Conflict-Averse Gradient Descent for Multi-task Learning

Anonymous Author(s)
Affiliation
Address
email

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.

1 Introduction

Multi-task learning (MTL) refers to learning a single model that can tackle multiple different tasks [10, 23, 37, 31]. By sharing parameters across tasks, MTL methods learn more efficiently with an overall smaller model size compared to learning with separate models [31, 33, 20]. Moreover, it has been shown that MTL could in principle improve the quality of the learned representation and therefore benefit individual tasks [29, 36, 28]. 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 [31]. 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* [34]: 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.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

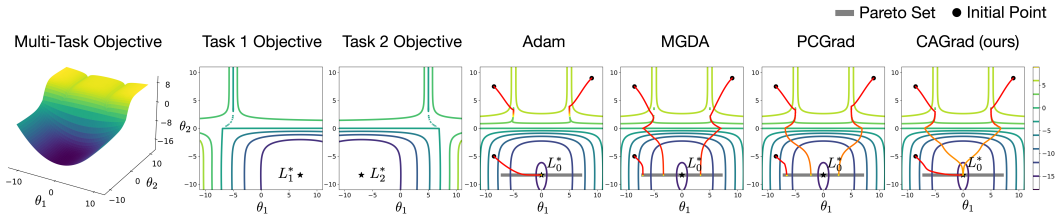

Figure 1: The optimization challenges faced by gradient descent (GD) and existing gradient manipulation methods like the multiple gradient descent algorithm (MGDA) [5] and PCGrad [34]. MGDA, PCGrad and CAGrad are applied on top of the Adam optimizer [14]. For each methods, we repeat 3 runs of optimization 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, 13] or seeks a better update vector [25, 34] 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 [34].

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 [34] 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) [5] for MTL, which directly optimizes towards the Pareto set [25]. 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 [34]. 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,

$$R(\theta, d) = \max_i \left\{ \frac{1}{\alpha} \left(L_i(\theta - \alpha d) - L_i(\theta) \right) \right\} \approx - \min_i \langle g_i, d \rangle, \tag{2}$$

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:

$$\min_{d \in \mathbb{R}^m} R(\theta,\ d) \approx \max_{d \in \mathbb{R}^m} \min_i \langle g_i, d \rangle \quad \text{s.t.} \quad \|d - g_0\| \leq c \|g_0\|, \tag{3}$$

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 [6]. 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

$$\max_d \min_{\lambda \geq 0, w} g_w^\top d - \lambda(\|g_0 - d\|^2 - \phi)/2.$$

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:

$$\min_{\lambda \geq 0, w} \max_d g_w^\top d - \lambda \|g_0 - d\|^2 / 2 + \lambda \phi / 2.$$

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,

$$w^* = \arg\min_w g_w^\top g_0 + \sqrt{\phi}\,\|g_w\|\,,$$

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) [5], 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 [12].

Definition 3.1 (Pareto-optimality and Pareto Set)**.** *Given a set of loss functions $\{L_i \colon \mathbb{R}^m \to \mathbb{R}, 1 \le i \le K\}$, a point $\theta \in \mathbb{R}^m$ achieves Pareto-optimality if there does not exist any other feasible $\theta' \in \mathbb{R}^m$ such that $L_i(\theta') \le L_i(\theta)$ for all $i \in [K]$ and $L_i(\theta') < L_i(\theta)$ for some $i \in [K]$. The set of all Pareto-optimal points is called the Pareto set.*

When individual loss functions $\{L_i\}$ are differentiable, previous work has identified the following necessary condition for Pareto-optimality [5].

Lemma 3.2. *Given a set of smooth loss functions $\{L_i \colon \mathbb{R}^m \to \mathbb{R}, 1 \le i \le K\}$, for a point θ, if the following condition holds:*

$$\exists \{w_i\} \text{ s.t. } \sum_{i=1}^K w_i \nabla L_i(\theta) = 0, \text{ where } \forall i, w_i \ge 0 \text{ and } \sum_i w_i = 1.$$

then θ is a Pareto-stationary point. Note that all the points in the Pareto set are Pareto-stationary.

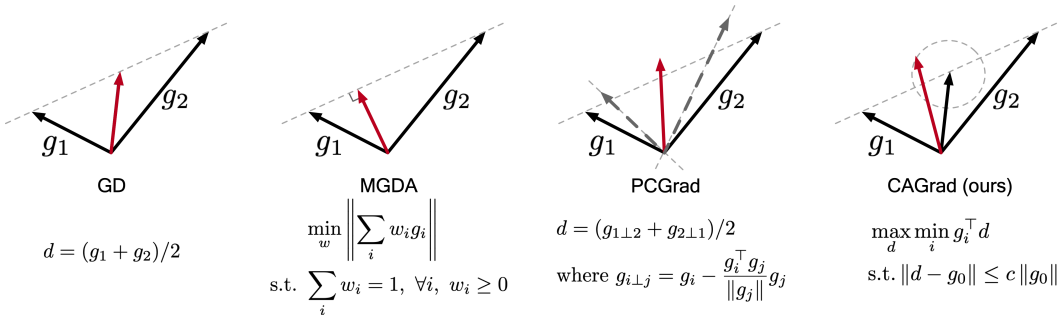

Figure 2: The combined update vector d (in red) of a two-task learning problem with gradient descent (GD), multiple gradient descent algorithm (MGDA), PCGrad and Conflict-Averse Gradient descent (CAGrad). The two task-specific gradients are labeled g_1 and g_2. For PCGrad, each gradient is first projected 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.

As a result, the Pareto set defines the feasible parameter vectors that are not *dominated* by any other parameter vectors (e.g. θ_1 dominates θ_2 if $L_i(\theta_1) \leq L_i(\theta_2)$ for all i and $L_i(\theta_1) < L_i(\theta_2)$ for some i).

Theorem 3.3 (Convergence of CAGrad)**.** *Assume individual loss functions L_i are differentiable and their gradients $\nabla L_i(\theta)$ are all H-Lipschitz, i.e. $\|\nabla L_i(x) - \nabla L_i(y)\| \leq H\,\|x - y\|$ for any task i and x, y. Then, with a fixed step size $\alpha \leq 1/H$, consider CAGrad in Alg. 1, we have the following:*
1) If $0 \leq c < 1$, then CAGrad converges to a stationary point of $L_0 = \frac{1}{K}\sum_{i=1}^K L_i$ with a $\mathcal{O}(\frac{1}{T})$ convergence rate in that

$$\min_{t \leq T} \|g_0(\theta_t)\|^2 \leq \frac{2(L_0(\theta_0) - L_0(\theta_{T+1}))}{\alpha(1 - c^2)(T + 1)}.$$

2) CAGrad converges to a Pareto-stationary point when $c \geq 1$.

The proof of Thm. 3.3 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 of similar property is PCGrad [34], 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 [34], PCGrad also converges to an arbitrary Pareto point without 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 $S \subset [K]$, compute their corresponding gradients $\{g_i \mid i \in S\}$ and the averaged gradient g_0. Then we optimize d in:

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

Remark Note that the convergence guarantee still holds 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 [10, 23, 37, 31]. A number of

MTL-friendly architectures have been proposed using task-specific modules [20, 10], attention-based mechanisms [18] or activating different paths along the deep networks to tackle MTL [22, 33]. 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 [24, 21, 30, 7]. Then a unified policy is learned from the smaller models using knowledge distillation [11].

MTL Optimization In this work, we focus on the optimization challenge of MTL [31]. 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 [13], gradient norm [3], or difficulty [8]. These methods are mostly heuristics and their performance can be unstable. Recently, two methods [25, 34] that manipulate the gradients to find a better local update vector have become popular. Sener et al [25] view MTL as a multi-objective optimization problem, and use multiple gradient descent algorithm for optimization. PCGrad [34] 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. 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 [17] with a shrinked LeNet architecture [16] (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. [34] 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).

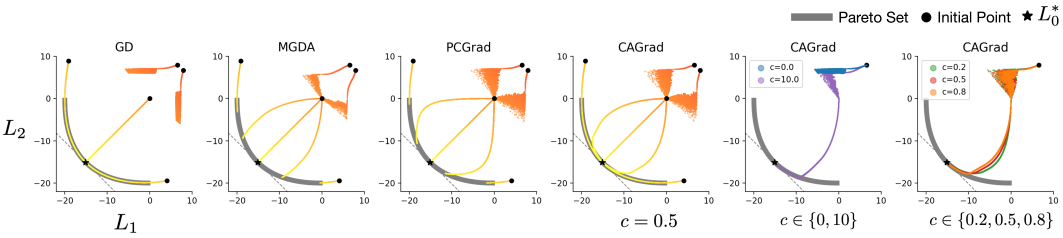

Figure 3: The left four plots are 5 runs of each algorithms from 5 different initial parameter vectors, where trajectories are colored from red to yellow. The right two plots are CAGrad's results with a varying $c \in \{0, 0.2, 0.5, 0.8, 10\}$.

Next, we apply the same set of experiments on the multi-task classification benchmark Multi-Fashion+MNIST [17]. This benchmark consists of images that are generated by overlaying an image from FashionMNIST dataset [32] on top of another image from MNIST dataset [4]. 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 [14] 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.

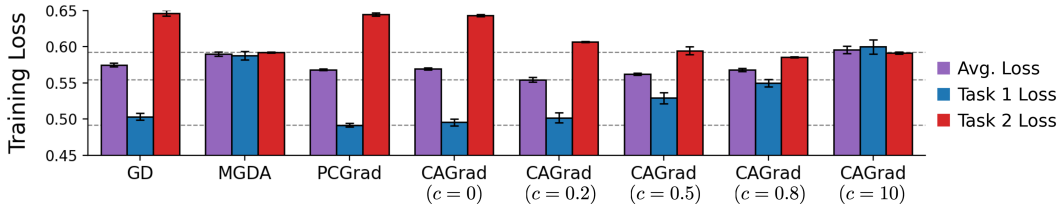

Figure 4: The average and individual training losses on the Fashion-and-MNIST benchmark by running GD, MGDA, PCGrad and CAGrad with different c values. GD gets stuck at the steep valley (the area with a cloud of dots), which other methods can pass. MGDA and PCGrad converge randomly on the Pareto set.

5.2 Multi-task Supervised Learning

To answer question **(3)** in the supervised learning setting, we follow the experiment setup from Yu et al. [34] 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 [34] and combine CAGrad with a state-of-the-art MTL method MTAN [18], which applies attention mechanism on top of the SegNet architecture [1]. We compare CAGrad with PCGrad, vanilla MTAN and Cross-Stitch [20], which is another MTL method that modifies the network architecture. MTAN originally experiments with equal loss weighting and two other dynamic loss weighting heuristics [13, 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.[19], 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					$\Delta m\% \downarrow$
						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	
$\approx$3	Cross-Stitch [20]	37.42	63.51	0.5487	**0.2188**	*28.85	*24.52	*22.75	*46.58	*59.56	6.96
1.77	MTAN [3]	38.66	64.43	0.5483	0.2248	*28.08	*23.79	*22.30	*47.86	*59.80	5.53
1.77	PCGrad [34]	38.89	65.17	0.5592	0.2387	*27.95	*23.57	*22.52	*48.26	*61.82	5.66
1.77	CAGrad (ours)	**39.30**	**65.70**	**0.5481**	0.2281	**26.16**	**21.38**	**25.31**	**52.75**	**66.12**	**0.18**

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. 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 0.05.

#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 [20]	*73.08	*92.79	*0.0165	*118.5	90.03
1.77	MTAN [3]	**75.40**	**93.56**	0.0150	*49.87	24.40
1.77	PCGrad [34]	75.30	93.55	0.0151	*44.18	19.42
1.77	CAGrad (ours)	**75.40**	93.50	**0.0143**	**38.49**	**12.67**

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 0.08.

performance on the rest of the tasks. We provide the final test losses and the per-epoch training time of each method in Fig. 5. CAGrad achieves lower losses than PCGrad using similar or shorter time.

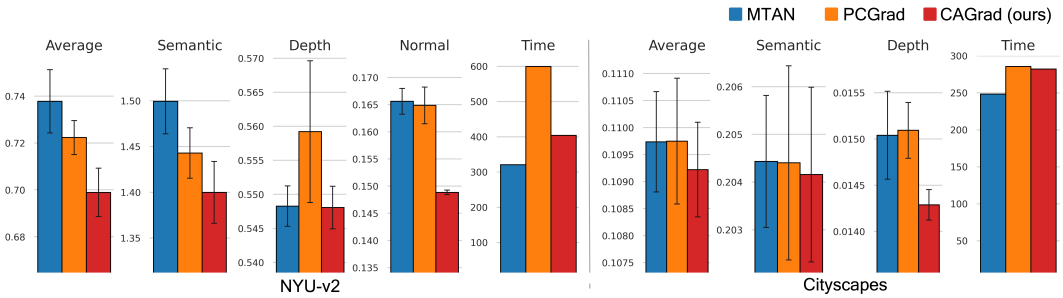

Figure 5: Test loss and training time comparison on NYU-v2 and Cityscapes.

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 [35]. In particular, MT10 and MT50 contains 10 and 50 robot manipulation tasks. Following [27], we use Soft Actor-Critic (SAC) [9] 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) [35] and PCGrad [34]. We also compare with Soft Modularization [33] 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 [27]. 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 12x (MT10) and 3x (MT50) speedup over PCGrad. We provide a visualization of tasks from MT10 and MT50, and the comparison of computational efficiency in Appendix B.3.

Method	Metaworld MT10 success (mean ± stderr)	Metaworld MT50 success (mean ± stderr)
Multi-task SAC [35]	0.49 ±0.073	0.36 ±0.013
Multi-task SAC + Task Encoder [35]	0.54 ±0.047	0.40 ±0.024
Multi-headed SAC [35]	0.61 ±0.036	0.45 ±0.064
PCGrad [34]	0.72 ±0.022	0.50 ±0.017
Soft Modularization [33]	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 [27]	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.

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 algorithms with Auxiliary Task Reweighting for Minimum-data Learning (ARML) [26], a state-of-the-art semi-supervised learning algorithm. The loss function is composed of the main task and two auxiliary tasks:

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

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 [15], 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 [14] optimizer. 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. Please refer to Appendix B.4 for more experimental details.

Method	500 labels	1000 labels	2000 labels
ARML [26]	67.05 ±0.16	73.22 ±0.26	81.35 ±0.36
ARML + PCGrad [34]	67.49 ±0.64	73.23 ±0.62	81.91 ±0.19
ARML + MGDA [25]	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

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.

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

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

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

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

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

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

[26] 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.

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

[28] 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.

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

[30] 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.

[31] 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.

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

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

[34] 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.

[35] 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.

[36] 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.

[37] 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. 3.3.
   (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]