cifar-100	8	3 042 546	1024	60 000	10 000	100
Fashion	7	1 163 330	784	70 000	10 000	10

From Table 2 and charts in Fig. 3, it can be seen that the algorithm at a heuristic ratio equal to 20 performed very well across all datasets and almost for all numbers of cycles while increasing this ratio to more than 50 did not give better results. Comparing the results obtained for a heuristic ratio of 0 and 20 can tell us that quickly exploring the search domain and continuing with SGD will give optimal results. Since traditional SGD-based optimizers start from a random point, results proved that investing a few cycles in finding a good initial point resulted in better results.

Table 2
Accuracy per dataset/epoch/heuristic ratio

		Heuristic ratio				
Epoc	Dataset	0	20	50	70	100
20	Mnist	0.953	0.974	0.967	0.893	0.806
30	Mnist	0.967	0.984	0.979	0.914	0.860
40	Mnist	0.994	0.995	0.995	0.953	0.894
20	CIFAR10	0.714	0.759	0.747	0.660	0.413
30	CIFAR10	0.807	0.848	0.748	0.694	0.518
40	CIFAR10	0.859	0.897	0.884	0.837	0.649
20	CIFAR100	0.618	0.654	0.542	0.494	0.453
30	CIFAR100	0.737	0.777	0.698	0.640	0.582
40	CIFAR100	0.924	0.953	0.815	0.868	0.628
20	IRIS	0.618	0.620	0.653	0.647	0.721
30	IRIS	0.737	0.720	0.734	0.751	0.769
40	IRIS	0.823	0.848	0.857	0.860	0.868
20	FASHION	0.858	0.894	0.879	0.758	0.404
30	FASHION	0.914	0.937	0.885	0.774	0.423
40	FASHION	0.925	0.931	0.913	0.851	0.474

It can be concluded that the algorithm is effective in initializing the training process of an NN, experiments showed that

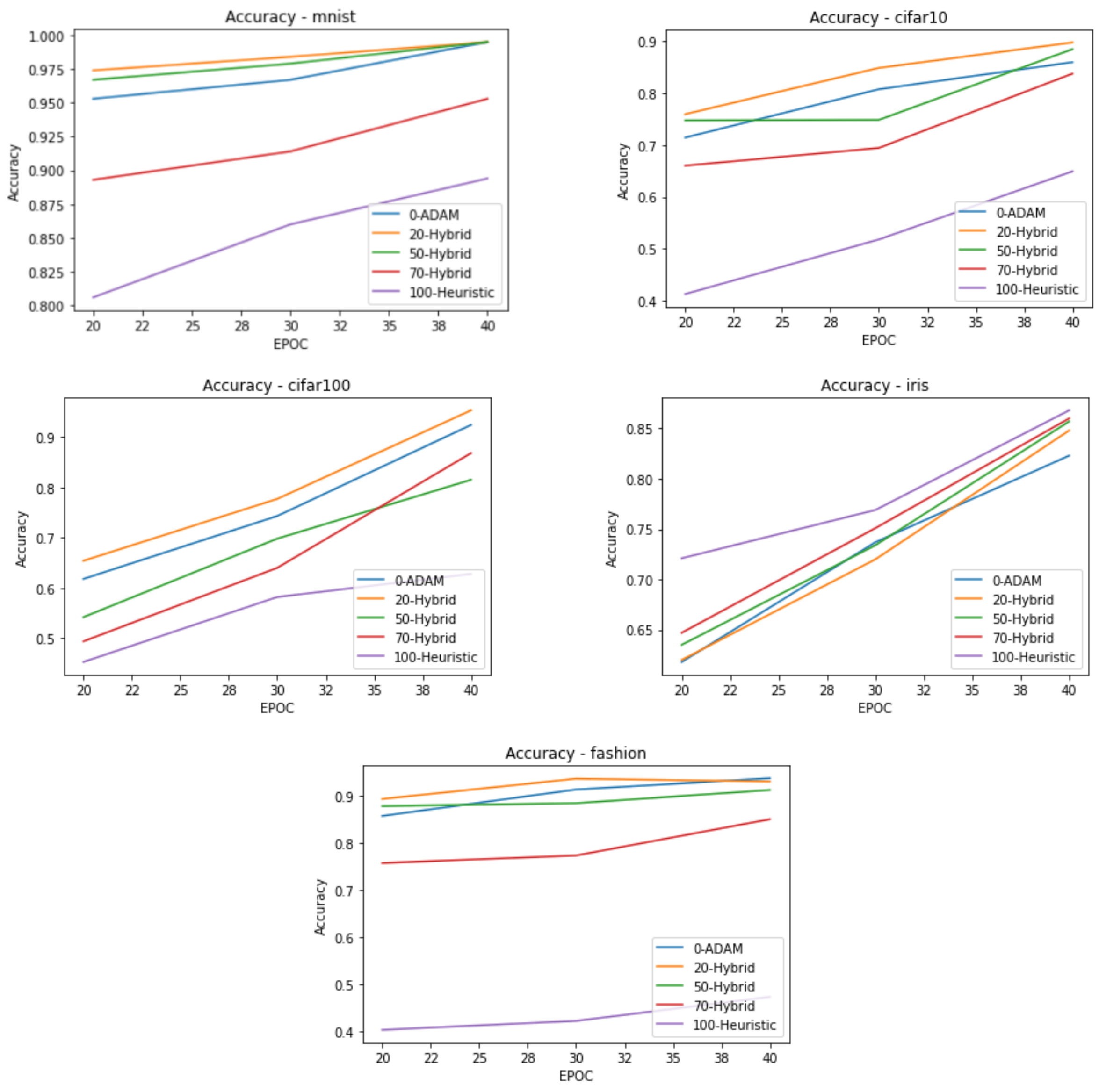

Fig. 3. Accuracy per dataset according to heuristic ratio

allocating around 20 percent of the training time before starting the traditional training will result in higher accuracy and faster convergence.

SGD is very powerful in local search because it depends on the derivation of the loss function to decide the direction to move. But it may fall in local optimums, especially when starting with a bad solution, although modern SGD-based algorithms are not suffering from the local optima dilemma, and they still need long training cycles to achieve high accuracy.

It can also be noticed that using the heuristic ratio of 100 produced bad results except for the Iris dataset, and this can be due to the complex models used, consisting of a very high number of parameters to be optimized, which resulted in a huge number of combinations that cannot be traversed using stochastic methods.

According to [42], the time complexity of training a model is mainly based on the number of epochs, number of training instances, and number of parameters to be optimized. The experiments listed in Table 2 are performed on the same model and same dataset, so when applying the hybrid approach the same accuracy with the same number of training cycles can be obtained. Then it can be concluded that our approach is more efficient in terms of time complexity, as the number of parameters and number of training samples is considered constant in all runs.

The explained experiments took place with a population size equal to 20. However, this population size is an important input parameter for the algorithm and should be selected carefully because it affects both the runtime and the obtained accuracy. We believe that this parameter should be selected according to the number of parameters in the model. Increasing the number of parameters will increase the search domain and will require more agents to explore the search space, to prove that more detailed experiments were needed.

Table 3
Accuracy per dataset/population size/heuristic ratio

	Population size				
Dataset	10	20	40	60	80
Mnist	0.884	0.901	0.965	0.985	0.994
CIFAR10	0.756	0.850	0.897	0.902	0.907
CIFAR100	0.827	0.903	0.953	0.983	0.994
IRIS	0.815	0.841	0.848	0.851	0.853
FASHION	0.842	0.842	0.931	0.937	0.943

The best obtained heuristic rate at 20 was selected from the previous experiments and the algorithm was tested at different values for population size at 10, 20, 40, 60, and 80. Results are represented in Table 3 and Fig. 4, where the obtained accuracy was registered for each data set and population size by having the heuristic ratio fixed at 20.

The algorithm is trying to find a good initialization point in a very high dimensional search space within a limited number of iterations and search agents. We believe that the algorithm performs well and can outperform similar heuristic algorithms. Due to the dynamic population feature which facilitates exploring the search domain efficiently by keeping focused on promising areas and not wasting time in hopeless ones. To prove that another experiment was done by selecting the best values for the input parameters obtained by the previously discussed experiments and then the same algorithm was run without applying the dynamic population part of the algorithm, lines from 7 to 11 of Algorithm-1, results listed in Table 4 showed that disabling this part of the algorithm resulted in lower accuracy due to ineffective technique in navigating the very big search space.

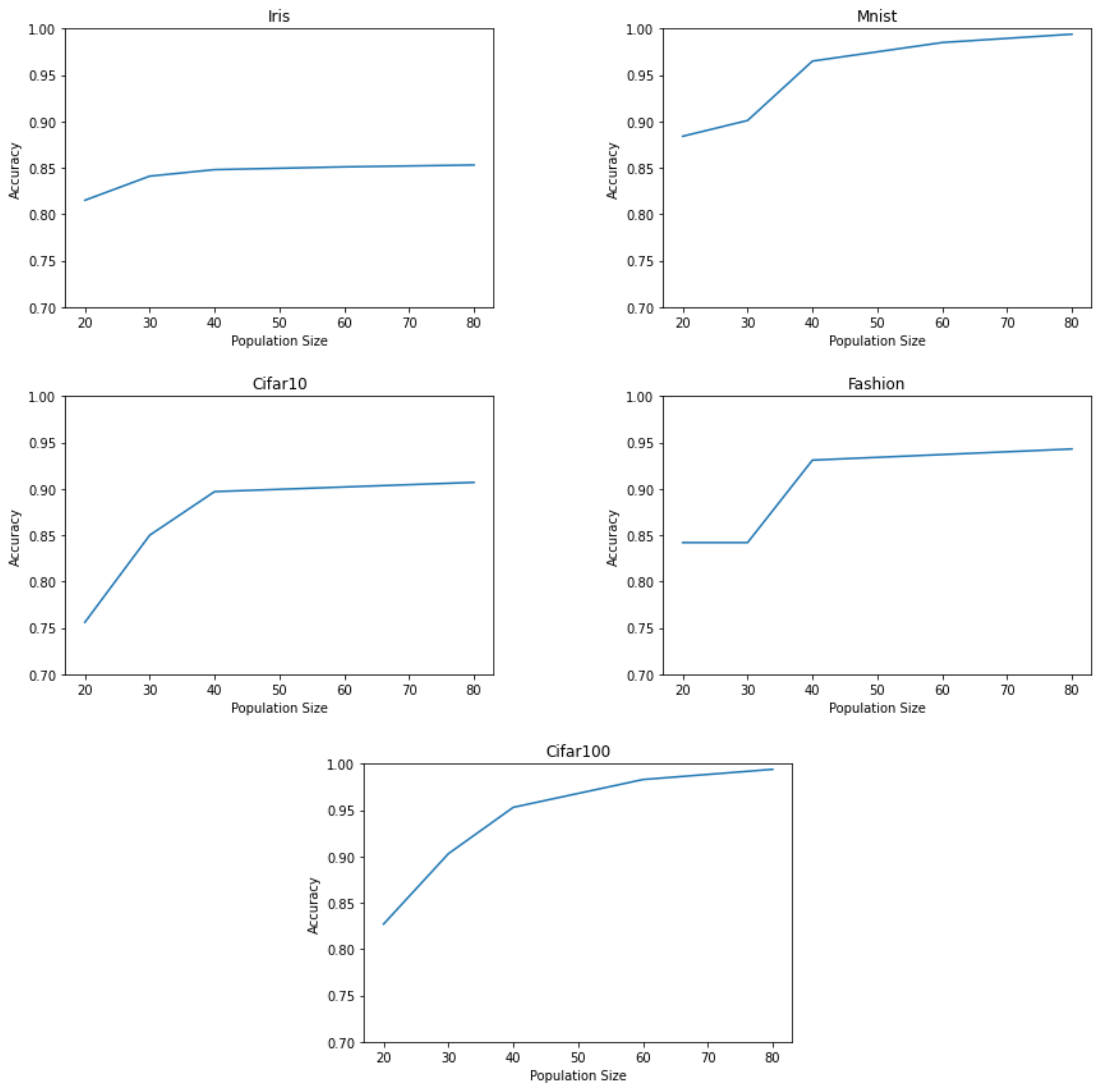

Fig. 4. Accuracy per dataset according to population size

Table 4
Impact of dynamic population

Dataset	Dynamic population	
	Enabled	Disabled
Mnist	0.995	0.994
CIFAR10	0.897	0.859
CIFAR100	0.953	0.924
IRIS	0.848	0.823
FASHION	0.931	0.925

Except for the Iris dataset, where the model was very simple, it can be noticed that the results are close to the one in the first column of Table 1, where the heuristic ratio was zero. In other words, in high-dimensional search spaces (models) the dynamic population strategy was able to find good initialization points while in relatively simple models traditional heuristic search approaches were able to do so.

5. DISCUSSION AND FUTURE WORK

In this study, a new dynamic population-based heuristic algorithm is proposed to initialize the training process of neural networks as a hybrid model of a heuristic algorithm along with traditional derivative-based techniques. The algorithm showed good results when evaluated by five benchmark datasets with a high number of parameters to be optimized. The method showed satisfactory results in initializing the training process when setting the heuristic ratio to 20%. This approach was applied to fixed neural network models. However, based on the promising results we got from the algorithm. We believe that the same approach can even be used to generate the NN model itself in addition to tuning the hyper. Moreover, since the initialization process to explore the search domain is done using multiple agents, the ability to make this process in parallel can be also investigated.

REFERENCES

[1] I.H. Osman and G. Laporte, "Metaheuristics: A bibliography," *Ann. Oper. Res.*, vol. 63, no. 5, pp. 511–623, Oct. 1996, doi: 10.1007/BF02125421.

[2] X.-S. Yang, *Nature-inspired metaheuristic algorithms*, 2. ed. Frome: Luniver Press, 2010.

[3] S. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4–5, pp. 185–196, Jun. 1993, doi: 10.1016/0925-2312(93)90006-O.

[4] D.P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." arXiv, Jan. 29, 2017. Accessed: Apr. 26, 2023. [Online]. Available: http://arxiv.org/abs/1412.6980.

[5] Y. Nesterov, "Implementable tensor methods in unconstrained convex optimization," *Math. Program.*, vol. 186, no. 1–2, pp. 157–183, Mar. 2021, doi: 10.1007/s10107-019-01449-1.

[6] T. Dozat, "Incorporating Nesterov Momentum into Adam," 2016.

[7] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[8] M.D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method." arXiv, Dec. 22, 2012. Accessed: Apr. 26, 2023. [Online]. Available: http://arxiv.org/abs/1212.5701.

[9] P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS." arXiv, Dec. 12, 2019. Accessed: Apr. 26, 2023. [Online]. Available: http://arxiv.org/abs/1912.06059.

[10] J.S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," in *Advances in Neural Information Processing Systems*, 2011, vol. 24..

[11] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 218–305, 2012.

[12] T. Domhan, J.T. Springenberg, and F. Hutter, "Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves," in *Proceedings of the 24th International Conference on Artificial Intelligence*, 2015, pp. 3460–3468.

[13] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts." arXiv, May 03, 2017. Accessed: Apr. 26, 2023. [Online]. Available: http://arxiv.org/abs/1608.03983.

[14] J. Rasley, Y. He, F. Yan, O. Ruwase, and R. Fonseca, "HyperDrive: exploring hyperparameters with POP scheduling," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, Las Vegas Nevada: ACM, Dec. 2017, pp. 1–13, doi: 10.1145/3135974.3135994.

[15] A. Gyorgy and L. Kocsis, "E?cient Multi-Start Strategies for Local Search Algorithms," arXiv, 16 Jan. 2014. [Online]. Available: https://arxiv.org/abs/1401.3894.

[16] P. Koch, O. Golovidov, S. Gardner, B. Wujek, J. Griffin, and Y. Xu, "Autotune: A Derivative-free Optimization Framework for Hyperparameter Tuning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &Data Mining*, London United Kingdom: ACM, Jul. 2018, pp. 443–452, doi: 10.1145/3219819.3219837.

[17] A. Aly, G. Guadagni, and J.B. Dugan, "Derivative-Free Optimization of Neural Networks using Local Search," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics &Mobile Communication Conference (UEMCON)*, New York City, NY, USA: IEEE, Oct. 2019, pp. 0293–0299, doi: 10.1109/UEMCON47517.2019.8993007.

[18] L.M. Rios and N.V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *J. Glob. Optim.*, vol. 56, no. 3, pp. 1247–1293, Jul. 2013, doi: 10.1007/s10898-012-9951-y.

[19] N.H. Kadhim and Q. Mosa, "Review Optimized Artificial Neural Network by Meta-Heuristic Algorithm and its Applications," *J.-Qadisiyah Comput. Sci. Math.*, vol. 13, no. 3, pp. 2021–2021, doi: 10.29304/jqcm.2021.13.3.825.

[20] Z. Tian and S. Fong, "Survey of Meta-Heuristic Algorithms for Deep Learning Training," in *Optimization Algorithms – Methods and Applications, InTech*, 2016, doi: 10.5772/63785.

[21] R. Mohapatra, S. Saha, C.A.C. Coello, A. Bhattacharya, S.S. Dhavala, and S. Saha, "AdaSwarm: Augmenting Gradient-Based optimizers in Deep Learning with Swarm Intelligence," arXiv, May 2020, [Online]. Available: http://arxiv.org/abs/2006.09875.

[22] M. Kaminski, "Neural Network Training Using Particle Swarm Optimization – a Case Study," in 2019 24th *International Conference on Methods and Models in Automation and Robotics*

(MMAR), Aug. 2019, pp. 115–120, doi: 10.1109/MMAR.2019.8864679.

[23] K.H. Lai, Z. Zainuddin, and P. Ong, "A study on the performance comparison of metaheuristic algorithms on the learning of neural networks," in *AIP Conference Proceedings, American Institute of Physics Inc.*, Aug. 2017, doi: 10.1063/1.4995871.

[24] M. Jaderberg *et al.*, "Population Based Training of Neural Networks," arXiv, Nov. 2017, [Online]. Available: http://arxiv.org/abs/1711.09846.

[25] S.R.Young, D.C. Rose, T.P. Karnowski, S.H. Lim, and R.M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, Austin, USA, Nov. 2015, doi: 10.1145/2834892.2834896.

[26] M. Mavrovouniotis and S. Yang, "Training neural networks with ant colony optimization algorithms for pattern classification," *Soft Comput.*, vol. 19, no. 6, pp. 1511–1522, Jun. 2015, doi: 10.1007/s00500-014-1334-5.

[27] X. Cui, W. Zhang, Z. Tüske, and M. Picheny, "Evolutionary Stochastic Gradient Descent for Optimization of Deep Neural Networks," arXiv, Oct. 2018, [Online]. Available: http://arxiv.org/abs/1810.06773.

[28] F.P. Such, V. Madhavan, E. Conti, J. Lehman, K.O. Stanley, and J. Clune, "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," arXiv, Dec. 2017, [Online]. Available: http://arxiv.org/abs/1712.06567.

[29] G. Morse and K.O. Stanley, "Simple evolutionary optimization can rival stochastic gradient descent in neural networks," in *Proceedings of the 2016 Genetic and Evolutionary Computation Conference GECCO 2016*, Jul. 2016, pp. 477–484, doi: 10.1145/2908812.2908916.

[30] A. Khan, R. Shah, M. Imran, A. Khan, J.I. Bangash, and K. Shah, "An alternative approach to neural network training based on hybrid bio meta-heuristic algorithm," *J. Ambient Intell. Humaniz. Comput.*, vol. 10, no. 10, pp. 3821–3830, Oct. 2019, doi: 10.1007/s12652-019-01373-4.

[31] R. Poli, J. Kennedy, and T.M. Blackwell, "Particle swarm optimization," *Swarm Intell.*, vol. 1, pp. 33–57, 1995.

[32] M. Dorigo and L.M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," 1997. [Online]. Available: http://iridia.ulb.ac.be/dorigo/dorigo.html, http://www.idsia.ch/~luca.

[33] X.-S. Yang and S. Deb, "Cuckoo Search via Levy Flights," arXiv, Mar. 2010, [Online]. Available: http://arxiv.org/abs/1003.1594.

[34] N.F. Johari, A.M. Zain, N.H. Mustaffa, and A. Udin, "Firefly algorithm for optimization problem," in *Applied Mechanics and Materials*, 2013, pp. 512–517, doi: 10.4028/www.scientific.net/AMM.421.512.

[35] D. Karaboga, "An Idea Based on Honey Bee Swarm for Numerical Optimization", Technical Report, Erciyes University, 2005

[36] D. Polap and M. Woźniak, "Polar bear optimization algorithm: Meta-heuristic with fast population movement and dynamic birth and death mechanism," *Symmetry*, vol. 9, no. 10, p. 203, Oct. 2017, doi: 10.3390/sym9100203.

[37] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv, Sep. 2016, [Online]. Available: http://arxiv.org/abs/1609.04747.

[38] A. Mirkhan and N. Celebi, "Binary Representation of Polar Bear Algorithm for Feature Selection," *Comput. Syst. Sci. Eng.*, vol. 43, no. 2, pp. 767–783, 2022, doi: 10.32604/csse.2022.023249.

[39] M. Aghili Nasr, M. Zangian, M. Abbasi, and A. Zolfaghari, "Neutronic and thermal-hydraulic aspects of loading pattern optimization during the first cycle of VVER-1000 reactor using Polar Bear Optimization method," *Ann. Nucl. Energy*, vol. 133, pp. 538–548, Nov. 2019, doi: 10.1016/j.anucene.2019.06.042.

[40] V.K. Ojha, A. Abraham, and V. Snášel, "Metaheuristic design of feedforward neural networks: A review of two decades of research," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 97–116, Apr. 2017, doi: 10.1016/j.engappai.2017.01.013.

[41] D.A. Simovici, C. Djeraba, *Mathematical Tools for Data Mining*. Springer, 2008.

[42] R. Livni, S. Shalev-Shwartz, O. Shamir, "On the Computational Efficiency of Training Neural Networks" in *Advances in Neural Information Processing Systems*, 2014, vol. 27.