Multi-Objective Deep Learning with Adaptive Reference Vectors

Weiyu Chen  James T. Kwok
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong
{wchenbx, jamesk}@cse.ust.hk

Abstract

Many deep learning models involve optimizing multiple objectives. Since objectives are often conflicting, we aim to get diverse and representative trade-off solutions among these objectives. Gradient-based multi-objective optimization (MOO) algorithms using reference vectors have shown promising performance. However, they may still produce undesirable solutions due to mismatch between the pre-specified reference vectors and the problem’s underlying Pareto front. In this paper, we propose a novel gradient-based MOO algorithm with adaptive reference vectors. We formulate reference vector adaption as a bilevel optimization problem, and solve it with an efficient solver. Theoretical convergence analysis is also provided. Experiments on an extensive set of learning scenarios demonstrate the superiority of the proposed algorithm over the state-of-the-art.

1 Introduction

Deep learning models are often evaluated under multiple, potentially conflicting, criteria. For example, in multi-task learning [6], a single model is required to perform well on multiple tasks. In some scenarios, besides accuracy, model fairness is also important so as to ensure that the model is not biased against gender and race. These problems can all be formulated as multi-objective optimization (MOO) [39] problems, and have attracted attention from various fields such as energy resource optimization [9] and signal processing [4].

Since the multiple objectives usually cannot be optimized simultaneously by a single solution, the goal of MOO is to find a set of solutions with different trade-offs to approximate the true Pareto front (PF). The past decades have witnessed the birth of a large number of gradient-free MOO algorithms, such as evolutionary multi-objective optimization algorithms [11, 52] and Bayesian multi-objective algorithms [26, 3]. These algorithms perform well on small-scale problems but fail to provide useful solutions when facing the huge number of parameters in deep learning models.

Recently, gradient-based MOO algorithms [14] have demonstrated promising performance in deep learning. Following the pioneering work in [45], Lin et al. [31] and Mahapatra et al. [37] propose to use a set of reference vectors (RV), and generate multiple solutions on the PF each of which is closest to an RV. Later, strategies are further proposed to improve the efficiency by training a hypernetwork [40, 50] or reference-vector-conditioned network [44]. However, note that a set of uniformly distributed RVs may not lead to a set of uniformly distributed solutions. Hence, an important limitation of these algorithms is that solutions generated using fixed RVs may not cover some parts of the PF, thus failing to provide enough information about the PF. As an illustration, in Figure 1a solutions obtained by the fixed RVs are close to the ends of the PF. If the RVs are properly positioned, it is possible that the obtained solution set can uniformly cover the PF (Figure 1b).

Since it is impossible to know the true PF of the problem before optimization, how to generate a proper set of RVs is a big challenge. In this paper, we propose to learn the set of RVs simultaneously with the model parameters. This is formulated as a bilevel optimization problem \[13\], in which the lower-level optimization problem obtains the Pareto-optimal solutions with a given set of RVs, while the upper-level problem optimizes the RVs based on a quality measure. Moreover, we solve this optimization problem with an inexpensive solver, while still showing theoretically nice convergence properties. Experiments on an extensive set of learning scenarios demonstrate the superiority of the proposed algorithm.

Notations. In this paper, \([m]\) denotes the set \(\{1, \ldots, m\}\), and \(\mathbb{R}^m_+ \equiv \{u \in \mathbb{R}^m| u_i \geq 0, \forall i \in [m]\}\). For two vectors \(u, v \in \mathbb{R}^m\), \(u \geq v\) means that \(u_i \geq v_i\) for all \(i \in [m]\). \(|\cdot|\) denotes the Euclidean norm for vectors and the Frobenius norm for matrix.

2 Related Work: Multi-Objective Optimization (MOO)

In multi-objective optimization (MOO) \([39]\), one aims to minimize \(m \geq 2\) objectives \(\{f_1(\phi), f_2(\phi), \ldots, f_m(\phi)\}\), or, equivalently, the vector-valued function:

\[
\min_{\phi} f(\phi) = [f_1(\phi), \ldots, f_m(\phi)] \in \mathbb{R}^m.
\]  

(1)

A solution \(\phi_1\) is dominated by another solution \(\phi_2\) if and only if \(f_i(\phi_1) \geq f_i(\phi_2)\) for \(i \in [m]\), and \(\exists i \in [m], f_i(\phi_1) > f_i(\phi_2)\). A solution \(\phi^*\) is Pareto-optimal if and only if it is not dominated by any other \(\phi'\). A Pareto front (PF) is the set of multi-objective values of all Pareto-optimal solutions. A PF is regular \([43]\) if it is simplex-like (i.e., all vectors with positive directions intersect it when they start at the origin), and irregular otherwise. It is shown that regular PFs are not very realistic \([24]\).

Since the number of Pareto-optimal solutions is usually large or even infinite, a set of \(n\) Pareto-optimal solutions \(\Phi = \{\phi_1, \ldots, \phi_n\}\) is often used to approximate the PF. Let the corresponding multi-objective values be \(F = \{f(\phi_1), \ldots, f(\phi_n)\}\). The quality of \(\Phi\) can be evaluated from two perspectives: convergence and diversity \([27]\). Convergence refers to the distance between \(F\) and the true PF, while diversity measures whether the solutions are well-distributed in the space of objectives. A popular measure, which evaluates both convergence and diversity, is the hypervolume (HV) \([53]\). Given a reference point \(z \in \mathbb{R}^m\), the HV of \(F\) is:

\[
HV(F; z) = \lambda(\cup_{f_i \in F}[z, f_i]),
\]  

(2)

where \([z, f_i] \equiv \{q \in \mathbb{R}^m| f_i \leq q \leq z\}\), and \(\lambda(\cdot)\) is the Lebesgue measure of a set.

A reference vector (RV) \([52]\), sometimes called the weight vector or preference vector, can be used to guide the optimization algorithm by indicating the preferred point on the PF. Usually, an algorithm is expected to obtain the Pareto-optimal solution closest to the given RV in the objective space. Given a RV \(r \in \mathbb{R}^m_+\), the MOO problem in \([1]\) can be converted to a single-objective problem by using a scalarization function \(s(\phi; r)\). The most straightforward construction is linear scalarization:

\[
s(\phi; r) = \sum_{j=1}^{m} r(j) f_j(\phi),
\]  

(3)

which weights the \(m\) objectives with the elements \(r(j)\)’s of \(r\). It is known that the minimizer \(\phi\) of \(s(\phi; r)\) is also Pareto-optimal for the original problem in \([1]\) (Ch 4.7, \([15]\)). However, the vector of objectives \(f(\phi)\) obtained may be far away from the given RV. To encourage them to be closer to this vector, a penalty term can be added to the linear scalarization function, leading to \([44]\):

\[
s(\phi; r) = \sum_{j=1}^{m} r(j) f_j(\phi) + \gamma \frac{r^T f(\phi)}{||r|| \cdot ||f(\phi)||},
\]  

(4)
where $\gamma > 0$ is a constant. Given a (discrete) set of solutions, Ma et al. [36] generate exploration directions to spawn new solutions on the PF, leading to a continuous PF.

By using $n$ RVs from a subset $R \subseteq \mathbb{R}^{m \times n}$, a set of solutions can be obtained to approximate the entire PF. Usually, $R$ is simply set to $\mathbb{R}^{m \times n}$. However, sometimes the decision-makers may only be interested in a specific region of the PF [2]. For example, we may limit the angle between any RV and each coordinate axis. $R$ can then be changed to:

$$\left\{ r_1, \ldots, r_n \right\} \quad \cos \phi_2 \leq \frac{r_j^\top u_i}{\|r_j\| \cdot \|u_i\|} \leq \cos \phi_1, \quad \forall i \in [m], j \in [n]$$

where $u_i$ is the $i$th coordinate axis and $\phi_1, \phi_2$ are the maximum and minimum allowable angles, respectively. However, note that the set of pre-specified RVs may not fit the problem’s PF (e.g., some of them may not intersect the underlying PF), leading to an undesirable solution distribution.

**Gradient-Free MOO.** Evolutionary MOO algorithms (e.g., NSGA-II [11], MOEA/D [52]) and Bayesian MOO algorithms (e.g., BMOA [26], USeMO [2]) are widely used for small-scale black-box problems. These algorithms assume that gradient information is not available. Hence, they often fail to converge on deep learning problems where the solution space can be very large.

**Gradient-Based MOO.** Gradient-based MOO algorithms are more efficient when problems have differentiable objectives and a large number of parameters. Sener and Koltun [45] propose to apply Multiple-Gradient Descent Algorithm (MGDA) [14] to multi-task learning. Liu and Vicente [33] provide theoretical analysis of stochastic MGDA. MGDA can be further extended to incorporate RVs (e.g., EPO [37, 38] and Pareto-MTL [31]). Some algorithms (e.g., MOO-SVGD [34], HIGA [49]) can output a solution set without using RVs. However, they optimize several neural networks simultaneously, and so are computationally expensive and need large GPU memory (especially when the neural network is large).

### 3 Proposed Algorithm

As mentioned in Section 1, the fixed uniformly-distributed RVs used in common practice may result in undesirable solution distributions. Instead of using a fixed set of $n$ RVs ($R = [r_1, \ldots, r_n] \in R$), we propose to adapt them so that the resultant solution set is well-distributed. In Section 3.1, we first introduce a reference vector-conditioned neural network so that reference vectors can be easily handled without using a lot more parameters. Section 3.2 then formulates reference vector adaptation as a bilevel optimization problem, and an efficient solver is presented in Section 3.3. Its convergence properties are then studied in Section 3.4.

#### 3.1 Reference Vector-Conditioned Neural Network

In deep learning models, $\phi$ corresponds to the network parameters. Optimizing $\Phi = \{\phi_1, \ldots, \phi_n\}$ means optimizing $n$ neural networks (as in EPO [37] and MOO-SVGD [34]), which is highly inefficient. To alleviate this problem in deep MOO algorithms, Navon et al. [40] and Lin et al. [30] propose to train a single hypernetwork [22] that can output neural network parameters based on the RV. In particular, the Pareto Hypernetwork (PHN) in [40] proposes two ways to optimize the hypernetwork: (i) PHN-LS, which uses linear scalarization, and (ii) PHN-EPO, which uses EPO. However, the hypernetwork still incurs significant computational overhead. For example, the hypernetwork in [40] is around 100 times larger than the base neural network. In this paper, we use the more efficient conditioned network [15, 44]. Specifically, COSMOS [44] concatenates the RV and data sample, and treats this as a joint input to the network. On the other hand, YOTO [15] is originally developed for use with a family of parameterized loss functions, in which the loss parameter is incorporated with the sample into the network via FilM layers [42]. It is theoretically shown that YOTO is as powerful as using $n$ neural networks [15]. In this work, we adapt the YOTO architecture by replacing the loss parameter with the RV. In this way, the proposed model only has a small parameter overhead compared to the single deep network.

#### 3.2 Reference Vector Adaptation via Bilevel Optimization

Consider an RV-conditioned neural network $f(\phi; r)$ with parameter $\phi$ and RV $r$ as input. With a set of RVs $R = [r_1, \ldots, r_n]$, the multi-objective values of a solution set can be written as
Algorithm 1 Gradient-based Multi-Objective Optimization with Adaptive Reference vectors (GMOOAR).

Input: learnable RVs $R$, learning rates $\{\alpha_k, \beta_k\}$, initial parameter $\phi$, number of iterations $K$.  
1: for $k = 1$ to $K$ do  
2: sample a mini-batch $\xi_k$ of samples;  
3: $\phi_{k+1} \leftarrow \phi_k - \beta_k \nabla_{\phi} S(R_k, \phi_k; \xi_k); /* optimize network parameters */$  
4: sample a mini-batch $\pi_k$ of samples;  
5: $R_{k+1} \leftarrow \text{proj}_R(R_k - \alpha_k \nabla_R Q(R_k, \phi_{k+1}; \pi_k)); /* optimize reference vectors */$  
6: end for

\footnote{As we use the RV-conditioned network here, $f_j(\phi)$ (resp. $f(\phi)$) in $[3, 4]$ becomes $f_j(\phi; r)$ (resp. $f(\phi; r)$).}
3.4 Convergence

In this section, we provide convergence analysis for Algorithm\[1\] As in [23], we make the following assumptions on $S(R, \phi)$ and $Q(R, \phi)$.

**Assumption 1.** (i) $S(R, \phi)$ is twice-differentiable in $(R, \phi)$. (ii) $\nabla_\phi S(R, \phi)$, $\nabla^2_{R\phi} S(R, \phi)$, $\nabla^2_{\phi\phi} S(R, \phi)$, $\nabla R Q(R, \phi)$, and $\nabla_\phi Q(R, \phi)$ are Lipschitz continuous w.r.t. $\phi$ with constants $L_s, L_{s,1}, L_{s,2}, L_{q,1}$, and $L_{q,2}$, respectively. (iii) $\nabla^2_{R\phi} S(R, \phi)$, $\nabla^2_{\phi\phi} S(R, \phi)$, and $\nabla Q(R, \phi)$ are Lipschitz continuous w.r.t. $R$ with constants $Ls,3$ and $Ls,4$, and $Lq,3$, respectively. (iv) $S(R, \phi)$ is $\mu_s$-strongly convex in $\phi$. (v) $\|\nabla^2_{R\phi} S(R, \phi)\| \leq C_s$ and $\|\nabla Q(R, \phi)\| \leq C_q$.

Given $\phi^\ast(R)$, the gradient of the upper-level objective $u(R) = Q(R, \phi^\ast(R))$ can be obtained as

$$
\nabla u(R) = \nabla R Q(R, \phi^\ast(R)) - \nabla^2_{R\phi} S(R, \phi^\ast(R))[\nabla^2_{\phi\phi} S(R, \phi^\ast(R))]^{-1}\nabla R Q(R, \phi^\ast(R)),
$$

where $\nabla R Q(R, \phi^\ast(R))$ is the direct gradient with respect to $R$. In [23], $\nabla u(R)$ is evaluated using a surrogate constructed by replacing $\phi^\ast(R)$ in (10) with $\phi_{k+1}$:

$$
\nabla R Q(R_k, \phi_{k+1}) = \nabla R Q(R_k, \phi_{k+1}) - \nabla^2_{R\phi} S(R_k, \phi_{k+1})[\nabla^2_{\phi\phi} S(R_k, \phi_{k+1})]^{-1}\nabla R Q(R_k, \phi_{k+1}).
$$

However, this involves computing the Hessian and is expensive. On the other hand, the proposed algorithm uses $h_k = \nabla R Q(R_k, \phi_{k+1}; \pi_k)$, which is the stochastic estimate of $\nabla R Q(R_k, \phi_{k+1})$. Note that $\nabla R Q(R_k, \phi_{k+1})$ is the first-order approximation of the gradient in (11). Such an approximation is also used in [13, 32]. It greatly reduces the time and space complexities, but leads to a bias that can be bounded by a constant:

$$
\|\nabla R Q(R_k, \phi_{k+1}) - E_{\pi_k}[h_k]\| = \|\nabla^2_{R\phi} S(R_k, \phi_{k+1})[\nabla^2_{\phi\phi} S(R_k, \phi_{k+1})]^{-1}\nabla R Q(R_k, \phi_{k+1})\| 
\leq C_s C_q/\mu_s.
$$

Next, we also make the following assumption similar to [23]. Let $h_k = \nabla \phi S(R_k, \phi_k; \xi_k)$.

**Assumption 2.** For any $k \geq 0$, there exist constants $\sigma_s, \sigma_q$, and $b_q$ such that:

$$
\mathbb{E}_{\xi_k}[h_k] = \nabla \phi S(R_k, \phi_k), \quad \mathbb{E}_{\pi_k}[h_k^2] = \nabla R Q(R_k, \phi_{k+1}) + B_k, \|B_k\| \leq b_q,
$$

$$
\mathbb{E}_{\xi_k}[\|h_k - \nabla \phi S(R_k, \phi_k)\|^2] \leq \sigma_s^2(1 + \|\nabla \phi S(R_k, \phi_k)\|^2),
$$

$$
\mathbb{E}_{\pi_k}[\|h_k - B_k - \nabla R Q(R_k, \phi_{k+1})\|^2] \leq \sigma_q^2.
$$

Let $\Delta^k = \mathbb{E}_{\xi_k}[\|\phi_k - \phi^\ast(R_{k-1})\|$, the expected gap between $\phi_k$ and the optimal network parameter given reference vectors $R_{k-1}$. Similarly, denote the expected gap between $R_k$ and the optimal reference vectors $R^\ast$ in [8] by $\Delta^k = \mathbb{E}_{\pi_k}[\|R_k - R^\ast\|].$

**Theorem 1.** Assume further that $u(R)$ is $\mu_q$-strongly convex, and the step sizes $(\alpha_k, \beta_k)$ satisfy

$$
\alpha_k \leq \min \left\{ c_0 \beta_k^{-3/2}, \frac{1}{\mu_q} \right\}, \quad \frac{\beta_k}{\beta_k - 1} \leq 1 + \beta_k \mu_s/8, \frac{\alpha_k - 1}{\alpha_k} \leq 1 + 3\alpha_k \mu_q/4, \quad (12a)
$$

$$
\beta_k \leq \min \left\{ c_1 \beta_k^{-2/3}, \frac{1}{\mu_s}, \frac{\mu_s}{L_q(1 + \sigma_q^2)}, \frac{\mu_s^2}{48c_0^2 L_q L_2^2} \right\}, \quad 8\mu_q \alpha_k \beta_k \leq \mu_s \beta_k, \forall k \geq 0, \quad (12b)
$$

where $L, L_q$ are constants and $c_0, c_1 > 0$ are free parameters. For any $k \geq 1$, the iterates generated by Algorithm [7] satisfy

$$
\Delta^k_R \preceq \prod_{i=0}^{k-1} (1 - \alpha_i \mu_q), \quad \Delta^k_R + \frac{L_q^2}{\mu_q} \Delta^0_R + \frac{L_q^2}{\mu_q} \left[ \frac{\sigma_s^2}{\mu_s} + \frac{c_1 L_q^2}{\mu_s^3} \sigma_q^2 \right] \alpha_{k-1}^2 + \frac{\beta_k^2}{\mu_q},
$$

$$
\Delta^k_\phi \preceq \prod_{i=0}^{k-1} (1 - \beta_i \mu_s/4), \quad \Delta^k_\phi + \left[ \frac{\sigma_s^2}{\mu_s} + \frac{c_1 L_q^2}{\mu_s^3} \sigma_q^2 \right] \beta_{k-1}, \quad (13)
$$

where $\preceq$ denotes that numerical constants are omitted.
With diminishing step sizes $\alpha_k = c_\alpha/(k + k_\alpha)$, $\beta_k = c_\beta/(k + k_\beta)^{2/3}$, where

$$k_\alpha = \max \left\{ 35 \left( \frac{L_s}{\mu_s} \right)^3 (1 + \sigma_s^2) \frac{(512)^2 L_s^2}{\mu_q^2}, \left( \frac{512}{L_s^2} \right)^2 L_s^2 \left( \frac{L_s}{\mu_s} \right)^3 \left( 1 + \sigma_s^2 \right) \right\},$$

$$c_\alpha = \frac{8}{3\mu_q}, \quad k_\beta = \frac{k_\alpha}{4}, \quad c_\beta = \frac{32}{3\mu_q},$$

the following Corollary shows that the assumptions (12a) and (12b) in Theorem 1 are satisfied, and thus $\Delta_k^\phi$ converge to 0, i.e., $\phi_k$ converges to the optimal solution $\phi^*(R_{k-1})$.

**Corollary 1.** $\phi_k$ converges to the optimal $\phi^*(R_{k-1})$ of (9).

## 4 Experiments

In this section, extensive experiments are performed, including synthetic problems (Section 4.1), multi-task learning (Section 4.2), accuracy-fairness trade-off (Section 4.3), and usage on larger networks (Section 4.4). Finally, ablation study is presented in Section 4.5. All experiments are conducted on an RTX-2080Ti with 11GB memory.

### 4.1 Synthetic Problems

Experiments are performed on four commonly used multi-objective benchmark problems [11, 52, 34] with different numbers of objectives: (i) 2-objective DTLZ2 [12], (ii) 3-objective DTLZ2 [12], (iii) 2-objective scaled-DTLZ2 [12], and (iv) 3-objective MaF1 [8]. Their detailed definitions are in Appendix B.1. The PFs of problems (i) and (ii) are regular, while those of (iii) and (iv) are irregular. The number of inputs is set to 30. We aim to get 15 non-dominated solutions for each 2-objective problem, and 36 non-dominated solutions for each 3-objective problem.

The proposed algorithm (GMOOAR-U using quality function (6) and GMOOAR-HV using quality function (7)) is compared with the state-of-the-art COSMOS [44], which uses fixed reference vectors. For the 2-objective problems, reference vectors for COSMOS are generated by following their strategy in [44]. For the 3-objective problems, we generate reference vectors for COSMOS by the method in [10], which can obtain more uniform reference vectors. For GMOOAR, the reference vectors are initialized randomly. As in [40], a neural network (with 2 hidden layers, each with 20 units) is used. More experimental details can be found in Appendix B.2.

Figure 2 shows the solutions and HV values obtained. For the 2-objective and 3-objective DTLZ2 (Figures 2a and 2b), the solutions obtained by COSMOS are relatively uniform. However, on the scaled-DTLZ2 (Figure 2c) and MaF1 (Figure 2d) with irregular PFs, many of its solutions are near the boundary and close to each other. In contrast, solutions obtained by GMOOAR-U are distributed more uniformly. Solutions obtained by GMOOAR-HV are not uniform, but have higher HV than COSMOS and GMOOAR-U.

Figure 3 shows the solution sets obtained by GMOOAR-U on the 3-objective DTLZ2 when the region of interest is a subspace $\mathcal{R}$ constrained as in (5). Note that imposing this constraint on COSMOS is difficult as (i) it is hard to generate a set of uniformly distributed RVs; and (ii) even with a set of uniformly distributed RVs, they may not lead to a uniformly distributed set of solutions.

### 4.2 Multi-Task Learning

In this experiment, we use three benchmark datasets from [31]: Multi-MNIST, Multi-Fashion, and Multi-Fashion+MNIST. In Multi-MNIST, each image is constructed by putting two different MNIST images together, one at the bottom-right (BR) and the other at the top-left (TL). Similarly, Multi-Fashion images are constructed by combining images from FashionMNIST [50], while Multi-Fashion+MNIST images are constructed by combining one MNIST image with one FashionMNIST image. More details can be found in [31]. The goal is to classify both the BR and TL images correctly, by minimizing the two cross-entropy losses using a single neural network. As in [31, 37, 44], we use the LeNet [28] with multi-head as base network. Details of the reference-vector-conditioned network are in Appendix B.3.

We compare the proposed algorithms (GMOOAR-U and GMOOAR-HV) with (i) EPO [37], (ii) Pareto hypernetworks (PHN-LS and PHN-EPO) [40], (iii) MOO-SVGD [34], and (iv) COSMOS.
Following common practice [44], we obtain a set of $n$ solutions in each iteration (with $n = 15$ in all experiments). They are evaluated on the validation set every 5 epochs. We only keep the solutions of iteration $k_{best}$ as the final solution set, where $k_{best}$ is the iteration that yields the solution set with the largest validation HV. Note that the original implementation of MOO-SVGD (obtained from the [44]. For EPO, PHN-LS, PHN-EPO and COSMOS, we generate reference vectors following the strategy in [44]. For GMOOAR, the reference vectors are initialized randomly. The experiment is repeated 10 times with different random seeds.

Figure 2: Solution sets (red) and HV values obtained on the synthetic datasets. The Pareto-optimal solutions is in blue.

Figure 3: Solution sets obtained by GMOOAR-U with different ($\varphi_1$, $\varphi_2$) settings.
Table 1: Average HV, number of parameters and runtime on the multi-task learning datasets.

<table>
<thead>
<tr>
<th></th>
<th>Multi-MNIST</th>
<th>Multi-Fashion</th>
<th>Fashion-MNIST</th>
<th># parameters</th>
<th>runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPO [27]</td>
<td>2.95±0.02</td>
<td>2.31±0.01</td>
<td>2.86±0.02</td>
<td>478,650</td>
<td>8,370</td>
</tr>
<tr>
<td>PHN-EPO [40]</td>
<td>2.82±0.04</td>
<td>2.16±0.05</td>
<td>2.74±0.05</td>
<td>3,243,410</td>
<td>1,515</td>
</tr>
<tr>
<td>PHN-LS [40]</td>
<td>2.79±0.04</td>
<td>2.14±0.04</td>
<td>2.67±0.06</td>
<td>3,243,410</td>
<td>644</td>
</tr>
<tr>
<td>MOO-SVGD [34]</td>
<td>2.67±0.02</td>
<td>2.02±0.02</td>
<td>2.54±0.04</td>
<td>478,650</td>
<td>8,661</td>
</tr>
<tr>
<td>COSMOS [44]</td>
<td>2.95±0.02</td>
<td>2.31±0.03</td>
<td>2.82±0.03</td>
<td>43,058</td>
<td>260</td>
</tr>
<tr>
<td>GMOOAR-U</td>
<td>3.02±0.01</td>
<td>2.33±0.10</td>
<td>2.91±0.02</td>
<td>43,685</td>
<td>288</td>
</tr>
<tr>
<td>GMOOAR-HV</td>
<td>3.02±0.01</td>
<td>2.33±0.09</td>
<td>2.92±0.02</td>
<td>43,685</td>
<td>348</td>
</tr>
</tbody>
</table>

4.3 Accuracy-Fairness Trade-off

In this experiment, we follow [44] and aim to achieve both high accuracy and fairness on three tabular datasets: Adult [16], Compass [11], and Default [51]. The accuracy is measured by the cross-entropy loss, while fairness is measured by a hyperbolic tangent relaxation of the Difference of Equality of Opportunity (DEO) [41]. As in [40, 44], a 2-hidden-layer multilayer perceptron is used as the base network. More details can be found in Appendix B.4. The experiment is repeated 10 times.

Figure 5 shows the test loss and fairness measure obtained by the solution set with median HV over the 10 runs. Since the datasets are not difficult, the approximated PFs obtained by various algorithms are close. Solutions obtained by GMOOAR are uniformly distributed, while those obtained by EPO, PHN-LS, PHN-EPO and COSMOS are very dense in the top-left region. Solutions obtained by
MOO-SVGD are clustered and only cover parts of the PF. Table 2 shows the HV values obtained. It can be seen that GMOOAR achieves better HVs than the baselines.

<table>
<thead>
<tr>
<th></th>
<th>Adult</th>
<th>Compass</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPO [37]</td>
<td>3.342±0.001</td>
<td>3.709±0.002</td>
<td>3.119±0.001</td>
</tr>
<tr>
<td>PHN-EPO [40]</td>
<td>3.340±0.006</td>
<td>3.709±0.004</td>
<td>3.111±0.005</td>
</tr>
<tr>
<td>PHN-LS [40]</td>
<td>3.341±0.008</td>
<td>3.698±0.007</td>
<td>3.121±0.003</td>
</tr>
<tr>
<td>MOO-SVGD [34]</td>
<td>3.330±0.008</td>
<td>3.716±0.011</td>
<td>3.110±0.005</td>
</tr>
<tr>
<td>COSMOS [44]</td>
<td>3.336±0.006</td>
<td>3.710±0.004</td>
<td>3.114±0.005</td>
</tr>
<tr>
<td>GMOOAR-U</td>
<td>3.344±0.004</td>
<td>3.719±0.008</td>
<td>3.123±0.004</td>
</tr>
<tr>
<td>GMOOAR-HV</td>
<td>3.345±0.005</td>
<td>3.714±0.008</td>
<td>3.123±0.002</td>
</tr>
</tbody>
</table>

Table 2: Average HV and standard deviation on the fairness datasets.

4.4 Larger Networks

To demonstrate that the proposed method can be used on larger networks, we apply GMOOAR on the EfficientNet-B4 [46] with about 17 million parameters. Following [44], we perform experiments on two easy tasks ("Goatee" and "Mustache") and two hard tasks ("Oval Face" and "Pointy Nose") selected from the 40 tasks in CelebA [35]. Since EfficientNet-B4 is around 400 times larger than the LeNet used in previous experiments, PHN-LS, PHN-EPO, and MOO-SVGD cannot be run on our machine. For performance evaluation, the testing cross-entropy loss of each selected task is used. The experiment is repeated 5 times with different random seeds.

Figure 6 shows the test losses obtained by the solution set with median HV over the 5 runs. The corresponding HV values are shown in Table 3. On CelebA-Easy, both GMOOAR-U and GMOOAR-HV outperform COSMOS in terms of HV. On CelebA-Hard, all three algorithms achieve similar HVs, though that of GMOOAR-HV is slightly better.

![Figure 6](image)

(a) CelebA-Easy. (b) CelebA-Hard.

Figure 6: Test losses on the two easy tasks (left) and two hard tasks (right) of CelebA.

Next, we select more tasks from CelebA. Table 4 shows the average HV over 3 runs with different random seeds on CelebA with 3, 5, 10, and 15 tasks. Details of the selected tasks are in Appendix B.4. As can be seen, as the number of tasks increases, GMOOAR exhibits more significant performance...
gains compared to COSMOS. In particular, GMOOAR-HV always achieves the highest HV, which is then closely followed by GMOOAR-U. However, the computation of HV is NP-hard with respect to the number of objectives [53]. Hence, as can be seen from Figure 7, the runtime of GMOOAR-HV grows much faster with the number of tasks than COSMOS and GMOOAR-U. On the other hand, GMOOAR-U is very scalable and its running time changes little with the number of tasks.

<table>
<thead>
<tr>
<th></th>
<th>CelebA-Easy</th>
<th>CelebA-Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSMOS [44]</td>
<td>3.700±0.005</td>
<td>2.217±0.002</td>
</tr>
<tr>
<td>GMOOAR-U</td>
<td>3.710±0.005</td>
<td>2.217±0.002</td>
</tr>
<tr>
<td>GMOOAR-HV</td>
<td>3.711±0.005</td>
<td>2.222±0.006</td>
</tr>
</tbody>
</table>

Table 3: Average HV and standard deviation of solution sets obtained on CelebA with 2 tasks.

<table>
<thead>
<tr>
<th></th>
<th>3 tasks</th>
<th>5 tasks</th>
<th>10 tasks</th>
<th>15 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSMOS [44]</td>
<td>6.925 ± 0.005</td>
<td>11.56 ± 0.09</td>
<td>118.1 ± 1.0</td>
<td>1805 ± 52</td>
</tr>
<tr>
<td>GMOOAR-U</td>
<td>6.953 ± 0.004</td>
<td>15.19 ± 0.18</td>
<td>215.0 ± 6.1</td>
<td>4057 ± 47</td>
</tr>
<tr>
<td>GMOOAR-HV</td>
<td>6.967 ± 0.007</td>
<td>15.34 ± 0.06</td>
<td>217.8 ± 1.5</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4: Average HV and standard deviation of solution sets obtained on CelebA with 3, 5, 10, and 15 tasks. GMOOAR-HV cannot be run on 15 tasks as it takes more than a month on our machine.

4.5 Ablation Study

In this experiment, we study the effects of the RV learning rate $\alpha$ and bandwidth $h$ in (6) on the performance of GMOOAR-U. We use the same setting as in Section 4.3. The experiment is repeated 10 times with different random seeds.

Figure 8 shows the variations of HV with $\alpha$ ($h$ is fixed to 0.01). As can be seen, an $\alpha$ too small results in almost no RV adaption and thus poor performance, while an $\alpha$ too large may lead to unstable learning. Figure 8b shows the variation of HV with $h$ ($\alpha$ is fixed to 0.005). When $h$ is too small (resp. too large), $Q(R, \phi)$ is close to zero (resp. $n$) and the gradient vanishes, making learning difficult.

5 Conclusion

In this paper, we present a novel gradient-based MOO algorithm with adaptive RVs. The proposed algorithm can efficiently adapt the RVs during optimization and provide diverse solutions with a small overhead compared to single-objective optimization. Experiments show the ability of the proposed strategies to obtain well-distributed solutions based on the specified quality function. In the future, we will consider the incorporation of [36] (to obtain a continuous PF) and other state-of-the-art MOO algorithms.

Acknowledgements

This research was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region (Grant 16200021).
References


A Proofs

First, to make the paper self-contained, we state the Lemmas in [23] (with different notations) here.

**Lemma 1.** [23] Lemma 2] Under Assumption 1, for any \( R, \phi \), it holds that
\[
\| \nabla_R Q(R, \phi) - \nabla u(\phi) \| \leq L \| \phi^* - \phi \|
\]
where
\[
L = L_{q,1} + \frac{L_{q,2} C_s}{\mu_s} + C_q \left( \frac{L_{s,1}}{\mu_s} + \frac{L_{s,2} C_s}{\mu_s^2} \right).
\]

**Lemma 2.** [23] Lemma 9] Let \( \alpha > 0 \), \( \{\gamma_j\}_{j \geq 0} \) be a non-increasing, non-negative sequence such that \( \gamma_0 < 1/a \). For any \( k \geq 0 \), it holds that
\[
\sum_{j=0}^{k} \gamma_j \prod_{\ell=j+1}^{k} (1 - \gamma_{\ell} a) \leq \frac{1}{a}
\]

**Lemma 3.** [23] Lemma 10] Fix a real number \( 1 < q \leq 2 \). Let \( \alpha > 0 \), \( \{\gamma_j\}_{j \geq 0} \) be a non-increasing, non-negative sequence such that \( \gamma_0 < 1/(2a) \). Suppose that \( \frac{\gamma_{j+1}}{\gamma_j} \leq 1 + \frac{1}{2(1-J)} \gamma_{\ell} \). For any \( k \geq 0 \), it holds that
\[
\sum_{j=0}^{k} \gamma_j^q \prod_{\ell=j+1}^{k} (1 - \gamma_{\ell} a) \leq \frac{2}{a} \gamma_k^{q-1}.
\]

### A.1 Proof of Theorem 1

We define the following quantities:
\[
G_{m:n}^{(1)} = \prod_{i=m}^{n} (1 - \beta_i \mu_s/4), \quad G_{m:n}^{(2)} = \prod_{i=m}^{n} (1 - \alpha_i \mu_q).
\]
(14)

First, we introduce the following lemma about the optimality bound for the lower-level objective.

**Lemma 4.** Under Assumptions 1–2, suppose that the step sizes satisfy the condition in Theorem 1. For any \( k \geq 1 \), it holds that
\[
\Delta_{\phi}^{k+1} \leq \sum_{\ell=0}^{k} \left( 1 - \beta_{\ell} \mu_s/2 \right) \Delta_{\phi}^{0} + \frac{8}{\mu_s} \left\{ \sigma_s^2 + \frac{4c_{L_s}^2 L_{s,1}^2}{\mu_s} \left[ \sigma_q^2 + 3b_{q}^2 \right] \right\} \beta_k.
\]
(15)

**Proof.** The proof is almost the same as in [23] Lemma 3, except that the bias \( B_k \) in Assumption 2 is bounded by a constant \( b_q \) (instead of a decreasing sequence as in [23]).

**Lemma 5.** Under Assumptions 1–2, suppose that the step sizes satisfy the condition in Theorem 1. For any \( k \geq 1 \), it holds that
\[
\Delta_{R}^{k+1} \leq \prod_{\ell=0}^{k} \left( 1 - \alpha_{\ell} \mu_q \right) \Delta_{R}^{0} + \left[ \frac{2 \sigma_q^2}{\mu_q} + 6 b_q^2 \right] \alpha_k + \frac{2 b_q^2}{\mu_q^2} \sum_{j=0}^{k} \alpha_j \prod_{\ell=j+1}^{k} \left( 1 - \alpha_{\ell} \mu_q \right) \Delta_{\phi}^{j+1}.
\]
(16)

**Proof.** The first few steps below follow [23] Lemma 4. Since \( B_k \) in Assumption 2 is bounded by a constant \( b_q \) (but not by a decreasing sequence as in [23]), our analysis leads to a different bound.

First, we get the following inequality using the projection property:
\[
\| R^{k+1} - R^* \|^2 \leq \| R^k - \alpha_k h_{k} - R^* \|^2 = \| R^k - R^* \|^2 - 2 \alpha_k \langle h_{k}, R^k - R^* \rangle + \alpha_k^2 \| h_{k} \|^2.
\]

Taking the conditional expectation given \( \pi_k \) and using Assumption 2 gives
\[
\mathbb{E}[\| R^{k+1} - R^* \|^2 | \pi_k] \leq \| R^k - R^* \|^2 - 2 \alpha_k \langle \nabla u(R^k), R^k - R^* \rangle + \alpha_k^2 \mathbb{E}[\| h_{k} \|^2 | \pi_k] - 2 \alpha_k \langle \nabla R Q(R^k, \phi^{k+1}) - \nabla u(R^k) \rangle R^k - R^*.
\]
(17)
Due to strong convexity of \( u(R) \), we get
\[
\mathbb{E}[\|R^{k+1} - R^*\|^2 | \pi_k] \leq (1 - 2\alpha_k \mu_q) \|R^k - R^*\|^2 - 2\alpha_k (\nabla_R \mathcal{Q}(R^k, \phi^{k+1}) - \nabla u(R^k) + B_k, R^k - R^*) + \alpha_k^2 \mathbb{E}[\|h^k_q\|^2 | \pi_k]
\]
\[
\leq (1 - \alpha_k \mu_q) \|R^k - R^*\|^2 + \frac{\alpha_k}{\mu_q} \|\nabla_R \mathcal{Q}(R^k, \phi^{k+1}) - \nabla u(R^k) + B_k\|^2 + \alpha_k^2 \mathbb{E}[\|h^k_q\|^2 | \pi_k]
\]
\[
\leq (1 - \alpha_k \mu_q) \|R^k - R^*\|^2 + (2\alpha_k / \mu_q) \{L^2 \|\phi^{k+1} - \phi^*(R^k)\|^2 + b^2_q\} + \alpha_k^2 \mathbb{E}[\|h^k_q\|^2 | \pi_k],
\]  
(18)

where the last inequality is from Lemma[1]

Combining Assumption[2] and Lemma[1] leads to the following inequality:
\[
\mathbb{E}_{\pi}[\|h^k_q\|^2] \leq \tilde{\sigma}_q^2 + 3\tilde{b}_q^2 + 3L^2 \|\phi^{k+1} - \phi^*(R^k)\|^2, \tilde{\sigma}_q^2 \equiv \sigma_q^2 + 3 \sup_{R \in \mathcal{R}} \|\nabla u(R)\|^2.
\]

Substituting the above inequality into (18) and taking the total expectation
\[
\Delta^{k+1}_R \leq \left\{ 1 - \alpha_k \mu_q \right\} \Delta^k_R + \left[ 2\alpha_k / \mu_q \right] L^2 \Delta^{k+1}_\phi + 2\alpha_k b^2_q / \mu_q + \alpha_k^2 \left( \tilde{\sigma}_q^2 + 3\tilde{b}_q^2 + 3L^2 \Delta^{k+1}_\phi \right)
\]
\[
= \left\{ 1 - \alpha_k \mu_q \right\} \Delta^k_R + \left[ 2\alpha_k / \mu_q + 3\alpha_k^2 \right] L^2 \Delta^{k+1}_\phi + \alpha_k^2 \left( \tilde{\sigma}_q^2 + 3\tilde{b}_q^2 \right) + 2\alpha_k b^2_q / \mu_q.
\]

Then, we obtain the following by recursion
\[
\Delta^{k+1}_R \leq G_{0:k}^{(2)} \Delta^k_R + \sum_{j=0}^{k} \left\{ \left( \tilde{\sigma}_q^2 + 3\tilde{b}_q^2 \right) \alpha_j G_{j+1:k}^{(2)} + \frac{2\alpha_j^2}{\mu_q} \alpha_j G_{j+1:k}^{(2)} + \left( \frac{2L^2}{\mu_q} + 3\alpha_0 L^2 \right) \sum_{j=0}^{k} \alpha_j G_{j+1:k}^{(2)} \right\} \cdot \Delta^{j+1}_\phi.
\]

By applying Lemmas[2] and[3] we get
\[
\sum_{j=0}^{k} \alpha_j G_{j+1:k}^{(2)} \leq \frac{1}{\mu_q}, \quad \sum_{j=0}^{k} \alpha_j^2 G_{j+1:k}^{(2)} \leq \frac{2\alpha_k}{\mu_q}.
\]

Therefore,
\[
\Delta^{k+1}_R \leq G_{0:k}^{(2)} \Delta^k_R + \frac{2\alpha_k}{\mu_q} \left( \tilde{\sigma}_q^2 + 3\tilde{b}_q^2 \right) \cdot \alpha_k + \frac{2\alpha_k^2}{\mu_q} + \left( \frac{2L^2}{\mu_q} + 3\alpha_0 L^2 \right) \sum_{j=0}^{k} \alpha_j^2 G_{j+1:k}^{(2)} \Delta^{j+1}_\phi.
\]

Finally, by substituting (15) into (16), we can get
\[
\Delta^{k+1}_R \leq G_{0:k}^{(2)} \left\{ \Delta^k_R + \left[ \frac{2L^2}{\mu_q} + 3\alpha_0 L^2 \right] \Delta^0_\phi \right\} + \frac{2\alpha_k}{\mu_q} \left( \tilde{\sigma}_q^2 + 3\tilde{b}_q^2 \right) \alpha_k + \frac{2\alpha_k^2}{\mu_q} + \frac{8}{\mu_s} \sum_{j=0}^{k} \left( \frac{4\alpha_j^2 L^2}{\mu_s} \tilde{\sigma}_q^2 + 3\tilde{b}_q^2 \right) \Delta^{j+1}_\phi.
\]

(19)

The result shown in Theorem[1] is obtained by only considering the dominating terms.

A.2 Proof of Corollary 1

Recall from (15) that
\[
\Delta^k_\phi \leq \prod_{i=0}^{k-1} \left( 1 - \beta_i \mu_s / 4 \right) \Delta^0_\phi + \left[ \frac{\sigma_q^2}{\mu_s} + \frac{c^2 \mu_s^2 L^2}{\mu_s^2 \tilde{\sigma}_q^2} \right] \beta_{k-1}.
\]

With diminishing step size \( \beta_k = c_k / (k + k_\beta)^{2/3} \), it is obvious that \( \lim_{k \to \infty} \beta_k \to 0 \). Consider the convergence of the first term:
\[
\log(\prod_{i=0}^{k-1} (1 - \beta_i \mu_s / 4)) = \sum_{i=0}^{k-1} \log(1 - \beta_i \mu_s / 4) \leq -\frac{1}{4} \sum_{i=0}^{k-1} \beta_i \mu_s = -\frac{c_k \mu_s}{4} \sum_{i=0}^{k-1} (i + k_\beta)^{-2/3},
\]
where the inequality follows \( \log(1 - x) = -\sum_{i=1}^{\infty} x^i / i \leq -x^2 / 2, \) for \( 0 < x < 1 \). Since \( (i + k_\beta)^{-2/3} \geq (i + k_\beta)^{-1} \) and that the harmonic series is a divergent series, we get \( \sum_{i=0}^{\infty} (i + k_\beta)^{-2/3} = \infty \). Therefore, as \( k \to \infty, \) \( \sum_{i=0}^{k-1} (1 - \beta_i \mu_s / 4) \to -\infty \) and \( \prod_{i=0}^{k-1} (1 - \beta_i \mu_s / 4) \to 0 \). Combining the convergence of the two terms gives \( \lim_{k \to \infty} \Delta^k_\phi \to 0 \), which means \( \phi_k \) converges to the optimal \( \phi^*(R_{k-1}) \) of (9).
B Experimental Details

B.1 Definitions of the Synthetic Problems

The DTZL2 problem \cite{12} minimizes the following \( m \) objectives functions with input \( x = [x_1, \ldots, x_d] \in \mathbb{R}^d \) (where \( 0 \leq x_i \leq 1 \) for \( i = 1, 2, \ldots, d \)):

\[
\begin{align*}
    f_1(x) &= (1 + g(x)) \prod_{i=1}^{m-1} \cos(x_i \pi/2), \\
    f_2(x) &= (1 + g(x)) \prod_{i=1}^{m-2} \cos(x_i \pi/2), \\
    f_3(x) &= (1 + g(x)) \prod_{i=1}^{m-3} \cos(x_i \pi/2), \\
    &\vdots \\
    f_m(x) &= (1 + g(x)) \sin(x_1 \pi/2),
\end{align*}
\]

where \( g(x) = \sum_{i=m}^{d} (x_i - 0.5)^2 \).

The corresponding Pareto front is \( \{ f^*(x) = [f_1^*(x), \ldots, f_m^*(x)] \in \mathbb{R}^m \mid \sum_{i=1}^{m} (f_i^*(x))^2 = 1, f_i^*(x) \geq 0 \text{ for } i = 1, 2, \ldots, m \} \).

In DTLZ2, the minimum value of each objective is zero, which is usually not the case for most machine learning objectives (e.g., when the loss is used as objective, typically it cannot be minimized to zero). To make the synthetic problems more similar to typical machine learning problems, we apply a linear transform on the objectives \( f_i(x) \)'s of DTLZ2, as:

\[
f_i(x) = 0.7 f_i(x) + 0.3, \quad i = 1, 2, \ldots, m.
\]

We call this Scaled-DTLZ2. Its Pareto front is \( \{ f^*(x) \mid \sum_{i=1}^{m} (f_i^*(x) - 0.3)/0.7)^2 = 1, f_i^*(x) \geq 0 \text{ for } i = 1, 2, \ldots, m \} \). It can be shown that the optimal value of each objective is nonzero.

MaF1 \cite{8} minimizes the following \( m \) objectives with input \( x = [x_1, \ldots, x_d] \in \mathbb{R}^d \) (where \( 0 \leq x_i \leq 1 \) for \( i = 1, 2, \ldots, d \)):

\[
\begin{align*}
    f_1(x) &= (1 + g(x)) (1 - \prod_{i=1}^{m-1} x_i), \\
    f_2(x) &= (1 + g(x)) (1 - x_{m-1}) \prod_{i=1}^{m-2} x_i, \\
    f_3(x) &= (1 + g(x)) (1 - x_{m-2}) \prod_{i=1}^{m-3} x_i, \\
    &\vdots \\
    f_m(x) &= (1 + g(x)) (1 - x_1),
\end{align*}
\]

where \( g(x) = \sum_{i=m}^{d} (x_i - 0.5)^2 \). Its PF is \( \{ f^* \mid \sum_{i=1}^{m} (1 - f_i^*) = 1, f_i^* \leq 1 \text{ for } i = 1, 2, \ldots, m \} \).

B.2 Details on Synthetic Problems

We use Adam optimizer \cite{25} and constant learning rates \( \alpha = 0.005, \beta = 0.01 \). The termination condition is set to 5000 iterations. The penalty strength \( \gamma \) in (4) is 5, and the bandwidth \( h \) in (6) for GMOOAR-U is 0.02.

B.3 Reference-Vector-Conditioned Network

For the convolutional neural networks in Sections 4.2 and 4.4, we follow the condition method using the FiLM layer \cite{42} in \cite{15}. Let \( x_i \) be the output of layer \( i \), and \( c \) be the channel index. The
FiLM layer applies a linear transformation to each channel of $x_i$, as: $x_i' = a_{i,c}x_{i,c} + b_{i,c}$. In the experiment, we generate the FiLM parameters $a_i, b_i$ according to reference vector $r$ using an MLP with one hidden layer, i.e., $[a, b] = MLP(r)$. Figure 9 is an illustration of the FiLM layer.

![FiLM layer for CNN](image)

For the LeNet in Section 4.2, we add a FiLM layer after each convolution layer, and the hidden layer dimension of $MLP$ is 20. For the EfficientNet-B4 in Section 4.4, we add a FiLM layer after each of the first four blocks, and the hidden layer dimension of $MLP$ is 20.

On the fairness datasets in Section 4.3, similar to convolutional neural networks, we add a linear transformation layer $x' = ax + b$ after each hidden layer of the MLP. The hidden layer dimension of the $MLP$ for FiLM parameter generation is 20.

### B.4 Details on the Real-World Problems

For most hyperparameters, we follow the settings in [44]. Note that we do not perform grid search for hyperparameters. The Adam optimizer[25] is used on all datasets.

**Multi-MNIST, Multi-Fashion, Multi-Fashion+MNIST** We set the initial learning rates $\alpha_1, \beta_1$ to 0.005 and 0.001, respectively. The learning rate is decayed at epochs 20 and 40 by a factor of 0.01. The batch size is 256. We set aside 10% of the training data as the validation set, and perform validation every 5 epochs. Algorithms terminate after 50 epochs. The penalty strength $\gamma$ is 2 for Multi-MNIST and Multi-Fashion, and 8 for Multi-Fashion+MNIST. The bandwidth $h$ for GMOOAR-U is 0.01.

**Adult, Compass, Default** We use an MLP with two hidden layers as the base network. The numbers of hidden units are 60 and 25 for the first and second hidden layers, respectively. We use the constant learning rate $\alpha = 0.005$ and $\beta = 0.001$. The batch size is 256. We use the provided validation set for validation every 5 epochs. Algorithms terminate after 50 epochs. The penalty strength $\gamma$ is set to 0.01 for all three datasets. The bandwidth $h$ for GMOOAR-U is 0.01.

**CelebA** We use the constant learning rate $\alpha = 0.005$ and $\beta = 0.001$. The batch size is 32. We use the provided validation set for validation every 3 epochs. Algorithms terminate after 30 epochs. The penalty strength $\gamma$ is 3, and the bandwidth $h$ for GMOOAR-U is 0.01.

For experiments on CelebA with 3 tasks, we select "Goatee", "Mustache", and "No_Beard" from CelebA as in [44]. For experiments on CelebA with $m \geq 5$ tasks, we select the first $m$ tasks from CelebA:

**CelebA-5**: "5_o_Clock_Shadow", "Arched_Eyebrows", "Attractive", "Bags_Under_Eyes", "Bald"
**CelebA-10**: "5_o_Clock_Shadow", "Arched_Eyebrows", "Attractive", "Bags_Under_Eyes", "Bald", "Bangs", "Big_Lips", "Big_Nose", "Black_Hair", "Blond_Hair"
The total amount of computing for experiments in this paper is around 600 GPU hours on an Nvidia GTX-2080Ti. All the datasets used in this paper are widely-used public datasets without personally identifiable information or offensive content.

C Full Results on Multi-Task Learning in Section 4.2

As mentioned in Section 4.2 of the main text, many of the solutions obtained by MOO-SVGD are much inferior, and only one of them is in the range shown in Figure 4 (in the main text). Figure 10 below shows a complete plot on the three datasets in Section 4.2. As can be seen, most of the solutions obtained by MOO-SVGD are dominated by solutions obtained by the other algorithms.

Figure 10: BR and TL test losses obtained on the multi-task learning datasets.