2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 1-4, 2023, Oahu, Hawaii, USA

Multi-objective ADAM Optimizer (MAdam)

Farzaneh Nikbakhtsarvestani[1], Mehran Ebrahimi[2] and Shahryar Rahnamayan[3]

Abstract— **Multi-objective optimization is a prevalent challenge in the area of deep learning. There is a lack of robust multi-objective optimization methods applicable in deep learning capable of training networks by simultaneously optimizing conflicting multiple loss functions. Its applications include a wide range of deep neural network branches such as multi-loss, multi-task, multi-modal, and cross-modal learning. In this paper, we develop MAdam as a multi-objective extension of the well-known Adam optimization algorithm. MAdam is a classical population-based approach that uses the gradient information of multiple objectives to accelerate population convergence toward an optimal minimum. The method applied a non-dominated sorting algorithm to keep selective population members and improve the diversity across the landscape. The performance of MAdam is evaluated on the standard ZDT test functions as the proof of concept. Promising results show the capability of this approach to converge towards an estimated Pareto front and to generate a well-distributed set of non-dominated solutions.**

I. INTRODUCTION

In many real-world problems, optimization of a single objective may not be sufficient to fully capture the whole requirements of a real-world, such as, minimizing cost and maximizing the performance. Improving one objective might come at the expense of deteriorating another conflicting objective. Any multi-objective optimization (MOO) problem involves a trade-off between the objective in order to achieve an acceptable balance. This requires the MOO algorithm to consider multiple objectives simultaneously to create a set of trade-off solutions for a decision-maker. Generally speaking, machine learning is based on optimizing some decision variables; which is inherently a multi-objective task for many real-world problems. This means that the learner has to come up with a model that performs well across multiple losses $L_1, \cdots, Lp$, as opposed to a single one [3]. Particularly, multi-task learning refers to learning a prediction model for solving multiple tasks as a multi-objective problem since different tasks may be in conflict, necessitating a trade-off. Multi-task learning in computer vision has gained significant success in deep learning. An ultimate visual system for full scene understanding must be able to perform diverse perceptual tasks simultaneously and efficiently, especially within the limited computing environments of embedded systems such as smartphones, wearable devices, and robots/drones [4]. Multi-Objective-Optimization (MOO) aims to find a set of solutions as close as possible to the optimal Pareto front and as diverse as possible. MOO formulates the problem of learning from multiple objectives as discovering a set of Pareto optimal solutions expressing trade-offs among the objectives [5]. There has been a steadily increasing interest in the application of MOO in deep learning, where the learning process is framed as an optimization problem. In such cases, multiple loss functions are simultaneously optimized and used to evaluate whether the forecast distribution matches the target variables in the training data. At the stage of designing a machine learning model, the designer examines many parameters that should be included in the model. For instance, in addition to demanding high prediction accuracy, it may be beneficial to have a simple classifier that could be scaled to previously unknown data. Likewise, while training a lossy image compression model, one optimizes the size and quality of the compressed images. Multiple loss functions should be minimized concurrently. In such cases, representing a distinct aspect of the problem is a crucial challenging requirement. The role of an optimizer in deep learning is to adjust the weights and learning rate of the nodes of the model under the training process, such that it successfully minimizes the loss function. Adaptive moment estimation (Adam) is the most popular of all the optimizers in this area. Adam algorithm is relatively simple to implement and is suitable for problems with very large datasets. Adam is a desirable algorithm for deep learning since it has a low error rate and a high rate of accuracy [1]. Here are some of the advantages of using Adam Optimizer.

- Adaptive learning rate: Adam uses adaptive learning rates for each parameter, which means it can adjust the learning rate on a per-parameter basis. This helps the algorithm converge faster and can prevent overfitting.
- Momentum-based updates: Similar to other optimization methods such as stochastic gradient descent with momentum (SGDM), Adam uses momentum-based updates that help the algorithm overcome local minima and saddle points in the loss landscape.
- No need for manual tuning of hyperparameters: Adam has fewer hyperparameters to tune compared to other optimization methods, making it easier to use by reducing the need for manual tuning.
- Efficient memory usage: Adam uses only first-order gradients and second-order moments of gradients, which require less memory than other optimization methods, e.g., second-order methods such as Newton's method.

[1]Farzaneh Nikbakhtsarvestani, Faculty of Science, Ontario Tech University, 2000 Simcoe St N, Oshawa, ON, Canada, L1G 0C5 farzaneh.nikbakhtsarvestani@ontariotechu.net
[2]Mehran Ebrahimi, Faculty of Science, Ontario Tech University, 2000 Simcoe St N, Oshawa, ON, Canada, L1G 0C5 mehran.ebrahimi@ontariotechu.ca
[3]Shahryar Rahnamayan, SMIEEE, Department of Engineering, Brock University, 1812 Sir Isaac Brock Way, St. Catharines, ON, Canada, L2S 3A1 srahnamayan@brocku.ca

2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC) | 979-8-3503-3702-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/SMC53992.2023.10394533

- Robustness to noisy gradients: Adam is also robust to noisy gradients, which can occur in some cases, such as when using batch normalization or dropout layers.
- Proper default settings: Adam has good default settings for the hyperparameters, which makes it a good choice for most deep-learning tasks.

The existing machine learning tools are currently based on standard optimizers with a single loss (objective) function. Committing to a single objective (loss function) often fails to capture the full complexity of the underlying problem and causes models to overfit to that individual objective. Multi-loss functions may be able to prevent the algorithm away from overfitting to one single loss function. In the basic form, Adam is subject to a single objective optimizer. As a result, Adam cannot easily be applied to training a lossy image compression model whose target is optimizing bi-objective as the size and the quality of the compressed images concurrently. On the other hand, the goal of MOO is to generate a Pareto front of non-dominating solutions, in a way that a practitioner selects a post hoc solution based on the achieved trade-offs among objectives. Several attempts to tackle MOO for deep learning focused on addressing multi-task learning through gradient descent to find a single solution on the Pareto front [7], [2].

Inspired by the success of Adam optimizer and the importance of using MOO in deep learning, in this paper we propose a new approach to close this gap. The proposed algorithm, which will be referred to, from this point of this paper, as MAdam, is a hybrid optimizer.

The classical Adam algorithm was carefully modified and expanded to develop an Adam-based MOO optimizer. The extensions of the work have the potential to be utilized for multi-loss optimization in deep learning. MAdam was designed for solving population-based MOO problems and applying ADAM, as shown in Fig.1.

Adam optimizer was applied to accelerate the convergence. NDS and crowding distance algorithms were interfaced with Adam for the simultaneous exploration of objectives in order to obtain uniformly distributed MOO solutions. The significance of MAdam is due to its suitability for being used as the optimizer of choice when scaling deep learning for solving MOO problems. This is achieved by fulfilling two important advantages: i) starting off with uniform random initialization which helps to explore the search space more widely; ii) it also comes along with an increased chance of finding multiple trade-off solutions and preventing optimal points from being stuck in a local optimum; and iii) it generates the entire Pareto front in a single run.

II. BACKGROUND

A. Preliminaries

1) Multi-Objective Optimization: In multi-objective optimization, the goal is to minimize a vector-valued objective function $f(x) = [f^1(x), \cdots, f^{(M)}(x)] \in \mathbb{R}^M$, where $M \geq 2$ while satisfying black-box constraints $g(x) \geq 0$ and $x \in \chi \subset \mathbb{R}^d$, and χ is a compact set. Usually, there is no single solution x^* that achieves the optimum that simultaneously minimizes all M objectives; when the objectives are in conflict. We can obtain a set of Pareto optimal solutions according to the following definitions.

Definition 2.1: (Pareto dominance). An objective vector $f(x)$ pareto-dominates $f(x')$, denoted as $f(x) \prec f(x')$, if $f^{(m)}(x) \leq f^{(m)}(x')$ for all $m = 1, \cdots, M$ and there exists at least one $m \in \{1, \cdots, M\}$ such that $f^{(m)}(x) < f^{(m)}(x')$. When there is no solution that dominate any of the members of the Pareto front, it is called an optimal Pareto front.

2) Adam Optimization: The Adaptive Movement Estimation algorithm, or Adam for short, is an extension to the gradient descent method and combines the advantages of two recently popular methods AdaGrad [10] and RMSProp [11]. Adam is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method automatically adapts a learning rate for each input variable for the objective function and further enhances the search process by using an exponentially decreasing moving average of the gradient to make updates to variables. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, the step sizes are approximately bounded by a hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs as a form of step size annealing [1]. That is better to say than converting multi-objective to single objective by using weighted summation is not proper to solve a multi-objective problem, because a portion of the solution space would not be acceptable and also it ends up with a single solution not optimal PF solutions. In addition, the number of function evaluations is significantly reduced, resulting in a more efficient computation. This concept is wisely adopted in Adam. The step size for each input parameter is automatically adapted throughout the search process, based on the gradients (partial derivatives) with respect to the variable.

3) Non-Dominated Sorting Algorithm: Non-dominated sorting (NDS) is a commonly used algorithm in multi-objective optimization, where the goal is to optimize multiple objectives simultaneously. The algorithm is used to partition a set of candidate solutions into different fronts, where the solutions in the first front are not dominated by any other solutions, the solutions in the second front are not dominated by the solutions in the first front, and so on. This algorithm is efficient in obtaining optimal Pareto fronts for any number of objectives and can accommodate any number of constraints as well. The pseudo-code of NDS algorithm is provided in algorithm 1.

4) Crowding Distance: Each candidate solution is associated with a rank based on the NDS criteria, which is a diversity maintenance mechanism, called crowding distance. It provides NDS with an estimation of the distance between the closest two members among the entire solution. The idea is to measure the distance between neighboring solutions in the objective space and use this measure to encourage the

Algorithm 1 Non-dominated sorting

input: Front (F), Population (pop)
output: rank
1: Let rank counter r be zero
2: Increase: $r = r + 1$
3: Find the non-dominated individuals, S, from population pop
4: $\text{rank}_p \leftarrow r \quad \forall p \in S$
5: Remove these individuals from pop and continue.
6: If Pop is empty then stop, else go to step 2.

algorithm to find a diverse set of solutions that cover the entire Pareto front. The pseudo-code of the crowding distance algorithm is provided in algorithm 2.

Algorithm 2 Crowding Distance

input: Front (F); number of individuals in the front (nf); the number of objective functions (n_{obj})
output: distance (d)
1: Let $d_k = 0$ for $k = 1, 2, \cdots, nf$
2: For each objective function f_i, $i = 1, 2, \cdots, n_{obj}$ (sort the set in ascending order)
3: Let d_1 and d_{nf} be maximum values, e.g. $d_1 = d_{nf} = \infty$
4: For $j = 2$ to $nf - 1$, set $d_j = d_j + \Sigma_{k=1}^{n_{\text{obj}}} \frac{f_i(j+1) - f_i(j-1)}{f_i(nf) - f_i(1)}$

B. Related Work

Previous attempts for the development of MOO optimizer for deep learning were concentrated on solving the problem of multi-task learning [7] by descending via the Pareto front until a single solution is found. Additional follow-up strategies offer to populate a set of Pareto optimal solutions by learning multiple neural networks along preference vectors [8], [9]. Preference vectors encode the predefined relative importance of each objective. Two recent ideas proposed conditioning the network's weights to the preference vector through hyper-networks [12]. A novel method that scales MOO to deep learning is proposed in [6]. This method conditions the predictions of the neural network to the preference vectors. Through this method, the full Pareto front of solutions is generated in a single optimization run, contrary to methods that train one network per point in the Pareto front. However, MOO in [6] was treated as single-objective optimization through linear scalarization and utilizing this method for MOO does not allow controlling values of individual objective functions during the optimization process. It creates serious problems in finding an evenly distributed optimal Pareto front.

III. PROPOSED ALGORITHM

We propose MAdam to find the Pareto front via hybridization of two optimization procedures. We use the Adam optimizer to construct descent direction of each multi-objective toward the optimal point. In addition, we adopt a multi-objective evolutionary algorithm based on non-dominated sorting and crowding distance to select the best solutions and improve the diversity, respectively. Also, MAdam includes a population-base approach. This idea is originated from evolutionary methods, while most of classical methods are based on a single solution approach. Generally, to balance the convergence and the diversity, a mechanism alternately using Adam optimizer and NDS is designed for solving MOO problems in this article. The population-based Adam is used for the exploitation to accelerate the convergence and NDS is used for the exploration and to find a better distribution of Pareto front solutions. As shown in Fig.1, MAdam algorithm consists of five major components: Random initialization, the core of MAdam optimizer, finding non-dominated sorting, crowding distance calculation, and finding elitist population for the next iteration.

A. Initialization of MAdam

Considering a population-based Adam the random population initialization of MAdam is bounded by $[0, 1]$, whereas the data is scaled to this range. MAdam is sensitive to the initialization used within the first raw population which is similar to single-solution based Adam. We used a single objective Adam optimizer for two different objective functions. Adam starts the search from a random point in the bounds of the problem and returns the optimal points for each objective. We consider the optimal point obtained via Adam optimizer as the first and the second individuals of MAdam initialization.

B. MAdam

MAdam is designed to incorporate Adam in a MOO optimizer. It includes the following components:

- In MAdam, we have two or more conflicting objectives, which is significant because it allows decision-makers to consider and to evaluate trade-offs among competing objectives. Each objective has a different descent direction.
- The dataset of MAdam stores information of each population member. During each iteration, this dataset is updated with the latest gradient direction. This dataset includes the first and the second moments of gradients with respect to each variable, the bias corrections of the first and the second moments of the gradients, the updated variable in decision space, and their associated scores in objective space (Algorithm 4).
- MAdam identifies Pareto-optimal solutions, which are solutions that cannot be improved upon in one objective without sacrificing performance in another objective. This allows decision-makers to make informed decisions and trade-offs based on their priorities and preferences. To do so, all points move progressively toward the optimal point in each iteration, therefore MAdam terminates when the maximum number of iterations is reached.

The framework of modified Adam is shown in Algorithm 3. The random population is generated first, and the first individuals are assigned to Adam search performance. Next,

we run a fixed number of iterations of the algorithm defined by $\max_{\text{iter}}$ for each gradient direction. In this step, MAdam dataset is called to update the dataset information of each member of the population and then two datasets are interconnected. Then, the objective values of concatenating dataset are excluded to identify solutions of the first non-dominated front. In the last stage, the elitist individuals which correspond to the first front are selected for the next iteration. When the maximum number of iterations is reached, MAdam terminates.

C. Selecting Elitist Population

For each member of the population, MAdam database stores the values of objective functions of updated points along each descent direction. After that, by concatenating the objective values of $f_1, f_2, \cdots, f_M$ for the entire population, non-dominated sorting (Algorithm 1) and crowding distance (Algorithm 2) are employed to remove the dominated solutions and to keep the best individuals in the population and maintain the diversity in search space, respectively. Once the sorting is performed with respect to crowding distance, then the elitist individuals in the population are selected from the dataset based on the rank indices to exploit and explore the landscape for the next iteration. According to the information from the previous iteration, the new database is updated.

Algorithm 3 MAdam Framework

Input:

- maximum number of iteration $\max_{\text{iter}}$

Output: elitist individuals S

- **1:** Adam optimizer for each objective function
- **2:** Pop_{set} = Generate random numbers in $[0, 1]$ with population size Npop that start from the points achieved from 1
- **3:** Define MAdam datasets (MDs) corresponding to gradients $\nabla f_1, \nabla f_2, \cdots, \nabla f_M$ by (Algorithm 4)
- **4:** initialize MD's with Pop_{set}

for $t \leftarrow 1$ *to* max_{iter} **do**

- **5:** $S = \emptyset$

/*Exploitation*/

- **6:** update MD's
- **7:** concatenate two database

/*Exploration*/

- **8:** excluding objective values information from concatenated MD
- **9:** NDS (Algorithm 1) to find front
- **10:** crowding distance (Algorithm 2) to find rank
- **11:** Finding corresponding population and their information with rank indices from concatenated MD and update S

D. Exploitation

The exploitation is to speed up the convergence by making the population search. Algorithm 4 presents the scheme of the exploitation. The first step is to calculate the partial derivative of each dimension. For number of objectives $j = 1, \cdots, M$ and a vector $Z = (z_1, \cdots, z_n)$,

$$\frac{df_j}{dZ} = \left(\frac{\partial f_j}{\partial z_1}, \cdots, \frac{\partial f_j}{\partial z_n} \right).$$

Next, MAdam calculation updates the information dataset of each population. Lines 2 and 3 calculate the first and the second moments, respectively. Lines 4 and 5 indicate the bias correction of the first and the second moments. Then, in line 6 the variable values are updated. This is then repeated for each dimension. In line 7 and 8, the functions of an updated variable are evaluated and are assigned as the input of Algorithm 1.

Algorithm 4 Pseudocode for updating MAdam dataset for M=2 objectives

Input:

- Dimension D.
- objective functions $f_j, \quad j = 1, \cdots, M = 2$,
- partial derivatives $\nabla_i f_j(x), \quad i = 1, \cdots, D$
- Population Pop_{set}
- t: Time
- α: Step size
- $\beta_1, \beta_2 \in [0, 1)$ Exponential decay rates for the moment estimate
- $\epsilon = 10^{-8}$
- bounds: define range for input population

Output: updated MAdam dataset (MD)

1 **for** $x \in Pop_{set}$ **do**

2 **1:** $g_t = \nabla_{x_k} f_j(x_{k-1})$ for $k = 1, \cdots, D$

$j = 0$

for $i \leftarrow 0$ *to* $D - 1$ **do**

3 **2:** $\text{MD}[x, i+j] \leftarrow \beta_1 * m[x, i] + (1 - \beta_1) * g_t[i]$

3: $\text{MD}[x, i+1+j] \leftarrow \beta_2 * v[x, i+1] + (1-\beta_2) * g_t[i]^2$

4: $\hat{m} \leftarrow \frac{\text{MD}[x, i+j]}{(1-\beta_1^{t+1})}$

5: $\hat{v} \leftarrow \frac{\text{MD}[x, i+1+j]}{(1-\beta_2^{t+1})}$

6: $\text{MD}[x, i+2+j] \leftarrow \text{MD}[x, i+2+j] - \alpha . \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$

$j = j + 2$

4 **7:** $\text{MD}[x, 6] \leftarrow f_1(\text{MD}[x, 0], \cdots, \text{MD}[x, D])$ (Update f_1-values)

8: $\text{MD}[x, 7] \leftarrow f_2(\text{MD}[x, 0], \cdots, \text{MD}[x, D])$ (Update f_2-values)

E. Exploration

The next step of MAdam explores the search space to find the best individuals for the next iteration. The objective values of the previous step information dataset compete together while the best solution (i.e, non-dominated solutions) are selected for the next generation. We applied the well-known multi-objective optimization algorithm NDS

(Algorithm 1) which is one of the most robust algorithms widely used to find the uniform spread of solutions and desirable convergence near the true Pareto-optimal front. In the exploration stage, the algorithm starts with determining all non-dominated solutions in the first rank. The non-dominated vectors are eliminated from the set and the remaining candidate solutions are then processed in the same manner in order to determine the second rank of individuals. The second level of individuals is made by solutions from this step that are not dominant (second Pareto). After that, the second-ranked individuals will be eliminated to determine the third Pareto. This procedure will continue until all individuals are sorted into distinct Pareto ranks. After assigning a rank to each solution based on the non-domination criterion, to maintain the diversity in search space, a niching approach crowding distance (Algorithm 2) is used to estimate the distance between the nearest two members for each solution.

The crowding distance parameter that is incorporated in MAdam algorithm serves as an estimate of the perimeter of the cuboids formed by using the nearest neighbors. It is performed by obtaining the average Euclidean distance of two points on either side of the point along each of the objectives in objective space. Fig.1 represents the schematic demonstration of MAdam algorithm.

IV. EXPERIMENTAL RESULTS

In this section, to empirically evaluate the proposed method, we investigate the MAdam capability to get close to the optimal PF. The results are based on the well-known multi-objective benchmark ZDT functions, ZDT1, ZDT2, ZDT3 and ZDT4 borrowed from the literature (Table I). The Pareto fronts of the ZDT functions are well-defined, which makes it easy to evaluate the performance of multi-objective optimization algorithms. All of these problems have two objective functions. These MOO functions are based on the following definition

$$\begin{cases} \min f_1(\boldsymbol{x}) \\ \min f_2(\boldsymbol{x}) = g(\boldsymbol{x})h(f_1(\boldsymbol{x}), g(\boldsymbol{x})). \end{cases} \tag{1}$$

Each call of MAdam performance is reported in terms of the objective functions and their partial derivatives with respect to each dimension, the last updated dataset, the number of iterations, and the hyper-parameters, such as learning rate and momentum. The number of MAdam calls is

$$\text{Number of MAdam Call} = \max_{iter} * N_{\text{pop}} * M,$$

where M is the number of objectives. MAdam inherits Adam hyperparameters that are tuned based on empirical testing. The parameters for MAdam are adjusted as input range [0, 1] , i.e., random points are initialized in this range as starting points, furthermore the number of objectives ($M = 2$), initial step size (learning rate) $\alpha = 0.02$, decay factor for first momentum $\beta_1 = 0.4$, and decay factor for infinity norm $\beta_2 = 0.999$. We assume 100 iterations while at each iteration we have a population of candidate solutions with the size of $N_{\text{pop}} = 100$. Since MAdam is evaluated twice for each member of the population, the number of selected populations may expand exponentially in each iteration. As a result, in each step of convergence along the descent direction, we maintain the fixed number of population ($N_{\text{pop}} = 100$). After concatenating the values of the objective function from MAdam dataset, NDS is invoked in each iteration. Fig. 2 shows the states of the Pareto fronts of the ZDT benchmarks. The candidate solutions are sorted based on their non-domination and placed into different fronts. Interval $[0, 1]$ was selected to explore the dynamic competition between the two objective in each ZDT functions and finding trade-off based on non-domination. The trajectory of each front toward non-dominated solution are traced during the optimization process. Individuals from the first Parteo front dominate solutions in the second front. Identically, the individuals from the second front dominate the third front. The same principle applies to the dominance of Pareto front at step "n" with respect to Pareto front at step "n+1". After extracting the Pareto fronts, the crowding distance algorithm was applied front-wise to maintain the diversity of the candidate solutions. The average crowding distance was used to determine population diversity. The computational time of MAdam is influenced by the number of population and maximum number of iterations.

In ZDT1, ZDT2, and ZDT4 there are multiple trajectories of points seeking the solution of the MOO problem and eventually shaping the last state of the Pareto front, as shown in Fig. 2. Both in ZDT1 and ZDT2 the resulted Pareto front is uniformly distributed. ZDT4 has a wider spectrum than the other ZDT benchmarks. A wider Pareto front in MOO problems is more promising since it provides a broader range of optimal solutions that cover a wider spectrum of trade-offs between the objectives. Due to the comparatively lower rate of convergence of ZDT3, higher rank of fronts are included for this benchmark. Because of this, solutions from the prior Pareto fronts are added to the last state of the Pareto front, which results in a comparatively denser population in Fig. 2. The segregate Pareto front of ZDT3 is a set of non-dominated solutions but not as optimal as the ones on the other ZDT's. These solutions can still provide a useful trade-off between the objectives. These for case studies can be seen as proof of concept which the proposed MAdam performs well on multi-objective optimization problems to find the optimal Pareto-front solutions.

V. CONCLUSION

In this paper, we proposed a hybrid gradient-based search method for solving smooth MOO problems. MAdam is the first attempt for incorporating Adam with multi-objective optimization. MAdam is based on searching the landscape by starting from a random population and finding the updated points by gradient descent in each iteration. In each iteration, the core of MAdam keeps the information of each population member and updates it for each gradient direction. MAdam is iterative in nature. Non-dominated sorting determines the rank of values of updated points. Furthermore, crowding distance algorithm is used to maintain the diversity of non-

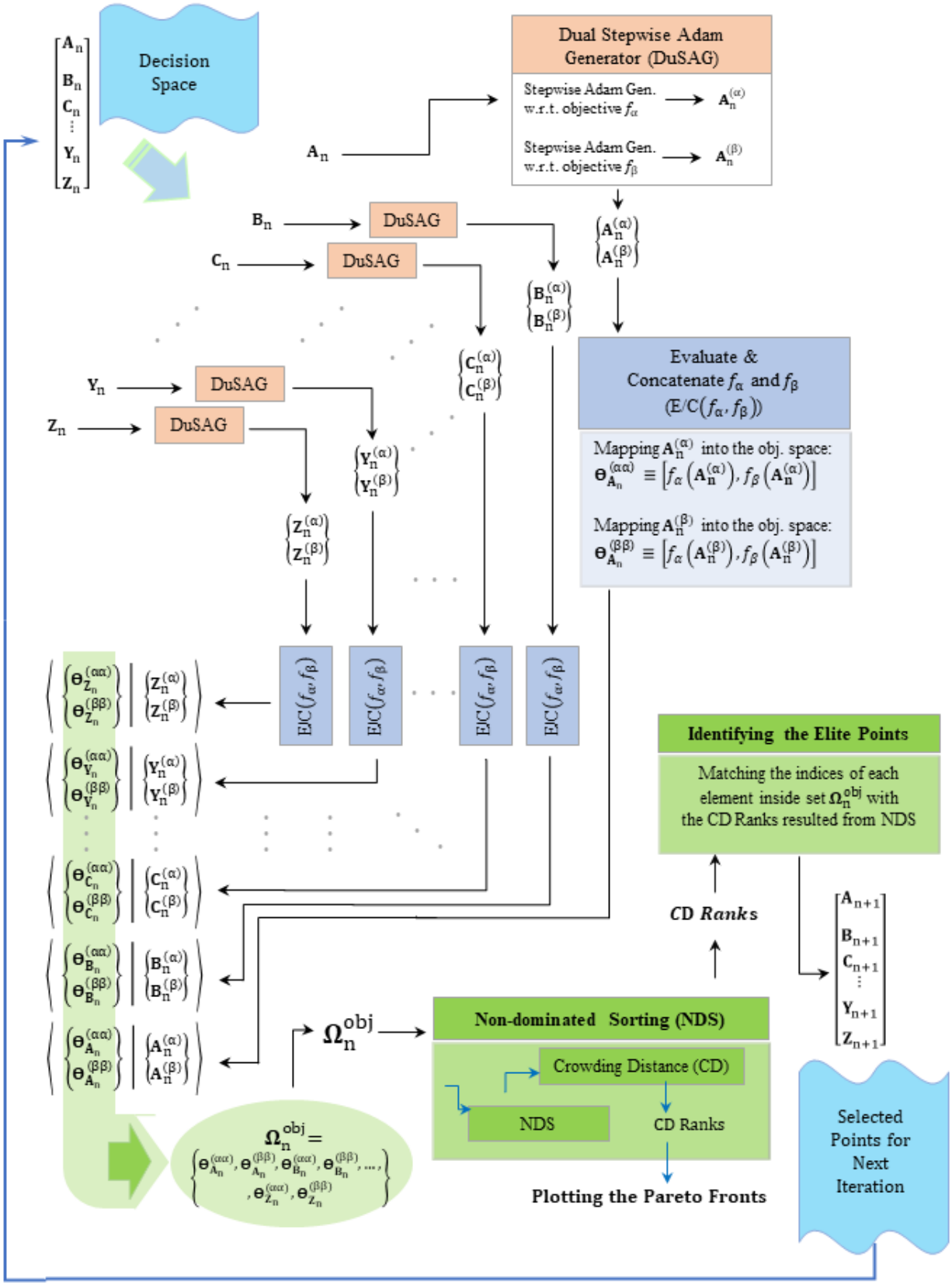

Fig. 1. Flowchart of MAdam

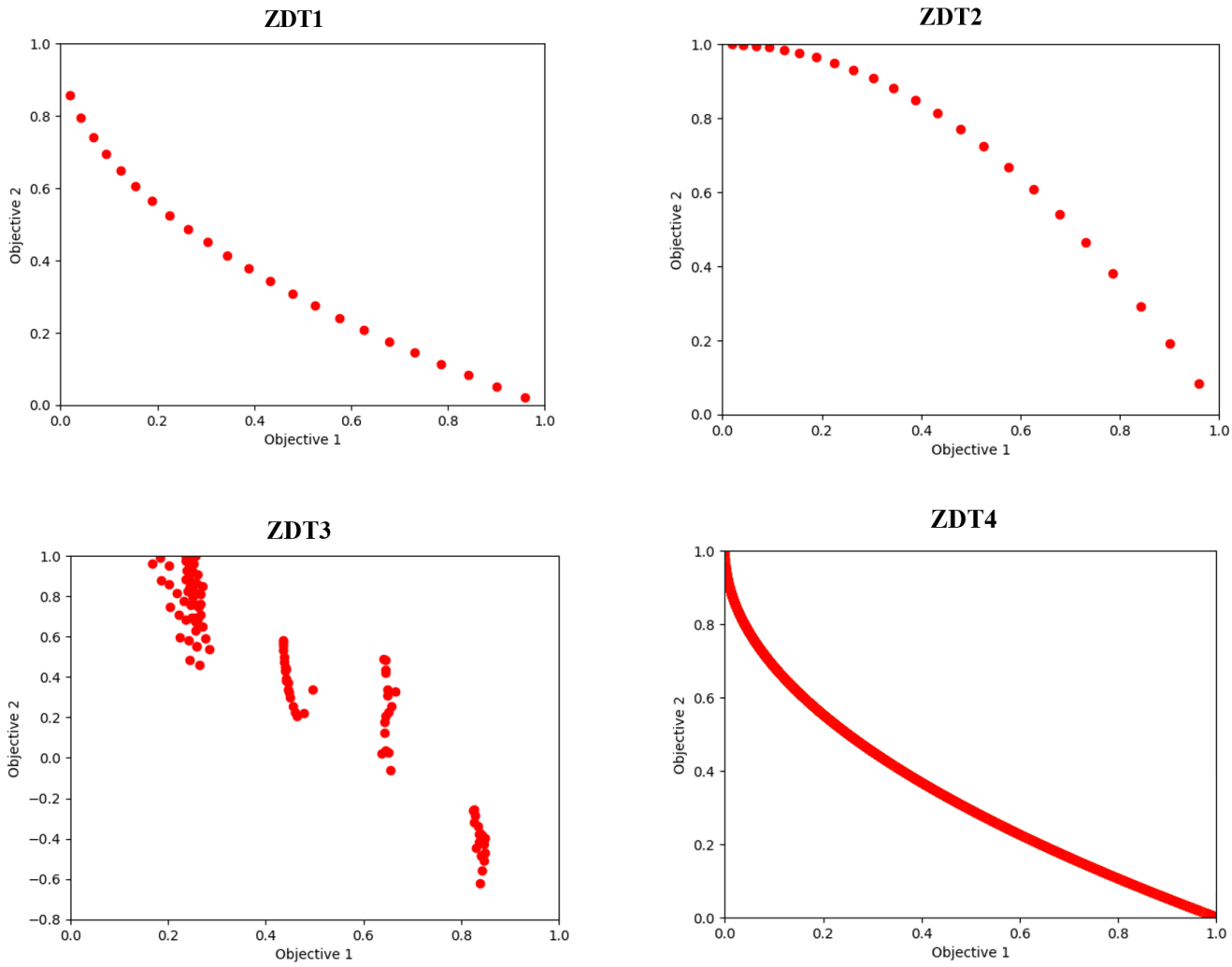

Fig. 2. Pareto fronts of ZDT1, ZDT2, ZDT3, and ZDT4, resulted from MAdam. ZDT1, ZDT2, and ZDT4 represent the last state of the Pareto front. ZDT3 includes both the last state of the Pareto front and higher rank fronts.

TABLE I

ZDT FUNCTIONS DEFINITIONS

ZDT functions		
Name	Problem	Domain
ZDT1	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{D-1}\Sigma_{i=2}^{D} x_i$ $h(f_1, g) = 1 - \sqrt{f1/g}$	$[0, 1]$
ZDT2	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{D-1}\Sigma_{i=2}^{D} x_i$ $h(f_1, g) = 1 - (f1/g)^2$	$[0, 1]$
ZDT3	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{D-1}\Sigma_{i=2}^{D} x_i$ $h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1)$	$[0, 1]$
ZDT4	$f_1(x) = x_1$ $g(x) = 1 + 10(D-1) + \Sigma_{i=2}^{D}(x_i^2 - 10\cos(4\pi x_i))$ $h(f_1, g) = 1 - \sqrt{f_1/g}$	$x_1 \in [0, 1]$ $x_i \in [-10, 10]$, for $i > 1$

dominated solutions. This algorithm is used in each iteration to select the population from the least crowded area in the objective space which results in generating uniformly distributed elitist solutions and reducing running time. Finally, we demonstrated the effectiveness of MAdam to provide a new approach to systematically traverse the Pareto front for two multi-objective benchmarks in our experiments that provides results consistent to the literature for these benchmarks.

This work can be considered as a novel direction opening paper, which proposes the Multi-objective Adam Algorithm to train deep numeral networks for multi-loss, multi-task, and multi-modal learning real-world problems. Furthermore, MAdam output would have a high potential for further development of deep neural networks algorithms and allows practitioners to make real-time trade-off adjustment among multi-loss functions after training. However, this scheme has some hereditary shortcomings from its parent algorithm. Selecting hyperparameters such as learning rate, decay rates, and epsilon requires manual tuning and may

depend on the specific problem domain, making it less automated compared to some other optimizers. Furthermore, due to the gradient-based nature of proposed scheme, the algorithm may struggle to escape saddle points efficiently, especially in high-dimensional spaces. MAdam's adaptive learning rate can slow down convergence in such regions. As a potential research path, we could examine the algorithm on other multi-objective benchmarks. We can also consider high-dimensional problems. In addition, we could potentially make a comparison of MAdam with other state-of the-art multi-objective strategies.

ACKNOWLEDGEMENTS

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
[2] Désidéri, J. A. (2009). Multiple-gradient descent algorithm (MGDA) (Doctoral dissertation, INRIA).
[3] Cortes, C., Mohri, M., Gonzalvo, J., Storcheus, D. (2020). Agnostic learning with multiple objectives. Advances in Neural Information Processing Systems, 33, 20485-20495.
[4] Chen, Z., Badrinarayanan, V., Lee, C. Y., Rabinovich, A. (2018, July). Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In International conference on machine learning (pp. 794-803). PMLR.
[5] Miettinen, K. (2002). Interactive nonlinear multiobjective procedures. Multiple criteria optimization: state of the art annotated bibliographic surveys, 227-276.
[6] Ruchte, M., Grabocka, J. (2021, December). Scalable pareto front approximation for deep multi-objective learning. In 2021 IEEE international conference on data mining (ICDM) (pp. 1306-1311). IEEE.
[7] Sener, O., Koltun, V. (2018). Multi-task learning as multi-objective optimization. Advances in neural information processing systems, 31.
[8] Lin, X., Zhen, H. L., Li, Z., Zhang, Q. F., Kwong, S. (2019). Pareto multi-task learning. Advances in neural information processing systems, 32.
[9] Mahapatra, D., Rajan, V. (2021). Exact Pareto optimal search for multi-task learning: touring the Pareto front. arXiv preprint arXiv:2108.00597.
[10] Duchi, J., Hazan, E., Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).
[11] Tieleman, T., Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.
[12] Navon, A., Shamsian, A., Chechik, G., Fetaya, E. (2020). Learning the pareto front with hypernetworks. arXiv preprint arXiv:2010.04104.