Controllable Pareto Multi-Task Learning

Xi Lin\textsuperscript{1}, Zhiyuan Yang\textsuperscript{1}, Qingfu Zhang\textsuperscript{1}, Sam Kwong\textsuperscript{1}
\textsuperscript{1}City University of Hong Kong,
{xi.lin, zhiyuan.yang}@my.cityu.edu.hk, {qingfu.zhang, cssamk}@cityu.edu.hk

Abstract

A multi-task learning (MTL) system aims at solving multiple related tasks at the same time. With a fixed model capacity, the tasks would be conflicted with each other, and the system usually has to make a trade-off among learning all of them together. Multiple models with different preferences over tasks have to be trained and stored for many real-world applications where the trade-off has to be made online. This work proposes a novel controllable Pareto multi-task learning framework, to enable the system to make real-time trade-off switch among different tasks with a single model. To be specific, we formulate the MTL as a preference-conditioned multiobjective optimization problem, for which there is a parametric mapping from the preferences to the optimal Pareto solutions. A single hypernetwork-based multi-task neural network is built to learn all tasks with different trade-off preferences among them, where the hypernetwork generates the model parameters conditioned on the preference. At the inference time, MTL practitioners can easily control the model performance based on different trade-off preferences in real-time. Experiments on different applications demonstrate that the proposed model is efficient for solving various multi-task learning problems.

1 Introduction

Multi-task learning (MTL) is important for many real-world applications, such as computer vision (Kokkinos, 2017), natural language processing (Subramanian et al., 2018), and reinforcement learning (Van Moffaert and Nowé, 2014). In these problems, multiple tasks are needed to be learned at the same time. An MTL system usually builds a single model to learn several related tasks together, in which the positive knowledge transfer could improve the performance for each task. In addition, using one model to conduct multiple tasks is also good for saving storage costs and reducing the inference time, which could be crucial for many applications (Standley et al., 2020).

However, with fixed learning capacity, different tasks could be conflicted with each other, and can not be optimized simultaneously (Zamir et al., 2018). The practitioners might need to carefully design and train the tasks into different groups to achieve the best performance (Standley et al., 2020). A considerable effort is also needed to find a set of suitable weights to balance the performance of each task (Kendall et al., 2018; Chen et al., 2018; Sener and Koltun, 2018). For some applications, it might need to train and store multiple models for different trade-off preferences among the tasks (Lin et al., 2019).

Figure 1: Controllable Pareto MTL allows practitioners to control the trade-offs among tasks in real time with a single model, which could be desirable for many MTL applications.
In many real-world MTL applications, the system will need to make a trade-off among different tasks in real time, and it is desirable to have the whole set of optimal trade-off solutions. For example, in a self-driving system, multiple tasks must be conducted simultaneously but also compete for a fixed resource (e.g., fixed total inference time threshold), and their preferences could change in real time for different scenarios (Karpathy, 2019). A recommendation system needs to balance multiple criteria among different stakeholders simultaneously, and making trade-off adjustment would be a crucial component (Milojkovic et al., 2019). Consider the huge storage cost, it is far from ideal to train and store multiple models to cover different trade-off preferences, which is also not good for real-time adjustment.

In this paper, we propose a novel controllable Pareto multi-task learning framework, to learn the whole Pareto optimal front for all tasks with a single model. As shown in Fig. 1, MTL practitioners can easily control the trade-off among tasks based on their preferences. To our best knowledge, this is the first approach to learn the whole MTL Pareto front. The main contributions are:

- We formulate solving an MTL problem as a preference-conditioned multiobjective optimization problem, and propose a novel Pareto solution generator to learn the whole Pareto front of optimal trade-off solutions. The proposed Pareto solution generator is also a novel contribution to multiobjective optimization.
- We propose a general hypernetwork-based multi-task neural network framework, and develop an end-to-end optimization algorithm to train a single model concerning different trade-off preferences among all tasks simultaneously.

2 Related Work

Multi-Task Learning. The current works on deep multi-task learning mainly focus on designing novel network architecture and constructing efficient shared representation among tasks (Zhang and Yang, 2017; Ruder, 2017). Different deep MTL networks, with hard or soft parameters sharing structures, have been proposed in the past few years (Misra et al., 2016; Long et al., 2017; Yang and Hospedales, 2017). However, how to properly combine and learn different tasks together remains a basic but challenging problem for MTL applications. Although it has been proposed for more than two decades, the simple linear tasks scalarization approach is still the current default practice to combine and train different tasks in MTL problems (Caruana, 1997).

Some adaptive weight methods have been proposed to better combine all tasks in MTL problems with a single model (Kendall et al., 2018; Chen et al., 2018; Liu et al., 2019; Yu et al., 2020). However, analysis on the relations among tasks in transfer learning (Zamir et al., 2018) and multi-task learning (Standley et al., 2020) show that some tasks might conflict with each other and cannot be optimized at the same time. Sener and Koltun (Sener and Koltun, 2018) propose to treat MTL as a multiobjective optimization problem, and find a single Pareto solution among different tasks. Pareto MTL (Lin et al., 2019) generalizes the multiobjective optimization idea, and proposes to generate a set of Pareto solutions with different optimal trade-off preferences. Recent works focus on generating diverse and dense Pareto solutions (Mahapatra and Rajan, 2020; Ma et al., 2020).

Multiobjective Optimization. Multiobjective optimization itself is a popular research topic in the optimization community. Many gradient-based and gradient-free algorithms have been proposed in the past decades (Fliege and Svaiter, 2000; Miettinen, 2012). In addition to MTL, they also can be used in reinforcement learning (Van Moorsel and Nave, 2014) and neural architecture search (Elsken et al., 2019). However, these methods directly use or modify well-studied multiobjective algorithms to find a single solution or a finite number of Pareto solutions. Our method for learning the whole Pareto set is a novel contribution to multiobjective optimization as well.

HyperNetworks. The hypernetwork approach is initially proposed for dynamic modeling and model compression (Schmidhuber, 1992; Ha et al., 2017). It also leads to various novel applications such as hyperparameter optimization (Brock et al., 2018; MacKay et al., 2019), Bayesian inference (Krueger et al., 2018; Dwaracherla et al., 2020), and transfer learning (von Oswald et al., 2020; Meyerson and Miikkulainen, 2019). Recently, some discussions have been made on its initialization method (Chang et al., 2020), the optimization dynamic (Littwin et al., 2020), and its relation to other multiplicative interaction methods (Jayakumar et al., 2020). This paper uses a hypernetwork to generate the weights of the main multi-task neural network conditioned on different preferences.
3 MTL as Multi-Objective Optimization

An MTL problem involves learning multiple related tasks at the same time. For training a deep multi-task neural network, it is also equal to minimize the losses for multiple tasks:

$$\min_{\theta} \mathcal{L}(\theta) = (\mathcal{L}_1(\theta), \mathcal{L}_2(\theta), \cdots, \mathcal{L}_m(\theta)),$$

where $\theta$ is the neural network parameters and $\mathcal{L}_i(\theta)$ is the loss of the $i$-th task. For learning all $m$ tasks together, an MTL algorithm usually aims to minimize all the losses at the same time. However, in many MTL problems, it is impossible to find a single best solution to optimize all the losses simultaneously. With different trade-offs among the tasks, the problem (1) could have a set of Pareto solutions which satisfy the following definitions (Zitzler and Thiele, 1999):

**Pareto dominance.** Let $\theta_a, \theta_b$ be two solutions for problem (1), $\theta_a$ is said to dominate $\theta_b$ ($\theta_a \prec \theta_b$) if and only if $\mathcal{L}_i(\theta_a) \leq \mathcal{L}_i(\theta_b), \forall i \in \{1, ..., m\}$ and $\mathcal{L}_j(\theta_a) < \mathcal{L}_j(\theta_b), \exists j \in \{1, ..., m\}$.

**Pareto optimality.** $\theta^*$ is a Pareto optimal solution if there does not exist $\hat{\theta}$ such that $\hat{\theta} \prec \theta^*$. The set of all Pareto optimal solutions is called the Pareto set. The image of the Pareto set in the objective space is called the Pareto front.

3.1 Finite Pareto Set Approximation

The number of Pareto optimal solutions could be infinite, and their objective values are on the boundary of the valid value region (Boyd and Vandenberghe, 2004). Under mild conditions, the whole Pareto set and Pareto front are both $(m - 1)$ dimensional manifolds in the solution space and objective space, respectively (Miettinen, 2012). As shown in Fig. 2(b)(c), for an MTL with two tasks, the Pareto set and Pareto front are one dimensional curves. Traditional multiobjective optimization algorithms aim at finding a set of finite solutions to approximate the whole Pareto set:

$$\hat{S} = \{\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_K\}, \quad \hat{F} = \{\mathcal{L}(\hat{\theta}_1), \mathcal{L}(\hat{\theta}_2), \cdots, \mathcal{L}(\hat{\theta}_K)\},$$

where $\hat{S}$ is the approximated Pareto set of $K$ estimated solutions, and $\hat{F}$ is the set of corresponding objective vectors in the objective space. Sener and Koltun (2018) first formulated MTL as solving a multiobjective optimization problem, and proposed a gradient-based algorithm to find a single Pareto solution ($K = 1$). A recently proposed Pareto MTL algorithm (Lin et al., 2019) can generate a set of multiple $K$ widely distributed Pareto solutions on the Pareto front for a given MTL problem.

This finite Pareto set approximation approach has many drawbacks for MTL. First of all, a large number of solutions are needed to cover the whole Pareto set, which would increase exponentially to the number of tasks. For deep MTL, each solution is a deep multi-task neural network. The training and storing cost would be expensive. In addition, this approach does not allow MTL practitioners to make real-time trade-off adjustments, which could be useful for many real-world applications.
4 Preference-Conditioned Pareto Solution Generator

4.1 Pareto Solution Generator

Instead of finite Pareto set estimation, we propose to directly learn the manifold of the whole Pareto set in this paper. It would be a novel and principal way to solve MTL as a multiobjective optimization problem. As shown in Fig. 1, we want to build a Pareto solution generator to map a preference vector \( p \) to the corresponding Pareto solution \( \theta_p \). If an optimal generator \( \theta_p = g(p|\phi^*) \) is obtained, MTL practitioners can assign their preference via the preference vector \( p \), and directly obtain the corresponding Pareto solution \( \theta_p \) with the specific trade-off among tasks.

With the Pareto solution generator, we can obtain the approximate Pareto set/front as:

\[
\hat{S} = \{ \theta_p \} = \{ g(p|\phi^*) | p \in P \}, \quad \hat{F} = \{ L(\theta_p) | \theta_p \in \hat{S} \},
\]

where \( P \) is the set of all valid preference vectors, \( g(p|\phi^*) \) is the Pareto generator with optimal parameters \( \phi^* \). Once we have a proper generator \( g(p|\phi^*) \), we can reconstruct the whole approximate Pareto set \( \hat{S} \) and the corresponding Pareto front \( \hat{F} \) by going through all possible preference vector \( p \) from \( P \). To solve an MTL problem in this way, we need to define:

- **Preference and Corresponding Solution**: Define the form of preference vector \( p \in P \), the corresponding Pareto solution \( \theta_p \), and how they are connected by the Pareto generator \( g(p|\phi^*) \). Two different approaches are proposed and discussed in the rest of this section.
- **Model and Training Procedure**: Build a deep multi-task network with a hypernetwork-based Pareto generator \( g(p|\phi) \), along with an end-to-end training procedure to train the model and obtain the optimal \( \phi^* \). We will propose the algorithm framework in section 5.

4.2 Preference-Conditioned Linear Scalarization

A simple and straightforward approach is to define the preference vector \( p \) and the corresponding solution \( \theta_p \) via the weighted linear scalarization:

\[
\theta_p = g(p|\phi^*) = \arg \min_{\theta} \sum_{i=1}^{m} p_i L_i(\theta),
\]

where the preference vector \( p = (p_1, p_2, \cdots, p_m) \) is the weight for each task, and the corresponding solution \( \theta_p \) is the minimum of the weighted linear scalarization. If we further require \( \sum p_i = 1 \), the set of all valid preference vector \( P \) is an \((m-1)\)-dimensional manifold in \( \mathbb{R}^m \). Therefore, the valid preference set and the Pareto set are both \((m-1)\)-dimensional manifolds. It should be noticed that the solution for the right-hand side of equation (4) might be not unique. For simplicity, we assume there is always a one-to-one mapping in this paper.

Although this approach is straightforward, it is not optimal for multiobjective optimization. Linear scalarization can not find any Pareto solution on the non-convex part of the Pareto front (Das and Dennis 1997; Boyd and Vandenberghe 2004). In other words, unless the problem has a convex Pareto front, the generator defined by linear scalarization cannot cover the whole Pareto set manifold.

4.3 Preference-Conditioned Multi-Objective Optimization

To better approximate the manifold of Pareto set, we generalize the idea of decomposition-based multiobjective optimization (Zhang and Li 2007; Liu et al. 2014) and Pareto MTL (Lin et al. 2019) to connect the preference vector and the corresponding Pareto solution. To be specific, we define the preference \( p \) as an \( m \) dimensional unit vector in the loss space, and the corresponding solution \( \theta_p \) is the one on the Pareto front which has the smallest angle with \( p \).

The idea is illustrated in Fig. 3. With a set of randomly generated unit reference vectors \( U = \{ u^{(1)}, \cdots, u^{(K)} \} \) and the preference vector \( p \), an MTL problem is decomposed into different regions in the loss space. We call the region closest to \( p \) as its preferred region. The corresponding Pareto solution \( \theta_p \) is the solution that always belongs to the preferred region and on the Pareto front. Formally, we can define the corresponding Pareto solution as:

\[
\theta_p = g(p|\phi^*) = \arg \min_{\theta} L(\theta) \quad \text{s.t.} \quad L(\theta) \in \Omega(p, U),
\]
The Controllable Pareto Multi-Task Network builds a hypernetwork to directly generate parameters for the main MTL network from the preference vector. In model training, the loss function is also characterized by the preference vector to define the corresponding Pareto solution:

\[ L(\theta) = (L_1(\theta), L_2(\theta), \cdots, L_m(\theta)), \]

\[ \Omega(p, U) = \{ v \in \mathbb{R}_m^m | \angle(v, p) \leq \angle(v, u^{(j)}), u^{(j)} \in U, \forall j = 1, \ldots, K \}. \]

The constraint is satisfied if the loss vector \( L(\theta^*) \) has the smallest angle with the preference vector \( p \).

For a given preference region \( \Omega(p, U) \), we can define the restricted Pareto solution:

\[ \theta^* \text{ is a Pareto optimal solution restricted on the region } \Omega(p, U) \text{ if } L(\theta^*) \in \Omega(p, U) \text{ and there does not exist } \hat{\theta} \prec \theta^* \text{ and } L(\hat{\theta}) \in \Omega(p, U). \]

The corresponding solution \( \theta_p \) is restricted Pareto optimal in \( \Omega(p, U) \) for any reference set \( U \). Since we require the preference vectors should be unit vectors \( ||p||^2 = 1 \) in the \( m \)-dimensional space, the set of all valid preference vectors \( P \) is an \((m-1)\) manifold in \( \mathbb{R}^m \). Similar to the linear scalarization case, a preference vector might have more than one corresponding solution, and a Pareto solution can correspond to more than one preference vector. For simplicity, we assume there is a one-to-one mapping between the preference vector and the corresponding Pareto solution in this paper.

## 5 Controllable Pareto Multi-Task Learning

In this section, we propose a hypernetwork-based deep multi-task neural network framework, along with an efficient end-to-end optimization procedure to solve the MTL problem.

### 5.1 Hypernetwork-based Deep Multi-Task Network

For a deep multi-task neural network, a solution is the set of all parameters for the network, and the objective vector is the loss functions for different tasks. We propose to use a hypernetwork to directly generate the parameters for the main multi-task network based on the preference vector.

The proposed controllable Pareto multi-task network is shown in Fig. 4. As discussed in the previous section, we use a preference vector to represent a practitioner’s trade-off preference among different tasks. The preference vector \( p \) is the input of the hypernetwork and also defines the corresponding loss function. The hypernetwork generates \( \theta_p = g(p|\phi) \) as the set of parameters for the main MTL network. The losses vector for the whole network is \( L(\theta_p) = L(g(p|\phi)) \) and the only trainable parameters to be optimized is the hypernetwork parameters \( \phi \).

Therefore, with the hypernetwork-based structure, we can simultaneously solve the MTL problem while learning the Pareto solution generator. Once we have optimal \( \phi^* \), the hypernetwork \( g(p|\phi^*) \) would be the Pareto solution generator. Practitioners can easily control the MTL network’s parameters, and hence the performances on different tasks in real time, by simply adjusting the preference vector \( p \). More discussions on the hypernetwork structure can be found in the Appendix Section.
5.2 End-to-End Optimization: Learning Pareto Generator via Solving MTL

Since we want to control the trade-off preference at the inference time, we need to train a model suitable for all preference vector \( p \) in the valid set \( P \) but not just a single point. Suppose we have a probability distribution on all possible preference vector \( p \sim P_p \), a general goal would be:

\[
\min_{\phi} \mathbb{E}_{p \sim P_p} \mathcal{L}(g(p|\phi)).
\]

(7)

It is hard to optimize the parameters within the expectation directly. We use Monte Carlo method to sample the preference vector, and use the stochastic gradient descent algorithm to train the deep neural network. At each step, we can sample one preference vector \( p \) and optimize the loss function:

\[
\min_{\phi} \mathcal{L}(g(p|\phi)), \quad p \sim P_p.
\]

(8)

At each iteration \( t \), if we have a valid gradient direction \( d_t \) for the loss \( \mathcal{L}(g(p|\phi_t)) \), we can simply update the parameters with gradient descent \( \phi_{t+1} = \phi_t - \eta d_t \). For the preference-conditioned linear scalarization case as in problem (4), the calculation of \( d_t \) is straightforward:

\[
d_t = \sum_{i=1}^{m} p_i \nabla_{\phi_i} \mathcal{L}_i(g(p|\phi_t)), \quad p \sim P_p.
\]

(9)

For the preference-conditioned multiobjective optimization problem (5), to deal with the angle constraints, we can rewrite it as the following equivalent problem:

\[
\theta_p = g(p|\phi) = \arg \min_{\theta} \mathcal{L}(\theta)
\]

\[
\text{s.t. } \mathcal{G}_j(\theta|p, U) = (u^{(j)} - p)^T \mathcal{L}(\theta) \leq 0, \forall j = 1, \ldots, K.
\]

(10)

Since the preference vector \( p \) and all reference vectors \( U = \{u^{(1)}, u^{(2)}, \ldots, u^{(K)}\} \) are unit vectors, a smaller angle with the vectors is equal to a larger inner product. The smallest angle between \( \mathcal{L}(\theta) \) and the preference vector \( p \) is equal to the largest inner product \( p^T \mathcal{L}(\theta) \). A valid descent direction for problem (10) should reduce all losses and activated constraints. One valid descent direction can be written as a linear combination of all tasks with dynamic weight \( \alpha_i(t) \):

\[
d_t = \sum_{i=1}^{m} \alpha_i(t) \nabla_{\phi_i} \mathcal{L}_i(g(p|\phi_t)), \quad p \sim P_p, \quad U \sim P_U,
\]

(11)

where the coefficients \( \alpha_i(t) \) is depended on both the loss functions \( \mathcal{L}_i(g(p|\phi_t)) \) and the constraints \( \mathcal{G}_j(\theta|p, U) \). We give the detailed derivation in the Appendix Section A.2 due to page limit.

5.3 Algorithm Framework

![Algorithm 1 Controllable Pareto Multi-Task Learning](image)

For the preference-conditioned multiobjective optimization problem (5), to deal with the angle constraints, we can rewrite it as the following equivalent problem:

\[
\theta_p = g(p|\phi) = \arg \min_{\theta} \mathcal{L}(\theta)
\]

\[
\text{s.t. } \mathcal{G}_j(\theta|p, U) = (u^{(j)} - p)^T \mathcal{L}(\theta) \leq 0, \forall j = 1, \ldots, K.
\]

(10)

Since the preference vector \( p \) and all reference vectors \( U = \{u^{(1)}, u^{(2)}, \ldots, u^{(K)}\} \) are unit vectors, a smaller angle with the vectors is equal to a larger inner product. The smallest angle between \( \mathcal{L}(\theta) \) and the preference vector \( p \) is equal to the largest inner product \( p^T \mathcal{L}(\theta) \). A valid descent direction for problem (10) should reduce all losses and activated constraints. One valid descent direction can be written as a linear combination of all tasks with dynamic weight \( \alpha_i(t) \):

\[
d_t = \sum_{i=1}^{m} \alpha_i(t) \nabla_{\phi_i} \mathcal{L}_i(g(p|\phi_t)), \quad p \sim P_p, \quad U \sim P_U,
\]

(11)

where the coefficients \( \alpha_i(t) \) is depended on both the loss functions \( \mathcal{L}_i(g(p|\phi_t)) \) and the constraints \( \mathcal{G}_j(\theta|p, U) \). We give the detailed derivation in the Appendix Section A.2 due to page limit.

5.3 Algorithm Framework

![Algorithm 1 Controllable Pareto Multi-Task Learning](image)

For the preference-conditioned multiobjective optimization problem (5), to deal with the angle constraints, we can rewrite it as the following equivalent problem:

\[
\theta_p = g(p|\phi) = \arg \min_{\theta} \mathcal{L}(\theta)
\]

\[
\text{s.t. } \mathcal{G}_j(\theta|p, U) = (u^{(j)} - p)^T \mathcal{L}(\theta) \leq 0, \forall j = 1, \ldots, K.
\]

(10)

Since the preference vector \( p \) and all reference vectors \( U = \{u^{(1)}, u^{(2)}, \ldots, u^{(K)}\} \) are unit vectors, a smaller angle with the vectors is equal to a larger inner product. The smallest angle between \( \mathcal{L}(\theta) \) and the preference vector \( p \) is equal to the largest inner product \( p^T \mathcal{L}(\theta) \). A valid descent direction for problem (10) should reduce all losses and activated constraints. One valid descent direction can be written as a linear combination of all tasks with dynamic weight \( \alpha_i(t) \):

\[
d_t = \sum_{i=1}^{m} \alpha_i(t) \nabla_{\phi_i} \mathcal{L}_i(g(p|\phi_t)), \quad p \sim P_p, \quad U \sim P_U,
\]

(11)

where the coefficients \( \alpha_i(t) \) is depended on both the loss functions \( \mathcal{L}_i(g(p|\phi_t)) \) and the constraints \( \mathcal{G}_j(\theta|p, U) \). We give the detailed derivation in the Appendix Section A.2 due to page limit.
6 Synthetic Multi-Objective Optimization Problem

In the previous section, we propose to use a hypernetwork to generate the parameters for the main deep multi-task network based on a preference vector. The network parameters are in a high-dimensional decision space, and the optimization landscape would be complicated. To better analyze the behavior and convergence performance of the proposed algorithm, we first use it to generate the Pareto solutions for a low-dimensional multi-objective optimization problem.

The experimental results are shown in Fig. 7. This synthetic problem has a sin-curve-like Pareto set in the solution space, and its corresponding Pareto front is a concave curve in the objective space. The detailed definition is given in the appendix. Our proposed controllable Pareto MTL method can cover and reconstruct the whole manifold of Pareto front from the preference vectors, while the traditional methods can only find a finite point set approximation. It should be noticed that the simple linear scalarization method has poor performance for the problem with concave Pareto front.

7 Experiments

Figure 7: Results on MultiMNIST, CityScapes and NYUv2: Our proposed algorithm can generate the Pareto front for each problem with a single model respectively. For CityScapes and NYUv2, we report pixel accuracy for segmentation and (1 - relative error) for depth estimation.
In this section, we validate the performance of the proposed controllable Pareto MTL to generate a set of Pareto solutions for different MTL problems. We compare it with the following MTL algorithms: 1) Linear Scalarization: simple linear combination of different tasks with fixed weights; 2) Uncertainty (Kendall et al., 2018): adaptive weight assignments with balanced uncertainty; 3) DWA (Liu et al., 2019): dynamic weight average for the losses; 4) MGDA (Sener and Koltun, 2018): finds one Pareto solution; 5) Pareto MTL (Lin et al., 2019): generates a set of wildly distributed Pareto solutions; and 6) Single Task: the single task baseline.

MultiMNIST: We first conduct experiments on the MultiMNIST problem (Sabour et al., 2017). In the dataset, each image has two digits, where are on the top left and bottom right respectively. The MTL problem is to classify these two digits simultaneously. We build a LeNet-based MTL network as used in the current work. For the hypernetwork-based model, we use MLP to generate all parameters for the main network.

CityScapes: We compare the controllable Pareto MTL with other algorithms on the CityScapes problem (Cordts et al., 2016). This dataset has street-view RGB images, and involves two tasks to be solved, which are pixel-wise semantic segmentation and depth estimation. We follow the experimental setting as in Liu et al. (2019). The images are resized to $[128 \times 256]$, and the 7-class coarser labels are used for segmentation. We use SegNet as the shared representation encoder, and both tasks have a small size task-specific output layers.

NYUv2: We also test our proposed algorithm on the NYUv2 problem (Silberman et al., 2012). This dataset is for indoor scene understanding with two tasks: a 13-class semantic segmentation and indoor depth estimation. Similar to Liu et al. (2019), we resize all images into $[288 \times 384]$ for speed up, and use a SegNet-based network as the MTL model.

The experimental results in Fig. 7 show that our proposed algorithm can successfully generate approximated Pareto fronts for each problem. With a single hypernetwork-based MTL model, practitioners can easily control the trade-off preference among tasks, and obtain the corresponding Pareto solution in real time. Other algorithms need to train a new model for each solution, and do not support trade-off adjustment at the inference time.

For all experiments, the single task model is a strong baseline in performance. However, it cannot dominate most of the approximated Pareto front learned by our model. Our models provide diverse optimal trade-offs among tasks (e.g., in the upper left and lower right area) for different problems. In addition, the inference times would be a concern for the single task baseline ($m$ models).

CIFAR-100 with 20 Tasks: Follow a similar setting in (Rosenbaum et al., 2018; von Oswald et al., 2020), we split the original CIFAR-100 dataset (Krizhevsky and Hinton, 2009) into 20 five-class classification tasks. The experimental results are shown in Fig. 8. The controllable Pareto MTL can achieve the best overall performance among different tasks by making a real-time trade-off adjustment. It also outperforms the balanced hypernet approach, which uses the same hypernetwork-based model to simultaneously optimize all tasks. These results confirm the usefulness of the multiobjective optimization reformulation. More details can be found in the Appendix Section D.
8 Conclusion

In this paper, we proposed a novel controllable Pareto multi-task learning framework for solving MTL problems. With a hypernetwork-based Pareto solution generator, our method can directly learn the whole Pareto front for all tasks with a single model. It allows practitioners to easily make real-time trade-off adjustment among tasks at the inference time. Experimental results on various MTL applications demonstrated the usefulness and efficiency of the proposed method.

References

V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12):2481–2495, 2017.

L. Bertinetto, J. F. Henriques, J. Valmadre, P. H. S. Torr, and A. Vedaldi. Learning feed-forward one-shot learners. In NIPS’16 Proceedings of the 30th International Conference on Neural Information Processing Systems, pages 523–531, 2016.

S. Boyd and L. Vandenberghe. Convex optimization. Cambridge university press, 2004.

A. Brock, T. Lim, J. Ritchie, and N. Weston. Smash: One-shot model architecture search through hypernetworks. In ICLR 2018 : International Conference on Learning Representations 2018, 2018.

R. Caruana. Multitask learning. Machine learning, 28(1):41–75, 1997.

O. Chang, L. Flokas, and H. Lipson. Principled weight initialization for hypernetworks. In ICLR 2020 : Eighth International Conference on Learning Representations, 2020.

Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In Proceedings of the 35th International Conference on Machine Learning, pages 794–803, 2018.

M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3213–3223, 2016.

I. Das and J. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. Structural Optimization, 14(1):63–69, 1997.

K. Deb. Multi-objective optimization using evolutionary algorithms, volume 16. John Wiley & Sons, 2001.

J.-A. Désidéri. Mutiple-gradient descent algorithm for multiobjective optimization. In European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2012), 2012.

V. Dwaracherla, X. Lu, M. Ibrahimi, I. Osband, Z. Wen, and B. V. Roy. Hypermodels for exploration. In ICLR 2020 : Eighth International Conference on Learning Representations, 2020.

T. Elsken, J. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In International Conference on Learning Representations, 2019.

J. Fliege and B. F. Svaiter. Steepest descent methods for multicriteria optimization. Mathematical Methods of Operations Research, 51(3):479–494, 2000.

J. Fliege, A. I. F. Vaz, and L. N. Vicente. Complexity of gradient descent for multiobjective optimization. Optimization Methods and Software, 34(5):949–959, 2019.

B. Gebken, S. Peitz, and M. Dellnitz. A descent method for equality and inequality constrained multiobjective optimization problems. In Numerical and Evolutionary Optimization, pages 29–61. Springer, 2017.
D. Golovin et al. Random hypervolume scalarizations for provable multi-objective black box optimization. *arXiv preprint arXiv:2006.04655*, 2020.

D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. In *ICLR 2017: International Conference on Learning Representations*, 2017.

M. Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th international conference on machine learning*, pages 427–435, 2013.

S. M. Jayakumar, J. Menick, W. M. Czarnecki, J. Schwarz, J. Rae, S. Osindero, Y. W. Teh, T. Harley, and R. Pascanu. Multiplicative interactions and where to find them. In *ICLR 2020: Eighth International Conference on Learning Representations*, 2020.

A. Karpathy. Multi-task learning in the wilderness. *AMTL Workshop, ICML2019*, 2019.

A. Kendall, Y. Gal, and R. Cipolla. 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 (CVPR)*, 2018.

I. Kokkinos. 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*, pages 6129–6138, 2017.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical report*, https://www.cs.toronto.edu/~kriz/cifar.html, 2009.

D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian hypernetworks. *arXiv: Machine Learning*, 2018.

X. Lin, H. Zhen, Z. Li, Q. Zhang, and S. Kwong. Pareto multi-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. accepted, 2019.

E. Littwin, T. Galanti, and L. Wolf. On the optimization dynamics of wide hypernetworks. *arXiv preprint arXiv:2003.12193*, 2020.

H.-L. Liu, F. Gu, and Q. Zhang. Decomposition of a multiobjective optimization problem into a number of simple multiobjective subproblems. *IEEE Trans. Evolutionary Computation*, 18(3): 450–455, 2014.

S. Liu, E. Johns, and A. J. Davison. End-to-end multi-task learning with attention. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1871–1880, 2019.

M. Long, Z. Cao, J. Wang, and P. S. Yu. Learning multiple tasks with multilinear relationship networks. In *Advances in Neural Information Processing Systems*, pages 1594–1603, 2017.

P. Ma, T. Du, and W. Matusik. Efficient continuous pareto exploration in multi-task learning. *Thirty-seventh International Conference on Machine Learning (ICML 2020)*, 2020.

M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *ICLR 2019: 7th International Conference on Learning Representations*, 2019.

D. Mahapatra and V. Rajan. Multi-task learning with user preferences: Gradient descent with controlled ascent in pareto optimization. *Thirty-seventh International Conference on Machine Learning (ICML 2020)*, 2020.

E. Meyerson and R. Miikkulainen. Modular universal reparameterization: Deep multi-task learning across diverse domains. In *Advances in Neural Information Processing Systems*, pages 7903–7914, 2019.

K. Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

N. Milojkovic, D. Antognini, G. Bergamin, B. Faltings, and C. Musat. Multi-gradient descent for multi-objective recommender systems. *arXiv preprint arXiv:2001.00846*, 2019.
I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3994–4003, 2016.

C. Rosenbaum, T. Klinger, and M. Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In ICLR 2018: International Conference on Learning Representations 2018, 2018.

S. Ruder. An overview of multi-task learning in deep neural networks. arXiv preprint arXiv:1706.05098, 2017.

S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In Advances in Neural Information Processing Systems, pages 3856–3866, 2017.

J. Schmidhuber. Learning to control fast-weight memories: an alternative to dynamic recurrent networks. Neural Computation, 4(1):131–139, 1992.

O. Sener and V. Koltun. Multi-task learning as multi-objective optimization. In Advances in Neural Information Processing Systems, pages 525–536, 2018.

N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In European conference on computer vision, pages 746–760. Springer, 2012.

T. Standley, A. R. Zamir, D. Chen, L. J. Guibas, J. Malik, and S. Savarese. Which tasks should be learned together in multi-task learning? International Conference on Machine Learning, 2020.

K. O. Stanley, D. B. D’Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. Artificial Life, 15(2):185–212, 2009.

S. Subramanian, A. Trischler, Y. Bengio, and C. J. Pal. Learning general purpose distributed sentence representations via large scale multi-task learning. In International Conference on Learning Representations, 2018.

K. Van Moffaert and A. Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. The Journal of Machine Learning Research, 15(1):3483–3512, 2014.

J. von Oswald, C. Henning, J. Sacramento, and B. F. Grewe. Continual learning with hypernetworks. In ICLR 2020: Eighth International Conference on Learning Representations, 2020.

Y. Yang and T. M. Hospedales. Deep multi-task representation learning: A tensor factorisation approach. In ICLR 2017: International Conference on Learning Representations 2017, 2017.

T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient surgery for multi-task learning. arXiv preprint arXiv:2001.06782, 2020.

A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3712–3722, 2018. URL https://academic.microsoft.com/paper/2964185501

Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on evolutionary computation, 11(6):712–731, 2007.

Y. Zhang and Q. Yang. A survey on multi-task learning. arXiv preprint arXiv:1707.08114, 2017.

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

As mentioned in the main paper, we provide more discussion and analysis in this appendix, which can be summarized as follows:

- **Pareto Generator and Multiobjective Gradient Descent**: We discuss the idea of Pareto solution generator from the view of multiobjective optimization in Section A. We also give the details of preference-conditioned multiobjective gradient descent for generator training.

- **Convergence-Diversity Analysis for Multiobjective Optimization**: In Section B, we provide a brief analysis on the convergence-diversity trade-off for multiobjective optimization. We discuss the convergence from multiple points to the manifold learning.

- **Scalability for Hypernetwork-based Pareto Generator**: We discuss the scalability for our proposed hypernetwork-based Pareto generator, and introduce two easy-to-use approaches to improve scalability in Section C.

- **Experimental Setting**: We provide detailed experimental setting in Section 7.

A Pareto Generator and Multiobjective Gradient Descent

This work follows a recent line of research on introducing multiobjective optimization into multi-task learning (Sener and Koltun, 2018; Lin et al., 2019). The previous works simply bridge the two fields, and use or combine existing multiobjective optimization methods to solve MTL problems. In contrast, we find that the current multiobjective optimization methods have shortcomings and are insufficient for many MTL applications. This work proposes a novel Pareto solution generator for multiobjective optimization, which is useful for solving MTL problems. The proposed method to learn the whole Pareto set manifold is a new contribution and could be useful for both communities.

In this section, we discuss the idea of Pareto solution generator from the view of multiobjective optimization, and show how this algorithm can be used to solve MTL problems.

A.1 Pareto Solution Generator

![Figure 9](image)

\[ S = \{ \theta_1, \theta_2, \ldots, \theta_K \}, \quad F = \{ \mathcal{L}(\theta_1), \mathcal{L}(\theta_2), \ldots, \mathcal{L}(\theta_K) \} \]

\[ \hat{S} = \{ \theta_p = g(p, \phi^*) | p \in P \}, \quad \hat{F} = \{ \mathcal{L}(\theta_p) | \theta_p \in \hat{S} \}. \]

The goal of most traditional multiobjective optimization algorithms is to find a set of finite solutions to approximate the Pareto set. However, under mild condition, since the Pareto set is a manifold, no set of finite solutions would be a perfect estimator. This approximation also restricts the information provided to the practitioners to make a decision. If no solution in the approximation set satisfies the specific trade-off preference, the practitioners need to rerun the algorithm to obtain other solutions.

As shown in Fig. 9, our proposed Pareto solution generator directly maps a specific preference vector to the corresponding Pareto solution. In the ideal case, the generator can cover the whole manifold of the Pareto set with a single model in real time.

In the viewpoint of multiobjective optimization, we provide a controllable Pareto solution generator, instead of a set of finite solutions, to approximate the manifold of Pareto set:

\[ \hat{S} = \{ \theta_p = g(p, \phi^*) | p \in P \}, \quad \hat{F} = \{ \mathcal{L}(\theta_p) | \theta_p \in \hat{S} \}. \]
Figure 10: **The training process of the Pareto solution generator.** At each iteration, the proposed algorithm randomly samples a preference vector and a set of reference vectors to decompose the loss space, and calculates a valid gradient direction to update the Pareto solution generator. The algorithm iteratively learns and improves the generated manifold during the optimization process.

$(m - 1)$ dimensional Pareto set manifold by exploring the suitable preference set. The Pareto solution generator also provides more information to the practitioners, and allow them to make flexible real-time control over the Pareto solutions. The practitioners can adjust the preference vector to obtain a suitable Pareto solution, or adaptively set the trade-off preference in real time to make a dynamic adjustment, which could be crucial for many real-world applications.

A.2 Preference-Conditioned Multiobjective Gradient Descent

The Pareto solution generator generates a solution $\theta_p = g(p|\phi)$ from the preference vector $p$. Under ideal condition, if we take all valid preference vectors $p \in P$ (which is an $(m - 1)$-dimensional manifold) as input, the output would be another $(m - 1)$-dimensional manifold. At the end of the algorithm, we hope the generated manifold is close to the manifold of true Pareto set.

In the proposed algorithm, we randomly sample a preference vector $p$ at each iteration to update the generator parameters $\phi$. As illustrated in Fig. 10, it continually learns and improves the current manifold around the preference vector at each iteration. It should be noticed that the sampled preference vector and all reference vectors are different at each iteration, so as the decomposition.

As mentioned in the Algorithm 1 in the main paper, we use simple gradient descent to update the Pareto solution generator at each iteration. For the case of linear scalarization, calculating the gradient direction is straightforward. In this subsection, we discuss how to obtain a valid gradient direction for the preference-conditioned multiobjective optimization. Similar to the previous work (Sener and Koltun 2018; Lin et al. 2019), we use multiobjective gradient descent (Fliege and Svaiter 2000; Désidéri 2012) to solve the MTL problem. However, in our algorithm, the optimization parameter is $\phi$ for the Pareto solution generator rather than $\theta$ for a single solution.

A simple illustration of the multiobjective gradient descent idea for a problem with two tasks is shown in Fig. 11. At each iteration, the algorithm samples a preference vector $p$ and obtains its current corresponding solution $\theta_p$. A valid gradient direction should reduce all the losses and guide the generated solution $\theta_p = g(p|\phi)$ toward the preference region $\Omega(p, U)$ around the preference vector $p$.

Figure 11: For a preference vector, the valid gradient direction should reduce all losses and activated constraints for the generated solution.
At iteration \( t \), for a given preference vector \( p \), the preference-conditioned multiobjective problem with the Pareto solution generator can be written as:

\[
\min_{\phi_t} (L_1(g(p|\phi_t)), L_2(g(p|\phi_t)), \ldots, L_m(g(p|\phi_t))),
\]

\[
\text{s.t. } G_j(g(p|\phi_t)|p, U) = (u^{(j)} - p)^T L(g(p|\phi_t)) \leq 0, \forall j = 1, \ldots, K.
\]

Since the parameter \( \theta_p \) is generated by the Pareto solution generator \( g(p|\phi_t) \), the generator’s parameter \( \phi_t \) is the only trainable parameters to be optimized. What we need is to find a valid gradient direction \( d_t \) to reduce all the losses \( L_i(g(p|\phi_t)) \) and activated constraints \( G_j(g(p|\phi_t)|p, U) \) with respect to \( \phi_t \).

We follow the methods proposed in [Fliege and Svaiter 2000; Gebken et al. 2017], and calculate a valid descent direction \( d_t \) by solving the optimization problem:

\[
(d_t, \alpha_t) = \arg\min_{d \in \mathbb{R}^n, \alpha \in \mathbb{R}} \alpha + \frac{1}{2}||d||^2
\]

\[
\text{s.t. } \nabla_{\phi_t} L_i(g(p|\phi_t))^T d \leq \alpha, i = 1, \ldots, m
\]

\[
\nabla_{\phi_t} G_j(g(p|\phi_t))^T d \leq \alpha, j \in I(\phi_t).
\]

where \( d_t \) is the obtained gradient direction, \( \alpha \) is an auxiliary parameter for optimization, and \( I(\theta) = \{ j \in I \mid G_j(g(p|\phi_t)) \geq 0 \} \) is the index set of all activated constraints. By solving the above problem, the obtained direction \( d_t \) and parameter \( \alpha_t \) will satisfy the following lemma [Gebken et al. 2017]:

Lemma 1: Let \((d_t, \alpha_t)\) be the solution of problem (15), we either have:

1. A non-zero \( d_t \) and \( \alpha_t \) with

\[
\alpha_t \leq -(1/2)||d_t||^2 < 0,
\]

\[
\nabla_{\phi_t} L_i(g(p|\phi_t))^T d_t \leq \alpha_t, i = 1, \ldots, m
\]

\[
\nabla_{\phi_t} G_j(g(p|\phi_t))^T d_t \leq \alpha_t, j \in I(\phi_t).
\]

2. or \( d_t = 0 \in \mathbb{R}^n, \alpha_t = 0 \), and \( \theta_p = g(p|\phi_t) \) is local Pareto critical restricted on \( \Omega(p, U) \).

In case 1, we obtain a valid descent direction \( d_t \neq 0 \) which has negative inner products with all \( \nabla_{\phi_t} L_i(g(p|\phi_t)) \) and \( \nabla_{\phi_t} G_j(g(p|\phi_t)) \) for \( j \in I(\phi_t) \). With a suitable step size \( \eta \), we can update \( \phi_{t+1} = \phi_t + \eta d_t \) to reduce all losses and activated constraints. In case 2, we cannot find any nonzero valid descent direction, and obtain \( d_t = 0 \). In other words, there is no valid descent direction to simultaneously reduce all losses and activated constraints. Improving the performance for one task would deteriorate the other(s) or violate some constraints. Therefore, the current solution \( \theta_p = g(p|\phi_t) \) is a local restricted Pareto optimal solution on \( \Omega(p, U) \).

A.3 Adaptive Linear Scalarization

As mentioned in the main paper, we can rewrite the gradient direction \( d_t \) as a dynamic linear combination with the gradient of all losses \( \nabla_{\phi_t} L_i(g(p|\phi_t)) \). Similar to the approach in [Fliege and Svaiter 2000; Lin et al. 2019], we reformulate the problem (15) in its dual form:

\[
\max_{\lambda_i, \beta_j} -\frac{1}{2} \sum_{i=1}^{m} \lambda_i \nabla_{\phi_t} L_i(g(p|\phi_t)) + \sum_{j \in I(\phi_t)} \beta_j \nabla_{\phi_t} G_j(g(p|\phi_t)) ||^2
\]

\[
\text{s.t. } \sum_{i=1}^{m} \lambda_i + \sum_{j \in I(\phi_t)} \beta_j = 1, \quad \lambda_i \geq 0, \beta_j \geq 0, \forall i = 1, \ldots, m, \forall j \in I(\phi_t),
\]

where \( d_t = \sum_{i=1}^{m} \lambda_i \nabla_{\phi_t} L_i(g(p|\phi_t)) + \sum_{j \in I(\phi_t)} \beta_j \nabla_{\phi_t} G_j(g(p|\phi_t)) \) is the obtained valid gradient direction, \( \lambda_i \) and \( \beta_i \) are the Lagrange multipliers for the linear inequality constraints in problem (15).

Based on the definition in problem (14), a constraint \( G_j(g(p|\phi_t)) \) can be rewritten as a linear combination of all losses:

\[
G_j(g(p|\phi_t)) = (u^{(j)} - p)^T L(g(p|\phi_t)) = \sum_{i=1}^{m} (u_i^{(j)} - p_i) L_i(g(p|\phi_t)).
\]
We use the Frank-Wolfe algorithm (Jaggi, 2013) to solve the problem (17) as in the previous work (Sener and Koltun, 2018; Lin et al., 2019). We use simple uniform distribution to sample both preference vectors to update the Pareto solution generator, as shown in Fig. 12.

The training process of the Pareto solution generator with multiple preferences. At each iteration, the proposed algorithm randomly samples multiple preference vectors, and calculates a valid gradient direction which can improve the performance for all subproblems.

The gradient of the constraint is also a linear combination of those for the losses:

$$\nabla_{\phi_t} g_j (g(p|\phi_t)) = (u^{(j)} - p)^T \nabla_{\phi_t} L(g(p|\phi_t)) = \sum_{i=1}^{m} (u^{(j)}_i - p_i) \nabla_{\phi_t} L_i(g(p|\phi_t)).$$  \hfill (19)

Therefore, we can rewrite the valid descent direction as a linear combination of the gradients for all subproblems with respect to the unit preference vector $p$:

$$d_t = \sum_{i=1}^{m} \alpha_i(t) \nabla g_i (g(p|\phi_t)), \quad \alpha_i(t) = \lambda_i + \sum_{j \in I_i(t)} \beta_j (u^{(j)}_i - p_i),$$  \hfill (20)

where $\lambda_i$ and $\beta_j$ are obtained by solving the dual problem (17). It should be noticed that the primal problem (15) directly optimizes the gradient direction $d_t$, which could be in millions of dimensions for training a deep neural network. In contrast, the dimension of dual problem (17) is the number of all tasks and activated constraints (up to a few dozens), which is much smaller than the primal problem. Therefore, in practice, we obtain the valid gradient direction $d_t$ by solving the dual problem (17) instead of the primal problem (15).

We use the Frank-Wolfe algorithm (Jaggi, 2013) to solve the problem (17) as in the previous work (Sener and Koltun, 2018; Lin et al., 2019). We use simple uniform distribution to sample both the unit preference vector $p$ and unit reference vectors $U$ in this paper. The number of preference vector is 1 in the previous discussion, and the number of reference vectors is a hyperparameter, which we set it as 3 in this paper. In the next subsection, we will discuss how to use more than one preference vector at each iteration to update the Pareto solution generator.

### A.4 Batched Preferences Update

To obtain an optimal Pareto generator $g(p|\phi^*)$, we need to find:

$$\phi^* = \arg \min_{\phi} \mathbb{E}_{p \sim P_p} \mathcal{L}(g(p|\phi)).$$  \hfill (21)

In the main paper, we sample one preference vector at each iteration to calculate a valid direction $d_t$ to update the Pareto generator. A simple and straightforward extension is to sample multiple preference vectors to update the Pareto solution generator, as shown in Fig. 12.

At each iteration, we simultaneously sample and optimize multiple subproblems with respect to the Pareto generator:

$$\min_{\phi} \{ \mathcal{L}(g(p_1|\phi)), \mathcal{L}(g(p_2|\phi)), \ldots, \mathcal{L}(g(p_K|\phi)) \}, \quad p_1, p_2, \ldots, p_K \sim P_p$$  \hfill (22)

where $p_1, p_2, \ldots, p_K$ are $K$ randomly sampled preference vectors and each $\mathcal{L}(g(p_k|\phi))$ is a multi-objective optimization problem. Therefore, we now have a hierarchical multiobjective optimization problem. If we do not have a specific preference among the sampled preference vectors, the above problem can be expanded as a problem with $Km$ objectives:

$$\min_{\phi} \{ \mathcal{L}_1(g(p_1|\phi)), \ldots, \mathcal{L}_m(g(p_1|\phi)), \ldots, \mathcal{L}_1(g(p_K|\phi)), \ldots, \mathcal{L}_m(g(p_K|\phi)) \}.$$  \hfill (23)
For the linear scalarization case, the calculation of valid gradient direction at each iteration $t$ is straightforward:

$$
\mathbf{d}_t = \sum_{k=1}^{K} \nabla_{\phi_t} \mathcal{L} (g(\mathbf{p}_k | \phi_t)) = \sum_{k=1}^{K} \sum_{i=1}^{m} \mathbf{p}_i \nabla_{\phi_t} \mathcal{L}_i (g(\mathbf{p}_j | \phi_t)), \quad \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_K \sim \mathcal{P}_p,
$$

where we assume all sampled subproblems are equally important.

It is more interesting to deal with the preference-conditioned multiobjective optimization problem. For each preference vector $\mathbf{p}_k$, suppose we use the rest preference vectors as its reference vectors, the obtained preference-conditioned multiobjective optimization problem would be:

$$
\begin{align*}
\min_{\phi_t} & \quad \mathcal{L}_1 (g(\mathbf{p}_1 | \phi_t)), \ldots, \mathcal{L}_m (g(\mathbf{p}_1 | \phi_t)), \ldots, \mathcal{L}_1 (g(\mathbf{p}_K | \phi_t)), \ldots, \mathcal{L}_m (g(\mathbf{p}_K | \phi_t)) \\
\text{s.t.} & \quad \mathcal{G}_{1j} (g(\mathbf{p}_1 | \phi_t)) = (\mathbf{p}_j - \mathbf{p}_1)^T \mathcal{L}(g(\mathbf{p}_1 | \phi_t)) \leq 0, \forall j \in \{1, \ldots, K\} \setminus \{1\}, \\
& \quad \mathcal{G}_{2j} (g(\mathbf{p}_2 | \phi_t)) = (\mathbf{p}_j - \mathbf{p}_2)^T \mathcal{L}(g(\mathbf{p}_2 | \phi_t)) \leq 0, \forall j \in \{1, \ldots, K\} \setminus \{2\}, \\
& \quad \ldots \\
& \quad \mathcal{G}_{Kj} (g(\mathbf{p}_K | \phi_t)) = (\mathbf{p}_j - \mathbf{p}_K)^T \mathcal{L}(g(\mathbf{p}_K | \phi_t)) \leq 0, \forall j \in \{1, \ldots, K\} \setminus \{K\}. 
\end{align*}
$$

There are $(K - 1)$ constraints for each preference vector, hence total $K(K - 1)$ constraints for the preference-conditioned multiobjective problem, although some of them could be inactivated. Similar to the single point case, we can also calculate the valid gradient direction in the form of adaptive linear combination as:

$$
\mathbf{d}_t = \sum_{k=1}^{K} \sum_{i=1}^{m} \beta_i(t) \nabla_{\phi_t} \mathcal{L}_i (g(\mathbf{p}_k | \phi_t)),
$$

where the adaptive weight $\beta_i(t)$ depends on all loss functions $\mathcal{L}_i (g(\mathbf{p}_k | \phi_t))$ and activated constraints $\mathcal{G}_{kj} (g(\mathbf{p}_k | \phi_t))$. 


B Convergence-Diversity Analysis

In the previous section, we have discussed how to find a valid gradient direction to update the Pareto solution generator by sampling one or a set of preference vectors at each iteration. For a preference vector $p$, if no valid descent direction can be found, the generated solution $\theta_p$ converges to a local restricted Pareto optimal point. It would be much more complicated to study the convergence of the whole manifold, especially when we use a neural network to approximate the unknown Pareto front.

Instead of giving proof and theorem, we give a general discussion on the convergence and diversity analysis from the view of multiobjective optimization. We hope this discussion can lead to a better understanding of our proposed algorithm, and could be useful for potential future work on learning the whole Pareto front.

B.1 Solving MOP with Finite Solutions: Convergence and Diversity

For a single objective optimization problem, we care about the convergence of a single point to the local or global optimum. For a multiobjective optimization problem, the traditional algorithms usually focus on finding one Pareto solution or a set of finite solutions to approximate the Pareto front. Since the Pareto front is a $(m-1)$-manifold in the objective space, no set of finite solutions can perfectly represent the whole Pareto front. With a fixed number of approximate solutions, there is a convergence-diversity dilemma for the multiobjective optimization algorithms. The current convergence analysis is still mainly on the single point-based algorithm (Fliege et al., 2019).

As shown in Fig. 13, a good approximation for the Pareto front should have a good balance between the convergence and diversity. However, there is no unique best way to define the optimal balance between convergence and diversity for a multiobjective optimization problem. In fact, the definition of diversity is also not unique, and different methods have their pros and cons for different problems. Different criteria and indicators have been proposed to measure the quality of a set of approximated solutions (Deb 2001, Miettinen 2012). They have different concerns and balances between the convergence and diversity, but there is no single best one for all problems. How to fairly evaluate the quality for a set of approximated solutions (e.g., the level it "converges" to the whole Pareto front) is still an open question in the multiobjective optimization community.

B.2 Pareto Solution Generator: Convergence

Different from the finite Pareto solutions approximation, our work proposes to use a Pareto solution generator to learn the whole Pareto front. In the optimal case, by taking all valid preference vectors as input (which is an $(m-1)$-manifold), it should generate all solutions on the Pareto set in the solution space, with the corresponding Pareto front in the objective space (we assume one-to-one mapping for all cases). As shown in Fig. 13(d), for an optimal generator, all generated solutions should be on the Pareto front, and the Pareto front should be well covered by the generated solutions.

For a fixed preference vector $p$, the valid gradient $d_t = 0$ means the corresponding solution $\theta_p = g(p|\phi_t)$ is on the Pareto set and hence it is a Pareto solution. For the Pareto solution generator,
if we obtain optimal parameters $\phi^*$, the gradient direction $d$ should be 0 for all preference vector $p$ in $P$. In this case, all solutions generated from valid preference vectors are Pareto solution as in Fig. 14(a). If we only want to obtain a single Pareto solution ($K = 1$), we can adapt the current theory results on single-point convergence to analyze the generator’s convergence performance (Fliege et al., 2019).

The case could become difficult and challenging once we have infinite preferences. As discussed in the main paper and previous section, the goal to be optimized for a Pareto generator is:

$$\min_{\phi} E_{p \sim P} \mathcal{L}(g(p|\phi)).$$  \hspace{1cm} (27)

If the number of preferences is finite (e.g., $P_p$ is a discrete distribution), we can use the batched preference method (A.4) to update the generator at each iteration. If we have a large number of even infinite preferences (e.g., $P_p$ is a continuous distribution), the single and batch method can be seemed as a sampling approximation to optimize the expectation, which is similar to SGD or batched SGD against full gradient descent.

### B.3 Pareto Solution Generator: Diversity

The previous subsection discusses the convergence issue. However, as shown in Fig. 14(b), only requiring all preferences have gradient direction $d_t = 0$ is not sufficient to define the optimality of the Pareto generator, since it can not avoid convergence to a narrow local region on the Pareto front.

In our proposed algorithm, we connect each preference vector to its corresponding Pareto solution via the Pareto solution generator $g(p|\phi_t)$. During the optimization process, we also require the generated solution should be close to the region around the preference vector. At each iteration, we randomly sample the preference vector $p$ and a set of reference vectors $U$ to decompose the loss space into different regions. A given preference vector $p$ might have different preference regions at different iterations. However, all the corresponding regions should be around the preference vector, and their overlapped region should be very close to the preference vector as in Fig. 15.

As shown in Fig. 14(c), if the preference vectors are widely distributed in the loss space, and the generated solutions are on the Pareto front and close to their own

---

**Figure 14:** The convergence-diversity trade-off for our proposed Pareto solution generator: (a) If all generated solutions are on the Pareto front, there is no valid gradient direction to improve their performance on all tasks; (b) No valid gradient direction is not sufficient for an optimal generator, since all generated solutions can be located in a narrow local region on the Pareto front; (c) To enhance diversity, the proposed algorithm restricts the generated solutions to be close to their widely distributed preference vectors.

**Figure 15:** Different preference regions for a given preference vector $p$ at different iterations.
preference vectors, the proposed algorithm can provide a good approximation to the ground-truth Pareto set and Pareto front.

The remaining question is how to guarantee the preferences vectors can represent and cover the whole Pareto front. In our proposed algorithm, the preference vectors are sampled from the distribution $P_p$ at each iteration. The remaining question is how to properly set the distribution function to cover the ground-truth but unknown Pareto front well. If we have enough knowledge about the given MTL problem, we might define a fixed and well-behaved distribution $P_p$ before the optimization process. However, in real-world applications, the more realistic case is that we know nothing about the Pareto front before actually training the MTL model. Some recent works have studied the relation between the decomposition method and the convergence-diversity indicator (e.g., hypervolume) \cite{Golovin et al. 2020}, which might be useful to adaptively adjust the preference distribution $P_p$ during the optimization process.
C Scalability Improvement

In this paper, we propose to learn the whole Pareto front for an MTL problem, and generate corresponding Pareto solutions based on given preferences in real time. Since both the preference vector and the Pareto front are \((m - 1)\)-dimensional manifolds, if we assume the one-to-one mapping does exist, there could be suitable parametric functions to connect them. However, since we are training large scale neural networks, it still has concerns on learning and generating the network parameters. The generated network parameters are in high dimensional parameter space, although the targeted Pareto set is a low-dimensional manifold.

We believe the hypernetwork can work for large-scale real-world applications for the following reasons: (i) Recent studies make training hypernetwork more robust for large-scale networks (Chang et al., 2020). (ii) The idea of hypernetwork is originally designed for model compression (Schmidhuber, 1992; Stanley et al., 2009; Bertinetto et al., 2016; Ha et al., 2017). Many methods can be used to reduce the size of hypernetwork-based model. A chunking method (Ha et al., 2017; von Oswald et al., 2020) is shown in Fig. 16(a). With different low-dimensional trainable embeddings, one can use a small-size hypernetwork to generate a larger network’s parameters. (iii) The broader multiplicative interaction methods (Jayakumar et al., 2020) might lead to further improvement.

This paper does not focus on designing sophisticated MTL networks, but the following two easy-to-use approaches would be useful to improve the scalability of the proposed Pareto solution generator:

Parameter Sharing: As shown in Fig. 16(b), we can force different trade-off solutions to share the same low-level feature extractor (part of the encoder). In other words, the hypernetwork only generates part of the main MTL network, and the rest parameters are shared among different Pareto solutions. Now the obtained manifold would be a restrict Pareto front, with constraints on the sharing parameters. This form could also be beneficial to knowledge transfer. For example, we can use a pretrained feature extractor (frozen or unfrozen) as the shared parameters, and then build a hypernetwork-based structure on top to generate the Pareto front. The hypernetwork-based model itself is a dynamic feature extractor, can be used to transfer knowledge to other models. More sophisticated parameter sharing methods could be studied in the future.

Preference Embedding: The original input for the hypernetwork is the preference vector, which is typically in a low-dimensional space. One straightforward way to expand the input dimension is to use preference embedding. As shown in Fig. 16(c), if we assign a trainable \(q\)-dimensional embedding \(e_i\) for each task, then the input to the hypernetwork would be \(\sum_i^{m} p_i e_i\), which is also a \(q\)-dimensional vector. We can set a larger \(q\) (e.g., \(q = 100\)) for problem with small \(m\) (e.g., \(m = 2\)). If we set each \(e_i\) as a one-hot (untrainable) \(m\)-dimensional unit vector, the input will reduce to the original preference vector \(p\). In the simples case \(\sum_i^{m} p_i e_i\), we are doing learnable linear interpolation among the preference embedding \(e_i\), which can be trained during the optimization process. Then the highly nonlinear hypernetwork maps the learned embedding into the main MTL network’s parameters. We can also use other powerful nonlinear embedding methods, such as the kernel method and multiplicative interaction method, to further improve this approach.
D Experimental Setting

**Synthetic Example:** The synthetic example we use in section 6 is defined as:

\[
\begin{align*}
\min_{\theta} f_1(\theta) &= 1 - \exp \left( - (\theta_1 - 1)^2 - \frac{1}{n-1} \sum_{i=2}^{n} \left[ \theta_i - \sin(5\theta_1) \right]^2 \right), \\
\min_{\theta} f_2(\theta) &= 1 - \exp \left( - (\theta_1 + 1)^2 \right).
\end{align*}
\] (28)

We set \( n = 10 \) and use a simple two-layer MLP network to generate the Pareto solutions based on the preference vectors.

**MultiMNIST:** In this problem, the goal is to classify two overlapped items in an image. The size of the original MNIST image is 28 × 28. We randomly choose two items from a dataset, and move one item to the upper-left and the other one to the bottom right with up to 4 pixels. Therefore, the input image is 36 × 36. Similar to the previous work (Sener and Koltun, 2018; Lin et al., 2019), we use a LeNet-based network with task-specific FC layers as the MTL model. For the hypernetwork-based model, we use simple MLP as the hypernetwork. For all methods, the optimizer is Adam with learning rate \( lr = 3e^{-4} \), the batch size is 256, and the number of epochs is 200.

**CityScapes:** This problem has two tasks to be learned together, which is the semantic segmentation and depth estimation. We follow the setting used in (Liu et al., 2019), and resize all images into 128 × 256. For the semantic segmentation, the model predicts the 7 coarser labels for each pixel. We use the L1 loss for the depth estimation. We report the experimental results on the Cityscapes validation set. We use the MTL network proposed in (Liu et al., 2019), which has SegNet (Badrinarayanan et al., 2017) as the shared representation encoder, and two task-specific lightweight convolution layers. For all experiments, we use Adam with learning rate \( lr = 3e^{-4} \) as the optimizer, and the batch size is 12. We train the model from scratch with 200 epochs.

**NYUv2:** This problem has a 13-class segmentation task and a depth estimation task for indoor scenes. We resize all images into 288 × 384. We use a similar MTL network and hyperparameter setting as for the CityScapes problem, except the batch size is 3 in this problem.

**CIFAR100 with 20 Tasks:** To validate the algorithm performance on MTL problem with many tasks, we split the CIFAR-100 dataset (Krizhevsky and Hinton, 2009) into 20 tasks, where each task is a 5-class classification problem. Similar setting has been used in MTL learning (Rosenbaum et al., 2018) and continual learning (von Oswald et al., 2020). The MTL neural network has four convolution layers as the shared architecture, and 20 task-specific FC layers. The optimizer is Adam with learning rate \( lr = 3e^{-4} \), and the batch size is 128. We report the test accuracy for all 20 tasks.