Conflict-Averse Gradient Descent for Multi-task Learning

[†]**Bo Liu,** [†]**Xingchao Liu,** [‡]**Xiaojie Jin,** [†,§]**Peter Stone,** [†]**Qiang Liu**
[†]The University of Texas at Austin, [§]Sony AI, [‡]Bytedance Research
{bliu,xcliu,pstone,lqiang}@cs.utexas.edu, xjjin0731@gmail.com

Abstract

The goal of multi-task learning is to enable more efficient learning than single task learning by sharing model structures for a diverse set of tasks. A standard multi-task learning objective is to minimize the average loss across all tasks. While straightforward, using this objective often results in much worse final performance for each task than learning them independently. A major challenge in optimizing a multi-task model is the *conflicting gradients*, where gradients of different task objectives are not well aligned so that following the average gradient direction can be detrimental to specific tasks' performance. Previous work has proposed several heuristics to manipulate the task gradients for mitigating this problem. But most of them lack convergence guarantee and/or could converge to any Pareto-stationary point. In this paper, we introduce Conflict-Averse Gradient descent (CAGrad) which minimizes the average loss function, while leveraging the worst local improvement of individual tasks to regularize the algorithm trajectory. CAGrad balances the objectives automatically and still provably converges to a minimum over the average loss. It includes the regular gradient descent (GD) and the multiple gradient descent algorithm (MGDA) in the multi-objective optimization (MOO) literature as special cases. On a series of challenging multi-task supervised learning and reinforcement learning tasks, CAGrad achieves improved performance over prior state-of-the-art multi-objective gradient manipulation methods. Code is available at `https://github.com/Cranial-XIX/CAGrad`.

1 Introduction

Multi-task learning (MTL) refers to learning a single model that can tackle multiple different tasks [11, 28, 44, 38]. By sharing parameters across tasks, MTL methods learn more efficiently with an overall smaller model size compared to learning with separate models [38, 40, 25]. Moreover, it has been shown that MTL could in principle improve the quality of the learned representation and therefore benefit individual tasks [35, 43, 34]. For example, an early MTL result by [2] demonstrated that training a neural network to recognize doors could be improved by simultaneously training it to recognize doorknobs.

However, learning multiple tasks simultaneously can be a challenging optimization problem because it involves multiple objectives [38]. The most popular MTL objective in practice is the average loss over all tasks. Even when this average loss is exactly the true objective (as opposed to only caring about a single task as in the door/doorknob example), directly optimizing the average loss could lead to undesirable performance, e.g. the optimizer struggles to make progress so the learning performance significantly deteriorates. A known cause of this phenomenon is the *conflicting gradients* [41]: gradients from different tasks 1) may have varying scales with the largest gradient dominating the update, and 2) may point in different directions so that directly optimizing the average loss can be quite detrimental to a specific task's performance.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

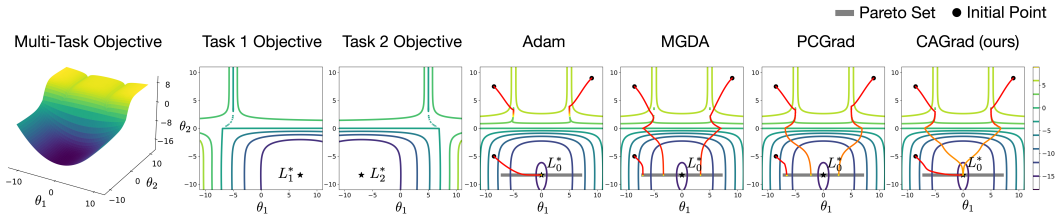


Figure 1: The optimization challenges faced by gradient descent (GD) and existing gradient manipulation methods like the multiple gradient descent algorithm (MGDA) [6] and PCGrad [41]. MGDA, PCGrad and CAGrad are applied on top of the Adam optimizer [16]. For each methods, we repeat 3 runs of optimization from different initial points (labeled with •). Each optimization trajectory is colored from red to yellow. GD with Adam gets stuck on two of the initial points because the gradient of one task overshadows that of the other task, causing the algorithm to jump back and forth between the walls of a steep valley without making progress along the floor of the valley. MGDA and PCGrad stop optimization as soon as they reach the Pareto set.

To address this problem, previous work either adaptively re-weights the objectives of each task based on heuristics [3, 15] or seeks a better update vector [30, 41] by manipulating the task gradients. However, existing work often lacks convergence guarantees or only provably converges to any point on the Pareto set of the objectives. This means the final convergence point of these methods may largely depend on the initial model parameters. As a result, it is possible that these methods over-optimize one objective while overlooking the others (See Fig. 1).

Motivated by the limitation of current methods, we introduce Conflict-Averse Gradient descent (CAGrad), which reduces the conflict among gradients and still provably converges to a minimum of the average loss. The idea of CAGrad is simple: it looks for an update vector that maximizes the worst local improvement of any objective in a neighborhood of the average gradient. In this way, CAGrad automatically balances different objectives and smoothly converges to an optimal point of the average loss. Specifically, we show that vanilla gradient descent (GD) and the multiple gradient descent algorithm (MGDA) are special cases of CAGrad (See Sec. 3.1). We demonstrate that CAGrad can improve over prior state-of-the-art gradient manipulation methods on a series of challenging multi-task supervised, semi-supervised, and reinforcement learning problems.

2 Background

In this section, we first introduce the problem setup of multi-task learning (MTL). Then we analyze the optimization challenge of MTL and discuss the limitation of prior gradient manipulation methods.

2.1 Multi-task Learning and its Challenge

In multi-task learning (MTL), we are given $K \geq 2$ different tasks, each of which is associated with a loss function $L_i(\theta)$ for a shared set of parameters θ. The goal is to find an optimal $\theta \in \mathbb{R}^m$ that achieves low losses across all tasks. In practice, a standard objective for MTL is minimizing the average loss over all tasks:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^m} \left\{ L_0(\theta) \triangleq \frac{1}{K} \sum_{i=1}^{K} L_i(\theta) \right\}. \tag{1}$$

Unfortunately, directly optimizing (1) using gradient descent may significantly compromise the optimization of individual losses in practice. A major source of this phenomenon is known as the conflicting gradients [41].

Optimization Challenge: Conflicting Gradients Denote by $g_i = \nabla L_i(\theta)$ the gradient of task i, and $g_0 = \nabla L_0(\theta) = \frac{1}{K}\sum_i^K g_i$ the averaged gradient. With learning rate $\alpha \in \mathbb{R}^+$, $\theta \leftarrow \theta - \alpha g_0$ is the steepest descent update that appears to be the most natural update to follow when optimizing (1). However, g_0 may conflict with individual gradients, i.e. $\exists\ i,\ \langle g_i, g_0 \rangle < 0$. When this conflict is large, following g_0 will decrease the performance on task i. As observed by [41] and illustrated in Fig. 1, when θ is near a steep "valley", where a specific task's gradient dominates the update, manipulating the direction and magnitude of g_0 often leads to better optimization.

2.2 Prior Attempts and Convergence Issues

Several methods have been proposed to manipulate the task gradients to form a new update vector and have shown improved performance on MTL. Sener et al. apply the multiple-gradient descent algorithm (MGDA) [6] for MTL, which directly optimizes towards the Pareto set [30]. Chen et al. dynamically re-weight each L_i using a pre-defined heuristic [3]. More recently, PCGrad identifies conflicting gradients as the motivation behind manipulating the gradients and projects each task gradient to the normal plane of others to reduce the conflict [41]. While all these methods have shown success at improving the learning performance of MTL, they manipulate the gradient without respecting the original objective (1). Therefore, these methods could in principle converge to any point in the Pareto set (See Fig. 1 and Sec. 3.2). We provide the detailed algorithms of MGDA and PCGrad in Appendix A.1 and A.2, and a visualization of the update vector by each method in Fig. 2.

3 Method

We introduce our main algorithm, Conflict-Averse Gradient descent in Sec. 3.1, and then show theoretical analysis in Sec. 3.2.

3.1 Conflict-Averse Gradient Descent

Assume we update θ by $\theta' \leftarrow \theta - \alpha d$, where α is a step size and d an update vector. We want to choose d to decrease not only the average loss L_0, but also every individual loss. To do so, we consider the minimum decrease rate across the losses,

$$R(\theta, d) = \max_{i\in[K]} \left\{ \frac{1}{\alpha} \left(L_i(\theta - \alpha d) - L_i(\theta) \right) \right\} \approx - \min_{i\in[K]} \langle g_i, d \rangle, \tag{2}$$

where we use the first-order Taylor approximation assuming α is small. If $R(\theta, d) < 0$, it means that all losses are decreased with the update given a sufficiently small α. Therefore, $R(\theta, d)$ can be regarded as a measurement of conflict among objectives.

With the above measurement, our algorithm finds an update vector that minimizes such conflict to mitigate the optimization challenge while still converging to an optimum of the main objective $L_0(\theta)$. To this end, we introduce Conflict-Averse Gradient descent (CAGrad), which on each optimization step determines the update d by solving the following optimization problem:

$$\max_{d\in\mathbb{R}^m} \min_{i\in[K]} \langle g_i, d \rangle \quad \text{s.t.} \quad \|d - g_0\| \leq c \|g_0\|, \tag{3}$$

Here, $c \in [0, 1)$ is a pre-specified hyper-parameter that controls the convergence rate (See Sec. 3.2). The optimization problem (3) looks for the best update vector within a local ball centered at the averaged gradient g_0, which also minimizes the conflict in losses measured by (2). Since we focus on MTL and choose the average loss as the main objective, g_0 is the average gradient. However, CAGrad also applies when g_0 is the gradient of some other user-specified objective. We leave exploring this possibility as a future direction.

Dual Objective The optimization problem (3) involves decision variable d that has the same dimension as the number of parameters in θ, which could be millions for a deep neural network. It is not practical to directly solve for d on every optimization step. However, the dual problem of Eq. (3), as we will derive in the following, only involves solving for a decision variable $w \in \mathbb{R}^K$, which can be efficiently found using standard optimization libraries [7]. Specifically, first note that $\min_i \langle g_i, d \rangle = \min_{w\in\mathcal{W}} \langle \sum_i w_i g_i, d \rangle$, where $w = (w_1, \ldots, w_K) \in \mathbb{R}^K$ and $\mathcal{W}$ denotes the probability simplex, i.e. $\mathcal{W} = \{w \colon \sum_i w_i = 1 \text{ and } w_i \geq 0\}$. Denote $g_w = \sum_i w_i g_i$ and $\phi = c^2 \|g_0\|^2$. The Lagrangian of the objective in Eq. (3) is

$$\max_{d\in\mathbb{R}^m} \min_{\lambda\geq 0, w\in\mathcal{W}} g_w^\top d - \lambda(\|g_0 - d\|^2 - \phi)/2.$$

Since the objective for d is concave with linear constraints, by switching the $\min$ and $\max$, we reach the dual form without changing the solution by Slater's condition:

$$\min_{\lambda\geq 0, w\in\mathcal{W}} \max_{d\in\mathbb{R}^m} g_w^\top d - \lambda \|g_0 - d\|^2 / 2 + \lambda\phi/2.$$

Algorithm 1 Conflict-averse Gradient Descent (CAGrad) for Multi-task Learning

Input: Initial model parameter vector θ_0, differentiable loss functions $\{L_i\}_{i=1}^K$, a constant $c \in [0, 1)$ and learning rate $\alpha \in \mathbb{R}^+$.

repeat

At the t-th optimization step, define $g_0 = \frac{1}{K}\sum_{i=1}^K \nabla L_i(\theta_{t-1})$ and $\phi = c^2 \|g_0\|^2$.

Solve

$$\min_{w\in\mathcal{W}} F(w) := g_w^\top g_0 + \sqrt{\phi}\,\|g_w\|\,, \text{ where } g_w = \frac{1}{K}\sum_{i=1}^K w_i \nabla L_i(\theta_{t-1}).$$

Update $\theta_t = \theta_{t-1} - \alpha\left(g_0 + \frac{\phi^{1/2}}{\|g_w\|} g_w\right)$.

until convergence

We end up with the following optimization problem w.r.t. w after several steps of calculus,

$$w^* = \arg\min_{w\in\mathcal{W}} g_w^\top g_0 + \sqrt{\phi}\,\|g_w\|\,,$$

where the optimal $\lambda^* = \|g_{w^*}\|/\phi^{1/2}$ and the optimal update $d^* = g_0 + g_{w^*}/\lambda^*$. The detailed derivation is provided in Appendix A.3 and the entire CAGrad algorithm is summarized in Alg. 1. The dimension of w equals to the number of objectives K, which usually ranges from 2 to tens and is much smaller than the number of parameters in a neural network. Therefore, in practice, we solve the dual objective to perform the update of CAGrad.

Remark In Alg. 1, when $c = 0$, CAGrad recovers the typical gradient descent with $d = g_0$. On the other hand, when $c \to \infty$, then minimizing $F(w)$ is equivalent to $\min_w \|g_w\|$. This coincides with the multiple gradient descent algorithm (MGDA) [6], which uses the minimum norm vector in the convex hull of the individual gradients as the update direction (see Fig. 2; second column). MGDA is a gradient-based multi-objective optimization designed to converge to an arbitrary point on the Pareto set, that is, it leaves all the points on the Pareto set as fixed points (and hence can not control which specific point it will converge to). It is different from our method which targets to minimize L_0 while using gradient conflict to regularize the optimization trajectory. As we will analyze in the following section, to guarantee that CAGrad converges to an optimum of $L_0(\theta)$, we have to ensure $0 \le c < 1$.

3.2 Convergence Analysis

In this section we first formally introduce the related Pareto concepts and then analyze CAGrad's convergence property. Particularly, in Alg. 1, when $c < 1$, CAGrad is guaranteed to converge to a minimum point of the average loss L_0.

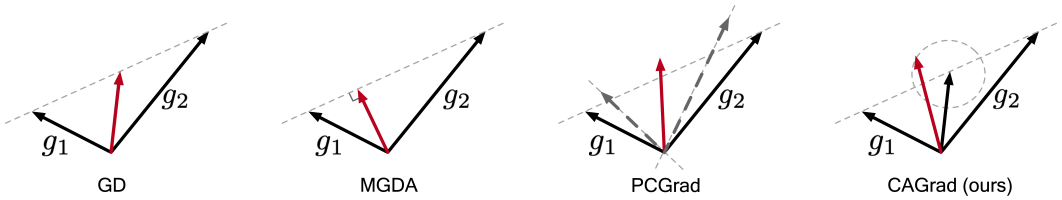
Pareto Concepts Unlike single task learning where any two parameter vectors θ_1 and θ_2 can be ordered in the sense that either $L(\theta_1) \le L(\theta_2)$ or $L(\theta_1) \ge L(\theta_2)$ holds, MTL could have two parameter vectors where one performs better for task i and the other performs better for task $j \ne i$. To this end, we need the notion of Pareto-optimality [13].

Definition 3.1 (Pareto optimal and stationary points)**.** *Let $\boldsymbol{L}(\theta) = \{L_i(\theta)\colon i \in [K]\}$ be a set of differentiable loss functions from $\mathbb{R}^m$ to $\mathbb{R}$. For two points $\theta, \theta' \in \mathbb{R}^m$, we say that θ is Pareto dominated by θ', denoted by $\boldsymbol{L}(\theta') \prec \boldsymbol{L}(\theta)$, if $L_i(\theta') \le L_i(\theta)$ for all $i \in [K]$ and $\boldsymbol{L}(\theta') \ne \boldsymbol{L}(\theta)$. A point $\theta \in \mathbb{R}^m$ is said to be Pareto-optimal if there exists no $\theta' \in \mathbb{R}^m$ such that $\boldsymbol{L}(\theta') \prec \boldsymbol{L}(\theta)$. The set of all Pareto-optimal points is called the Pareto set. A point θ is called Pareto-stationary if we have $\min_{w\in\mathcal{W}} \|g_w(\theta)\| = 0$, where $g_w(\theta) = \sum_{i=1}^K w_i \nabla L_i(\theta)$, and $\mathcal{W}$ is the probability simplex on $[K]$.*

Similar to the case of single-objective differentiable optimization, a local Pareto optimal point θ must be Pareto stationary (see e.g., [6]).

Theorem 3.2 (Convergence of CAGrad)**.** *Assume the individual loss functions $L_0, L_1, \ldots, L_K$ are differentiable on $\mathbb{R}^m$ and their gradients $\nabla L_i(\theta)$ are all H-Lipschitz, i.e. $\|\nabla L_i(x) - \nabla L_i(y)\| \le H\|x - y\|$ for $i = 0, 1, \ldots, K$ where $0 \le H \le \infty$. Assume $L_0^* = \inf_{\theta\in\mathbb{R}^m} L_0(\theta) > -\infty$.*

With a fixed step size α satisfying $0 < \alpha \le 1/H$, we have for the CAGrad in Alg. 1:



$d = (g_1 + g_2)/2$

$\max_d \min_i g_i^\top d$ s.t. $\|d\| \leq 1$

$d = (g_{1\perp 2} + g_{2\perp 1})/2$ where $g_{i\perp j} = g_i - \frac{g_i^\top g_j}{\|g_j\|} g_j$

$\max_d \min_i g_i^\top d$ s.t. $\|d - g_0\| \leq c\,\|g_0\|$

Figure 2: The combined update vector d (in red) of a two-task learning problem with gradient descent (GD), multiple gradient descent algorithm (MGDA), PCGrad and Conflict-Averse Gradient descent (CAGrad). The two task-specific gradients are labeled g_1 and g_2. MGDA's objective is given in its primal form (See Appendix A.1). For PCGrad, each gradient is first projected onto the normal plane of the other (the dashed arrows). Then the final update vector is the average of the two projected gradients. CAGrad finds the best update vector within a ball around the average gradient that maximizes the worse local improvement between task 1 and task 2.

1) For any $c \geq 1$, all the fixed points of CAGrad are Pareto-stationary points of $(L_0, L_1, \ldots, L_K)$.

2) In particular, if we take $0 \leq c < 1$, then CAGrad satisfies

$$\sum_{t=0}^{T} \|\nabla L_0(\theta_t)\|^2 \leq \frac{2(L_0(\theta_0) - L_0^*)}{\alpha(1 - c^2)}.$$

This means that the algorithm converges to a stationary point of ∇L_0 if we take $0 \leq c < 1$. The proof is in Appendix A.3. As we discuss earlier, unlike our method, MGDA is designed to converge to an arbitrary point on the Pareto set, without explicit control of which point it will converges to. Another algorithm with similar property is PCGrad [41], which is a gradient-based algorithm that mitigates the conflicting gradients problem by removing the conflicting components of each gradient with respect to the other gradients before averaging them to form the final update; see Fig. 2, third column for the illustration. Similar to MGDA, as shown in [41], PCGrad also converges to an arbitrary Pareto point without explicit control of which point it will arrive at.

3.3 Practical Speedup

A typical drawback of methods that manipulate gradients is the computation overhead. For computing the optimal update vector, a method usually requires K back-propagations to find all individual gradients g_i, in addition to the time required for optimization. This can be prohibitive for the scenario with many tasks. To this end, we propose to only sample a subset of tasks $S \subseteq [K]$, compute their corresponding gradients $\{g_i \mid i \in S\}$ and the averaged gradient g_0. Then we optimize d in:

$$\max_{d \in \mathbb{R}^m} \min \left(\langle \frac{K g_0 - \sum_{i \in S} g_i}{K - |S|}, d \rangle, \quad \min_{i \in S} \langle g_i, d \rangle \right) \quad \text{s.t.} \quad \|d - g_0\| \leq c\,\|g_0\| \tag{4}$$

Remark Note that the convergence guarantee in Thm. 3.2 still holds for Eq. 4 as the constraint does not change (See Appendix A.3). The time complexity is $\mathcal{O}((|S|N + T)$, where N denotes the time for one pass of back-propagation and T denotes the optimization time. For few-task learning ($K < 10$), usually $T \ll N$. When $S = [K]$, we recover the full CAGrad algorithm.

4 Related Work

Multi-task Learning Due to its benefit with regards to data and computational efficiency, multi-task learning (MTL) has broad applications in vision, language, and robotics [11, 28, 22, 44, 38]. A number of MTL-friendly architectures have been proposed using task-specific modules [25, 11], attention-based mechanisms [21] or activating different paths along the deep networks to tackle MTL [27, 40]. Apart from designing new architectures, another branch of methods focus on decomposing a large problem into smaller local problems that could be quickly learned by smaller models [29, 26, 37, 8]. Then a unified policy is learned from the smaller models using knowledge distillation [12].

MTL Optimization In this work, we focus on the optimization challenge of MTL [38]. Gradient manipulation methods are designed specifically to balance the learning of each task. The simplest form of gradient manipulation is to re-weight the task losses based on specific criteria, e.g., uncertainty [15], gradient norm [3], or difficulty [9]. These methods are mostly heuristics and their performance can be unstable. Recently, two methods [30, 41] that manipulate the gradients to find a better local update vector have become popular. Sener et al [30] view MTL as a multi-objective optimization problem, and use multiple gradient descent algorithm for optimization. PCGrad [41] identifies a major optimization challenge for MTL, the conflicting gradients, and proposes to project each task gradient to the normal plane of other task gradients before combining them together to form the final update vector. Though yielding good empirical performance, both methods can only guarantee convergence to a Pareto-stationary point, but not knowing where it exactly converges to. More recently, GradDrop [4] randomly drops out task gradients based on how much they conflict. IMTL-G [20] seeks an update vector that has equal projections on each task gradient. RotoGrad [14] separately scales and rotates task gradients to mitigate optimization conflict.

Our method, CAGrad, also manipulates the gradient to find a better optimization trajectory. Like other MTL optimization techniques, CAGrad is model-agnostic. However, unlike prior methods, CAGrad converges to the optimal point in theory and achieves better empirical performance on both toy multi-objective optimization tasks and real-world applications.

5 Experiment

We conduct experiments to answer the following questions:

Question (1) Do CAGrad, MGDA and PCGrad behave consistently with their theoretical properties in practice? (yes)

Question (2) Does CAGrad recover GD and MGDA when varying the constant c? (yes)

Question (3) How does CAGrad perform in both performance and computational efficiency compared to prior state-of-the-art methods, on challenging multi-task learning problems under the supervised, semi-supervised and reinforcement learning settings? (CAGrad improves over prior state-of-the-art methods under all settings)

5.1 Convergence and Ablation over c

To answer questions **(1)** and **(2)**, we create a toy optimization example to evaluate the convergence of CAGrad compared to MGDA and PCGrad. On the same toy example, we ablate over the constant c and show that CAGrad recovers GD and MGDA with proper c values. Next, to test CAGrad on more complicated neural models, we perform the same set of experiments on the Multi-Fashion+MNIST benchmark [19] with a shrinked LeNet architecture [18] (in which each layer has a reduced number of neurons compared to the original LeNet). Please refer to Appendix B for more details.

For the toy optimization example, we modify the toy example used by Yu et al. [41] and consider $\theta = (\theta_1, \theta_2) \in \mathbb{R}^2$ with the following individual loss functions:

$$
\begin{aligned}
&L_1(\theta) = c_1(\theta) f_1(\theta) + c_2(\theta) g_1(\theta) \;\text{ and }\; L_2(\theta) = c_1(\theta) f_2(\theta) + c_2(\theta) g_2(\theta), \text{ where} \\
&f_1(\theta) = \log\big(\max(|0.5(-\theta_1 - 7) - \tanh(-\theta_2)|,\;\; 0.000005)\big) + 6, \\
&f_2(\theta) = \log\big(\max(|0.5(-\theta_1 + 3) - \tanh(-\theta_2) + 2|,\;\; 0.000005)\big) + 6, \\
&g_1(\theta) = \big((-\theta_1 + 7)^2 + 0.1 * (-\theta_2 - 8)^2\big)/10 - 20, \\
&g_2(\theta) = \big((-\theta_1 - 7)^2 + 0.1 * (-\theta_2 - 8)^2\big)/10 - 20, \\
&c_1(\theta) = \max(\tanh(0.5 * \theta_2),\; 0) \;\text{ and }\; c_2(\theta) = \max(\tanh(-0.5 * \theta_2),\; 0).
\end{aligned}
$$

The average loss L_0 and individual losses L_1 and L_2 are shown in Fig. 1. We then pick 5 initial parameter vectors $\theta_{\text{init}} \in \{(-8.5, 7.5), (-8.5, 5), (0, 0), (9, 9), (10, -8)\}$ and plot the corresponding optimization trajectories with different methods in Fig. 3. As shown in Fig. 3, GD gets stuck in 2 out of the 5 runs while other methods all converge to the Pareto set. MGDA and PCGrad converge to different Pareto-stationary points depending on θ_{init}. CAGrad with $c = 0$ recovers GD and CAGrad with $c = 10$ approximates MGDA well (in theory it requires $c \to \infty$ to exactly recover MGDA).

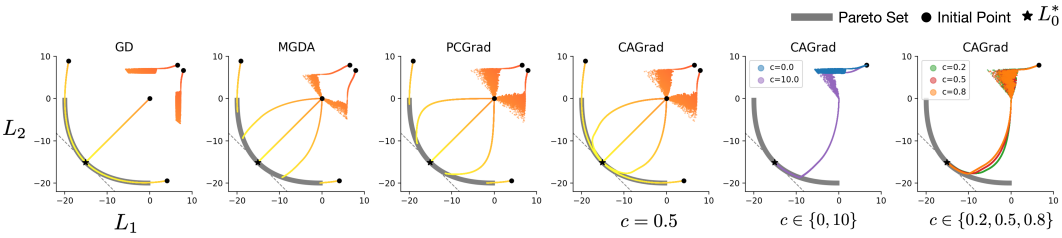


Figure 3: The left four plots are 5 runs of each algorithms from 5 different initial parameter vectors, where trajectories are colored from red to yellow. The right two plots are CAGrad's results with a varying $c \in \{0, 0.2, 0.5, 0.8, 10\}$.

Next, we apply the same set of experiments on the multi-task classification benchmark Multi-Fashion+MNIST [19]. This benchmark consists of images that are generated by overlaying an image from FashionMNIST dataset [39] on top of another image from MNIST dataset [5]. The two images are positioned on the top-left and bottom-right separately. We consider a shrinked LeNet as our model, and train it with Adam [16] optimizer with a 0.001 learning rate for 50 epochs using a batch size of 256. Due to the highly non-convex nature of the neural network, we are not able to visualize the entire Pareto set. But we provide the final training losses of different methods over three independent runs in Fig. 4. As shown, CAGrad achieves the lowest average loss with $c = 0.2$. In addition, PCGrad and MGDA focus on optimizing task 1 and task 2 separately. Lastly, CAGrad with $c = 0$ and $c = 10$ roughly recovers the final performance of GD and MGDA. By increasing c, the model performance shifts from more GD-like to more MGDA-like, though due to the non-convex nature of neural networks, CAGrad with $0 \leq c < 1$ does not necessarily converge to the exact same point.

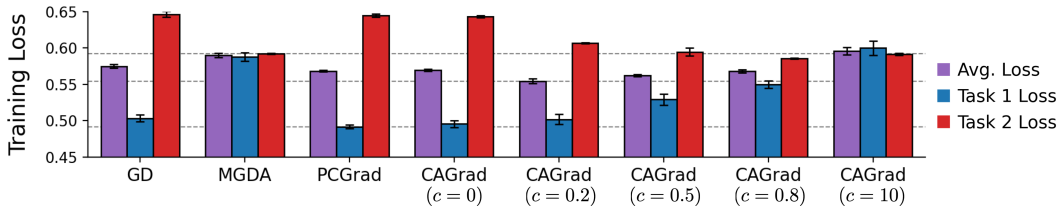


Figure 4: The average and individual training losses on the Fashion-and-MNIST benchmark by running GD, MGDA, PCGrad and CAGrad with different c values. GD gets stuck at the steep valley (the area with a cloud of dots), which other methods can pass. MGDA and PCGrad converge randomly on the Pareto set.

5.2 Multi-task Supervised Learning

To answer question **(3)** in the supervised learning setting, we follow the experiment setup from Yu et al. [41] and consider the NYU-v2 and CityScapes vision datasets. NYU-v2 contains 3 tasks: 13-class semantic segmentation, depth estimation, and surface normal prediction. CityScapes similarly contains 2 tasks: 7-class semantic segmentation and depth estimation. Here, we follow [41] and combine CAGrad with a state-of-the-art MTL method MTAN [21], which applies attention mechanism on top of the SegNet architecture [1]. We compare CAGrad with PCGrad, vanilla MTAN and Cross-Stitch [25], which is another MTL method that modifies the network architecture. MTAN originally experiments with equal loss weighting and two other dynamic loss weighting heuristics [15, 3]. For a fair comparison, all methods are applied under the equal weighting scheme and we use the same training setup from [3]. We search $c \in \{0.1, 0.2, \ldots 0.9\}$ with the best average training loss for CAGrad on both datasets (0.4 for NYU-v2 and 0.2 for Cityscapes). We perform a two-tailed, Student's t-test under *equal sample sizes, unequal variance* setup and mark the results that are significant with an $*$. Following Maninis et al.[24], we also compute the average per-task performance drop of method m with respect to the single-tasking baseline b: $\Delta m = \frac{1}{K}\sum_{i=1}^{K}(-1)^{l_i}(M_{m,i} - M_{b,i})/M_{b,i}$ where $l_i = 1$ if a higher value is better for a criterion M_i on task i and 0 otherwise. The single-tasking baseline (independent) refers to training individual tasks with a vanilla SegNet. Results are shown in Tab. 1 and Tab. 2.

Given the single task performance, CAGrad performs better on the task that is overlooked by other methods (Surface Normal in NYU-v2 and Depth in CityScapes) and matches other methods'

#P.	Method	Segmentation (Higher Better)		Depth (Lower Better)		Surface Normal: Angle Distance (Lower Better)		Surface Normal: Within t° (Higher Better)			$\Delta m\% \downarrow$
		mIoU	Pix Acc	Abs Err	Rel Err	Mean	Median	11.25	22.5	30	
3	Independent	38.30	63.76	0.6754	0.2780	25.01	19.21	30.14	57.20	69.15	
$\approx$3	Cross-Stitch [25]	37.42	63.51	0.5487	**0.2188**	*28.85	*24.52	*22.75	*46.58	*59.56	6.96
1.77	MTAN [21]	39.29	65.33	0.5493	0.2263	*28.15	*23.96	*22.09	*47.50	*61.08	5.59
1.77	MGDA [30]	*30.47	*59.90	*0.6070	*0.2555	**24.88**	**19.45**	**29.18**	**56.88**	**69.36**	1.38
1.77	PCGrad [41]	38.06	64.64	0.5550	0.2325	*27.41	*22.80	*23.86	*49.83	*63.14	3.97
1.77	GradDrop [4]	39.39	65.12	**0.5455**	0.2279	*27.48	*22.96	*23.38	*49.44	*62.87	3.58
1.77	CAGrad (ours)	**39.79**	**65.49**	0.5486	0.2250	26.31	21.58	25.61	52.36	65.58	**0.20**

Table 1: Multi-task learning results on NYU-v2 dataset. #P denotes the relative model size compared to the vanilla SegNet. Each experiment is repeated over 3 random seeds and the mean is reported. The best average result among all multi-task methods is marked in bold. MGDA, PCGrad, GradDrop and CAGrad are applied on the MTAN backbone. CAGrad has statistically significant improvement over baselines methods with an $*$, tested with a p-value of 0.1.

#P.	Method	Segmentation (Higher Better)		Depth (Lower Better)		$\Delta m\% \downarrow$
		mIoU	Pix Acc	Abs Err	Rel Err	
2	Independent	74.01	93.16	0.0125	27.77	
$\approx$3	Cross-Stitch [25]	*73.08	*92.79	*0.0165	*118.5	90.02
1.77	MTAN [21]	75.18	93.49	*0.0155	*46.77	22.60
1.77	MGDA [30]	*68.84	*91.54	0.0309	**33.50**	44.14
1.77	PCGrad [41]	75.13	93.48	0.0154	42.07	18.29
1.77	GradDrop [4]	**75.27**	**93.53**	*0.0157	*47.54	23.73
1.77	CAGrad (ours)	75.16	93.48	**0.0141**	37.60	**11.64**

Table 2: Multi-task learning results on CityScapes Challenge. Each experiment is repeated over 3 random seeds and the mean is reported. The best average result among all multi-task methods is marked in bold. PCGrad and CAGrad are applied on the MTAN backbone. CAGrad has statistically significant improvement over baselines methods with an $*$, tested with a p-value of 0.1.

performance on the rest of the tasks. We also provide the final test losses and the per-epoch training time of each method in Fig. 5 in Appendix B.2.

5.3 Multi-task Reinforcement Learning

To answer question **(3)** in the reinforcement learning (RL) setting, we apply CAGrad on the MT10 and MT50 benchmarks from the Meta-World environment [42]. In particular, MT10 and MT50 contains 10 and 50 robot manipulation tasks. Following [33], we use Soft Actor-Critic (SAC) [10] as the underlying RL training algorithm. We compare against Multi-task SAC (SAC with a shared model), Multi-headed SAC (SAC with a shared backbone and task-specific head), Multi-task SAC + Task Encoder (SAC with a shared model and the input includes a task embedding) [42] and PCGrad [41]. We also compare with Soft Modularization [40] that routes different modules in a shared model to form different policies. Lastly, we also include a recent method (CARE) that considers language metadata and uses a mixture of expert encoder for MTL. We follow the same experiment setup from [33]. The results are shown in Tab. 3. CAGrad outperforms all baselines except for CARE which benefits from extra information from the metadata. We also apply the practical speedup in Sec. 3.3 and sub-sample 4 and 8 tasks for MT10 and MT50 (CAGrad-Fast). CAGrad-fast achieves comparable performance against the state-of-the-art method while achieving a 2x (MT10) and 5x (MT50) speedup over PCGrad. We provide a visualization of tasks from MT10 and MT50, and the comparison of computational efficiency in Appendix B.3.

5.4 Semi-supervised Learning with Auxiliary Tasks

Training with auxiliary tasks to improve the performance of a main task is another popular application of MTL. Here, we take semi-supervised learning as an instance. We combine different optimization

Method	Metaworld MT10	Metaworld MT50
	success (mean ± stderr)	success (mean ± stderr)
Multi-task SAC [42]	0.49 ±0.073	0.36 ±0.013
Multi-task SAC + Task Encoder [42]	0.54 ±0.047	0.40 ±0.024
Multi-headed SAC [42]	0.61 ±0.036	0.45 ±0.064
PCGrad [41]	0.72 ±0.022	0.50 ±0.017
Soft Modularization [40]	0.73 ±0.043	0.50 ±0.035
CAGrad (ours)	**0.83** ±0.045	**0.52** ±0.023
CAGrad-Fast (ours)	0.82 ±0.039	0.50 ±0.016
CARE [33]	0.84 ±0.051	0.54 ±0.031
One SAC agent per task (upper bound)	0.90 ±0.032	0.74 ±0.041

Table 3: Multi-task reinforcement learning results on the Metaworld benchmarks. Results are averaged over 10 independent runs and the best result is marked in bold.

algorithms with Auxiliary Task Reweighting for Minimum-data Learning (ARML) [31], a state-of-the-art semi-supervised learning algorithm. The loss function is composed of the main task and two auxiliary tasks:

$$L_0 = L_{CE}(\theta; D_l) + w_1 L^1_{aux}(\theta; D_u) + w_2 L^2_{aux}(\theta; D_u), \tag{5}$$

where L_{CE} is the main cross-entropy classification loss on the labeled dataset D_l, and L^1_{aux}, L^2_{aux} are auxiliary unsupervised learning losses on the unlabeled dataset D_u. We use the same w_1 and w_2 from ARML, and use the CIFAR10 dataset [17], which contains 50,000 training images and 10,000 test images. 10% of the training images is held out as the validation set. We test PCGrad, MGDA and CAGrad with 500, 1000 and 2000 labeled images. The rest of the training set is used for auxiliary tasks. For all the methods, we use the same labeled dataset, the same learning rate and train them for 200 epochs with the Adam [16] optimizer. Please refer to Appendix B.4 for more experimental details. Results are shown in Tab. 4. With all the different number of labels, CAGrad yields the best averaged test accuracy. We observed that MGDA performs much worse than the ARML baseline, because it significantly overlooks the main classification task. We also compare different gradient manipulation methods on the same task with GradNorm [3], which dynamically adjusts w_1 and w_2 during training. The results and conclusions are similar to those for ARML.

Method	500 labels	1000 labels	2000 labels
ARML [31]	67.05 ±0.16	73.22 ±0.26	81.35 ±0.36
ARML + PCGrad [41]	67.49 ±0.64	73.23 ±0.62	81.91 ±0.19
ARML + MGDA [30]	49.27 ±0.68	60.11 ±2.35	60.78 ±0.17
ARML + CAGrad (Ours)	**68.25** ±0.37	**74.37** ±0.42	**82.81** ±0.48
GradNorm [3]	67.35 ±0.15	73.53 ±0.23	81.03 ±0.71
GradNorm + PCGrad [41]	**67.83** ±0.19	73.91 ±0.09	82.72 ±0.19
GradNorm + MGDA [30]	36.99 ±2.11	57.94 ±0.92	59.12 ±0.63
GradNorm + CAGrad (Ours)	67.53 ±0.26	**74.72** ±0.19	**83.15** ±0.56

Table 4: Semi-supervised Learning with auxiliary tasks on CIFAR10. We report the average test accuracy over 3 independent runs for each method and mark the best result in bold.

6 Conclusion

In this work, we introduce the Conflict-Averse Gradient descent (CAGrad) algorithm that explicitly optimizes the minimum decrease rate of any specific task's loss while still provably converging to the optimum of the average loss. CAGrad generalizes the gradient descent and multiple gradient descent algorithm, and demonstrates improved performance across several challenging multi-task learning problems compared to the state-of-the-art methods. While we focus mainly on optimizing the average loss, an interesting future direction is to look at main objectives other than the average loss under the multi-task setting.

Acknowledgements

The research was conducted in the statistical learning and AI group (SLAI) and the Learning Agents Research Group (LARG) in computer science at UT Austin. SLAI research is supported in part by CAREER-1846421, SenSE-2037267, EAGER-2041327, and Office of Navy Research, and NSF AI Institute for Foundations of Machine Learning (IFML). LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, FAIN-2019844), ONR (N00014-18-2243), ARO (W911NF-19-2-0333), DARPA, Lockheed Martin, GM, Bosch, and UT Austin's Good Systems grand challenge. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research. Xingchao Liu is supported in part by a funding from BP.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[2] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[3] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803. PMLR, 2018.

[4] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. *arXiv preprint arXiv:2010.06808*, 2020.

[5] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[6] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5-6):313–318, 2012.

[7] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[8] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

[9] Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. Dynamic task prioritization for multitask learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 270–287, 2018.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[11] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[13] Harold M Hochman and James D Rodgers. Pareto optimal redistribution. *The American economic review*, 59(4):542–557, 1969.

[14] Adrián Javaloy and Isabel Valera. Rotograd: Dynamic gradient homogenization for multi-task learning. *arXiv preprint arXiv:2103.02631*, 2021.

[15] Alex Kendall, Yarin Gal, and Roberto 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*, pages 7482–7491, 2018.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qingfu Zhang, and Sam Kwong. Pareto multi-task learning. *arXiv preprint arXiv:1912.12854*, 2019.

[20] Liyang Liu, Yi Li, Zhanghui Kuang, Jing-Hao Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. In *International Conference on Learning Representations*, 2020.

[21] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019.

[22] Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks. *arXiv preprint arXiv:2011.10219*, 2020.

[23] Debabrata Mahapatra and Vaibhav Rajan. Multi-task learning with user preferences: Gradient descent with controlled ascent in pareto optimization. In *International Conference on Machine Learning*, pages 6597–6607. PMLR, 2020.

[24] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1851–1860, 2019.

[25] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[26] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[27] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

[28] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[29] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[30] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *arXiv preprint arXiv:1810.04650*, 2018.

[31] Baifeng Shi, Judy Hoffman, Kate Saenko, Trevor Darrell, and Huijuan Xu. Auxiliary task reweighting for minimum-data learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[32] Shagun Sodhani and Amy Zhang. Mtrl - multi task rl algorithms. Github, 2021.

[33] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. *arXiv preprint arXiv:2102.06177*, 2021.

[34] Charles Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Contribution to the Theory of Statistics*, pages 197–206. University of California Press, 2020.

[35] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. 2013.

[36] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1195–1204, 2017.

[37] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

[38] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[39] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[40] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. *arXiv preprint arXiv:2003.13661*, 2020.

[41] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[42] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[43] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

[44] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Sec. 3.2 for the convergence analysis, Fig. 1 for the challenges faced by previous methods, and Sec. 5 for empirical evaluation of these challenges and the advantage of CAGrad.
   (b) Did you describe the limitations of your work? [Yes] See Sec. 6. Currently we mainly focus on optimizing the average loss, which could be replaced by other main objectives.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our method does not have potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] The assumptions are stated in Thm. 3.2.
   (b) Did you include complete proofs of all theoretical results? [Yes] The complete proof is included in Appendix A.3.
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We mention most of the details to reproduce the result in Sec. 5 and provide the rest of details of each experiment in Appendix.B. The code comes with the supplementary material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix.B and Sec. 5.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For each experiment except for the toy (since there is no stochasticity), we run over multiple (≥ 3) seeds.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We explicitly compare the computational efficiency in Fig. 5. More details on the resources are provided in the corresponding sections in Appendix.B.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] For most of the experiment, we follow the exact experiment setup and use the corresponding open-source code from previous works and have cited and compared against them.

(b) Did you mention the license of the assets? [Yes] All code and data are publicly available under MIT license

(c) Did you include any new assets either in the supplemental material or as a URL? [No] No new assets are introduced for our experiment. The only thing we modified is a shrinked LeNet, where the details are provided in Appendix.B.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The data we use are publicly available data that has been used by a lot of prior research. There should be no personally identifiable information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No human subjects involved.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Algorithm Details

In this section, we first formally introduce the Multiple Gradient Descent Algorithm and the Projecting Conflicting Gradients method. Then we provide the full proof of Thm. 3.2.

A.1 Multiple Gradient Descent Algorithm (MGDA)

The Multiple Gradient Descent Algorithm (MGDA) explicitly optimizes towards a Pareto-optimal point for multiple objectives (See the definition 3.1). It is known that a necessary condition for θ to be a Pareto-optimal point is that we could find a convex combination of the task gradients at θ that results in the 0 vector. Therefore, MGDA proposes to minimize the minimum possible convex combination of task gradients:

$$\min \frac{1}{2} \left\| \sum_{i=1}^{K} w_i g_i \right\|^2, \quad \text{s.t.} \quad \sum_{i=1}^{K} w_i = 1, \text{ and } \forall i, w_i \geq 0. \tag{6}$$

We call this the *dual* objective for MGDA, as the primal objective of MGDA has a close connection to CAGrad's primal objective in Eq. (3). Specifically, the *primal* objective of MGDA is

$$\max_{\|d\| \leq 1} \min_i \langle d, g_i \rangle. \tag{7}$$

To see the primal-dual relationship, denote $g_w = \sum_i w_i g_i$, where $w \in \mathcal{W} \triangleq \{w \in \mathbb{R}^K : \sum_i w_i = 1,\ w_i \geq 0, \forall i \in [K]\}$. Note that $\min_i \langle g_i, d\rangle = \min_{w\in\mathcal{W}} \langle \sum_i w_i g_i, d\rangle$. The Lagrangian of Eq. (7) is

$$\max_d \min_{\lambda\geq 0, w\in\mathcal{W}} \langle d, g_w\rangle - \frac{\lambda}{2}(\|d\|^2 - 1). \tag{8}$$

Since the problem is a convex programming and the Slater's condition holds when $c > 0$ (On the other hand, if $c = 0$, then it is easy to check that all the results hold trivially), the strong duality holds and we can exchange the min and max:

$$\min_{\lambda\geq 0, w\in\mathcal{W}} \max_d \langle d, g_w\rangle - \frac{\lambda}{2}(\|d\|^2 - 1). \tag{9}$$

The optimal $d^* = g_w/\lambda$ and the resulting primal objective is therefore

$$\min_{\lambda\geq 0, w\in\mathcal{W}} \lambda(\frac{1}{2}\|g_w\|^2 + 1). \tag{10}$$

Here, λ corresponds to the constraint $\|d\| \leq 1$. If we fix λ to be any constant, then we recover the dual objective in Eq. (6).

Remark Looking at the primal form of MGDA in Eq. (7), the major difference between MGDA and CAGrad is that the new update vector d is searched around the 0 vector for MGDA and g_0 for CAGrad. Therefore, theoretically both MGDA and CAGrad optimizes the worst local update, but MGDA is more conservative and can converge to any point on the Pareto set without explicit control (See Thm. 2 from [6]). This also explains MGDA's behavior in practice that it often learns much slower than other methods.

A.2 Projecting Conflicting Gradients (PCGrad)

Identifying that a major challenge for multi-task optimization is the conflicting gradient, Yu et al. [41] propose to project each task gradient to the normal plane of others before combining them together to form the final update vector. In the following, we provide the full algorithm of the Projecting Conflicting Gradients (PCGrad):

Algorithm 2 Projecting Conflicting Gradient Update Rule

Input: model parameter vector θ and differentiable loss functions $\{L_i\}_{i=1}^K$.
$g_i \leftarrow \nabla_\theta L_i(\theta)$.
$g_i^{\text{PC}} = g_i,\ \forall i$.
for task $i \in [K]$ **do**
  for $j \neq i \in [K]$ in random order **do**
    if $g_i^{\text{PC}} \cdot g_j < 0$ **then**
      $g_i^{\text{PC}} = g_i^{\text{PC}} - \frac{g_i^{\text{PC}} \cdot g_j}{\|g_j\|^2} g_j$.
    end if
  end for
end for
Return the new update vector $d = g^{\text{PC}} = \frac{1}{K}\sum_i g_i^{\text{PC}}$.

Fig. 2 provides a visualization of PCGrad's update rule for two-task learning (the 3rd column). Different from MGDA and CAGrad, PCGrad does not have a clear optimization objective at each step, which makes it hard to analyze PCGrad's convergence guarantee in general. In practice, the random ordering to do the projection is particularly important for PCGrad to work well [41], which suggests that the intuition of removing the "conflicting" part of each gradient might not be always correct. For the convergence analysis, Yu et al. establishes the convergence guarantee for PCGrad only under the two-task learning setting. Moreover, PCGrad is only guaranteed to converge to the Pareto set without explict control over which point it will arrive at (See Thm. A.1 in the following).

Theorem A.1 (Convergence of PCGrad [41])**.** *Consider two-task learning, assume the loss functions L_1 and L_2 are convex and differentiable. Suppose the gradient of $L_0 = (L_1 + L_2)/2$ is H-Lipschitz with $H > 0$. Then, the PCGrad update rule with step size $t \leq 1/H$ will converge to a Pareto-stationary point.*

A.3 Conflit-Averse Gradient descent (CAGrad)

We provide the full derivation of CAGrad and the proof for its convergence in this section. Our proof assumes L_0 is a general function with gradient $g_0 = \nabla L_0$, that is, it does not have to be the average of L_i as the case we focus on in the main paper.

Lemma A.2. *Let d^* be the solution of*

$$\max_{d\in\mathbb{R}^m} \min_{i\in[K]} g_i^\top d \quad s.t. \quad \|g_0 - d\| \le c\,\|g_0\|\,,$$

where $c \ge 0$, and $g_0, g_1, \dots, g_K \in \mathbb{R}^m$. Then we have

$$d^* = g_0 + \frac{c\,\|g_0\|}{\|g_{w^*}\|} g_{w^*}\,,$$

where $g_{w^*} = \sum_i w_i^* g_i$ and w^* is the solution of

$$\min_{w\ge\mathcal{W}} g_w^\top g_0 + c\,\|g_0\|\,\|g_w\|\,, \tag{11}$$

where $\mathcal{W} = \{w \in \mathbb{R}^K : \quad \sum_i w_i = 1, \;\; w_i \ge 0, \forall i \in [K]\}$. In addition,

$$\min_i g_i^\top d^* = g_{w^*}^\top g_0 + c\,\|g_0\|\,\|g_{w^*}\|\,. \tag{12}$$

Proof. Denote $\phi = c^2\,\|g_0\|^2$. Note that $\min_i \langle g_i, d\rangle = \min_{w\in\mathcal{W}} \langle \sum_i w_i g_i, d\rangle$. The Lagrangian of the objective in Eq. (3) is

$$\max_{d\in\mathbb{R}^m} \min_{\lambda\ge 0, w\in\mathcal{W}} g_w^\top d - \frac{\lambda}{2}(\|g_0 - d\|^2 - \phi).$$

Since the problem is a convex programming and the Slater's condition holds when $c > 0$ (On the other hand, if $c = 0$, then it is easy to check that all the results hold trivially), the strong duality holds and we can exchange the min and max:

$$\min_{\lambda\ge 0, w\in\mathcal{W}} \max_{d\in\mathbb{R}^m} g_w^\top d - \frac{\lambda}{2}\,\|g_0 - d\|^2 + \frac{\lambda\phi}{2}.$$

With λ, w fixing, the optimal d is achieved when $d = g_0 + g_w/\lambda$, yielding the following dual problem

$$\min_{w,\lambda\ge 0} g_w^\top (g_0 + g_w/\lambda) - \frac{\lambda}{2}\,\|g_w/\lambda\|^2 + \frac{\lambda}{2}\phi.$$

This is equivalent to

$$\min_{w,\lambda\ge 0} g_w^\top g_0 + \frac{1}{2\lambda}\,\|g_w\|^2 + \frac{\lambda\phi}{2}.$$

Optimizing out the λ we have

$$\min_{w\in\mathcal{W}} g_w^\top g_0 + \sqrt{\phi}\,\|g_w\|\,,$$

where the optimal $\lambda = \|g_w\|\,/\phi^{1/2}$. This solves the problem. (12) is the consequence of the strong duality. □

Convergence Analysis

Assumption A.3. *Assume individual loss functions $L_0, L_1, \dots, L_K$ are differentiable on $\mathbb{R}^m$ and their gradients $\nabla L_i(\theta)$ are all H-Lipschitz, i.e. $\|\nabla L_i(x) - \nabla L_i(y)\| \le H\,\|x - y\|$ for $i = 0, 1, \dots, K$, where $H \in (0, \infty)$. Assume $L_0^* = \inf_{\theta\in\mathbb{R}^m} L_0(\theta) > -\infty$.*

Theorem A.4 (Convergence of CAGrad)**.** *Assume Assumption A.3 holds. With a fixed step size α satisfying $0 < \alpha \le 1/H$, we have for the CAGrad in Alg. 1:*

1) If $0 \le c < 1$, then CAGrad converges to stationary points of L_0 convergence rate in that

$$\sum_{t=0}^{T} \|g_0(\theta_t)\|^2 \le \frac{2(L_0(\theta_0) - L_0^*)}{\alpha(1 - c^2)}.$$

2) For any $c \ge 0$, all the fixed point of CAGrad are Pareto-stationary points of $(L_0, L_1, \dots, L_K)$.

Proof. We will first prove 1). Consider the t-th optimization step and denote $d^*(\theta_t)$ the update direction obtained by solving (3) at the t-th iteration. Then we have

$$\begin{aligned}
L_0(\theta_{t+1}) - L_0(\theta_t) &= L_0(\theta_t - \alpha d^*(\theta_t)) - L_0(\theta_t) \\
&\leq -\alpha g_0(\theta_t)^\top d^*(\theta_t) + \frac{H\alpha^2}{2} \|d^*(\theta_t)\|^2 \\
&\leq -\alpha g_0(\theta_t)^\top d^*(\theta_t) + \frac{\alpha}{2} \|d^*(\theta_t)\|^2 \qquad //\alpha \leq 1/H \\
&\leq -\frac{\alpha}{2}\left(\|g_0(\theta_t\|^2 + \|d^*(\theta_t)\|^2 - \|g_0(\theta_t) - d^*(\theta_t)\|^2\right) + \frac{\alpha}{2}\|d^*(\theta_t)\|^2 \\
&= -\frac{\alpha}{2}\left(\|g_0(\theta_t)\|^2 - \|d^*(\theta_t) - g_0(\theta_t)\|^2\right) \\
&\leq -\frac{\alpha}{2}(1-c^2)\|g_0(\theta_t)\|^2 \qquad //\text{by the constraint in (3)}
\end{aligned}$$

Using telescoping sums, we have $L_0(\theta_{T+1}) - L_0(0) = -(\alpha/2)(1-c^2)\sum_{t=0}^T \|g_0(\theta_t)\|^2$. Therefore

$$\min_{t\leq T} \|g_0(\theta_t)\|^2 \leq \frac{1}{T+1}\sum_{t=0}^T \|g_0(\theta_t)\|^2 \leq \frac{2(L_0(0) - L_0(\theta_{T+1}))}{\alpha(1-c^2)(T+1)}.$$

Therefore, if L_0 is lower bounded, that is, $L_0^* := \inf_{\theta\in\mathbb{R}^m} L_0(\theta) > -\infty$, then $\min_{t\leq T}\|g_0(\theta_t)\|^2 = O(1/T)$.

For general $c \geq 0$, in the fixed point, we have $d^*(\theta) = g_0(\theta) + \lambda g_{w^*}(\theta) = 0$, which readily match the definition of Pareto Stationarity. □

In the following, we show an additional result that when $c \geq 1$, and we use a properly decaying step size, the limit points of CAGrad are either stationary points of L_0, or Pareto-stationary points of $(L_1, \ldots, L_K)$.

Theorem A.5. *Under Assumption A.3, assume $c \geq 1$ and we a time varying step size satisfying*

$$\alpha_t \leq \frac{\left\|g_{w_t^*}(\theta_t)\right\|}{H(c-1)\|g_0(\theta_t)\|},$$

where w_t^* is the solution of (11) *at the t-th iteration, then we have*

$$\sum_{t=0}^T \alpha_t \|g_0(\theta_t)\| \left\|g_{w_t^*}(\theta_t)\right\| \leq 2\frac{\min_i(L_i(\theta_0) - L_i(\theta_{T+1}))}{(c-1)}.$$

Therefore, if we hae $L_i^* = \inf_{\theta\in\mathbb{R}^m} L(\theta) > -\infty$ and $c > 1$, then we have $\alpha_t \|g_0(\theta_t)\| \left\|g_{w_t^*}(\theta_t)\right\| \to 0$ as $t \to \infty$, meaning that we have either $\alpha_t \to 0$, or $\|g_0(\theta_t)\| \to 0$ or $\left\|g_{w_t^*}(\theta_t)\right\| \to 0$.

In this case, the actual behavior of the algorithm depends on the specific choice of the step size. For example, if we take $\alpha_t = \frac{\left\|g_{w_t^*}(\theta_t)\right\|}{H(c-1)\|g_0(\theta_t)\|}$, then the result becomes

$$\sum_{t=0}^T \left\|g_{w_t^*}(\theta_t)\right\|^2 \leq 2H\min_i(L_i(\theta_0) - L_i(\theta_{T+1})).$$

which ensures $\left\|g_{w_t^*}(\theta_t)\right\|^2 \to 0$.

Proof. For any task $i \in [K]$,

$$\begin{aligned}
L_i(\theta_{t+1}) - L_i(\theta) &\leq -\alpha_t g_i(\theta_t)^\top d^*(\theta_t) + \frac{H\alpha_t^2}{2}\|d^*(\theta_t)\|^2 \\
&\leq -\alpha_t \min_i g_i(\theta_t)^\top d^*(\theta_t) + \frac{H\alpha_t^2}{2}\|d^*(\theta_t)\|^2 \\
&\leq -\alpha_t\left(g_{w_t^*}(\theta_t)^\top g_0(\theta_t) + c\|g_0(\theta_t)\|\left\|g_{w_t^*}(\theta_t)\right\|\right) + \frac{H\alpha_t^2}{2}\|d^*(\theta_t)\|^2 \qquad //\text{by (12)}
\end{aligned}$$

Meanwhile, note that

$$\begin{aligned}
\|d^*(\theta_t)\|^2 &= \left\| g_0(\theta_t) + \frac{c\,\|g_0(\theta_t)\|}{\|g_{w_t^*}(\theta_t)\|} g_{w_t^*}(\theta_t) \right\|^2 \\
&= (c^2+1)\,\|g_0(\theta_t)\|^2 + 2\frac{c\,\|g_0(\theta_t)\|}{\|g_{w_t^*}(\theta_t)\|} g_0(\theta_t)^\top g_{w_t^*}(\theta_t) \\
&= 2c\frac{\|g_0(\theta_t)\|}{\|g_{w_t^*}(\theta_t)\|}\left(g_{w_t^*}(\theta_t)^\top g_0(\theta_t) + c\,\|g_0(\theta_t)\|\,\|g_{w_t^*}(\theta_t)\|\right) + (1-c^2)\,\|g_0(\theta_t)\|^2 .
\end{aligned}$$

Therefore,

$$\begin{aligned}
&L_i(\theta_{t+1}) - L_i(\theta) \\
&\le -\alpha_t\left(1 - H\alpha_t c\frac{\|g_0(\theta_t)\|}{\|g_{w_t^*}(\theta_t)\|}\right)\left(g_{w_t^*}(\theta_t)^\top g_0(\theta_t) + c\,\|g_0(\theta_t)\|\,\|g_{w_t^*}(\theta_t)\|\right) + \frac{H\alpha_t^2}{2}(c^2-1)\,\|g_0(\theta_t)\|^2 \\
&\overset{(*)}{\le} -\alpha_t\left(1 - H\alpha_t c\frac{\|g_0(\theta_t)\|}{\|g_{w_t^*}(\theta_t)\|}\right)(c-1)\,\|g_0(\theta_t)\|\,\|g_{w_t^*}(\theta_t)\| - \frac{H\alpha_t^2}{2}(c^2-1)\,\|g_0(\theta_t)\|^2 \\
&= -\alpha_t(c-1)\,\|g_0(\theta_t)\|\,\|g_{w_t^*}(\theta_t)\| + \frac{H\alpha_t^2}{2}(c-1)^2\,\|g_0(\theta_t)\|^2 \\
&\le -\frac{1}{2}\alpha_t(c-1)\,\|g_0(\theta_t)\|\,\|g_{w_t^*}(\theta_t)\| \qquad \text{//assume } \alpha_t \le \frac{\|g_{w_t^*}(\theta_t)\|}{H(c-1)\,\|g_0(\theta_t)\|},\ c \ge 1
\end{aligned}$$

where inequality (*) uses Cauchy-Schwarz inequality. Therefore, a telescoping sum gives

$$\sum_{t=0}^{T} \alpha_t\,\|g_0(\theta_t)\|\,\|g_{w_t^*}(\theta_t)\| \le 2\frac{\min_i(L_i(\theta_0) - L_i(\theta_{T+1}))}{(c-1)},$$

when $c \ge 1$.

□

B Experiment Details

B.1 Multi-Fashion+MNIST

Experiment Details We follow the experiment setup from [23] and use the same shrinked LeNet that consists of the following layers as the shared base network: Conv(1,5,9,1), MaxPool2d(2), ReLU, BatchNorm2d(5), Conv2d(5,10,5,1), MaxPool2d(2), ReLU, BatchNorm1d(250), Linear(250, 50). Then a task-specific linear head Linear(50, 10) is attached to the shared base for the MNIST and FashionMNIST prediction. We use Adam [16] optimizer with a 0.001 learning rate and 0.01 weight decay, and then train for 50 epochs with a batch size of 256. The training set consists of 120000 images of size 36x36 and the test set consists of 20000 images of the same size.

B.2 Multi-task Supervised Learning

Experiment Details For the multi-task supervised learning experiments on the NYU-v2 and CityScapes datasets, we follow exactly the same setup from MTAN [21]. We describe the details in the following. We adopt the SegNet [1] architecture as the backbone network and apply the attention mechanism from MTAN [21] on top of it. For the CityScapes dataset, we use the 7-class semantics labels. We train MTAN, Cross-Stitch, PCGrad and CAGrad with 200 epochs with a batch size of 2 for NYU-v2 and a batch size of 8 for CityScapes, using the Adam [16] optimizer with a learning rate of 0.0001. We further decay the learning rate to 0.00005 at the 100th epoch. As Liu et al. do not separately create a validation set, they average the test performance of each method in the last 10 epochs. We follow this and also average the test performance over the last 10 epochs, but additionally run over 3 seeds and calculate the mean and the standard error. We train CAGrad with $c \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and pick the best c using their corresponding averaged training performance ($c = 0.4$ for NYU-v2 and $c = 0.4$ for CityScapes).

We also provide the final test losses and the per-epoch training times of each method in Fig. 5.

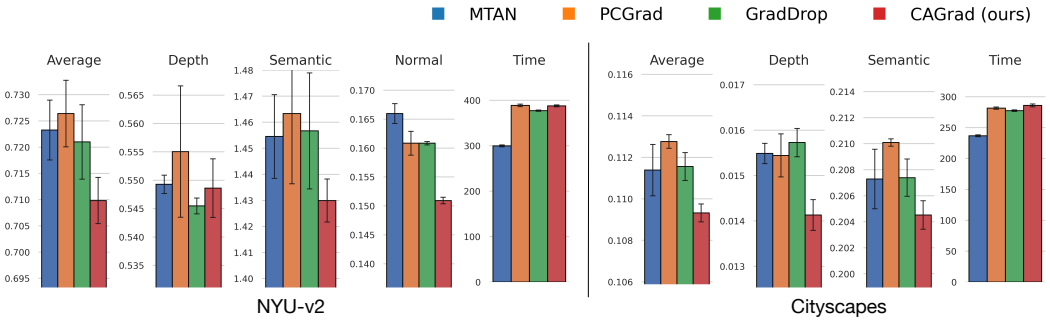


Figure 5: Test loss and training time comparison on NYU-v2 and Cityscapes.

#P.	Method	Segmentation (Higher Better)		Depth (Lower Better)		Surface Normal					$\Delta m\% \downarrow$
						Angle Distance (Lower Better)		Within t° (Higher Better)			
		mIoU	Pix Acc	Abs Err	Rel Err	Mean	Median	11.25	22.5	30	
3	Independent	38.30	63.76	0.6754	0.2780	25.01	19.21	30.14	57.20	69.15	
≈3	Cross-Stitch [25]	37.42	63.51	0.5487	0.2188	28.85	24.52	22.75	46.58	59.56	6.96
1.77	MTAN [21]	39.29	65.33	0.5493	0.2263	28.15	23.96	22.09	47.50	61.08	5.59
1.77	MGDA [30]	30.47	59.90	0.6070	0.2555	24.88	19.45	29.18	56.88	69.36	1.38
1.77	PCGrad [41] (lr=1e-4)	38.06	64.64	0.5550	0.2325	27.41	22.80	23.86	49.83	63.14	3.97
1.77	PCGrad [41] (lr=2e-4)	37.70	63.40	0.5871	0.2482	28.18	24.09	21.94	47.20	60.87	8.12
1.77	GradDrop [4]	39.39	65.12	0.5455	0.2279	27.48	22.96	23.38	49.44	62.87	3.58
1.77	CAGrad (c=0.2)	39.15	65.45	0.5563	0.2295	26.74	21.93	25.17	51.55	64.70	1.55
1.77	CAGrad (c=0.4)	39.79	65.49	0.5486	0.2250	26.31	21.58	25.61	52.36	65.58	0.20
1.77	CAGrad (c=0.6)	39.54	65.60	0.5340	0.2199	25.87	20.94	25.88	53.78	67.00	-1.36
1.77	CAGrad (c=0.8)	39.18	64.97	0.5379	0.2229	25.42	20.47	27.37	54.73	67.73	-2.29
1.77	MTAN [21] (Uncert. Weights)	38.74	64.70	0.5360	0.2243	26.52	21.71	25.50	52.02	65.14	0.75
1.77	PCGrad [41] (Uncert. Weights)	37.81	64.35	0.5318	0.2242	26.53	21.73	25.45	51.98	65.16	1.04
1.77	CAGrad (c=0.2) (Uncert. Weights)	38.87	65.19	0.5357	0.2227	26.38	21.64	25.66	52.21	65.39	0.319
1.77	CAGrad (c=0.4) (Uncert. Weights)	38.89	64.98	0.5313	0.2242	25.71	20.72	26.89	54.14	67.13	-1.59
1.77	CAGrad (c=0.6) (Uncert. Weights)	39.80	65.32	0.5334	0.2242	25.69	20.91	26.89	54.14	67.13	-1.59
1.77	CAGrad (c=0.8) (Uncert. Weights)	39.20	65.15	0.5322	0.2202	25.28	20.17	27.83	55.41	68.25	-3.14

Table 5: Multi-task learning results on NYU-v2 dataset. #P denotes the relative model size compared to the vanilla SegNet. Each experiment is repeated over 3 random seeds and the mean is reported.

More Ablation Studies on NYU-v2 and CityScapes Datasets We conduct the following additional studies on NYU-v2 and CityScapes datasets: 1) How do different methods perform when we additional apply the uncertain weight method [15]? 2) How do CAGrad perform with different values of c? 3) How does PCGrad perform when we enlarge the learning rate? Specifically we double the learning rate to 2e-4. Results are provided in Tab. 5 and Tab. 6. We can see that CAGrad perform consistently with different values of $0 < c < 1$. PCGrad with larger learning rate will not perform better. Under the uncertain weights, MTAN and PCGrad indeed perform better but CAGrad is still comparable or better than them.

B.3 Multi-task Reinforcement Learning

Experiment Details The multi-task reinforcement learning experiments follow the exact setup from CARE [33]. Specifically, it is built on top of the MTRL codebase [32]. We consider the MT10 and MT50 benchmarks from the MetaWorld environment [42]. A visualization of the 50 tasks from MT50 is provided in Fig. 6. The MT10 benchmark consists of a subset of 10 tasks from the MT50 task pool. For all methods, we use Soft Actor Critic (SAC) [10] as the underlying reinforcement learning algorithm. All methods are trained over 2 million steps with a batch size of 1280. Following CARE [32], we evaluate each method once every 10000 steps, and report the highest average test performance of a method over 10 random seeds over the entire training stage. For CAGrad-Fast, we sub-sample 4 and 8 tasks randomly at each optimization step as the S (See Eq. (4)) for the MT10 and MT50 experiments. For CAGrad, since MT10 and MT50 have 10 and 50 tasks, much more

#P.	Method	Segmentation (Higher Better) mIoU	Segmentation (Higher Better) Pix Acc	Depth (Lower Better) Abs Err	Depth (Lower Better) Rel Err	$\Delta m\% \downarrow$
2	Independent	74.01	93.16	0.0125	27.77	
≈3	Cross-Stitch [25]	73.08	92.79	0.0165	118.5	90.02
1.77	MTAN [21]	75.18	93.49	0.0155	46.77	22.60
1.77	MGDA [30]	68.84	91.54	0.0309	33.50	44.14
1.77	PCGrad [41]	75.13	93.48	0.0154	42.07	18.29
1.77	GradDrop [4]	75.27	93.53	0.0157	47.54	23.73
1.77	CAGrad (c=0.2)	75.18	93.49	0.0140	40.12	13.69
1.77	CAGrad (c=0.4)	75.16	93.48	0.0141	37.60	11.64
1.77	CAGrad (c=0.6)	74.31	93.39	0.0151	34.84	11.46
1.77	CAGrad (c=0.8)	74.95	93.50	0.0143	36.05	10.74
1.77	MTAN [21] (Uncert. Weights)	75.02	93.36	0.0139	35.56	9.48
1.77	PCGrad [41] (Uncert. Weights)	74.68	93.36	0.0135	34.00	7.26
1.77	CAGrad (c=0.2) (Uncert. Weights)	75.05	93.45	0.0140	34.33	8.40
1.77	CAGrad (c=0.4) (Uncert. Weights)	74.90	93.46	0.0141	34.84	9.13
1.77	CAGrad (c=0.6) (Uncert. Weights)	74.89	93.45	0.0136	35.17	8.48
1.77	CAGrad (c=0.8) (Uncert. Weights)	75.38	93.48	0.0141	35.54	9.63

Table 6: Multi-task learning results on CityScapes Challenge. Each experiment is repeated over 3 random seeds and the mean is reported.

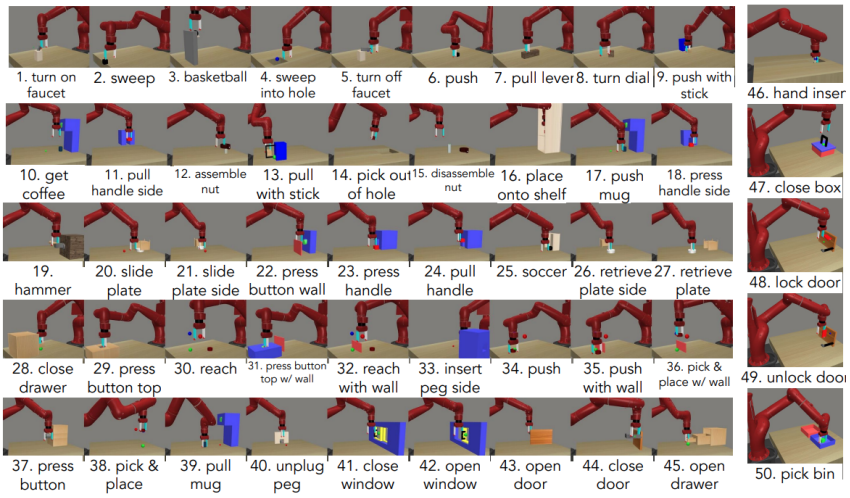


Figure 6: The 50 tasks in MT50 benchmark [42].

than the number of tasks in supervised MTL, so instead of using standard optimization library to solve the CAGrad objective, we apply 20 gradient descent steps to approximately solve the objective. The gradient descent is performed with a learning rate of 25 for MT10 and 50 for MT50, with a momentum of 0.5. We search the best c from $\{0.1, 0.5, 0.9\}$ for MT10 and MT50 ($c = 0.9$ for MT10 and $c = 0.5$ for MT50). The computation efficiency is compared in Tab. 7.

In principle, PCGrad should have the same time complexity as CAGrad. However, in practice,

Method	MT10 Time (sec)	MT50 Time (sec)
PCGrad	9.7	59.8
CAGrad	10.3	27.8
CAGrad-Fast	**4.8**	**11.4**

Table 7: The training time per update step for PCGrad, CAGrad and CAGrad-Fast on MT10/50.

PCGrad projects the gradients following a random ordering of the tasks in a sequential fashion (See Alg. 2), so it requires a for loop over that task ordering, which makes it slow for a large number of tasks. Combined with the results from Tab. 3, we see that CAGrad-Fast achieves comparable or better results than PCGrad with a roughly 2x and 5x speedup on MT10 and MT50.

B.4 Semi-Supervised Learning with Auxiliary Tasks

Experiment Details We provide the hyperparameters for reproducing the experiments in our main text. All the methods are applied upon the original ARML baseline, with the same configuration in [31]. Specifically, the batch size is 256 and the optimizer is Adam. The learning rate is initialized to 0.005 in the first $160,000$ iterations and decay to 0.001 in the rest iterations. The backbone networks is a `WRN-28-2` model. To stablize the training process, the features are extracted by a moving-averaged model like in [36] with a moving-average factor of 0.95. For PCGrad and MGDA, we use their official implementation without any change. For CAGrad (our method), we fix $c = 0.1$ in all the experiments. The labeled images are randomly selected from the whole training set, and we repeat the experiments for 3 times on the same set of labeled images. We report the test accuracy of the model with the highest validation accuracy.

Training Losses We analyze the training losses of different methods to demonstrate the difference between these optimization methods. We report the losses, L_{CE}, L^1_{aux} and L^2_{aux}, of the last epoch, when the number of labeled images is $2,000$. The losses are listed in Tab. 8. We have two key observations: (1) MGDA totally ignores the main task L_{CE}, yet it has the smallest loss on the second auxiliary task L^2_{aux}. This implies MGDA finds a sub-optimal solution on the Pareto front. (2) PCGrad and CAGrad can both decrease the averaged loss L_0 compared with the baseline ARML, however, CAGrad yields a smaller L_0 than PCGrad.

Method	L_{CE}	L^1_{aux}	L^2_{aux}	L_0
ARML [31]	**0.0** $\pm$0.0	0.0574 $\pm$0.0036	-0.4946 $\pm$0.0010	-0.4372 $\pm$0.0046
ARML + PCGrad [41]	**0.0** $\pm$0.0	0.0494 $\pm$0.0088	-0.4943 $\pm$0.0007	-0.4449 $\pm$0.0095
ARML + MGDA [30]	0.407 $\pm$0.018	0.0453 $\pm$0.0049	**-0.4980** $\pm$0.0007	-0.0463 $\pm$0.0233
ARML + CAGrad (Ours)	**0.0** $\pm$0.0	**0.0419** $\pm$0.0034	-0.4926 $\pm$0.0023	**-0.4507** $\pm$0.0058

Table 8: The Training Losses in the Last Epoch when the number of the labeled images is $2,000$. Values that are smaller than 10^{-6} are replaced by 0. We report the averaged losses over 3 independent runs for each method, and mark the smallest losses in bold.