In Defense of the Unitary Scalarization for Deep Multi-Task Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent multi-task learning research argues against *unitary scalarization*, where training simply minimizes the sum of the task losses. Several ad-hoc multi-task optimization algorithms have instead been proposed, inspired by various hypotheses about what makes multi-task settings difficult. The majority of these optimizers require per-task gradients, and introduce significant memory, runtime, and implementation overhead. We show that unitary scalarization, coupled with standard regularization and stabilization techniques from single-task learning, matches or improves upon the performance of complex multi-task optimizers in popular supervised and reinforcement learning settings. We then present an analysis suggesting that many specialized multi-task optimizers can be partly interpreted as forms of regularization, potentially explaining our surprising results. We believe our results call for a critical reevaluation of recent research in the area.

1 Introduction

Multi-Task Learning (MTL) [4] exploits similarities between tasks to yield models that are more accurate, generalize better and require less training data. Owing to the success of MTL on traditional machine learning models [2, 15, 21] and of deep single-task learning across a variety of domains, a growing body of research has focused on deep MTL. The most straightforward way to train a neural network for multiple tasks at once is to minimize the sum of per-task losses. Adopting terminology from multi-objective optimization, we call this approach *unitary scalarization*.

While some work shows that multi-task networks trained via unitary scalarization exhibit superior performance to independent per-task models [27, 33], others suggest the opposite [28, 51, 55]. As a result, many explanations for the difficulty of MTL have been proposed, each motivating a new Specialized Multi-Task Optimizer (SMTO) [10, 40, 51, 59, 62]. These works typically claim that the proposed SMTO outperforms unitary scalarization, in addition to relevant prior work. However, SMTOs usually require access to per-task gradients either with respect to the shared parameters, or to the shared representation. Therefore, their reported performance gain comes at significant computation and memory cost, the overhead scaling linearly with the number of tasks. By contrast, unitary scalarization requires only the average of the gradients across tasks, which can be computed via a single backpropagation.

Existing SMTOs were introduced to solve challenges related to the optimization of the deep MTL problem. We instead postulate that the reported weakness of unitary scalarization is linked to experimental variability or to a lack of regularization, leading to the following contributions:

- A comprehensive experimental evaluation (§4) of recent SMTOs on popular multi-task benchmarks, showing that no SMTO consistently outperforms unitary scalarization in spite of the added complexity and overhead. In particular, either the differences between unitary scalarization

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

and SMTOs are not statistically significant, or they can be bridged by standard regularization and stabilization techniques from the single-task literature. Our reinforcement learning (RL) experiments include optimizers previously applied only to supervised learning.

- An empirical and technical analysis of the considered SMTOs, suggesting that they reduce overfitting on the multi-task problem and hence act as regularizers (§5). We conduct an ablation study and provide a collection of novel and existing technical results that support this hypothesis.
- We will publicly release a unified PyTorch [47] codebase, currently provided in the supplementary material, for all SMTOs in the experimental evaluation.

We believe that our results suggest that the considered SMTOs can be often replaced by less expensive techniques. We hope that these surprising results stimulate the search for a deeper understanding of MTL.

2 Related Work

Before diving into details of specific SMTOs in Section 5, we provide a high-level overview of the deep MTL research. MTL is represented by a vast body of literature, including Caruana's pioneering work [5] exploring *hard parameter sharing*, i.e., sharing neural network parameters between all tasks with, possibly, a separate part of the model for each task. Hard parameter sharing is still the major MTL approach adopted in natural language processing [8, 11], computer vision [44], and speech recognition [50]. In this work, we implicitly assume that each parameter update employs information from all tasks. However, not all works satisfy this assumption, either due to a large number of tasks [3, 34], or simply as an implementation decision [24, 35]. In this setting, MTL resembles other problems dealing with multiple tasks, i.e., continual [30], curriculum [45], and meta-learning [23], which are not the focus of this work.

Many works strive to improve the performance of deep multi-task models. One line of research hypothesizes that conflicting per-task gradient directions lead to suboptimal models, and thus focuses on explicitly removing conflicts across task gradients [10, 26, 39, 40, 59, 62]. Some authors postulate that loss imbalances across tasks hinder learning, proposing loss reweighting methods [9, 28, 38]. Finally, Sener and Koltun [51] propose that tasks compete for model capacity and interpret MTL as multi-objective optimization in order to cope with inter-task competition. Here, we focus on algorithms that explicitly rely on per-task gradients to try to outperform unitary scalarization (§5). Research on multi-task architectures [18, 44] or MTL algorithms exclusively motivated by deterministic loss reweighting [17, 28, 41] are orthogonal to our work. Both topics are investigated by a recent survey on pixel-level multi-task computer vision problems [58], which found that the minimization of tuned weighted sums of losses is empirically competitive with deterministic loss reweighting and MGDA in the considered settings.

In addition to the common supervised settings, we also consider multi-task RL, whose research can be grouped into three categories: the first adds auxiliary tasks providing additional inductive biases to speed up learning [25] on a target task. The second, based on policy distillation, uses per-task teacher models to provide labels for a multi-task model or per-task policies as regularizers [46, 48, 54]. The third directly learns a shared policy [27], possibly via an SMTO [62]. We focus on the third category, whose literature reports varying performance for unitary scalarization (better [27] or worse [62] than per-task models), indicating confounding factors in evaluation pipelines and further motivating our work. PopArt [22, 57] performs scale-invariant value function updates in order to address differences in returns across environments, showing improvements in the multi-task setting while still using unitary scalarization. PopArt does not require per-task gradients but introduces additional hyperparameters. In our work, we address the differences in rewards by normalizing them at the replay buffer level. However, we believe both unitary scalarization and SMTOs might equally benefit from PopArt.

3 Multi-Task Learning Optimizers

We will now describe the deep MTL training problem and common algorithms employed for its solution. Let $(X, Y) \in \mathbb{R}^{d \times n} \times \mathbb{R}^{o \times n}$ be the training set, composed of n d-dimensional points and o-dimensional labels. In addition, $\mathcal{L}_i : \mathbb{R}^{o \times n} \times \mathbb{R}^{o \times n} \to \mathbb{R}$ denotes the loss for the i-th task, $\boldsymbol{\theta} \in \mathbb{R}^S$ the parameter space, $\mathcal{T} := \{1, \ldots, m\}$ the set of m tasks. The goal of MTL is to learn a single (generally task-aware) parametrized model $f : \mathbb{R}^S \times \mathbb{R}^{d \times n} \times \mathcal{T} \to \mathbb{R}^{o \times n}$ that performs well on all

tasks $\mathcal{T}$. The parameter space is often split into a set of shared parameters across tasks (generally the majority of the architecture), denoted $\boldsymbol{\theta}_{\|}$, and (possibly empty) task-specific parameters, denoted $\boldsymbol{\theta}_{\perp}$, so that $\boldsymbol{\theta} := [\boldsymbol{\theta}_{\|}, \boldsymbol{\theta}_{\perp}]^T$. In this context, the model f often takes on an encoder-decoder architecture, where the encoder g learns a shared representation across tasks, and the decoders h_i are task-specific predictive heads: $f(\boldsymbol{\theta}, X, i) = h_i(g(\boldsymbol{\theta}_{\|}, X), \boldsymbol{\theta}_{\perp})$. In this case, we denote by $\mathbf{z} = g(\boldsymbol{\theta}_{\|}, X) \in \mathbb{R}^{r \times n}$ the r-dimensional shared representation of X.

The training problem for MTL is typically formulated as the sum of the per-task losses [10, 51, 62]:

$$\min_{\boldsymbol{\theta}} \left[\ \mathcal{L}^{\text{MT}}(\boldsymbol{\theta}) := \textstyle\sum_{i \in \mathcal{T}} \mathcal{L}_i(f(\boldsymbol{\theta}, X, i), Y) \ \right]. \tag{1}$$

Unitary Scalarization The obvious way to minimize the multi-task training objective in equation (1) is to rely on a standard gradient-based algorithm. While, for simplicity, we focus on standard gradient descent rather than mini-batch stochastic gradient descent, the notation can be adapted by replacing the dataset size n by the mini-batch size b. Equation (1) corresponds to a linear scalarization with unitary weights under a multi-objective interpretation of MTL; hence, we call the direct application of gradient descent on equation (1) *unitary scalarization*. For vanilla gradient descent, this corresponds to taking a step in the opposite direction as the one given by the sum of per-task gradients: $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{MT}} = \sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i$. Per-task gradients are not required, as it suffices to directly compute the gradient of the sum $\mathcal{L}^{\text{MT}}$. Hence, when relying on deep learning frameworks based on reverse-mode differentiation, such as PyTorch [47], the backward pass is performed once per iteration (rather than m times). Furthermore, the memory cost is a factor m less than most SMTOs, which require access to each $\nabla_{\boldsymbol{\theta}} \mathcal{L}_i$. As a consequence, unitary scalarization is simple, fast, and memory efficient. Our experiments demonstrate that, when possibly coupled with single-task regularization such as early stopping, ℓ_2 penalty or dropout layers [53], this simple optimizer is strongly competitive with SMTOs.

MGDA Sener and Koltun [51] point out that equation (1) can be cast as a multi-objective optimization problem with the following objective: $\boldsymbol{\mathcal{L}}^{\text{MT}}(\boldsymbol{\theta}) := [\mathcal{L}_1(\boldsymbol{\theta}), \ldots, \mathcal{L}_m(\boldsymbol{\theta})]^T$. A commonly employed solution concept in multi-objective optimization is Pareto optimality. A point $\boldsymbol{\theta}^*$ is called Pareto-optimal if, for any another point $\boldsymbol{\theta}^\dagger$ such that $\exists i \in \mathcal{T} : \mathcal{L}_i(\boldsymbol{\theta}^\dagger) < \mathcal{L}_i(\boldsymbol{\theta}^*)$, then $\exists j \in \mathcal{T} : \mathcal{L}_j(\boldsymbol{\theta}^\dagger) > \mathcal{L}_j(\boldsymbol{\theta}^*)$. Sener and Koltun [51] rely on Multiple-Gradient Descent Algorithm (MGDA) [13] to reach a Pareto-optimal point for shared parameters $\boldsymbol{\theta}_{\|}$. MGDA takes steps in a direction that decreases the loss of all tasks at once [13, 16], which can be found by solving the following optimization problem:

$$\min_{\mathbf{g}, \epsilon} \left[\epsilon + {}^1\!/_2 \left\| \mathbf{g} \right\|_2^2 \right] \quad \text{s.t.} \ \ \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i^T \mathbf{g} \leq \epsilon \quad \forall \, i \in \mathcal{T}, \tag{2}$$

whose dual takes the following form (corresponding to the formulation from Désidéri [13]):

$$\max_{\boldsymbol{\alpha} \geq 0} -{}^1\!/_2 \left\| \mathbf{g} \right\|_2^2 \quad \text{s.t.} \ \ \sum_i \alpha_i \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i = -\mathbf{g}, \quad \sum_{i \in \mathcal{T}} \alpha_i = 1. \tag{3}$$

In other words, MGDA takes a step in a direction $\mathbf{g}$ given by the negative convex combination of per-task gradients, whose coefficients are given by solving equation (3). In practice, per-task gradients are rescaled before applying MGDA: the original authors' implementation [51] relies on $\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \leftarrow {}^{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}\!/ \left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\|_{\mathcal{L}_i(\boldsymbol{\theta})}$. The convergence of MGDA to a Pareto-optimal point is still guaranteed after normalization [13].

IMTL Impartial Multi-Task Learning (IMTL) [40] is presented as an SMTO that is not biased against any single task. It is composed of two complementary algorithmic blocks: IMTL-L, acting on task losses, and IMTL-G, acting on per-task gradients. IMTL-G follows the intuition that a multi-task optimizer should proceed along a direction $\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i$ that equally represents per-task gradients. This is formulated analytically by requiring that the cosine similarity between $\mathbf{g}$ and each $\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i$ be the same. To prevent the resulting problem from being underdetermined, Liu et al. [40] add the constraint $\sum_{i \in \mathcal{T}} \alpha_i = 1$, resulting in a problem that admits a closed-form solution for $\mathbf{g}$:

$$\mathbf{g}^T \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_1}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_1 \right\|} = \mathbf{g}^T \frac{\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i}{\left\| \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \right\|} \ \ \forall \, i \in \mathcal{T} \setminus \{1\}, \quad \mathbf{g} = -\textstyle\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i, \quad \sum_{i \in \mathcal{T}} \alpha_i = 1. \tag{4}$$

IMTL-L, instead, aims to reweight task losses so that they are all constant over time, and equal to 1. In order to limit oscillations of the scaling factors, the authors propose to learn them jointly

with the network by minimizing a common objective via gradient descent. In particular, given $s_i \in \mathbb{R}\ \forall i \in \mathcal{T}$, Liu et al. [40] derive the following form for the joint minimization problem: $\min_{\mathbf{s},\boldsymbol{\theta}} \left[\sum_i \left(e^{s_i}\mathcal{L}_i(\boldsymbol{\theta}) - s_i\right)\right]$. As proved by Liu et al. [40], IMTL-L only has a rescaling effect on the update direction of IMTL-G. Unlike IMTL-G and the other SMTOs presented in this section, IMTL-L rescaling is designed to affect the updates for task-specific parameters $\boldsymbol{\theta}_\perp$ as well.

PCGrad Let us write $\cos(\mathbf{x}, \mathbf{z})$ for the cosine similarity between vectors $\mathbf{x}$ and $\mathbf{z}$. Yu et al. [62] postulate that multi-task convergence is severely slowed down if the following three conditions (named the *tragic triad*) hold at once: (i) conflicting gradient directions: $\cos(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i, \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j) < 0$ for some $i, j \in \mathcal{T}$; (ii) differing gradient magnitudes: $\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i\right\| \gg \left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j\right\|$ for some $i, j \in \mathcal{T}$; and (iii) the unitary scalarization $\mathcal{L}^{\text{MT}}$ has high curvature along $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{MT}}$. The PCGrad [62] SMTO is presented as a solution to the tragic triad, targeted at the first condition. Consistent with the previous sections, let us denote the update direction by $\mathbf{g}$. Furthermore, let $[\mathbf{x}]_+ := \max(\mathbf{x}, \mathbf{0})$. Given per-task gradients $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i$, PCGrad iteratively projects each task gradient onto the normal plane of all the gradients with which it conflicts:

$$\left[\mathbf{g}_i \leftarrow \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i,\ \ \mathbf{g}_i \leftarrow \mathbf{g}_i + \left[\frac{-\mathbf{g}_i^T \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j\right\|^2}\right]_+ \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j\ \ \forall j \in \mathcal{T} \setminus \{i\}\right] \forall i \in \mathcal{T}, \quad \mathbf{g} = -\sum_{i \in \mathcal{T}} \mathbf{g}_i, \tag{5}$$

where the iterative updates of $\mathbf{g}_i$ with respect to $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j$ are performed in random order.

GradDrop Chen et al. [10] focus on conflicting signs across task gradient entries, arguing that such conflicts lead to gradient "tug-of-wars". The GradDrop SMTO [10], presented as a solution to this problem, proposes to randomly mask per-task gradients $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i$ so as to minimize such conflicts. Specifically, GradDrop computes the "positive sign purity" p_j for the task gradient's j-th entry and then masks the j-th entry of each per-task gradient with probability increasing with p_j, if the entry is negative, or decreasing with p_j, if the entry is positive. Let us write $\mathbf{p} := [p_1, \ldots, p_S]$, where S is the dimensionality of the parameter space (see §3), $\odot$ for the Hadamard product and $\mathbb{1}_{\mathbf{a}}$ for the indicator vector on condition $\mathbf{a}$. Given a vector $\mathbf{u}_i$, uniformly sampled in $[\mathbf{0}, \mathbf{1}]$ at each iteration, GradDrop takes a step in the direction given by:

$$\mathbf{g} = \sum_{i \in \mathcal{T}} \begin{pmatrix} -\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i \odot \mathbb{1}_{\left(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i > 0\right)} \odot \mathbb{1}_{(\mathbf{u}_i > \mathbf{p})} \\ -\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i \odot \mathbb{1}_{\left(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i < 0\right)} \odot \mathbb{1}_{(\mathbf{u}_i < \mathbf{p})} \end{pmatrix}, \text{ with } \ \mathbf{p} = \frac{1}{2}\left(1 + \frac{\sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i}{\sum_{i \in \mathcal{T}} \left|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i\right|}\right). \tag{6}$$

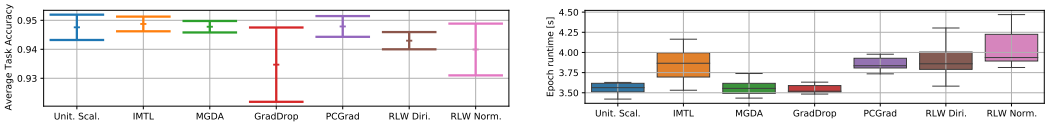
4 Experimental Evaluation

Relying on a unified experimental pipeline, we present an empirical evaluation on common MTL benchmarks of unitary scalarization (§3), of the popular SMTOs presented in §3, and of the recent RLW algorithms [38] due to their similarities with PCGrad and GradDrop (see §5.2). In particular, we benchmark against the two RLW instances that showed the best average performance in the original paper: RLW with weights sampled from a Dirichlet distribution ("RLW Diri."), and RLW with weights sampled from a Normal distribution ("RLW Norm."). Whenever appropriate, we employ "Unit. Scal." as shorthand for unitary scalarization. We first present supervised learning experiments (§4.1), and then evaluate on a popular reinforcement learning benchmark (§4.2).

Our experiments indicate that the performance of unitary scalarization has been consistently underestimated in the literature. By showing the variability between runs and by relying on standard regularization and stabilization techniques from the single-task literature, we demonstrate that *no SMTO consistently outperforms unitary scalarization across the considered settings*. This result holds in spite of the added complexity and computational overhead associated with most SMTOs. Furthermore, in supervised learning, most methods drive the training loss of all tasks in the proximity of the respective global minima. This suggests that the main difficulty of MTL is not associated with the optimization of its training objective, but rather to incorporating adequate regularization (cf. §5).

4.1 Supervised Learning

All the architectures employed in the supervised learning experiments conform to the encoder-decoder structure detailed in §3. Whenever suggested by the original authors for this context, the SMTO



(a) Avg. task test accuracy: mean and 95% CI (10 runs). (b) Box plots for the training time of an epoch (10 runs).

Figure 1: No algorithm outperforms unitary scalarization on the Multi-MNIST dataset.

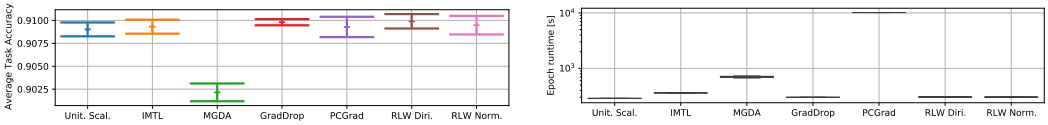
implementations rely on per-task gradients with respect to the last shared activation, $\nabla_{\mathbf{z}}$, rather than on the usually more expensive $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i$. In particular, this is the case for MGDA, IMTL and GradDrop. See appendix B for details concerning each individual algorithm. Surprisingly, several MTL works [10, 38, 40, 62] report validation results, making it easier to overfit. Instead, following standard machine learning practice, we select a model on the validation set, and later report test metrics for all benchmarks. Validation results are also available in appendix D. Appendix C.1 reports dataset descriptions, the computational setup, hyperparameter and tuning details.

4.1.1 Multi-MNIST

We present results on the Multi-MNIST [51] dataset, a simple two-task supervised learning benchmark. We employ a popular architecture from previous work [51, 62] (see appendix C.1), where a single dropout layer [53] (with dropout probability 0.5) is employed in both the encoder and the decoder. ℓ_2 regularization did not improve validation performance and was therefore omitted. Figure 1 reports the average task test accuracy, and the training time per epoch. For each run, the test model was selected as the model with the largest average task validation accuracy across the training epochs. Appendix D presents the results of Figure 1 in tabular form, as well as the average task validation accuracy per epoch. As seen from the overlapping confidence intervals, none of the considered algorithms clearly outperforms the others. However, GradDrop displays higher experimental variability. Furthermore, Figure 6(b) shows that the sums of the task cross-entropy losses is driven nearly to zero by most methods. Finally, Figure 1(b) shows that unitary scalarization also has among the lowest training times.

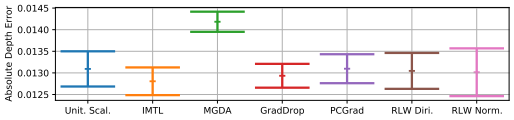
4.1.2 CelebA

We now show results for the CelebA [42] dataset, a challenging 40-task multi-label classification problem. We employ the same architecture as many previous studies [38, 40, 51, 62] (see appendix C.1). We tuned ℓ_2 regularization terms λ for all SMTOs in the following grid: $\lambda \in \{0, 10^{-4}, 10^{-3}\}$. The best validation performance was attained with $\lambda = 10^{-3}$ for unitary scalarization, IMTL and PCGrad, and with $\lambda = 10^{-4}$ for MGDA, GradDrop, and RLW. Validation performance was further stabilized by the addition of several dropout layers (see Figure 5), with dropout probabilities from 0.25 to 0.5. We present an ablation study on the effect of regularization on this experiment in §5.1. Figure 9 (appendix D.2) shows that regularization improves the peak average validation performance for all the considered methods. Analogously to our Multi-MNIST results, Figure 2 plots the distribution of the training time per epoch, and the average test task accuracy. As with Multi-MNIST, the test model for each run was the one with maximal average validation task accuracy across epochs. In other words, if the peak is attained before the last epoch, we perform early stopping: as shown in Figure 7(a) in appendix D this is the case for most methods. Due to the large number of tasks, Figure 2(b) shows relatively large runtime differences across methods. PCGrad is the slowest (roughly 35 times slower than unitary scalarization). In fact, amongst the considered algorithms, it is the only one that computes per-task gradients over the parameters ($\nabla_{\boldsymbol{\theta}}\mathcal{L}_i \; \forall i \in \mathcal{T}$) at each iteration. GradDrop, MGDA and IMTL have overhead factors (compared to unitary scalarization) ranging from roughly 1.05 to

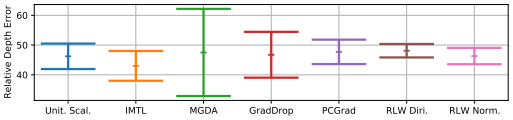


(a) Avg. task test accuracy: mean and 95% CI (3 runs). (b) Box plots for the training time of an epoch (10 runs).

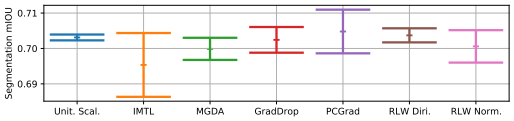
Figure 2: While SMTOs display larger runtimes, none of them outperforms the unitary scalarization on the CelebA dataset.



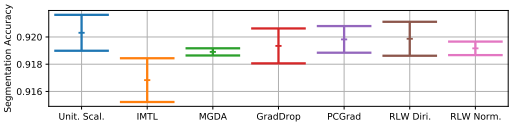
(a) Absolute depth test error: lower is better.



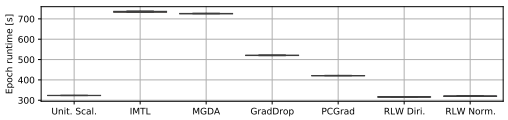
(b) Relative depth test error: lower is better.



(c) Test segmentation mIOU: higher is better.



(d) Test segmentation accuracy: higher is better.



(e) Box plots for the training time of a single epoch (10 repetitions).

Figure 3: On Cityscapes, none of the SMTOs outperforms unitary scalarization, which proves to be the most cost-effective algorithm. Subfigures (a)-(d) report means for three runs, and their 95% CIs.

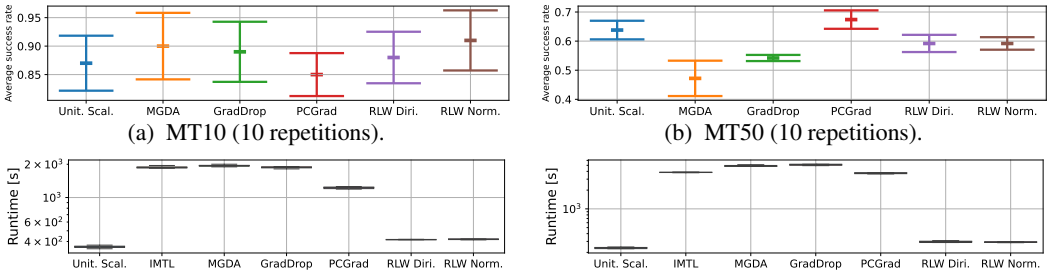
2.4 due to the relatively small size of $\mathbf{z}$ for the employed architecture. The overhead of RLW is negligible: roughly 5%. Nevertheless, due to largely overlapping confidence intervals in Figure 2(a), none of the methods consistently outperforms unitary scalarization. In fact, owing to our adoption of explicit regularization techniques (see §5.1) its average performance is superior to that reported in the literature [40, 51]. As with Multi-MNIST, Figure 8(a) demonstrates that the cross-entropy loss of each task can be driven near to its global optimum by most optimizers.

4.1.3 Cityscapes

In order to complement the multi-task classification experiments for Multi-MNIST and CelebA, we present results for Cityscapes [12], a dataset for semantic understanding of urban street scenes. We rely on a common encoder architecture from the literature [38, 40] (see appendix C.1), a single dropout layer is contained in the task-specific heads [38]. As for CelebA, unitary scalarization, IMTL, and PCGrad benefit from more regularization than the other optimizers: we employ $\lambda = 10^{-5}$ for these three algorithms, as it resulted in better validation performance on the majority of metrics, and $\lambda = 0$ for the remaining methods. Figure 3 shows test results for two metrics per task, and the distribution of the training time per epoch. The test model for each run and each metric was selected as the one associated to the best (maximal or minimal, depending on the metric) validation result across epochs. We therefore perform per-run early stopping for most methods (see Figure 16). As with Multi-MNIST and CelebA, no training algorithm clearly outperforms unitary scalarization (significant overlaps across confidence intervals exist), which is again the least expensive method. In contrast with a popular hypothesis [9, 28, 40], this holds in spite of relatively large loss imbalances. In fact, the loss for the depth task is roughly 10 times smaller than that of the segmentation task: see figures 16(f)-16(g). Nevertheless, both losses are rapidly driven towards their respective global minima. Unlike CelebA (see Figure 2(b)), IMTL, MGDA and GradDrop are significantly slower than unitary scalarization (factors from 1.6 to 2.3), due to the relatively (compared to the parameter space) large size of $\mathbf{z}$ in the employed architecture. PCGrad, instead, appears to be less expensive (30% more than the baseline), demonstrating the benefits of working on $\nabla_{\boldsymbol{\theta}} \mathcal{L}_i$ on this model.

4.2 Reinforcement Learning

For RL experiments, we use Meta-World [61] and the Soft Actor-Critic [19] implementation from [52]. Unlike §4.1, the employed network architecture (see appendix C.1) is fully shared across tasks. Therefore, all SMTO implementations for these experiments rely on per-task gradients with respect to network parameters $\nabla_{\boldsymbol{\theta}} \mathcal{L}_i$ (see §5). Among the SMTOs we consider, PCGrad is the only one developed with the RL setting in mind. For fairness and completeness, we add all the other SMTOs from the supervised learning experiments, and are the first to test these optimizers in the RL setting. To stabilize learning, we increase the replay buffer size, a well known technique in single-task RL, add actor l_2 regularization, and modify the reward normalization employed by Sodhani et al. [52]. The unitary scalarization performance reported by Yu et al. [62] is considerably lower than that of Sodhani et al. [52], which we believe is due to the lack of reward normalization in the former.



(a) MT10 (10 repetitions). (b) MT50 (10 repetitions).

(c) MT10 (10 repetitions). (d) MT50 (10 repetitions).

Figure 4: On Metaworld, none of the SMTOs significantly outperforms Unit. Scal., which is the least expensive method. Subfigures (a)-(b) report mean and 95% CI for the best (over the updates) average success rate. Subfigures (c)-(d) show box plots for the training time of 10,000 updates.

Sodhani et al. [52] keep a moving average of rewards in the environment, with a hyperparameter controlling the speed of the moving average. As we show in Figure 15, the learning algorithm is sensitive to that hyperparameter. Moreover, such normalization might make similar transitions have drastically different rewards stored in the replay buffer. To alleviate these issues, we store the raw rewards in the buffer, and normalize only when a mini-batch is sampled.

Figure 4 reports the best average success rate across the updates and the runtime for 10,000 updates. In addition to these summary statistics, reported for consistency with §4.1, the learning curves are shown in appendix E. Our MT10 (10 tasks) results in Figure 4(a) show that by stabilizing the baseline using standard RL techniques, unitary scalarization performs on par with other SMTOs, mirroring our findings in §4.1. This is in contrast with the previous literature, which reported that PCGrad outperforms unitary scalarization [52, 62]. Figure 4(b) presents results on MT50 (50 tasks): similarly to MT10, none of the SMTOs significantly outperforms unitary scalarization, with PCGrad's average being slightly above unitary scalarization. We speculate that the stochastic loss rescaling performed by PCGrad (see Proposition 3) reduces the differences in task return scales, and expect that methods like PopArt [57] would have a similar effect without requiring access to per-task gradients. While we did not tune hyperparameters for MT50 (we employed those found for MT10), it would be much easier to do that for unitary scalarization due to its lower runtime. In fact, Figure 4(d) shows that a single unitary scalarization run takes roughly 15 hours, whereas PCGrad, MGDA and GradDrop require more than a week. Similarly to MT10, actor regularization pushes the average performance of unitary scalarization higher (see in appendix E.2). Overall, as in the supervised learning setting, unitary scalarization performs comparably to SMTOs despite being simpler and less demanding in both memory and compute. IMTL was unstable on this RL benchmark and all of the runs crashed due to numerical overflow. We hence omit IMTL results from the main body of the paper and show its results in Figure 12 in appendix E, which also describes a possible explanation. We hypothesize that the instability of IMTL is due to lack of bounds on scaling coefficients. See appendix C.2 for hyperparameter settings and ablation studies.

5 Regularization in Specialized Multi-Task Optimizers

The empirical results presented in §4 motivate the need to carefully analyze existing SMTOs. We make an initial attempt in this direction by viewing their effects through the lens of regularization. Let us define a regularizer as a technique to reduce overfitting [14]. We first show that the SMTOs considered in §4 empirically act as regularizers via an ablation study (§5.1). We then take a closer look at their behavior, presenting technical results that support their alternative interpretation as regularizers (§5.2). Unless otherwise stated, we assume that MTL methods apply only to $\boldsymbol{\theta}_{||}$ and that standard gradient-based updates are employed for tasks-specific parameters $\boldsymbol{\theta}_{\perp}$. We furthermore adopt the following shorthands: $\mathcal{L}_i(\boldsymbol{\theta})$ for $\mathcal{L}_i(f(\boldsymbol{\theta}, X, i), Y)$, and $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i$ for $\nabla_{\boldsymbol{\theta}}\mathcal{L}_i(f(\boldsymbol{\theta}, X, i), Y)$.

5.1 Ablation Study

We repeat the experiment from §4.1.2 and remove explicit regularization: no dropout layers are added to the encoder-decoder architecture, and $\lambda = 0$ for all optimizers. In addition, we examine the behavior of two different ℓ_2-regularized instances of unitary scalarization: $\lambda = 10^{-4}$ for "Unit.

Scal. ℓ_2", $\lambda = 2 \times 10^{-3}$ for "Unit. Scal. ℓ_2+". Figure 5 shows that SMTOs behave similarly to an ℓ_2-penalized unitary scalarization. Importantly, SMTOs delay overfitting, requiring less early stopping compared to unitary scalarization to obtain comparable performance. In other words, early stopping is sufficient for unitary scalarization to perform on par with SMTOs. Finally, overfitting is further reduced by "Unit. Scal. Reg.", which plots the regularized unitary scalarization from §4.1.2, with dropout layers and a weight decay of $\lambda = 10^{-3}$. Further results are presented in appendix D.2.

5.2 Technical Results

All the methods considered in §5.1 regularize more than unitary scalarization. While RLW was shown to reduce overfitting by the original authors [38, theorem 2], we now provide a collection of novel and existing technical results that potentially explain the regularizing behavior of each of the other algorithms, complementing the presentation from §3. In particular, we show that MGDA, IMTL and PCGrad have a larger convergence set than unitary scalarization, reducing the chances to land on sharp local minima [14]. Furthermore, GradDrop and PCGrad introduce significant stochasticity, which is often linked to the same effect [29, 32]. We hope these observations will steer further research.

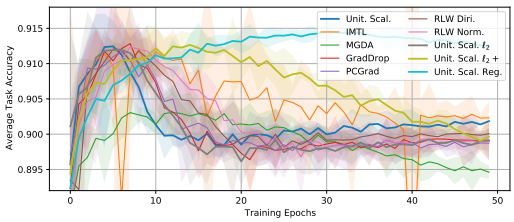


Figure 5: Mean and 95% CI (3 runs) avg. task validation accuracy over epochs on CelebA. SMTOs postpone the onset of overfitting, mirroring the effect of ℓ_2 regularization on unitary scalarization.

MGDA Let us denote the convex hull of a set $\mathcal{A}$ by $\text{Conv}(\mathcal{A})$. We now recall a well-known property of MGDA [13] and relate it to the behavior of unitary scalarization.

Proposition 1. *The* MGDA SMTO *[51] converges to a superset of the convergence points of unitary scalarization. More specifically, it converges to any point* $\boldsymbol{\theta}_{\|}^*$ *such that:* $\mathbf{0} \in \textit{Conv}(\{\nabla_{\boldsymbol{\theta}_{\|}^*} \mathcal{L}_i \mid i \in \mathcal{T}\})$.

See appendix B.1 for a simple proof. As a consequence of Proposition 1, MGDA does not necessarily reach a stationary point for $\mathcal{L}^{\text{MT}}$ (that is, a point for which $\sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i = \mathbf{0}$) or for any of the losses $\mathcal{L}_i$ ($\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i = \mathbf{0}$). For example, any point $\boldsymbol{\theta}_{\|}$ for which two per-task gradients point in opposite directions is Pareto optimal. On account of the well-known [14] relationship between under-optimizing (e.g., early stopping [6, 37]) and overfitting, proposition 1 supports the interpretation of MGDA as a regularizer for equation (1). Empirical evidence that MGDA under-optimizes is provided in Figure 8(a) and complemented by Figure 5, which shows over-regularization.

IMTL We now show that aggregating per-task gradients so that their cosine similarity is the same (equation (4)) yields a constrained steepest-descent algorithm (Proposition 2). This view on the update step of IMTL leads to a novel analysis of its convergence points (corollary 1). Proofs can be found in appendix B.2. We will denote by $\text{Aff}(\mathcal{A})$ the affine hull of a set $\mathcal{A}$.

Proposition 2. *IMTL by Liu et al. [40] updates* $\boldsymbol{\theta}_{\|}$ *by taking a step in the* steepest descent direction whose *cosine similarity with per-task gradients is the same across tasks.*

Corollary 1. IMTL *by Liu et al. [40] converges to a superset of the Pareto-optimal points for* $\boldsymbol{\theta}_{\|}$ *(and hence of the convergence points of the unitary scalarization). More specifically, it converges to any point* $\boldsymbol{\theta}_{\|}^*$ *such that:* $\mathbf{0} \in \textit{Aff}\left(\left\{\nabla_{\boldsymbol{\theta}_{\|}^*} \mathcal{L}_i / \left\|\nabla_{\boldsymbol{\theta}_{\|}^*} \mathcal{L}_i\right\| \mid i \in \mathcal{T}\right\}\right)$.

As seen for MGDA, corollary 1 implies that, even if the employed model f has the capacity to reach the minimal loss on $\mathcal{L}^{\text{MT}}$, IMTL may stop before reaching a stationary point. Recalling the relationship between under-optimizing and overfitting [14], this supports the interpretation of IMTL as a regularizer for equation (1). This is empirically shown in Figures 5 and 8(a). In particular, unitary scalarization reaches the same average performance of IMTL but requires earlier stopping.

PCGrad We provide an alternative characterization of the PCGrad update rule, highlighting its stochasticity in the context of its interpretation as loss rescaling [38, 40]. See appendix B.3 for a proof.

Proposition 3. *PCGrad is equivalent to a dynamic, and possibly stochastic, loss rescaling for* $\boldsymbol{\theta}_{\|}$. *At each iteration, per-task gradients are rescaled as follows:*

$$\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i \leftarrow \left(1 + \textstyle\sum_{j \in \mathcal{T} \setminus \{i\}} d_{ji}\right) \nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i, \; d_{ji} \in \left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_j\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\|}} \mathcal{L}_i\right\|}\right].$$

Furthermore, if $|\mathcal{T}| > 2$, d_{ji} is a random variable, and the above range contains its support.

The results from proposition 3 can be easily extended to GradVac [59], which generalizes PCGrad's projection onto the normal vector to arbitrary target cosine similarities between per-task gradients. When $|\mathcal{T}| > 2$, PCGrad corresponds to a stochastic loss re-weighting. As such, PCGrad bears many similarities with Random Loss Weighting (RLW) [38]. RLW proposes to sample scalarization weights from standard probability distributions at each iteration, and proves that this leads the better generalization [38, theorem 2]. Indeed, it is well-known that adding noise to stochastic gradient estimations leads the optimization towards flatter minima, and that such minima may reduce overfitting [29, 32]. In line with the main technical results by Yu et al. [62], we now restrict our focus to two-task problems, which allow for an easy description of PCGrad's convergence points. The result is largely based on [62, theorem 1]: we relax some of the assumptions and provide a proof in appendix B.3.

Corollary 2. *If $|\mathcal{T}| = 2$, PCGrad will stop at any point where $\cos(\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_2) = -1$. Furthermore, if $\mathcal{L}_1$ and $\mathcal{L}_2$ are differentiable, and $\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}^{MT}$ is L-Lipschitz with $L > 0$, PCGrad with step size $t < \frac{1}{L}$ converges to a superset of the convergence points of the unitary scalarization.*

Corollary 2 implies that, when $|\mathcal{T}| = 2$, PCGrad may induce early stopping on the unitary scalarization as MGDA and IMTL. In particular, if $\cos(\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_2) = -1$, then $\mathbf{0} \in \text{Conv}(\{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_2\})$ (see proposition 1). We believe that PCGrad's stochasticity and enlarged convergence set potentially explain its regularizing effect.

GradDrop While the motivation behind GradDrop is to avoid entry-wise gradient conflicts across tasks, the main property of the method is to drive the optimization towards "joint minima": points that are stationary for all the individual tasks at once [10, proposition 1]. In other words: $\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i = \mathbf{0} \; \forall \, i \in \mathcal{T}$. While this property is desirable, we show that it holds beyond GradDrop, and independently of the gradient directions. Under strong assumptions on the model capacity, the above property would trivially hold for unitary scalarization (proposition 5, appendix B.3). Proposition 4 shows that it holds for a simple randomized version of unitary scalarization, which we name Random Grad Drop (RGD).

Proposition 4. *Let $\mathcal{L}^{RGD}(\boldsymbol{\theta}_\parallel) := \sum_{i \in \mathcal{T}} u_i \mathcal{L}_i(\boldsymbol{\theta}_\parallel)$, where $u_i \sim$ Bernoulli(p) $\forall i \in \mathcal{T}$ and $p \in (0, 1]$. The gradient $\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}^{RGD}$ is always zero if and only if $\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i = \mathbf{0} \; \forall i \in \mathcal{T}$. In other words, the result from [10, proposition 1] can be obtained without any information on the sign of per-task gradients.*

Proposition 4 (see appendix B.4 for a simple proof) shows that an inexpensive sign-independent stochastic scalarization shares GradDrop's main reported property. $\mathcal{L}^{\text{RGD}}$ can be directly cast an instance of RLW, and hence as a regularization method [29, 32]. Furthermore, Figure 11 in appendix D.3 shows that the empirical results of GradDrop on CelebA [42] are closely matched by a sign-agnostic gradient masking, partly undermining the conflicting gradients assumption. We believe that the above results, along with the authors' original experiments showing that GradDrop delays overfitting on CelebA [10, figure 3], suggest that GradDrop behaves as a regularizer.

6 Conclusions

This paper made two main contributions. First, we evaluated popular SMTOs using a single experimental pipeline, including previously unpublished results of MGDA, IMTL, RLW, and GradDrop in the RL setting. Surprisingly, our evaluation showed that none of the SMTOs consistently outperform unitary scalarization, the simplest and least expensive method. Second, in order to explain our surprising results, we postulate that SMTOs act as regularizers and present an analysis that supports our hypothesis. We believe our work calls for further reevaluation of progress in developing principled and efficient MTL algorithms.

We conclude by addressing the limitations of our work. While we covered a wide range of popular benchmarks, we do not exclude the existence of settings where unitary scalarization underperforms: discovering them is an interesting direction for future work. Furthermore, our experimental results were obtained via grid searches under limited compute resources: some of the methods might benefit from further fine-tuning. Nevertheless, we remark that fine-tuning will be easier for unitary scalarization due to its shorter runtimes. Finally, we presented the regularization hypothesis only as a partial explanation of our results: we hope it will steer further analysis and consequently improve the understanding of MTL.

References

[1] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, 2019.

[2] B. Bakker and T. Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 2003.

[3] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Velickovic. Combinatorial optimization and reasoning with graph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 2021.

[4] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[5] R. Caruana. *Multitask learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1997.

[6] R. Caruana, S. Lawrence, and L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Neural Information Processing Systems*, 2000.

[7] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision*, 2018.

[8] S. Chen, Y. Zhang, and Q. Yang. Multi-task learning in natural language processing: An overview. *CoRR*, 2021.

[9] Z. Chen, V. Badrinarayanana, C.-Y. Lee, and A. Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, 2018.

[10] Z. Chen, J. Ngiam, Y. Huang, T. Luong, H. Kretzschmar, Y. Chai, and D. Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. In *Neural Information Processing Systems*, 2020.

[11] R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, 2008.

[12] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Conference on Computer Vision and Pattern Recognition*, 2016.

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

[14] T. Dietterich. Overfitting and undercomputing in machine learning. *ACM Computing Surveys*, page 326–327, sep 1995.

[15] T. Evgeniou and M. Pontil. Regularized multi–task learning. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[16] J. Fliege and B. F. Svaiter. Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 2000.

[17] M. Guo, A. Haque, D.-A. Huang, S. Yeung, and L. Fei-Fei. Dynamic task prioritization for multitask learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[18] P. Guo, C.-Y. Lee, and D. Ulbricht. Learning to branch for multi-task learning. 2020.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*, 2016.

[21] T. Heskes. Empirical bayes for learning to learn. In *International Conference on Machine Learning*, 2000.

[22] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. Multi-task deep reinforcement learning with popart. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3796–3803. AAAI Press, 2019.

[23] T. M. Hospedales, A. Antoniou, P. Micaelli, and A. J. Storkey. Meta-learning in neural networks: A survey. *CoRR*, 2020.

[24] W. Huang, I. Mordatch, and D. Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, 2020.

[25] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[26] A. Javaloy and I. Valera. Rotograd: Gradient homogenization in multitask learning. In *International Conference on Learning Representations*, 2022.

[27] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *CoRR*, 2021.

[28] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[29] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations*, 2017.

[30] K. Khetarpal, M. Riemer, I. Rish, and D. Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.

[31] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *International Conference on Learning Representations*, 2015.

[32] B. Kleinberg, Y. Li, and Y. Yuan. An alternative view: When does SGD escape local minima? In *International Conference on Machine Learning*, 2018.

[33] I. Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[34] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro. Can q-learning with graph networks learn a generalizable branching heuristic for a SAT solver? In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[35] V. Kurin, M. Igl, T. Rocktäschel, W. Boehmer, and S. Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

[37] M. Li, M. Soltanolkotabi, and S. Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2020.

[38] B. Lin, F. Ye, and Y. Zhang. A closer look at loss weighting in multi-task learning. In *arXiv preprint arXiv:2111.10603*, 2022.

[39] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 2021.

[40] L. Liu, Y. Li, Z. Kuang, J.-H. Xue, Y. Chen, W. Yang, Q. Liao, and W. Zhang. Towards impartial multi-task learning. In *International Conference on Learning Representations*, 2021.

[41] S. Liu, E. Johns, and A. J. Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019.

[42] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[43] S. Ma, R. Bassily, and M. Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, 2018.

[44] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016.

[45] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.*, 2020.

[46] E. Parisotto, L. J. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*. 2019.

[48] A. A. Rusu, S. G. Colmenarejo, Ç. Gülçehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[49] S. Sabour, N. Frosst, and G. E. Hinton. *Dynamic Routing between Capsules*. 2017.

[50] M. L. Seltzer and J. Droppo. Multi-task learning in deep neural networks for improved phoneme recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, 2013.

[51] O. Sener and V. Koltun. Multi-task learning as multi-objective optimization. In *Neural Information Processing Systems*, 2018.

[52] S. Sodhani, A. Zhang, and J. Pineau. Multi-task reinforcement learning with context-based representations. In M. Meila and T. Zhang, editors, *International Conference on Machine Learning*, 2021.

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[54] Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4496–4506, 2017.

[55] Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. In *Neural Information Processing Systems*, 2017.

[56] W.-C. Tseng. Weichengtseng/pytorch-pcgrad, 2020. URL `https://github.com/WeiChengTseng/Pytorch-PCGrad.git`.

[57] H. P. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. Learning values across many orders of magnitude. *Advances in Neural Information Processing Systems*, 29:4287–4295, 2016.

[58] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[59] Z. Wang, Y. Tsvetkov, O. Firat, and Y. Cao. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. In *International Conference on Learning Representations*, 2021.

[60] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *Computer Vision and Pattern Recognition*, 2017.

[61] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *3rd Annual Conference on Robot Learning*, 2019.

[62] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient surgery for multi-task learning. 2020.

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]
   (b) Did you describe the limitations of your work? [Yes] see §6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] due to space constraints, we provide a discussion in appendix A.
   (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]
   (b) Did you include complete proofs of all theoretical results? [Yes] we provide full proofs in the Appendix, and refer to them in the main body of the paper.
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 provide the code and the instructions in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(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] see appendix C.1.

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]
   (b) Did you mention the license of the assets? [Yes] appendix C.3 describes licenses of all benchmarks and implementations we used for our work.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] we include the code and the instructions on how to replicate the experiments into the supplemental material.
   (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]
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]
   (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 Societal Impact

Due to the object of its study, our work does not have a direct societal impact. However, as any machine learning paper, it can potentially negatively effect the society through automation and loss of jobs. While it is hard to anticipate any particular risk, as any technology, if not regulated properly, it might lead to growing social and economic inequality.

On the positive side, our work might have a positive environmental impact since it advocates for simpler and more economical methods which will reduce energy consumption in data centers. Finally, simpler methods are usually easier to understand, which is beneficial in terms of explainability, an important factor for real-life applications.

B Supplement to the Overview of Multi-Task Optimizers

This section presents the proofs and the technical results omitted from section 5, along with a description of the use of per-task gradients with respect to the last shared activation for encoder-decoder architectures (usually less expensive than per-task gradients with respect to shared parameters).

B.1 MGDA

Proposition 1. *The* MGDA SMTO *[51] converges to a superset of the convergence points of unitary scalarization. More specifically, it converges to any point* $\boldsymbol{\theta}_{\|}^*$ *such that:* $\mathbf{0} \in$ *Conv*$(\{\nabla_{\boldsymbol{\theta}_{\|}^*}\mathcal{L}_i \mid i \in \mathcal{T}\})$.

Proof. As shown by Désidéri [13], equation (3) is a simplex-constrained norm-minimization problem. In other words, the argument of the minimum is the projection of $\mathbf{0}$ onto the feasible set. Therefore:

$$\mathbf{g} = \mathbf{0} \iff \mathbf{0} \in \text{Conv}(\{\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i \mid i \in \mathcal{T}\}).$$

It then suffices to point out that $\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0} \iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|}\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0} \Rightarrow \mathbf{0} \in \text{Conv}(\{\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i \mid i \in \mathcal{T}\})$ to conclude the proof. □

Due to the cost of computing per-task gradients, Sener and Koltun [51] propose MGDA-UB, which replaces the gradients wrt the parameters $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$ with the gradients wrt the shared activation $\nabla_{\mathbf{z}}\mathcal{L}_i$ in the computation of the coefficients of $\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$. This yields an upper bound on the objective of equation (3), thus restricting the set of points the algorithm convergences to. Rather than directly relying on $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$, $\mathbf{g}$ can then be obtained by computing the gradient of $\sum_{i\in\mathcal{T}} \alpha_i \mathcal{L}_i$ via reverse-mode differentiation, hence saving memory and compute.

Corollary 3. *The MGDA-UB* SMTO *by Sener and Koltun [51] converges to any point such that:* $\mathbf{0} \in$ *Conv*$(\{\nabla_{\mathbf{z}}\mathcal{L}_i \mid i \in \mathcal{T}\})$. *Furthermore, if* $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}$ *is non-singular, it converges to a superset of the convergence points of the unitary scalarization.*

Proof. The first part of the proof proceeds as the proof of proposition 1, noting that the MGDA-UB update is associated to the following problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\|\mathbf{g}\|_2^2 \\ \text{s.t.} \quad & \sum_i \alpha_i \nabla_{\mathbf{z}}\mathcal{L}_i = -\mathbf{g}, \quad \sum_{i\in\mathcal{T}} \alpha_i = 1, \\ & \alpha_i \geq 0 \qquad \forall\, i \in \mathcal{T}. \end{aligned}$$

In order to show that a stationary point of the unitary scalarization satisfies $\mathbf{0} \in \text{Conv}(\{\nabla_{\mathbf{z}^*}\mathcal{L}_i \mid i \in \mathcal{T}\})$, we will assume $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}$ is non-singular, as done by Sener and Koltun [51, theorem 1]. Then, relying

on the chain rule, the result follows from:

$$\begin{aligned}
\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i = \mathbf{0} &\iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|} \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i = \mathbf{0} \\
&\iff \sum_{i\in\mathcal{T}} \frac{\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_\parallel}}{|\mathcal{T}|} \nabla_{\mathbf{z}} \mathcal{L}_i = \mathbf{0} \\
&\iff \left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_\parallel}\right)^{-1} \frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_\parallel} \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|} \nabla_{\mathbf{z}} \mathcal{L}_i = \mathbf{0} \\
&\iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|} \nabla_{\mathbf{z}} \mathcal{L}_i = \mathbf{0} \\
&\Rightarrow \mathbf{0} \in \mathrm{Conv}(\{\nabla_{\mathbf{z}} \mathcal{L}_i \mid i \in \mathcal{T}\})
\end{aligned}$$

□

B.2 IMTL

Proposition 2. *IMTL by Liu et al. [40] updates $\boldsymbol{\theta}_\parallel$ by taking a step in the* steepest descent direction whose *cosine similarity with per-task gradients is the same across tasks.*

Proof. First, equation (4) solves the linear system in $\boldsymbol{\alpha} := [\alpha_1, \ldots, \alpha_m]$ given by:

$$\begin{aligned}
&\mathbf{g}^T \left(\frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1\right\|} - \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|} \right) = \mathbf{0} \qquad \forall\, i \in \mathcal{T} \setminus \{1\}, \\
&\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i, \quad \sum_{i\in\mathcal{T}} \alpha_i = 1,
\end{aligned}$$

which corresponds to finding a point of $\mathcal{A}' := \mathrm{Aff}(\{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i | \; i \in \mathcal{T}\})$ which is orthogonal to $\mathcal{A} := \mathrm{Aff}\left(\left\{ \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|} \mid i \in \mathcal{T} \right\}\right)$. To see this, it suffices to point out that any point orthogonal to $\mathcal{A}$ is also orthogonal to the vector subspace spanned by differences of vectors belonging to $\mathcal{A}$. As this subspace has $m-1$ dimensions, any vector orthogonal to $\left(\frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_1\right\|} - \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|} \right)$ for each $i \in \mathcal{T} \setminus \{1\}$ is orthogonal to the entire subspace.

Second, consider the problem of finding a point in $\mathcal{A}$ that is orthogonal to the affine subspace $\mathcal{A}$. In other words, we seek the projection of $\mathbf{0}$ onto $\mathcal{A}$. Recalling the definition of $\mathcal{A}$, we can write:

$$\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2} \left\|\mathbf{g}'\right\|_2^2 \\
\text{s.t.} \quad & \sum_i \alpha_i \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|} = -\mathbf{g}', \quad \sum_i \alpha_i = 1.
\end{aligned} \tag{7}$$

The solution of equation (7) is always collinear to the solution of equation (4). In fact, if a vector $\mathbf{g} \in \mathcal{A}'$ is orthogonal to the affine subspace $\mathcal{A}$, then $\gamma\mathbf{g} = \left(\gamma \sum_i \left(\alpha_i \left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|\right) \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|}\right)$ is orthogonal to $\mathcal{A}$ as well, and $\gamma = \frac{1}{\sum_i \left(\alpha_i \left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|\right)} \implies \gamma\mathbf{g} \in \mathcal{A}$.

Finally, equation (7) differs from equation (3) in two aspects: $\boldsymbol{\alpha}$ is not constrained to be non-negative (hence the convex hull is replaced by the affine hull), and the task vectors are normalized. Therefore, equation (7) is the dual of:

$$\begin{aligned}
\min_{\mathbf{g},\epsilon} \quad & \epsilon + \frac{1}{2} \left\|\mathbf{g}\right\|_2^2 \\
\text{s.t.} \quad & \frac{\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i^T}{\left\|\nabla_{\boldsymbol{\theta}_\parallel} \mathcal{L}_i\right\|} \mathbf{g} = \epsilon \qquad \forall\, i \in [\![1, m]\!].
\end{aligned} \tag{8}$$

The proposition then follows by comparing equation (8) with equation (2), and recalling that IMTL-L only adds a scaling factor to the chosen update direction. □

Corollary 1. IMTL *by Liu et al. [40] converges to a superset of the Pareto-optimal points for* $\boldsymbol{\theta}_{\|}$ *(and hence of the convergence points of the unitary scalarization). More specifically, it converges to any point* $\boldsymbol{\theta}_{\|}^*$ *such that:* $\mathbf{0} \in \text{Aff}\left(\left\{\nabla_{\boldsymbol{\theta}_{\|}^*}\mathcal{L}_i / \left\|\nabla_{\boldsymbol{\theta}_{\|}^*}\mathcal{L}_i\right\| \mid i \in \mathcal{T}\right\}\right)$.

Proof. Inspecting equation (8), which yields a collinear point to the IMTL update, reveals that IMTL might converge to non Pareto-stationary points: due to the restrictive equality constraints, the minimizer of equation (8) might be $\mathbf{0}$ even if a descent direction exists. Furthermore, its dual, equation (7), implies that:

$$\begin{aligned}\mathbf{g} = \mathbf{0} &\iff \mathbf{0} \in \text{Aff}\left(\left\{\frac{\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i}{\left\|\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i\right\|} \mid i \in \mathcal{T}\right\}\right)\\ &\iff \mathbf{0} \in \text{Aff}\left(\left\{\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i \mid i \in \mathcal{T}\right\}\right),\end{aligned}$$

which, noting that $\text{Conv}(\mathcal{A}) \subseteq \text{Aff}(\mathcal{A})$ for any $\mathcal{A}$, concludes the proof. □

Similarly to MGDA-UB, Liu et al. [40] advocate using $\nabla_{\mathbf{z}}\mathcal{L}_i$ in place of $\nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$ while solving equation (4), typically reducing the cost of computing the coefficients of $\mathbf{g} = -\sum_i \alpha_i \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i$.

Corollary 4. *When employing the approximation of problem (4) that relies on* $\nabla_{\mathbf{z}}\mathcal{L}_i$*, IMTL by Liu et al. [40] converges to* $\mathbf{0} \in \text{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right)$*. If* $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}$ *is non-singular, this is a superset of of the convergence points of the unitary scalarization.*

Proof. Following the proof of proposition 2, the following problem yields a collinear point to the $\nabla_{\mathbf{z}}\mathcal{L}_i$-approximate IMTL update:

$$\begin{aligned}\max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\left\|\mathbf{g}'\right\|_2^2\\ \text{s.t.} \quad & \sum_i \alpha_i \frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} = -\mathbf{g}', \quad \sum_i \alpha_i = 1.\end{aligned}$$

Therefore:

$$\mathbf{g} = \mathbf{0} \iff \mathbf{0} \in \text{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right).$$

Finally, assuming $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}$ is non-singular, we can replicate the procedure in the proof of corollary 3 to get:

$$\begin{aligned}&\sum_{i\in\mathcal{T}} \nabla_{\boldsymbol{\theta}_{\|}}\mathcal{L}_i = \mathbf{0} \iff \sum_{i\in\mathcal{T}} \frac{1}{|\mathcal{T}|}\nabla_{\mathbf{z}}\mathcal{L}_i = \mathbf{0}\\ &\iff \sum_{i\in\mathcal{T}} \frac{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|}{|\mathcal{T}|}\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} = \mathbf{0}\\ &\iff \left(\frac{|\mathcal{T}|}{\sum_{i\in\mathcal{T}}\left(\|\nabla_{\mathbf{z}}\mathcal{L}_i\|\right)}\right)\sum_{i\in\mathcal{T}} \frac{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|}{|\mathcal{T}|}\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} = \mathbf{0}\\ &\Rightarrow \mathbf{0} \in \text{Conv}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right)\\ &\Rightarrow \mathbf{0} \in \text{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right),\end{aligned}$$

which shows that $\text{Aff}\left(\left\{\frac{\nabla_{\mathbf{z}}\mathcal{L}_i}{\|\nabla_{\mathbf{z}}\mathcal{L}_i\|} \mid i \in \mathcal{T}\right\}\right)$ contains the convergence points of the unitary scalarization. □

B.3 PCGrad

Proposition 3. *PCGrad is equivalent to a dynamic, and possibly stochastic, loss rescaling for $\boldsymbol{\theta}_{\parallel}$. At each iteration, per-task gradients are rescaled as follows:*

$$\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i \leftarrow \left(1 + \sum_{j \in \mathcal{T} \setminus \{i\}} d_{ji}\right) \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i, \; d_{ji} \in \left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i\right\|}\right].$$

Furthermore, if $|\mathcal{T}| > 2$, d_{ji} is a random variable, and the above range contains its support.

Proof. We start by pointing out that:

$$\begin{aligned}
\left[\frac{-\mathbf{g}_i^T \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|^2}\right]_+ &= \left[\frac{-\mathbf{g}_i^T \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|}\right]_+ \frac{1}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|} \\
&= \left[-\cos(\mathbf{g}_i, \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j) \left\|\mathbf{g}_i\right\|\right]_+ \frac{1}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|} \\
&\in \left[0, \frac{\left\|\mathbf{g}_i\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|}\right].
\end{aligned}$$

As $\mathbf{g}_i$ is obtained by iterative projections of $\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i$ onto the normals of $\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j \; \forall j \in \mathcal{T} \setminus \{i\}$, and the norm of a vector can only decrease or remain unvaried after projections, we can write the coefficient of the last $\mathbf{g}_i$ update as:

$$d_{ij} := \left[\frac{-\mathbf{g}_i^T \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j(\mathbf{x})}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|^2}\right]_+ \in \left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|}\right].$$

Furthermore, if $|\mathcal{T}| > 2$ the contraction factor $\frac{\|\mathbf{g}_i\|}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i\right\|}$ for the norm of g_i depends on the ordering of the projections, which is stochastic by design [62]. Therefore, d_{ij} a random variable whose support is contained in $\left[0, \frac{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_j\right\|}\right]$. Finally, exploiting the definition of d_{ij}, we can re-write equation (5) as:

$$\begin{aligned}
-\mathbf{g} &= \sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i + \sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{T} \setminus \{i\}} d_{ij} \\
&= \sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i + \sum_{j \in \mathcal{T}} \sum_{i \in \mathcal{T} \setminus \{j\}} d_{ji} \\
&= \sum_{i \in \mathcal{T}} \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_i \left(1 + \sum_{j \in \mathcal{T} \setminus \{j\}} d_{ji}\right),
\end{aligned}$$

from which the result trivially follows. □

Corollary 2. *If $|\mathcal{T}| = 2$, PCGrad will stop at any point where $\cos(\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_1, \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_2) = -1$. Furthermore, if $\mathcal{L}_1$ and $\mathcal{L}_2$ are differentiable, and $\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}^{MT}$ is L-Lipschitz with $L > 0$, PCGrad with step size $t < \frac{1}{L}$ converges to a superset of the convergence points of the unitary scalarization.*

Proof. Let us start from the first statement, which does not require any assumption on the loss landscape. From proposition 3, we get:

$$\begin{aligned}
-\mathbf{g} &= \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_1 \left(1 + d_{21}\right) + \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_2 \left(1 + d_{12}\right) \\
&= \left(1 + \left[\frac{-\cos(\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_1, \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_2) \left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_2\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_1\right\|}\right]_+\right) \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_1 \\
&+ \left(1 + \left[\frac{-\cos(\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_1, \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_2) \left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_1\right\|}{\left\|\nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_2\right\|}\right]_+\right) \nabla_{\boldsymbol{\theta}_{\parallel}} \mathcal{L}_2,
\end{aligned}$$

which shows that, in case of conflicting gradient directions, gradient norms are rebalanced proportionally to the angle between them. For $\cos(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2) = -1$, the above evaluates to:

$$-\mathbf{g} = \left(\frac{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1\right\| + \left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2\right\|}{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1\right\|}\right)\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1 + \left(\frac{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1\right\| + \left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2\right\|}{\left\|\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2\right\|}\right)\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2.$$

The first part of the result then follows by pointing out that, if $\cos(\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1, \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2) = -1$, then $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_1 = -\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_2$, and hence $\mathbf{g} = \mathbf{0}$. We remark that a similar proof appears in [62, theorem 1 and proposition 1]. However, our derivation relaxes the author's assumptions on $\mathcal{L}^{\text{MT}}$ and is therefore applicable to the training of neural networks.

Finally, given the assumptions on differentiability and smoothness, we need to prove that PCGrad converges to the stationary points of the unitary scalarization: this directly follows from [62, proposition 1]. □

B.4 GradDrop

Proposition 5. *Let us assume, as often demonstrated in the single-task case [1, 43], that the multi-task network has the capacity to interpolate the data on all tasks at once:* $\min_{\boldsymbol{\theta}} \mathcal{L}^{MT} = \sum_{i\in\mathcal{T}} \min_{\boldsymbol{\theta}} \mathcal{L}_i$*, and that its training by gradient descent attains such global minimum. Then, if* $\inf_{\boldsymbol{\theta}} \mathcal{L}_i > -\infty \; \forall \; i \in \mathcal{T}$*, unitary scalarization converges to a joint minimum.*

Proof. It suffices to point out that if $\mathcal{L}^{\text{MT}}(\boldsymbol{\theta}^*) = \sum_{i\in\mathcal{T}} \min_{\boldsymbol{\theta}} \mathcal{L}_i$, then the globally optimal loss is attained for all tasks. In other words $\mathcal{L}_i(\boldsymbol{\theta}^*) = \min_{\boldsymbol{\theta}} \mathcal{L}_i \; \forall i \in \mathcal{T}$, and hence $\nabla_{\boldsymbol{\theta}^*}\mathcal{L}_i = \mathbf{0} \; \forall \; i \in \mathcal{T}$ (joint minimum). Furthermore, running gradient descent on $\min_{\boldsymbol{\theta}} \mathcal{L}^{\text{MT}}$ corresponds to the unitary scalarization (§3), which concludes the proof. □

Proposition 4. *Let* $\mathcal{L}^{RGD}(\boldsymbol{\theta}_\parallel) := \sum_{i\in\mathcal{T}} u_i \mathcal{L}_i(\boldsymbol{\theta}_\parallel)$*, where* $u_i \sim$ *Bernoulli*$(p) \; \forall i \in \mathcal{T}$ *and* $p \in (0, 1]$*. The gradient* $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{RGD}$ *is always zero if and only if* $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i = \mathbf{0} \; \forall i \in \mathcal{T}$*. In other words, the result from [10, proposition 1] can be obtained without any information on the sign of per-task gradients.*

Proposition 4 can be proved by adapting the proof from Chen et al. [10, proposition 1]: it suffices to replace $f(\mathcal{P})$ with the Bernoulli parameter p, which is non-negative by definition. In our opinion, this seriously undermines the conflicting gradient hypothesis that motivated GradDrop. For the reader's convenience, we now provide a straightforward and self-contained proof.

Proof. Let us start from the statement on $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{RGD}}$. If $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i = \mathbf{0} \; \forall i \in \mathcal{T}$, then $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{RGD}} = \mathbf{0}$ with probability one. On the other hand, if $\exists j : \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j \neq \mathbf{0}$, then:

$$\begin{aligned}\mathbb{P}\left[\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{RGD}} \neq \mathbf{0}\right] &\geq \mathbb{P}\left[\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{RGD}} = \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j\right] \\ &= p(1-p)^{m-1} > 0,\end{aligned}$$

where the first inequality comes from the fact that $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{RGD}} = \nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_j$ is only one of the many instances of a non-null $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{RGD}}$. □

Let $\text{sign}(\mathbf{x})$ stand for the element-wise sign operator applied on $\mathbf{x}$. On encoder-decoder architectures, similarly to MGDA and IMTL (see appendices B.1 and B.2), the authors do not apply GradDrop on $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}_i$, but rather on a the usually less expensive $\nabla_{\mathbf{z}}\mathcal{L}_i$. In more detail, they compute the GradDrop sign purity scores $\mathbf{p}$ from equation (6) on $\sum_{i=1}^{n}\left(\text{sign}(\mathbf{z}) \odot \nabla_{\mathbf{z}}\mathcal{L}_i\right)[i] \in \mathbb{R}^r$, and then apply equation (6) on the $\nabla_{\mathbf{z}}\mathcal{L}_i$ gradients, yielding a vector $\mathbf{g}_z \in \mathbb{R}^{n\times r}$. Then, relying on reverse-mode differentiation, the update direction in the space of the parameters $\boldsymbol{\theta}_\parallel$ is obtained via a Jacobian-vector product: $\mathbf{g} = -\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_\parallel}\right)^T \mathbf{g}_z$. Such a computation replaces the similar $\nabla_{\boldsymbol{\theta}_\parallel}\mathcal{L}^{\text{MT}} = \left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_\parallel}\right)^T \nabla_{\mathbf{z}}\mathcal{L}^{\text{MT}}$ from the unitary scalarization.

C Experimental Setting, Reproducibility

We now present details concerning the experimental settings from §4, including details on the employed open-source software, dataset information, hardware specifications, and hyper-parameters.

C.1 Supervised Learning

All the experiments were run under Ubuntu 18.04 LTS, on a single GPU per run (using two 8-GPU machines in total). Timing experiments were all run on Nvidia GeForce GTX 1080 Ti GPUs, with an Intel Xeon E5-2650 CPU. The remaining experiments were run on either Nvidia GeForce RTX 2080 Ti GPUs or Nvidia GeForce GTX 1080 Ti GPUs, respectively using an Intel Xeon Gold 6230 CPU or an Intel Xeon E5-2650 CPU.

C.1.1 MultiMNIST

Multi-MNIST, originally introduced by Sabour et al. [49] and as modified by Sener and Koltun [51], is a simple two-task supervised learning benchmark dataset constructed by uniformly sampling MNIST [36] images, and placing one in the top-left corner, the other in the bottom-right corner. Each of the two overlaid images corresponds to a 10-class classification task. Using the above procedure, we generate the Multi-MNIST training set from the first 50000 MNIST training images, the validation set from the last 10000 training images, and the test set from the original MNIST test set. For consistency with the experimental setup of Sener and Koltun [51], we employ a modified encoder-decoder version of the LeNet architecture [36]. Specifically, the last layer is omitted from the encoder, and two fully-connected layers are employed as task-specific predictive heads. The cross-entropy loss is used for both tasks. All methods are trained for 100 epochs using Adam [31] in the stochastic gradient setting, with an initial learning rate of $\eta = 10^{-2}$ (tuned in $\eta \in \{10^{-3}, 10^{-2}, 10^{-1}\}$ and yielding the best validation results for all considered algorithms), exponentially decayed by 0.95 after each epoch, and a mini-batch size of 256.

C.1.2 CelebA

The CelebA [42] dataset consists of $200,000$ headshots (with standard training, validation and test splits) associated with the presence or absence of 40 attributes. In the MTL literature, is commonly treated as a 40-task classification problem, each task being a binary classification problem for an attribute. As commonly done in previous work [40, 51, 62], we employ an encoder-decoder architecture where the encoder is a ResNet-18 [20] (without the final layer), and the per-task decoders are linear classifiers. The cross-entropy loss is used for all tasks. The training is performed from scratch for 50 epochs using Adam, with a mini-batch size of 128 and a per-epoch exponential decay factor of 0.95. As common on this network-dataset combination [10, 38], the initial learning rate is $\eta = 10^{-3}$ for all methods except for MGDA and IMTL, for which $\eta = 5 \times 10^{-4}$ yielded a better validation performance. As done by the respective authors, for PCGrad, RLW and GradDrop we use the same learning rate as the unitary scalarization [10, 38, 62].

C.1.3 Cityscapes

We rely on the version of the dataset pre-processed by Liu et al. [41], which consists of $2,975$ training and 500 test images and presents two tasks: semantic segmentation on 7 classes, and depth estimation. We further split the original training set into a validation set of 595 images, employed to tune hyper-parameters, and a training set of 2380 images. Consistently with recent work [38], we rely on a dilated ResNet-50 architecture pre-trained on ImageNet [60] for the encoder, and on the Atrous Spatial Pyramid Pooling [7] as decoders. Cross-entropy loss is employed on the semantic segmentation task, whereas the ℓ_1 loss is used for the depth estimation. The training is performed by using Adam with a mini-batch size of 32 for 100 epochs, with an initial step size $\eta = 5 \times 10^{-4}$ resulting in the best validation performance for all algorithms, exponentially decayed by 0.95 at each epoch.

C.2 Reinforcement Learning

Similarly to the supervised learning experiments, we ran all the experiments under Ubuntu 18.04 LTS using one GPU per run (using six 8-GPU machines in total). Timing experiments were all run using

NVIDIA GeForce RTX 2080 Ti GPUs, with an Intel Xeon Gold 6230 CPU. The main bulk of the remaining experiments was run on Nvidia GeForce RTX 2080 Ti GPUs with either Intel Xeon Gold 6230 or Intel Xeon Silver 4216. We utilised NVIDIA GeForce RTX 3080 GPUs with Intel Xeon Gold 6230 CPUs for a small fraction of experiments.

We use Meta-World's MT10/MT50 for our experiment. The benchmark consists of ten/fifty tasks in which a simulated robot manipulator has to perform various actions, e.g., pressing a button, opening a door, or pushing the block. We use Sodhani et al. [52] for most of the hyperparameters and list them in Table 1. We use bold font where we use a hyperparameter different from Sodhani et al. [52]. Similarly to Sodhani et al. [52], we use the V1 version of Metaworld for our experiments[1]. Sodhani et al. [52] use a shared entropy loss weight α for PCGrad and separate α for unitary scalarization[2]. In our experiments, use shared α for all of the methods for fairness. Since it is a single number (rather than a vector), we used unitary scalarization to update α for all SMTOs apart from PCGrad which was already implemented in [52].

We use the same network architecture as in Sodhani et al. [52], i.e. a three-layered feedforward fully-connected network with 400 hidden units per layer for both, the actor and the critic. The actor is shared across all tasks as well as the critic.

To normalize rewards, we keep track of first and second moments in the buffer and normalise the rewards by their standard deviation: $r'_i = {}^{r_i}/_{\hat{\sigma}_i}$, where $\hat{\sigma}_i$ is the sample standard deviation of the rewards for environment i.

Sodhani et al. [52] average the gradient for unitary scalarisation and pcgrad, whereas our SMTO implementations sum the gradients, i.e. effectively using larger learning rates (apart from MGDA that assures that all the aggregation weights sum to 1). We tried reducing the learning rate for SMTOs that sum (RLW Norm., RLW Diri., and GradDrop) both for MT10 and MT50, but it worked worse for these methods and we kept the default learning rate for them as well. We had to use a smaller learning rate for IMTL, because with the default one it crashed at the beginning of training due to numerical overflow. Smaller learning rate did not prevent it from crashing, but this happened much later.

We tried 10^6, 2×10^6, and 4×10^6 for the replay buffer size with the last being superior in terms of stability. Additionally, for l_2 actor regularization, we tried 0.0001 and 0.0003 with the latter being slightly superior for the baseline. We tried the same options for other SMTOs, and picked the best option for each of the method. For MGDA, no regularisation works best, most likely due to a strong regularization effect of the method itself, which is mirrored by our supervised learning results. PCGrad and Graddrop work best with the regularization coefficient of 0.0001. Both RLW variants use the same coefficient as the baseline (0.0003).

For MT50, we took the best MT10 hyperparameters, and we believe one could obtain even better results for unitary scalarisation since it is much faster to tune compared to other SMTOs (e.g. 15 hours for unitary scalarisation vs 9 days for PCGrad).

C.3 Software Acknowledgments and Licenses

Our codebase is built upon several prior works: [51], [41], [38] and [52]: all of them were released under a MIT license. We also acknowledge Tseng [56], upon which we built some of our code. Multi-MNIST is based on MNIST dataset that is released under Creative Commons Attribution-Share Alike 3.0 license. The code for generating Multi-MNIST dataset was taken from Sener and Koltun [51] released under MIT license. CelebA dataset has a custom license allowing non-commercial research purposes. More details can be found on the project website:`http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`. Cityscapes also has a custom license allowing non-commercial research purposes. The full text of the license can be found on the project website:`https://www.cityscapes-dataset.com/license/`. Metaworld, used for RL experiments is released under MIT license.

D Supplementary Supervised Learning Experiments

[1] `https://github.com/rlworkgroup/metaworld.git@af8417bfc82a3e249b4b02156518d775f29eb289`

[2] `https://mtrl.readthedocs.io/en/latest/pages/tutorials/baseline.html`

Table 1: Hyperparameters of the RL experiments. Hyperparameters different from Sodhani et al. [52] are in bold.

Hyperparameter	**Value**
All methods	
– training steps	2,000,000
– batch size	1280
– **Replay buffer size**	**4,000,000**
– actor learning rate	0.0003
– critic learning rate	0.0003
– entropy α learning rate	0.0003
– shared entropy α	True
– runs	10
– discounting γ	0.99
Unit. Scal.	
– **actor l_2 coeff.**	**0.0003**
PCGrad	
– **actor l_2 coeff.**	**0.0001**
RLW Norm.	
– normal mean	0
– normal std	1
– actor l_2 coeff.	0.0003
RLW Diri.	
– α	1
– actor l_2 coeff.	0.0003
GradDrop	
– k	1
– p	0.5
– actor l_2 coeff.	0.0001
MGDA	
– gradient normalization	L_2
– actor l_2 coeff.	0.0
IMTL	
– actor learning rate	0.00003
– critic learning rate	0.00003
– entropy α learning rate	0.00003
– actor l_2 coeff.	0.0

This section presents supervised learning results omitted from §4.1. In particular, we show additional plots for the experiments of §4.1, then present an analysis of the regularising effect of SMTOs in the absence of single-task regularization (§D.2), and conclude with an ablation study on GradDrop's dependency on the sign of per-task gradients (§D.3).

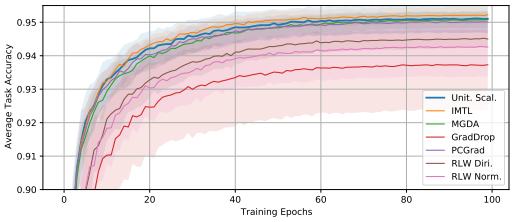
D.1 Addendum

This section complements the plots presented in §4.1. In particular, we show the test and runtime results in table form, along with the behavior of the validation metrics and of the training loss over the training epochs. Plots for Multi-MNIST, CelebA, and Cityscapes are reported in Figures 6, 7 and 16, respectively. The Multi-MNIST plots show that all optimizers are relatively stable and drive each task's loss towards its global minimum of 0. The behavior of the CelebA training loss demonstrates heavier regularization (compare with the unregularized plot in Figure 8(a)). Except IMTL and MGDA, for which the tuned values of the weight decay prevent overfitting, the other optimizers display very similar validation and training curves, and start overfitting around epoch 30. Considering that most SMTOs required less regularization (see §4.1.2), the results are consistent with our interpretation of SMTOs as regularizers in §5. The Cityscapes plots display a certain instability across training epochs, as demonstrated by the various peaks and valleys in the metrics. Nevertheless, in spite of a factor 10 difference in scale, both training losses are rapidly driven towards 0 by most optimizers.

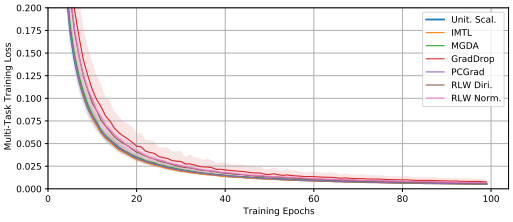
D.2 Unregularized Experiments

Figures 5, 8(a) and 8(b) respectively report the average task validation accuracy, the multi-task training loss, and the multi-task validation loss at each training epoch. The regularizing effect of SMTOs compared to unitary scalarization is shown by: (i) the delay of the onset of overfitting on the validation data in figure 5, (ii) the reduction of the convergence rate on the training loss in figure 8(a) (compare with figure 7(b)), and (iii) the fact that validation and training losses remain positively correlated for larger numbers of epochs. In fact, the behavior of both the training and validation loss for the SMTOs closely parallels that of ℓ_2-regularized unitary scalarization, with differing degrees of regularization. We further note that unregularized IMTL displays a certain instability (compare with the regularized version in figure 7(a)).

The addition of dropout layers further reduces overfitting, improves stability (reduced confidence intervals) and pushes the average validation curve upwards, motivating its use on all optimizers for the experiments of §4.1.2. Nevertheless, confidence intervals in Figure 5 still overlap due to the instability of the unregularized unitary scalarization. Figure 10 provides a more detailed comparison over 20 repetitions, confirming that the combined use of dropout layers and ℓ_2 regularization improves average performance and reduces the empirical variance for unitary scalarization. Furthermore, Figure 9



(a) Mean (and 95% CI) average task validation accuracy per training epoch.

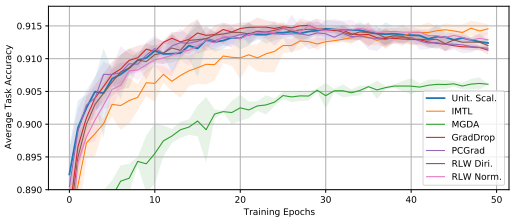


(b) Mean (and 95% CI) training multi-task loss $\mathcal{L}^{\text{MT}}$ per epoch.

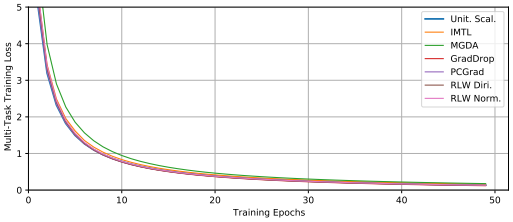
MTO	Average Task Accuracy	Epoch Runtime [s]
Unit. Scal.	9.476e-01 $\pm$ 4.368e-03	[3.510e+00, 3.617e+00]
IMTL	9.487e-01 $\pm$ 2.533e-03	[3.695e+00, 3.996e+00]
MGDA	9.478e-01 $\pm$ 1.977e-03	[3.491e+00, 3.617e+00]
GradDrop	9.347e-01 $\pm$ 1.282e-02	[3.508e+00, 3.589e+00]
PCGrad	9.479e-01 $\pm$ 3.578e-03	[3.807e+00, 3.928e+00]
RLW Diri.	9.430e-01 $\pm$ 2.973e-03	[3.790e+00, 4.005e+00]
RLW Norm.	9.399e-01 $\pm$ 8.929e-03	[3.894e+00, 4.225e+00]

(c) Mean and 95% CI of the avg. task test accuracy across runs, and interquartile range for the training time per epoch.

Figure 6: Additional figures for the comparison of various SMTOs with the unitary scalarization on the MultiMNIST dataset [51].



(a) Mean (and 95% CI) average task validation accuracy per training epoch.



(b) Mean (and 95% CI) training multi-task loss $\mathcal{L}^{\text{MT}}$ per epoch.

MTO	Average Task Accuracy	Epoch Runtime [s]
Unit. Scal.	9.090e-01 $\pm$ 7.568e-04	[2.869e+02, 2.878e+02]
IMTL	9.093e-01 $\pm$ 7.631e-04	[3.600e+02, 3.621e+02]
MGDA	9.022e-01 $\pm$ 9.687e-04	[6.859e+02, 7.194e+02]
GradDrop	9.098e-01 $\pm$ 3.383e-04	[3.001e+02, 3.008e+02]
PCGrad	9.093e-01 $\pm$ 1.108e-03	[1.015e+04, 1.016e+04]
RLW Diri.	9.099e-01 $\pm$ 7.845e-04	[3.040e+02, 3.054e+02]
RLW Norm.	9.095e-01 $\pm$ 1.012e-03	[3.028e+02, 3.037e+02]

(c) Mean and 95% CI of the avg. task test accuracy across runs, and interquartile range for the training time per epoch.

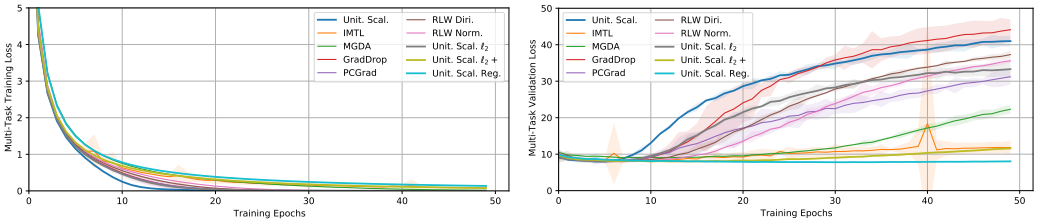
Figure 7: Additional figures for the comparison of various SMTOs with the unitary scalarization on the CelebA [42] dataset.

shows that regularization improves the peak average validation performance for all algorithms, demonstrating the need of tuning λ also for SMTOs. We conclude by pointing out that even without regularization, when carefully tuned, the maximal performance over epochs of unitary scalarization is comparable to SMTOs in Figure 5.

D.3 Sign-Agnostic GradDrop

We will now present an ablation study on GradDrop, investigating the effect of the sign of per-task gradients on the SMTO's performance. Specifically, we compare the performance of GradDrop with a sign-agnostic version of its stochastic gradient masking (which we refer to as "Sign-Agnostic GradDrop"), whose update direction is defined as follows:

$$\mathbf{g} = -\left(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}_{\|}}\right)^T \left(\sum_{i \in \mathcal{T}} \mathbf{u}_i \odot \nabla_{\mathbf{z}} \mathcal{L}_i\right),$$



(a) Mean and 95% CI (3 runs) multi-task training loss per epoch.

(b) Mean and 95% CI (3 runs) multi-task validation loss per training epoch.

Figure 8: Additional figures for the unregularized comparison of various SMTOs with the unitary scalarization on CelebA. SMTOs provide varying degrees of regularization.

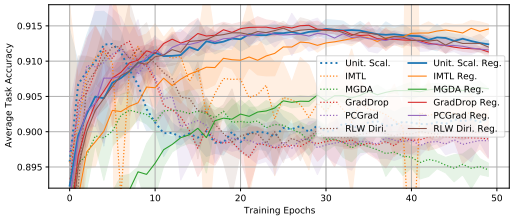


Figure 9: Effect of regularization (dropout layers and weight decay) on the average task validation accuracy for all considered optimizers on the CelebA dataset: regularization improves the average performance of all algorithms.

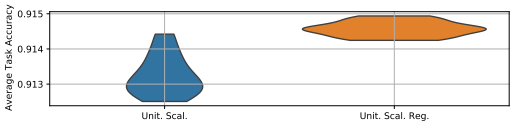


Figure 10: Effect of regularization (dropout layers and weight decay) on unitary scalarization on the CelebA dataset: violin plots (20 runs) for the best avg. task validation accuracy over epochs. The width at a given value represents the proportion of runs yielding that result. Regularization improves the average performance while decreasing its variability.

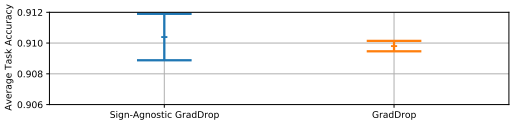
where $\mathbf{u}_i, \nabla_{\mathbf{z}} \mathcal{L}_i \in \mathbb{R}^{n \times r}$ and, for all $i \in \mathcal{T}$, $\mathbf{u}_i$ is i.i.d. according to $\mathbf{u}_i[j,k] \sim \text{Bernoulli}(p)\ \forall j \in \{1, \dots, n\}, k \in \{1, \dots, r\}$. Differently from a similar study carried out by Chen et al. [10], we tuned the hyper-parameter of the sign-agnostic masking in the following range: $p \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$.

The experimental setup complies with the one described in appendix C.1. Figure 11, plotting test and validation results for the CelebA dataset [42], shows that the performance of Sign-Agnostic GradDrop closely matches the original algorithm. Therefore, sign conflicts across per-task gradients do not seem to play a significant role in GradDrop's performance.

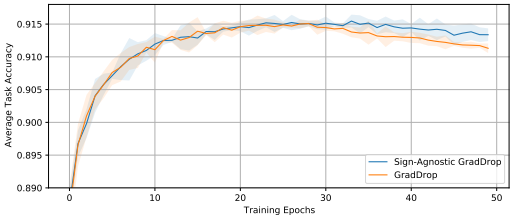
E Supplementary Reinforcement Learning Experiments

E.1 Addendum

This section presents additional plots for the RL experiments in §4.2. Specifically, Figure 12 re-plots Figure 4(a) and 4(b) with the omitted IMTL results, while Figure 13 shows the learning curves omitted from §4.2. As pointed out in §4.2, none of the IMTL runs successfully terminated due to numerical instability. Indeed, Liu et al. [40] show that, in supervised settings, coefficients do not fluctuate much across epochs [40, Figure 4, appendix B] and never become negative. By contrast, up to 50% of the scaling coefficients α are negative in our experiments, thus reversing subtask gradient directions. MGDA, which constrains the weights, is more stable and is comparable to unitary scalarization. In order to avoid incomplete curves and unfair calculations of the mean, Figure 13 plots the highest

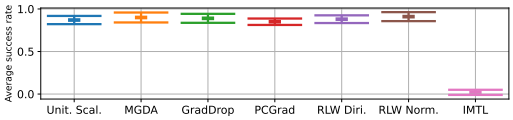


(a) Mean and 95% CI (3 runs) avg. task test accuracy.

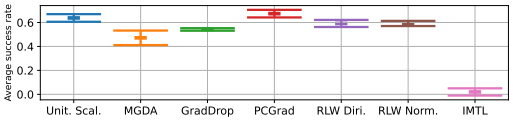


(b) Mean and 95% CI (3 runs) avg. task validation accuracy per training epoch.

Figure 11: Comparison of GradDrop [10] with sign-agnostic masking of the shared-representation gradients on the CelebA dataset [42]. No statistically relevant difference between the two methods can be observed for the majority of the epochs.

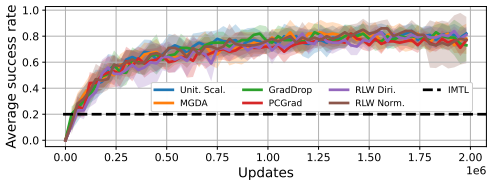


(a) MT10 (10 runs per method).

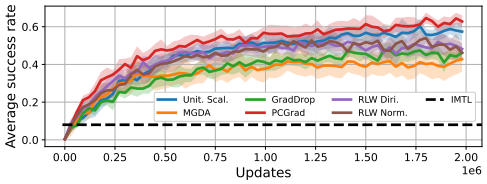


(b) MT50 (10 runs per method).

Figure 12: Mean and 95% CI for the best avg. success rate on Metaworld. None of the SMTOs significantly outperforms unitary scalarization.

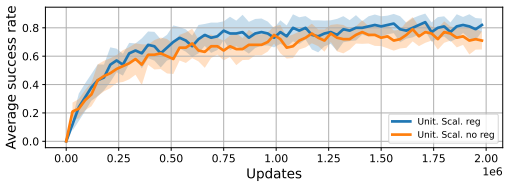


(a) MT10 (10 points per method).

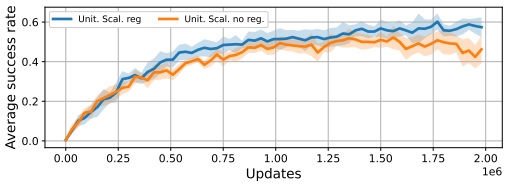


(b) MT50 (10 points per method).

Figure 13: Mean and 95% CI for the avg. success rate on Metaworld. None of the SMTOs significantly outperforms unitary scalarization.



(a) MT10 average performance (10 runs) and 95% CI.



(b) MT50 average performance (5 runs) and 95% CI.

Figure 14: For both MT10 and MT50, actor l_2 regularization pushes the average higher for unitary scalarization.

value ever achieved by *any* seed as a dashed horizontal line. The IMTL results in Figure 12, instead, report the best average success rate of each seed until its termination.

E.2 Ablation studies

Figure 17 presents our ablations for MT10 experiments. Due to computational constraints, we ran ablations on the unitary scalarization and PCGrad since these are the two methods previously tested in the RL setting.

Figure 14 shows ablation studies on the effect of regularization on MT10 and MT50. In spite of CI overlaps, actor l_2 regularization pushes the average higher on both benchmarks, motivating our use of regularization for the experiments in §4.2. Furthermore, the gap between the averages tends to widen with the number of updates on MT50, suggesting improved stabilization.

E.3 Sensitivity to Reward Normalization

Figure 15 shows that multitask agent performance is highly sensitive to the reward normalization moving average hyperparameter[3] motivating our buffer normalization in Section 4.2.

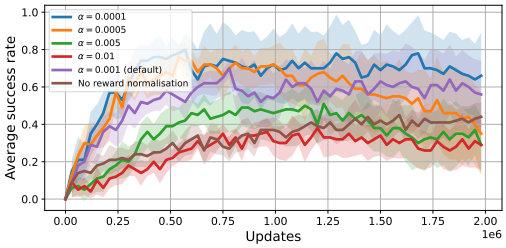
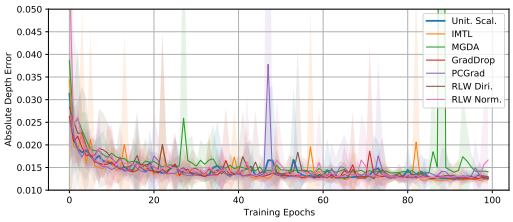


Figure 15: The learning outcomes of a Multitask SAC agent vary considerably depending on the reward normalisation hyperparameter. Each of the curves represents and average of 10 runs with shaded 95% confidence interval.

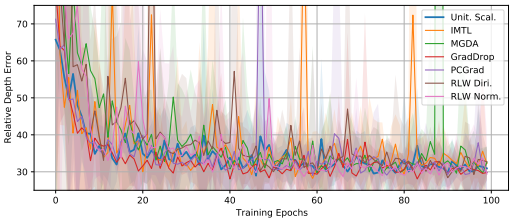
[3] https://github.com/facebookresearch/mtenv/blob/4a6d9d6fdfb321f1b51f890ef36b5161359e972d/mtenv/envs/metaworld/wrappers/normalized_env.py#L69

(a) Mean and 95% CI of the test metrics across runs, and interquartile range for the training time per epoch.

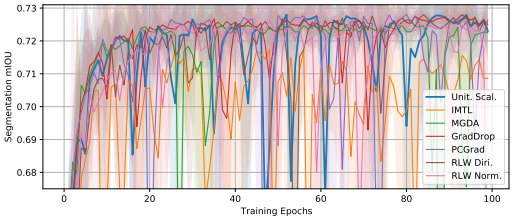
MTO	Absolute Depth Error	Relative Depth Error	Segmentation Accuracy	Segmentation mIOU	Epoch Runtime [s]
Unit. Scal.	1.301e-02 $\pm$ 2.342e-04	4.761e+01 $\pm$ 5.148e+00	9.196e-01 $\pm$ 2.913e-04	7.012e-01 $\pm$ 6.001e-04	[3.228e+02, 3.241e+02]
IMTL	1.281e-02 $\pm$ 7.521e-04	4.389e+01 $\pm$ 6.984e-01	9.164e-01 $\pm$ 2.828e-03	6.967e-01 $\pm$ 4.785e-03	[7.329e+02, 7.373e+02]
MGDA	1.418e-02 $\pm$ 2.331e-04	4.750e+01 $\pm$ 1.466e+01	9.189e-01 $\pm$ 2.636e-04	6.999e-01 $\pm$ 3.124e-03	[7.251e+02, 7.269e+02]
GradDrop	1.293e-02 $\pm$ 2.757e-04	4.674e+01 $\pm$ 7.709e+00	9.193e-01 $\pm$ 1.282e-03	7.024e-01 $\pm$ 3.628e-03	[5.196e+02, 5.215e+02]
PCGrad	1.294e-02 $\pm$ 2.284e-04	4.380e+01 $\pm$ 5.165e+00	9.198e-01 $\pm$ 9.119e-04	7.025e-01 $\pm$ 6.531e-04	[4.202e+02, 4.212e+02]
RLW Diri.	1.305e-02 $\pm$ 4.155e-04	4.810e+01 $\pm$ 2.259e+00	9.199e-01 $\pm$ 1.247e-03	7.037e-01 $\pm$ 1.989e-03	[3.161e+02, 3.164e+02]
RLW Norm.	1.301e-02 $\pm$ 5.528e-04	4.630e+01 $\pm$ 2.751e+00	9.192e-01 $\pm$ 4.962e-04	7.006e-01 $\pm$ 4.580e-03	[3.194e+02, 3.210e+02]



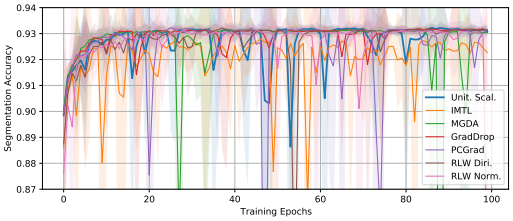
(b) Mean (and 95% CI) absolute depth validation error per training epoch.



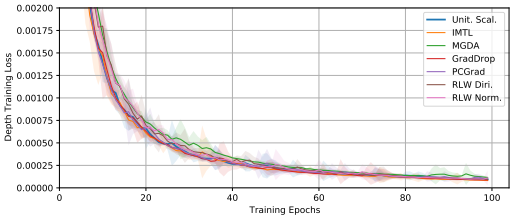
(c) Mean (and 95% CI) relative depth validation error per training epoch.



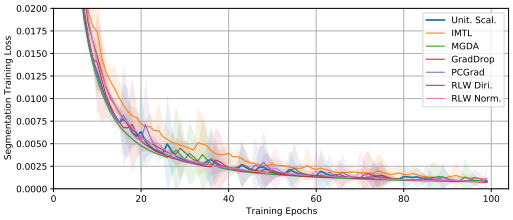
(d) Mean (and 95% CI) validation segmentation mIOU per training epoch.



(e) Mean (and 95% CI) validation segmentation accuracy per training epoch.



(f) Mean (and 95% CI) training depth loss per epoch.



(g) Mean (and 95% CI) training segmentation loss per epoch.

Figure 16: Additional figures for the comparison of SMTOs with the unitary scalarization on the Cityscapes [12] dataset.

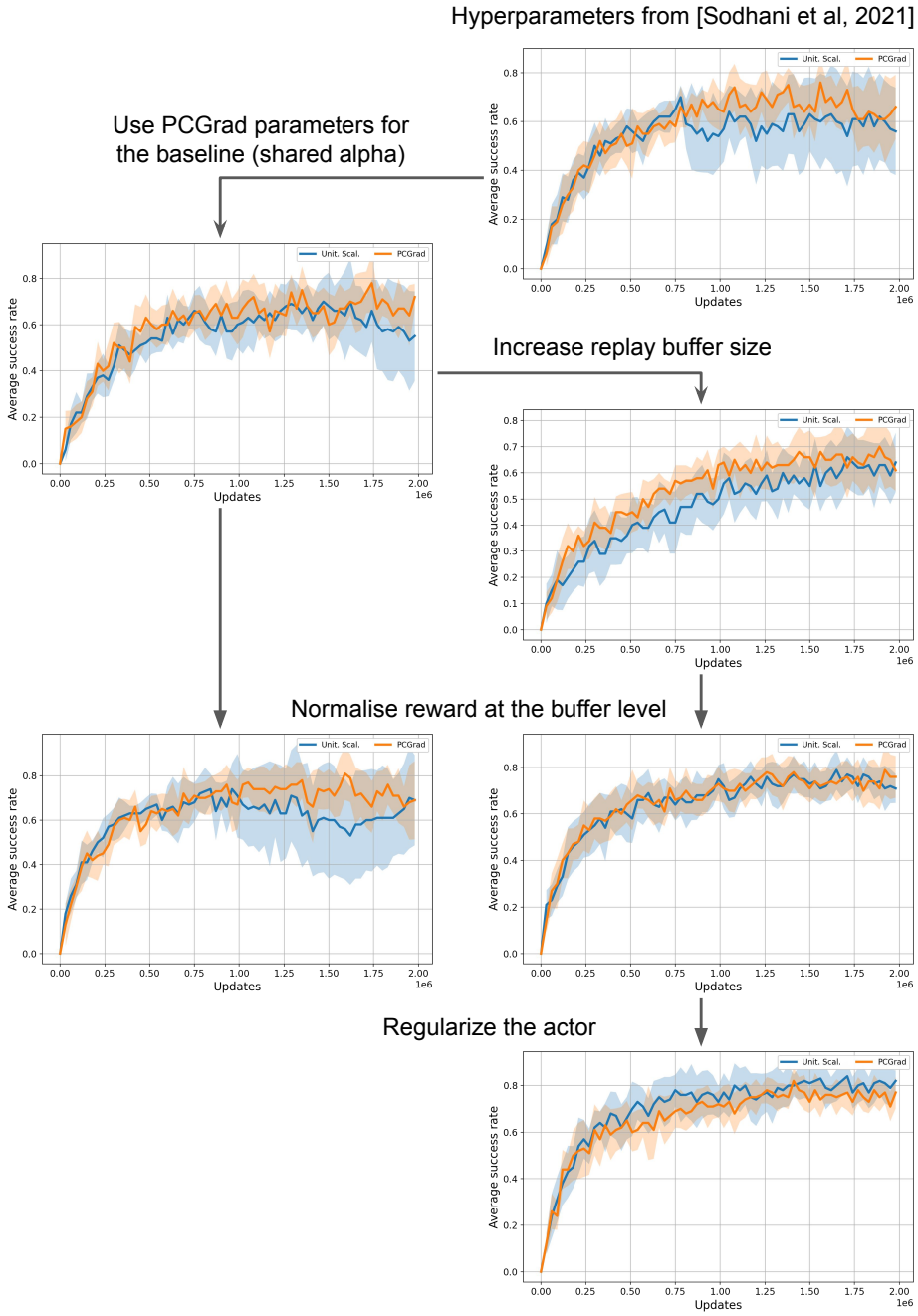


Figure 17: Metaworld's MT10 ablation experiments.