Efficient Continuous Pareto Exploration in Multi-Task Learning

Pingchuan Ma [*1] **Tao Du** [*1] **Wojciech Matusik** [1]

Abstract

Tasks in multi-task learning often correlate, conflict, or even compete with each other. As a result, a single solution that is optimal for all tasks rarely exists. Recent papers introduced the concept of Pareto optimality to this field and directly cast multi-task learning as multi-objective optimization problems, but solutions returned by existing methods are typically finite, sparse, and discrete. We present a novel, efficient method that generates locally continuous Pareto sets and Pareto fronts, which opens up the possibility of continuous analysis of Pareto optimal solutions in machine learning problems. We scale up theoretical results in multi-objective optimization to modern machine learning problems by proposing a sample-based sparse linear system, for which standard Hessian-free solvers in machine learning can be applied. We compare our method to the state-of-the-art algorithms and demonstrate its usage of analyzing local Pareto sets on various multi-task classification and regression problems. The experimental results confirm that our algorithm reveals the primary directions in local Pareto sets for trade-off balancing, finds more solutions with different trade-offs efficiently, and scales well to tasks with millions of parameters.

1. Introduction

Conflicting objectives are common in machine learning problems: designing a machine learning model takes into account model complexity and generalizability, training a model minimizes bias and variance errors from datasets, and evaluating a model typically involves multiple metrics that are, more often than not, competing with each other. Such trade-offs among objectives often invalidate the existence of one single solution optimal for all objectives. Instead, they give rise to a set of solutions, known as the Pareto set, with varying preferences on different objectives.

In this paper, we are interested in the topic of recovering Pareto sets in deep multi-task learning (MTL) problems. Despite that MTL is inherently a multi-objective problem and trade-offs are frequently observed in theory and practice, most of prior work focused on obtaining one optimal solution that is universally used for all tasks. To solve this problem, prior approaches proposed new model architectures (Misra et al., 2016) or developed new optimization algorithms (Kendall et al., 2018; Sener & Koltun, 2018). Work on exploring a diverse set of solutions with trade-offs is surprisingly rare and limited to finite and discrete solutions (Lin et al., 2019). In this work, we address this challenging problem by proposing an efficient method that reconstructs a first-order accurate continuous approximation to Pareto sets in MTL problems.

The significant leap from finding a discrete Pareto set to discovering a continuous one requires a fundamentally novel algorithm. Typically, generating one solution in a Pareto set is a time-consuming process that requires expensive optimization (e.g., training a neural network). In order to obtain an efficient algorithm for computing a continuous Pareto set, it is necessary to exploit local information. Our technical method is inspired by second-order methods in multi-objective optimization (MOO) (Hillermeier, 2001; Martín & Schütze, 2018; Schulz et al., 2018) which connect the local tangent plane, the gradient information, and the Hessian matrices at a Pareto optimal solution all in one concise linear equation. This theorem allows us to construct a continuous, first-order approximation of the local Pareto set. However, naively applying this method to deep MTL scales poorly with the number of parameters (e.g., the number of weights in a neural network) due to its need to compute full Hessian matrices. Motivated by other second-order methods in deep learning (Martens, 2010; Vinyals & Povey, 2012), we propose to resolve the scalability issue by using Krylov subspace iteration methods, a family of matrix-free, iterative linear solvers, and present a complete algorithm for generating families of continuous Pareto sets in deep MTL.

We empirically evaluate our method on five datasets with various size and model complexity, ranging from MultiMNIST (Sabour et al., 2017) that consists of 60k images

[*]Equal contribution [1]MIT CSAIL. Correspondence to: Pingchuan Ma <pcma@csail.mit.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

and requires a network classifier with only 20k parameters, to UTKFace (Zhang et al., 2017), an image dataset with 3 objectives and a modern network structure with millions of parameters. The code and data are available online[1]. Experimental results demonstrate that our method generates much denser Pareto sets and Pareto fronts than previous work with small computational overhead compared to the whole MTL training process. We also show in the experiments the continuous Pareto sets can be reparametrized into a low dimensional parameter space, allowing for intuitive manipulation and traversal in the Pareto set. We believe that our efficient and scalable algorithm can open up new possibilities in MTL and foster a deeper understanding of trade-offs between tasks.

2. Related work

Multi-task learning (MTL) is a learning paradigm that jointly optimizes a set of tasks with shared parameters. It is generally assumed that information across different tasks can reinforce the training of shared parameters and improve the overall performance in all tasks. However, since MTL problems share some parameters, performances on different tasks compete with each other. Therefore, trade-offs between performances on different tasks are usually prevalent in MTL. A standard strategy to deal with these trade-offs is to formulate a single-objective optimization problem which assigns weights to each task (Kokkinos, 2017). Choosing weights for each task is typically empirical, problem-specific, and tedious. To simplify the process of selecting weights, prior work suggests some heuristics on adaptive weights (Chen et al., 2018; Kendall et al., 2018). However, this family of methods aims to find one optimal solution for all tasks and is not designed for exploring trade-offs.

Instead of solving a weighted sum of tasks as a single objective, some recent papers directly cast MTL as a multi-objective optimization (MOO) problem and introduce multiple gradient-based methods (MGDA) (Fliege & Svaiter, 2000; Désidéri, 2012; Fliege & Vaz, 2016) to MTL. Sener and Koltun (2018) formally formulate MTL as an MOO problem and propose to use MGDA for training a single optimal solution for all objectives. Another recent approach (Lin et al., 2019), which is the most relevant to our setting, pushes the frontier further by pointing out the necessity of exploring Pareto fronts in MTL and presents an MGDA-based method to generate a discrete set of solutions evenly distributed on the Pareto front. Each solution in their method requires full training from an initial network, which limits its ability to generate a dense set of Pareto optimal solutions.

All the methods discussed so far are based on first-order algorithms in MOO and generate either one solution or a finite set of sparse solutions with trade-offs. A clear distinction between our paper and previous work is that we propose replacing discrete solutions with continuous solution families, allowing for a much denser set of solutions and continuous analysis on them. The advance from discrete to continuous solutions requires a second-order analysis tool in MOO (Hillermeier, 2001; Martín & Schütze, 2018; Schulz et al., 2018), which embeds tangent planes, gradients, and Hessians in one concise linear system. Our work is also related to Hessian-free methods in machine learning (Martens, 2010; Vinyals & Povey, 2012) which rely heavily on Hessian-vector products in neural networks (Pearlmutter, 1994) to solve Hessian systems efficiently.

[1] https://github.com/mit-gfx/ContinuousParetoMTL

3. Preliminaries

In this work, we consider an unconstrained multi-objective optimization problem described by $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ where each $f_i(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}, i = 1, 2, \cdots, m$ represents the objective function of the i-th task to be minimized. For any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{x}$ dominates $\mathbf{y}$ if and only if $\boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{f}(\boldsymbol{y})$ and $\boldsymbol{f}(\boldsymbol{x}) \neq \boldsymbol{f}(\boldsymbol{y})$. A point $\boldsymbol{x}$ is said to be Pareto optimal if $\boldsymbol{x}$ is not dominated by any points in $\mathbb{R}^n$. Similarly, $\boldsymbol{x}$ is locally Pareto optimal if $\boldsymbol{x}$ is not dominated by any points in a neighborhood of $\boldsymbol{x}$. The Pareto set of this problem consists of all Pareto optimal points, and the Pareto front is the image of the Pareto set. In the context of deep MTL, $\boldsymbol{x}$ represents the parameters of a neural network instance and each $f_i(\boldsymbol{x})$ represents one learning objective, e.g., a certain classification loss.

Similar to single-objective optimization, solving for local Pareto optimality is better established than global Pareto optimality. A standard way is to run gradient-based methods to solve for local Pareto optimality then prune the results. Hillermeier et al. (2001) describes the following necessary condition:

Definition 3.1 (Hillermeier et al. 2001)**.** *Assuming each $f_i(\boldsymbol{x})$ is continuously differentiable, a point $\boldsymbol{x}$ is called Pareto stationary if there exists $\boldsymbol{\alpha} \in \mathbb{R}^m$ such that $\alpha_i \geq 0$, $\sum_{i=1}^m \alpha_i = 1$, and $\sum_{i=1}^m \alpha_i \nabla f_i(\boldsymbol{x}) = \mathbf{0}$.*

Proposition 3.1 (Hillermeier et al. 2001)**.** *All Pareto optimal points are Pareto stationary.*

Once a Pareto optimal solution $\boldsymbol{x}^*$ is found, previous papers (Hillermeier, 2001; Martín & Schütze, 2018; Schulz et al., 2018) have proven a strong result revealing the first-order approximation of the local, continuous Pareto set:

Proposition 3.2 (Hillermeier 2001)**.** *Assuming that $\boldsymbol{f}(\boldsymbol{x})$ is smooth and $\boldsymbol{x}^*$ is Pareto optimal, consider any smooth curve $\boldsymbol{x}(t) : (-\epsilon, \epsilon) \to \mathbb{R}^n$ in the Pareto set and passing $\boldsymbol{x}^*$ at $t = 0$, i.e., $\boldsymbol{x}(0) = \boldsymbol{x}^*$, then $\exists \boldsymbol{\beta} \in \mathbb{R}^m$ such that:*

$$\boldsymbol{H}(\boldsymbol{x}^*)\boldsymbol{x}'(0) = \nabla \boldsymbol{f}(\boldsymbol{x}^*)^\top \boldsymbol{\beta} \tag{1}$$

where $\boldsymbol{H}(\boldsymbol{x}^*)$ *is defined as*

$$\boldsymbol{H}(\boldsymbol{x}^*) = \sum_{i=1}^{m} \alpha_i \nabla^2 f_i(\boldsymbol{x}^*) \tag{2}$$

and α_i *is given by Definition 3.1.*

In other words, in the Pareto set, for any smooth curve passing $\boldsymbol{x}^*$, $\boldsymbol{H}(\boldsymbol{x}^*)$ transforms its tangent at $\boldsymbol{x}^*$ to a vector in the space spanned by $\{\nabla f_i(\boldsymbol{x}^*)\}$. By gradually changing the curve, its tangent sweeps the tangent plane of the Pareto set at $\boldsymbol{x}^*$. Essentially, the theorem states that $\boldsymbol{H}(\boldsymbol{x}^*)$ connects the whole tangent plane to the column space of $\nabla \boldsymbol{f}(\boldsymbol{x}^*)^\top$. Note that, however, this theorem is not directly applicable to MTL because of its requirement of full Hessians.

4. Efficient Pareto Set Exploration

Given an initial $\boldsymbol{x}_0 \in \mathbb{R}^n$, our algorithm is executed in two phases: phase 1 uses gradient-based methods to generate a Pareto stationary solution $\boldsymbol{x}_0^*$ from $\boldsymbol{x}_0$. It then computes a few exploration directions to spawn new $\{\boldsymbol{x}_i\}$. We execute phase 1 recursively by feeding it with a newly generated $\boldsymbol{x}_i$. Phase 2 constructs continuous Pareto sets: we first build a local linear subspace at each Pareto stationary solution by linearly combining its exploration directions. We then check whether two local Pareto fronts collide and stitch them to form a larger continuous set. The major challenge brought by deep MTL is that $\mathbb{R}^n$ is the space of neural network parameters. Therefore, it is computationally prohibitive to explicitly calculate Hessian matrices. We describe phase 1 below and phase 2 will be explained in Section 5.

4.1. Gradient-Based Optimization

Our algorithm is compatible with any gradient-based local optimization methods as long as they can return a Pareto stationary solution from any initial $\boldsymbol{x} \in \mathbb{R}^n$. A standard method in MTL is to minimize a weighted sum of objectives with stochastic gradient descent (SGD) (Kokkinos, 2017; Chen et al., 2018; Kendall et al., 2018). Recent papers (Sener & Koltun, 2018; Lin et al., 2019) also proposed to determine a gradient direction online by solving a small convex problem. Essentially, they minimize a loss by combining gradients with fixed or adaptive weights.

4.2. First-Order Expansion

Once a Pareto stationary point $\boldsymbol{x}_0^*$ is found, we explore its local Pareto set by spawning new points $\{\boldsymbol{x}_i\}$. This is decomposed into two steps: computing $\boldsymbol{\alpha}$ in Definition 3.1 at $\boldsymbol{x}_0^*$ and estimating $\{\boldsymbol{v}_i\}$, the basis directions of the tangent plane, from Proposition 3.2. The new points $\{\boldsymbol{x}_i\}$ are then computed by $\boldsymbol{x}_i = \boldsymbol{x}_0^* + s\boldsymbol{v}_i$ where s is an empirical step size whose choice will be discussed in our experiments.

We acquire $\boldsymbol{\alpha}$ at $\boldsymbol{x}_0^*$ by solving the following convex problem (Désidéri, 2012), as suggested by Sener and Koltun (2018):

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \|\sum_{i=1}^{m} \alpha_i \nabla f_i(\boldsymbol{x}_0^*)\|_2 \\ s.t. \quad & \boldsymbol{\alpha} \geq \boldsymbol{0}, \quad \sum_{i=1}^{m} \alpha_i = 1 \end{aligned} \tag{3}$$

Note that the objective can be written as a quadratic form of dimension m. Since m is typically very small, solving it takes little time even for large neural networks.

Given $\boldsymbol{\alpha}$, finding $\{\boldsymbol{v}_i\}$ on the tangent plane at $\boldsymbol{x}_0^*$ can be transformed to finding a solution $(\boldsymbol{v}, \boldsymbol{\beta})$ from Equation (1):

$$\boldsymbol{H}(\boldsymbol{x}_0^*)\boldsymbol{v} = \nabla \boldsymbol{f}(\boldsymbol{x}_0^*)^\top \boldsymbol{\beta} \tag{4}$$

When n is small, we can apply classic $O(n^3)$ methods like Gram-Schmidt process or QR decomposition. However, directly applying them in deep MTL is difficult for two reasons: first, $\boldsymbol{x}_0^*$ is rarely a true Pareto stationary solution because of the early termination in training to avoid overfitting. Second, and more importantly, the large parameter space makes any $O(n^3)$ method prohibitive.

To address the first issue, we propose a variant to Problem (3) to find $\boldsymbol{\alpha}$ as well as a correction vector $\boldsymbol{c}$:

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{c}} \quad & \|\boldsymbol{c}\|_2 \\ s.t. \quad & \boldsymbol{\alpha} \geq \boldsymbol{0}, \quad \sum_{i=1}^{m} \alpha_i = 1 \\ & \sum_{i=1}^{m} \alpha_i (\nabla f_i(\boldsymbol{x}_0^*) - \boldsymbol{c}) = \boldsymbol{0} \end{aligned} \tag{5}$$

In other words, we seek the minimal modification to the gradients such that if we use $\nabla f_i(\boldsymbol{x}_0^*) - \boldsymbol{c}$ as if they were the true gradients, $\boldsymbol{x}_0^*$ would be Pareto stationary. It is easy to show that solving this new optimization problem brings little overhead to the original problem (see supplemental material for the proof):

Proposition 4.1. *Let* $\boldsymbol{\alpha}^*$ *be the solution to Problem (3), then the solution to Problem 5 is* $(\boldsymbol{\alpha}, \boldsymbol{c}) = (\boldsymbol{\alpha}^*, \nabla \boldsymbol{f}(\boldsymbol{x}_0^*)^\top \boldsymbol{\alpha}^*)$.

To address scalability, we consider the following sparse linear system with unknowns $\boldsymbol{v}$:

$$\boldsymbol{H}(\boldsymbol{x}_0^*)\boldsymbol{v} = (\nabla \boldsymbol{f}(\boldsymbol{x}_0^*)^\top - \boldsymbol{c}\boldsymbol{1}^\top)\boldsymbol{\beta} \tag{6}$$

where $\boldsymbol{1}$ is an m-dimensional column vector with all elements equal to 1 and $\boldsymbol{\beta} \in \mathbb{R}^m$ is randomly sampled. In other words, we solve a linear system with the right-hand side sampled from the space spanned by $\{\nabla f_i(\boldsymbol{x}_0^*) - \boldsymbol{c}\}$. Solving such a large linear system in MTL requires an efficient matrix solver. We propose to use Krylov subspace iteration methods because they are matrix-free and iterative solvers, allowing us to solve the system without complete Hessians and terminate with intermediate results. In our experiment, we choose to use the minimal residual method (MINRES), a classic Krylov subspace method designed for symmetric indefinite matrices (Choi et al., 2011).

We now discuss MINRES in more detail to better explain why it is the right tool for this problem. The time complexity of MINRES depends on the time spent on each iteration and the number of iterations. The cost of each iteration is dominated by calculating $\boldsymbol{H}\boldsymbol{v}$ for arbitrary $\boldsymbol{v}$, which is in general $O(n^2)$. However, it is well known that Hessian-vector products can be implemented in $O(n)$ time on computational graphs (Pearlmutter, 1994), giving us the first strong reason to use MINRES. Analyzing the number of iterations is hard because it heavily depends on the rarely available eigenvalue distribution. In practice, MINRES is known to converge very fast for systems with fast decay of eigenvalues (Fong & Saunders, 2012). In our experiments, we specify a maximum number of iterations k. We observed that $k = 50$ was usually sufficient to generate good exploration directions even for networks with millions of parameters. Note that early termination in MINRES still returns meaningful results because the residual error is guaranteed to decrease monotonically with iterations.

To summarize, the efficiency of our exploration algorithm comes from two sources: exploration on the tangent plane and early termination from a matrix-free, iterative solver. The time cost of getting one tangent direction is $O(kn)$, which scales linearly to the network size.

4.3. The Full Algorithm

We now state the complete algorithm for Pareto set exploration in Algorithm 1. It takes as input a seed network and spawns N Pareto stationary networks in a breadth-first style. Any networks put in the queue are returned by `ParetoOptimize` (Section 4.1) and therefore Pareto stationary by design. When such a network is popped out from the queue, `ParetoExpand` generates K exploration directions (Section 4.2) and spawns K child networks. The algorithm then calls `ParetoOptimize` to refine these networks before appending them to the queue, and terminates after M Pareto stationary networks are collected.

For each output network, we also return the objectives, the gradients, and a reference to its parent. This information is mostly used to construct a continuous linear subspace approximating the local Pareto set, which we will describe in Section 5. Another usage is to remove the sign ambiguity in $\boldsymbol{v}_i$: by definition, both $\boldsymbol{v}_i$ and $-\boldsymbol{v}_i$ are on the tangent plane, and an arbitrary choice can lead to a retraction instead of the desired expansion in the Pareto set. In this case, one can use $\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{f}(\boldsymbol{x}^* + s\boldsymbol{v}_i) - \boldsymbol{f}(\boldsymbol{x}^*) \approx s\nabla\boldsymbol{f}(\boldsymbol{x}^*)\boldsymbol{v}_i$ to predict the changes in the objectives and rule out the undesired direction.

When Algorithm 1 is applied to MTL, it is worth noting that `ParetoOptimize` and `ParetoExpand` rarely return the precise solutions because of stochasticity, early termination, and local minima. As a result, good choices of hyperparameters plays an important role. We discussed in more detail two crucial hyperparameters (k and s) and reported the ablation study in Section 6.

Algorithm 1 Efficient Pareto Set Exploration

Input: a random initial neural network $\boldsymbol{x}_0 \in \mathbb{R}^n$
Output: N Pareto stationary networks
$\boldsymbol{x}_0^* \leftarrow$`ParetoOptimize`$(\boldsymbol{x}_0)$
Initialize a queue $q \leftarrow [\boldsymbol{x}_0^*]$
Initialize an empty list to store the output: $output \leftarrow \emptyset$
repeat
  Pop a neural network $\boldsymbol{x}^*$ from q
  for $i = 1$ **to** K **do**
    $\boldsymbol{v}_i \leftarrow$`ParetoExpand`$(\boldsymbol{x}^*)$
    $\boldsymbol{v}_i /= \|\boldsymbol{v}_i\|_2$
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}^* + s\boldsymbol{v}_i$
    $\boldsymbol{x}_i^* \leftarrow$`ParetoOptimize`$(\boldsymbol{x}_i)$
    if No points in $output$ dominates $\boldsymbol{x}_i^*$ **then**
      Append $\boldsymbol{x}_i^*$ to q
      Append $(\boldsymbol{x}_i^*, \boldsymbol{f}(\boldsymbol{x}_i^*), \nabla\boldsymbol{f}(\boldsymbol{x}_i^*), \boldsymbol{x}^*)$ to $output$
    end if
  end for
until The size of $output$ reaches N

5. Continuous Parametrization

In this section, we describe a post-processing step that builds a continuous approximation to the local Pareto set based on the discrete points $\{\boldsymbol{x}_i^*\}$ returned by Algorithm 1. For each $\boldsymbol{x}_i^*$, we collect its K children $\{\boldsymbol{x}_{i_1}^*, \cdots, \boldsymbol{x}_{i_K}^*\}$ and assign a continuous variable $r_{i\to i_j} \in [0, 1]$ to a vector $\boldsymbol{v}_{i\to i_j} = \boldsymbol{x}_{i_j}^* - \boldsymbol{x}_i^*, j = 1, 2, \cdots, K$. The local Pareto set at $\boldsymbol{x}_i^*$ is then constructed by

$$S(\boldsymbol{x}_i^*) = \{\boldsymbol{x}_i^* + \sum_{i=1}^{K} r_{i\to i_j}\boldsymbol{v}_{i\to i_j} | r_{i\to i_j} \geq 0, \sum_{i=1}^{K} r_{i\to i_j} \leq 1\} \tag{7}$$

In other words, $S(\boldsymbol{x}^*)$ is the convex hull of $\boldsymbol{x}_i^*$ and its children $\{\boldsymbol{x}_{i_1}^*, \cdots, \boldsymbol{x}_{i_K}^*\}$. This construction is justified by the fact that a linear combination of tangent vectors is still on the tangent plane. As a special case, when there are only 2 objectives and $K = 1$, $\{\boldsymbol{x}^*\}$ forms a chain, and therefore $S = \cup_i S(\boldsymbol{x}_i^*)$ becomes a piecewise linear set in $\mathbb{R}^n$.

It is possible that two continuous families can collide in the objective space, creating a larger continuous Pareto front. In this case, we create a stitching point in both families and crop solutions dominated by the other family. By repeatedly applying this idea, a single continuous Pareto front covering all families can possibly be created, providing the ultimate solution to continuous traversal in the whole Pareto front. We illustrate this idea on MultiMNIST with our experimental results in Section 6.4.

Since the continuous approximation interpolates different tangent directions, having more directions can enrich the coverage of the continuous set and offer more options to users. It is therefore natural to ask whether the set of tangent directions discovered in the last section could be augmented even further by adding more directions without downgrading the quality of the Pareto front. For the special case of two objectives ($m = 2$), it turns out that we can augment the set of known tangent directions with a *null vector* of the Hessian matrix, as stated in the following proposition:

Proposition 5.1. *Assuming $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^2$ is sufficiently smooth. Let $\boldsymbol{x}^*$ be a Pareto optimal point and consider a curve $\boldsymbol{c_d}(t) : \mathbb{R} \to \mathbb{R}^2$ defined as $\boldsymbol{c_d}(t) = \boldsymbol{f}(\boldsymbol{x}^* + t\boldsymbol{d})$. If $\boldsymbol{x}(t) : (-\epsilon, \epsilon) \to \mathbb{R}^2$ is any smooth curve in Proposition 3.2 that satisfies $\boldsymbol{H}(\boldsymbol{x}^*)\boldsymbol{x}'(0) \neq \boldsymbol{0}$, then for any $\boldsymbol{u} \in \mathbb{R}^n$:*

1) $\boldsymbol{c}_{\boldsymbol{x}'(0)}$ and $\boldsymbol{c}_{\boldsymbol{x}'(0)+\boldsymbol{u}}$ have the same value and tangent direction $(-\alpha_2, \alpha_1)$ at $t = 0$;

2) Furthermore, if $\boldsymbol{u}$ is a null vector of $\boldsymbol{H}(\boldsymbol{x}^*)$, i.e., $\boldsymbol{H}(\boldsymbol{x}^*)\boldsymbol{u} = \boldsymbol{0}$, then $\boldsymbol{u}$ is not parallel to $\boldsymbol{x}'(0)$, and $\boldsymbol{c}_{\boldsymbol{x}'(0)}(t)$ and $\boldsymbol{c}_{\boldsymbol{x}'(0)+\boldsymbol{u}}(t)$ have the same curvature at $t = 0$.

In this proposition, $\boldsymbol{c_d}(t)$ is a parametrized 2D curve: it considers a straight-line trajectory in $\mathbb{R}^n$ that passes $\boldsymbol{x}^*$ in the direction of $\boldsymbol{d}$ and uses $\boldsymbol{f}$ to map this trajectory to the space of $\mathbb{R}^2$, generating a 2D curve. This proposition states that if a tangent direction $\boldsymbol{v}$ is known and if we also have a null vector $\boldsymbol{u}$, then the two curves $\boldsymbol{c_v}$ and $\boldsymbol{c_{v+u}}$ are very similar at $\boldsymbol{x}^*$ in the sense that they share the same value, gradients, and curvature. This means that for each tangent direction $\boldsymbol{v}$ found in the previous section, $\boldsymbol{v} + \boldsymbol{u}$ can also be used as a backbone direction together with $\boldsymbol{v}$ for continuous parametrization without downgrading the quality of the reconstructed Pareto front.

While this proposition is generally not applicable to real problems due to its need for null vectors, it still has interesting theoretical implications: the fact that $\boldsymbol{c_{v+u}}$ and $\boldsymbol{c_v}$ share the same gradients should not be surprising as $\boldsymbol{v} + \boldsymbol{u}$ also satisfies Equation (4), but it is less obvious to see that they actually share the same curvature at $\boldsymbol{f}(\boldsymbol{x}^*)$, which we illustrate in Section 6.4 and will prove in our supplemental material. In practice, we observed that neural networks typically have a Hessian matrix with a null space whose dimension is much higher than m. This means a very large set of bases, while not often accessible in real problems, can in theory be used to greatly enrich the Pareto set.

6. Experimental Results

6.1. Datasets, Metrics, and Baselines

We applied our method to five datasets in three categories: 1) MultiMNIST (Sabour et al., 2017) and its two variants FashionMNIST (Xiao et al., 2017) and MultiFashionMNIST, which are medium-sized datasets with two classification tasks; 2) UCI Census-Income (Kohavi, 1996), a medium-sized demographic dataset with three binary prediction tasks; 3) UTKFace (Zhang et al., 2017), a large dataset of face images. We used LeNet5 (LeCun et al., 1998) (22,350 parameters) for MultiMNIST and its variants, two-layered multilayer perceptron (158,598 parameters) for UCI Census-Income, and ResNet18 (He et al., 2016) (tens of millions of parameters) for UTKFace. Please refer to our supplemental material for more information about the network architectures, task descriptions, and implementation details in each dataset.

We measure the performance of a method by two metrics: the time cost and the hypervolume (Zitzler & Thiele, 1999). We measure the time cost by counting the evaluations of objectives, gradients, and Hessian-vector products. The hypervolume metric, explained in Figure 1, is a classic MOO metric for measuring the quality of exploration. More concretely, this metric takes as input a set of explored solutions in the objective space and returns a score. Larger hypervolume score indicates a better Pareto front. Using the two metrics, we define that a method is more efficient if, within the same time budget, it generates a Pareto front with a larger hypervolume, or equivalently, if it generates the Pareto front with a similar hypervolume but within shorter time. For all figures in this section, we use the same random seed whenever possible and report results from more random seeds in the supplemental material.

Our method is not directly comparable to any baselines because no prior work aims to recover a continuous Pareto front in MTL. Instead, we devised two experiments, which we call the sufficiency and necessity tests, to show its effectiveness (Section 6.3). In the sufficiency test, we consider four previous methods: GradNorm (Chen et al., 2018), Uncertainty (Kendall et al., 2018), MGDA (Sener & Koltun, 2018), and ParetoMTL (Lin et al., 2019). These methods aim at pushing an initial guess to one or a few discrete Pareto optimal solutions. For them, we show that our Pareto

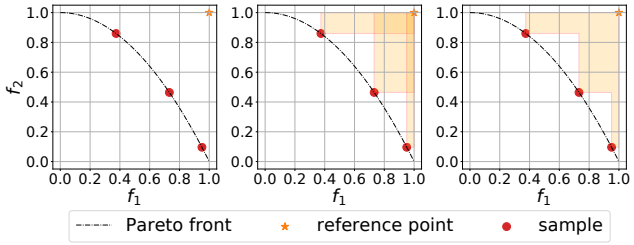

Figure 1. Definition of hypervolume. Given a set of sample points (red circles) in $\mathbb{R}^m$, the hypervolume is computed by picking a reference point (orange star), creating axis-aligned rectangles from each point, and calculating the size of their union (orange polygon).

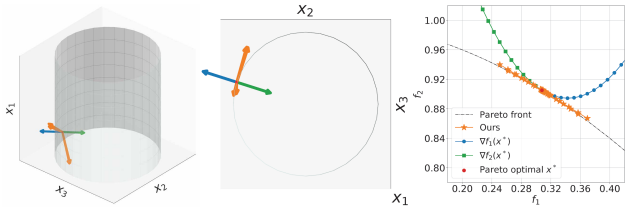

Figure 2. Comparisons of different exploration directions at a Pareto optimal solution $\boldsymbol{x}^*$ (red circle). Left: the analytic Pareto set (the cylindrical surface) of ZDT2-variant, the gradients $\nabla f_1(\boldsymbol{x}^*)$ (blue) and $\nabla f_2(\boldsymbol{x}^*)$ (green), and our exploration directions $\{\boldsymbol{v}_i\}$ (orange) predicted by MINRES. Middle: a top-down view to show ours are almost tangent to the Pareto set. Right: plots of $\boldsymbol{f}(\boldsymbol{x}^*+s\boldsymbol{d})$ where $s \in [-0.1, 0.1]$ and $\boldsymbol{d}$ is $\nabla f_1(\boldsymbol{x}^*)$ (blue circles), $\nabla f_2(\boldsymbol{x}^*)$ (green squares), and our directions (orange stars).

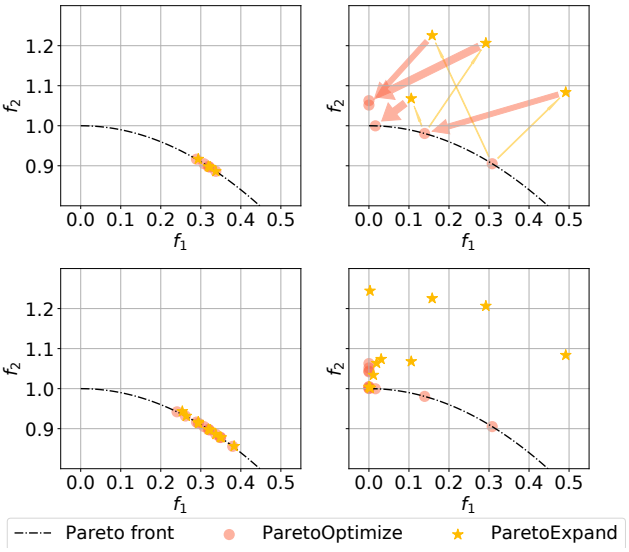

Figure 3. Comparisons of two expansion strategies in Algorithm 1. Starting with a given Pareto optimal point (red circle), the algorithm iteratively calls `ParetoExpand` (orange arrows from circles to stars) and `ParetoOptimize` (red arrows from stars to circles), generating a series of explored points (orange stars) and Pareto optimal solutions (red circles). Arrow thickness indicates the time cost of each function. Top row: expansion using our predicted tangent directions (top left) versus using gradients (top right). Bottom row: running both strategies until 10 Pareto optimal points were collected.

expansion procedure is a fast yet powerful complement by comparing the time and hypervolume before and after running it as a post-processing step. We call this experiment the sufficiency test as it demonstrates our method is able to quickly explore Pareto sets and Pareto fronts.

Our necessity test, which focuses on the value of the tangent directions in exploring Pareto fronts, deserves some discussions on its baselines. There is a trivial baseline for Pareto expansion: rerunning an SGD-based method from scratch to optimize a perturbed weight combination of objectives. Since each new run requires full training, our method clearly dominates this baseline (30 times faster on MultiMNIST). Another trivial baseline is to use a random direction instead of the tangent direction for Pareto expansion. We tested this idea but do not include it in our experiments as its performance is significantly worse than any other methods, which is understandable due to the high dimensionality of neural network parameters: with the increase of dimensionality, the chance of a random guess still staying on the tangent plane decays exponentially. The baseline we considered in this experiment is WeightedSum, which runs SGD from the last Pareto optimal solution but with weights on objectives different from the weights used in training. Specifically, we choose weights from one-hot vectors for each task as well as a vector assigning equal weights to every task. We call this experiment the necessity test as we use this experiment to establish that the choice of expansion strategies is not arbitrary, and tangent directions are indeed the source of efficiency in our method.

6.2. Synthetic Examples

6.2.1. ZDT2-VARIANT

Our first example, ZDT2-variant, was originated from ZDT2 (Zitzler et al., 2000), a classic benchmark problem in multi-objective optimization with $n = 3$ and $m = 2$. Both the Pareto set and the Pareto front of this example can be computed analytically. This makes ZDT2-variant an ideal example for visualizing Proposition 3.2 and Algorithm 1. Figure 2 compares the gradients to our tangent directions when used to explore the Pareto front. We used MINRES with $k = 1$ to solve 5 tangent directions. It can be seen that our directions are much closer to the Pareto set and tracked the true Pareto front much better than the gradients. We further compare their performances in Algorithm 1 with MGDA (Désidéri, 2012; Sener & Koltun, 2018) as the optimizer in Figure 3. This figure shows that the gradients expanded the neighborhood not on the Pareto set but to the dominated interior, resulting in a much more expensive correction step to follow. On the other hand, expanding with our predicted tangents steadily grew the solution set along the Pareto front.

6.2.2. MULTIMNIST SUBSET

To understand the behavior of our algorithm when neural networks are involved, we picked a subset of 2048 images from MultiMNIST and trained a simplified LeNet (LeCun et al., 1998) with 1500 parameters to minimize two classification errors. We generated an empirical Pareto front by optimizing the weighted sum of the two objectives with

varying weights. We then picked a Pareto optimal x^* and visualized trajectories generated by traversing along gradients and the approximated tangents after 10, 20, and 50 iterations of MINRES (Figure 4 left). Just as in ZDT2-variant, our approximated tangents tracked the Pareto front much more closely. We then compared using approximated tangents after 50 iterations of MINRES (MINRES-50) to the WeightedSum baseline (Section 6.1) after 50 iterations of SGD. The two methods had roughly the same time budgets, and MINRES-50 outperformed the WeightedSum baseline in that it explored a much wider Pareto front (Figure 4 middle). Specifically, its advantage comes from a much larger step size enabled by the approximated tangents (Figure 4 right).

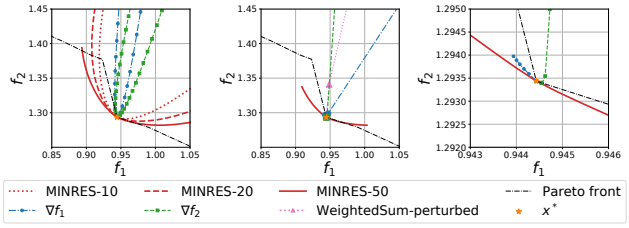

Figure 4. Comparisons of expansion strategies on MultiMNIST subset. Left: trajectories of different expansion strategies in the objective space. Curves closer to the Pareto front mean better expansion. Middle: trajectories generated by running SGD to minimize f_1 (blue circles), f_2 (green squares), or a weighted combination (pink triangles) within the same time budget as MINRES-50 (red). Right: a zoom-in version showing that tiny step sizes have to be used by SGD to avoid deviating from the Pareto front too much.

6.3. Pareto Expansion

We first conducted the sufficiency test described in Section 6.1 to analyze Pareto expansion, the core of our algorithm. We ran ParetoMTL, the state of the art, on all datasets to generate discrete seeds for Pareto expansion. Moreover, for smaller datasets (MultiMNIST and its variants), we also ran the other baselines for a more thorough analysis. Compared to the time cost of generating discrete solutions (Table 1 column 2), our Pareto expansion only used a small fraction of the training time (Table 1 column 4) but generated much denser Pareto fronts (Figure 5 and Table 1 column 5). This experiment, as a natural extension to the synthetic experiments, confirms the efficacy of Pareto expansion on large neural networks and datasets.

The sufficiency test has established that our expansion method has a positive effect on discovering more solutions. However, one can still argue there could be simpler expansion strategies that are as good as ours. It remains to show that the benefit indeed comes from approximated tangent directions. We verified this with the necessity test described in Section 6.1, which directly compared our Pareto expansion

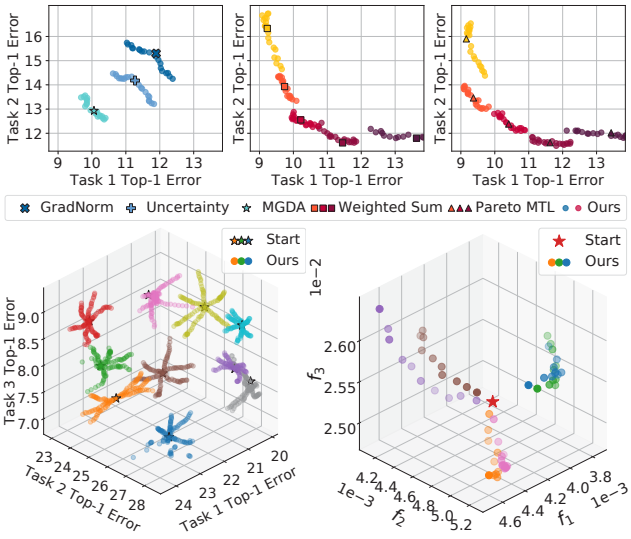

Figure 5. Using Pareto expansion to grow dense Pareto fronts (colorful circles) from discrete solutions generated by baselines on MultiMNIST and its variants (top row), UCI Census-Income (bottom left), and UTKFace (bottom right). Points expanded from the same discrete solution have the same color.

to the WeightedSum expansion strategy. Starting with the same seed solution, we gave both methods the same time budget, so the area of their expansions directly reflected their performances. We display the results on MultiMNIST, UCI Census-Income, and UTKFace in Figure 6. New solutions were generated after each run of MINRES in our method and after each epoch in WeightedSum. We provide more results in the supplementary material. We see from

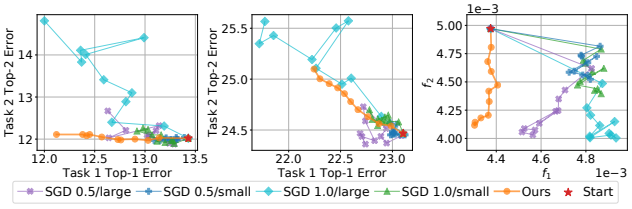

Figure 6. Comparisons of two expansion methods (ours and running SGD with a weighted sum) from a given Pareto optimal network. We display results on MultiMNIST (left), UCI Census-Income (middle), and UTKFace (right). In all figures, lower left regions mean better solutions. All SGD methods are labeled with preference on task 1/the type of learning rates (large or small). UCI Census-Income and UTKFace have three objectives and we show results from considering f_1 and f_2 only. Results on FashionMNIST and MultiFashionMNIST and other combinations of objectives in UCI Census-Income and UTKFace can be found in our supplemental material.

Table 1. A summary of the improvement brought by calling Pareto expansion from solutions generated by baselines. TRAIN: the training time used by each baseline, measured by the aggregated number of evaluations of objectives, gradients, and Hessian-vector products; HV: the hypervolume of the solution at the end of training; EXPAND: the time cost of our Pareto expansion; NEW HV: the hypervolume after expansion. Larger hypervolume is better.

MULTIMNIST	TRAIN	HV	EXPAND	NEW HV
GRADNORM	21150	7.463	4520	7.628
UNCERTAINTY	21150	7.615	4520	7.756
MGDA	21150	7.831	4520	7.896
WEIGHTEDSUM	70500	8.019	22600	8.034
PARETOMTL	106281	8.025	22600	8.046
UCI	TRAIN	HV	EXPAND	NEW HV
WEIGHTEDSUM	467400	5.685	165600	5.725
PARETOMTL	934888	5.642	165600	5.675
FACE	TRAIN	HV	EXPAND	NEW HV
PARETOMTL	35568	2.257	9920	5.030

these experiments that our method discovered solutions that clearly dominated what WeightedSum returned on 4 out of the 5 datasets except UCI Census-Income. From this experiment, we conclude the tangent directions in Pareto expansion are indeed the core reason for the good performance of our algorithm.

The effectiveness of our Pareto expansion method can also be understood by noticing it uses higher-order derivatives than previous work for determining the optimal expansion directions. Consider the three possible methods for the task of expanding the local Pareto set from a known Pareto optimal solution $\boldsymbol{x}^*$: simply retraining the neural network from scratch with a different initial guess reuses nothing from $\boldsymbol{x}^*$; rerunning SGD from $\boldsymbol{x}^*$ leverages the first-order gradient information at $\boldsymbol{x}^*$; our method exploits both the first-order and the second-order information at $\boldsymbol{x}^*$ and therefore is the most effective among the three.

It is worth mentioning that our Pareto expansion strategy is still a local optimization method, meaning that it inevitably suffers from being trapped in local minima. As a result, there is no theoretical guarantee on the resulting Pareto fronts being globally Pareto optimal. We alleviate this issue by exploring from multiple Pareto optimal solutions returned by previous methods and stitching them together, which we will explain shortly in the next section.

6.4. Continuous Parametrization

From discrete solutions returned by Algorithm 1, our continuous parametrization creates low-dimensional, locally smooth Pareto sets. Moreover, we stitch them together when

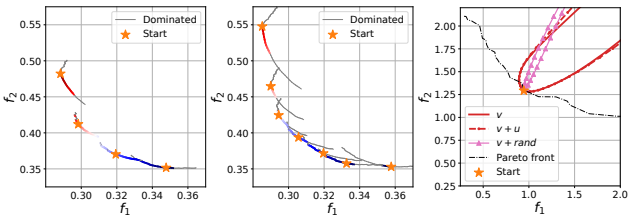

Figure 7. Illustrations of the continuous parametrization. Left: Continuous Pareto fronts grown from 4 Pareto optimal solutions (orange stars) on MultiMNIST. Curve colors indicate the value of t from -1 (dark blue) to 1 (dark red) and the thin gray lines indicated dominated solutions. Middle: larger approximations were formed by stitching 10 Pareto fronts. Right: comparisons between expansions with three directions: a tangent $\boldsymbol{v}$ (red and solid), $\boldsymbol{v}$ plus a null vector $\boldsymbol{u}$ (red and dash), and $\boldsymbol{v}$ plus a random direction (pink triangles).

their Pareto fronts collide, forming a larger continuous approximation. We illustrate this idea in Figure 7: we ran Algorithm 1 on MultiMNIST with $K = 2$ and $N = 10$ for each Pareto stationary solution $\boldsymbol{x}^*$, generating two chains of solutions favoring small f_1 and small f_2 respectively. As described in Section 5, we then constructed a piecewise linear curve parametrized by $t \in [-1, 1]$. By continuously varying t, we explore a diverse set of solutions from favoring small f_1 to small f_2. We highlight this mapping from a single control variable to a wider-range Pareto front because it demonstrates the real advantage of a continuous reconstruction over discrete solutions. As a straightforward application, one can analyze this mapping by running single-variable gradient-descent to pick an optimal solution, which would be impossible if only discrete solutions were provided. We give more results in the supplemental material.

We conclude our discussion on continuous parametrization by demonstrating Proposition 5.1 on MultiMNIST subset in Figure 7. We precomputed its full null space and revealed over 600 bases. We then expanded the Pareto set at a Pareto optimal $\boldsymbol{x}^*$ with three directions: a tangent direction $\boldsymbol{v}$, $\boldsymbol{v}$ plus a null vector $\boldsymbol{u}$, and $\boldsymbol{v}$ plus a random direction. As expected, expanding with the first two directions led to trajectories sharing the same gradient and curvature at $\boldsymbol{x}^*$, showing that we can enrich the Pareto set by adding null space bases without degrading its quality.

6.5. Ablation Study

Finally, we conducted ablations tests on two crucial hyperparameters in our algorithm: the maximum number of iterations k in MINRES and the step size s that controls the expansion speed. We started with a random Pareto stationary point $\boldsymbol{x}^*$ returned by ParetoMTL, followed by running Algorithm 1 with fixed parameters $K = 1$ and $N = 5$ on

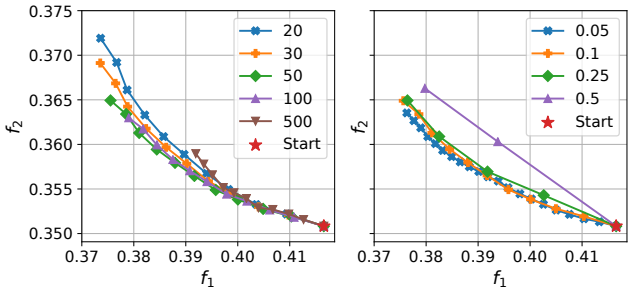

Figure 8. Pareto expansion from a Pareto optimal solution (red star) on MultiMNIST by various k and s. Left: expansion with fixed s and $k \in \{20, 30, 50, 100, 500\}$. Lower curves are more dominant. Right: expansion with fixed k and $s \in \{0.05, 0.1, 0.25, 0.5\}$. The number of runs was chosen such that its product with s equals 1.

MultiMNIST and its two variants. The results are summarized in Figure 8, Table 2, and the supplemental material.

To see the influence of k, we fixed $s = 0.1$ and ran experiments with $k \in \{20, 30, 50, 100, 500\}$, whose trajectories are in Figure 8. Between $k = 20$ and 50, the trajectories were pushed towards its lower left, indicating a better approximated Pareto front. This is as expected since more iterations in MINRES were consumed. This trend plateaued between $k = 50$ and 100. Moreover, the tail of the trajectory drifted away after $k = 500$ iterations. We hypothesized that the tangent after 500 iterations explored a new region in $\mathbb{R}^n$ where the constant step size $s = 0.1$ was not proper. Based on these observations, we used $k = 50$ in all experiments.

To understand how s affects expansion, we reran the same experiments with a fixed $k = 50$ and chose s from $\{0.05, 0.1, 0.25, 0.5\}$. For each s, we set the number of points to be generated to $1/s$, i.e., the product of the step size and the step number is constant. From Figure 8 right, we noticed a conservative s was likely to follow the Pareto front more closely while an aggressive step size quickly led the search to the dominated interior. This is consistent with the fact that our tangents are a first-order approximation to the true Pareto set.

7. Conclusions

We presented a novel, efficient method to construct continuous Pareto sets and fronts in MTL. Our method is originated from second-order analytical results in MOO, and we combined it with matrix-free iterative linear solvers to make it a practical tool for large-scale problems in MTL. We analyzed thoroughly the source of efficiency with demonstrations on synthetic examples. Moreover, experiments showed our method is scalable to modern machine learning datasets and networks with millions of parameters.

Table 2. Hypervolumes (HV) from the ablation study on hyperparameters k and s. The time cost of experiments is proportional to k and inverse proportional to s. Best results are in bold.

k	20	30	50	100	500
HV	7.731	**7.739**	7.734	7.727	7.669
s	0.05	0.10	0.25	0.50	
HV	**7.741**	7.733	7.728	7.712	

While the majority of work in MTL aims to find one near-optimal solution, we believe conflicting objectives in MTL are common and the full answer should be a wide range of candidates with varying trade-offs. Although we are not the first to explore Pareto fronts in MTL or apply second-order techniques to neural networks, we are, to our best knowledge, the first to introduce second-order analysis to Pareto exploration in MTL and the first to propose a continuous reconstruction. We believe our work enables lots of opportunities that would otherwise be impossible if only finite, sparse, and discrete solutions were given, for example, revealing the dimensionality and underlying structure of local Pareto sets, developing interpretable analysis tools for deep MTL networks, and encoding dense Pareto sets and fronts with limited storage.

Acknowledgments

We thank Tae-Hyun Oh for his insightful suggestions and constructive feedback on Krylov subspace methods. We also thank all reviewers for their comments. We thank Buttercup Foshey (and Michael Foshey) for her emotional support during this work. This work is supported by the Intelligence Advanced Research Projects Activity under grant 2019-19020100001, the Defense Advanced Research Projects Agency under grant N66001-15-C-4030, and the National Science Foundation under grant CMMI-1644558.

References

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pp. 794–803, 2018.

Choi, S.-C. T., Paige, C. C., and Saunders, M. A. MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems. *SIAM Journal on Scientific Computing*, 33(4):1810–1836, 2011.

Désidéri, J.-A. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5-6):313–318, 2012.

Fliege, J. and Svaiter, B. F. Steepest descent methods for

multicriteria optimization. *Mathematical Methods of Operations Research*, 51(3):479–494, 2000.

Fliege, J. and Vaz, A. I. F. A method for constrained multiobjective optimization based on SQP techniques. *SIAM Journal on Optimization*, 26(4):2091–2119, 2016.

Fong, D. C.-L. and Saunders, M. CG versus MINRES: An empirical comparison. *Sultan Qaboos University Journal for Science [SQUJS]*, 17(1):44–62, 2012.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hillermeier, C. Generalized homotopy approach to multiobjective optimization. *Journal of Optimization Theory and Applications*, 110(3):557–583, 2001.

Hillermeier, C. et al. *Nonlinear multiobjective optimization: a generalized homotopy approach*, volume 135. Springer Science & Business Media, 2001.

Kendall, A., Gal, Y., and Cipolla, R. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.

Kohavi, R. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pp. 202–207, 1996.

Kokkinos, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6129–6138, 2017.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lin, X., Zhen, H.-L., Li, Z., Zhang, Q.-F., and Kwong, S. Pareto multi-task learning. In *Advances in Neural Information Processing Systems*, pp. 12037–12047, 2019.

Martens, J. Deep learning via hessian-free optimization. In *ICML*, volume 27, pp. 735–742, 2010.

Martín, A. and Schütze, O. Pareto tracer: a predictor–corrector method for multi-objective optimization problems. *Engineering Optimization*, 50(3):516–536, 2018.

Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3994–4003, 2016.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. In *Advances in neural information processing systems*, pp. 3856–3866, 2017.

Schulz, A., Wang, H., Grinspun, E., Solomon, J., and Matusik, W. Interactive exploration of design trade-offs. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

Sener, O. and Koltun, V. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pp. 527–538, 2018.

Vinyals, O. and Povey, D. Krylov subspace descent for deep learning. In *Artificial Intelligence and Statistics*, pp. 1261–1268, 2012.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zhang, Z., Song, Y., and Qi, H. Age progression/regression by conditional adversarial autoencoder. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5810–5818, 2017.

Zitzler, E. and Thiele, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Zitzler, E., Deb, K., and Thiele, L. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.