# GRADIENT SURGERY FOR MULTI-TASK LEARNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

While deep learning and deep reinforcement learning systems have demonstrated impressive results in domains such as image classification, game playing, and robotic control, data efficiency remains a major challenge, particularly as these algorithms learn individual tasks from scratch. Multi-task learning has emerged as a promising approach for sharing structure across multiple tasks to enable more efficient learning. However, the multi-task setting presents a number of optimization challenges, making it difficult to realize large efficiency gains compared to learning tasks independently. The reasons why multi-task learning is so challenging compared to single task learning are not fully understood. Motivated by the insight that gradient interference causes optimization challenges, we develop a simple and general approach for avoiding interference between gradients from different tasks, by altering the gradients through a technique we refer to as "gradient surgery". We propose a form of gradient surgery that projects the gradient of a task onto the normal plane of the gradient of any other task that has a *conflicting* gradient. On a series of challenging multi-task supervised and multi-task reinforcement learning problems, we find that this approach leads to substantial gains in efficiency and performance. Further, it can be effectively combined with previously-proposed multi-task architectures for enhanced performance in a model-agnostic way.

## 1 INTRODUCTION

While deep learning and deep reinforcement learning (RL) have shown considerable promise in enabling systems to perform complex tasks, the data requirements of current methods make it difficult to learn a breadth of capabilities particularly when all tasks are learned individually from scratch. A natural approach to such multi-task learning problems is to train a single network on all tasks jointly, with the aim of discovering shared structure across the tasks in a way that achieves greater efficiency and performance than solving the tasks individually. However, learning multiple tasks all at once results in a difficult optimization problem, sometimes leading to *worse* overall performance and data efficiency compared to learning tasks individually (Parisotto et al., 2015; Rusu et al., 2016a). These optimization challenges are so prevalent that multiple multi-task RL algorithms have considered using independent training as a subroutine of the algorithm before distilling the independent models into a multi-tasking model (Levine et al., 2016; Parisotto et al., 2015; Rusu et al., 2016a; Ghosh et al., 2017; Teh et al., 2017), producing a multi-task model but losing out on the efficiency gains over independent training. If we could tackle the optimization challenges of multi-task learning effectively, we may be able to actually realize the hypothesized benefits of multi-task learning without the cost in final performance.

While there has been a significant amount of research in multi-task learning (Caruana, 1997; Ruder, 2017), the optimization challenges are not well understood. Prior work has described varying learning speeds of different tasks (Chen et al., 2017) and plateuas in the optimization landscape (Schaul et al., 2019) as potential causes, while a range of other works have focused on the model architecture (Misra et al., 2016b; Liu et al., 2018). In this work, we instead hypothesize that the central optimization issue in multi-task learning arises from gradients from different tasks conflicting with one another. In particular, we define two gradients to be conflicting if they point away from one another (i.e., have a negative cosine similarity). As a concrete example, consider the 2D optimization landscapes of two task objectives shown in Figure 1. The optimization landscape of each task consists of a deep valley, as has been characterized of neural network optimization landscapes in the past (Goodfellow et al., 2014). When considering the combined optimization landscape for multiple tasks, SGD produces gradients that struggle to efficiently find the optimum. This occurs due to a gradient thrashing

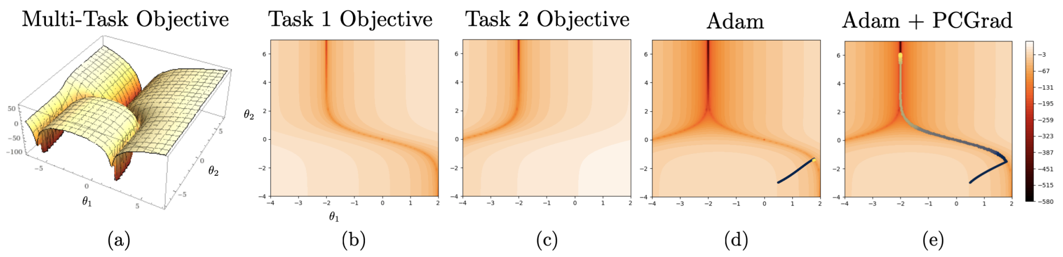

Figure 1: Visualization of gradient surgery's effect on a 2D multi-task optimization problem. (a) A multi-task objective landscape. (b) & (c) Contour plots of the individual task objectives that comprise the multi-task objective. (d) Trajectory of gradient updates on the multi-task objective using the Adam optimizer. (e) Trajectory of gradient updates on the multi-task objective using Adam with PCGrad. For (d) and (e), the optimization trajectory goes from black to yellow.

phenomenon, where the gradient of one task destabilizes optimization in the valley. We can observe this in Figure 1(d) when the optimization reaches the deep valley of task 1, but is prevented from traversing the valley to an optimum. In Section 6.2, we find experimentally that this thrashing phenomenon also occurs in a neural network multi-task learning problem.

The core contribution of this work is a method for mitigating gradient interference by altering the gradients directly, i.e. by performing "gradient surgery". If two gradients are conflicting, we alter the gradients by projecting each onto the normal plane of the other, preventing the interfering components of the gradient from being applied to the network. We refer to this particular form of gradient surgery as *projecting conflicting gradients* (PCGrad). PCGrad is model-agnostic, requiring only a single modification to the application of gradients. Hence, it is easy to apply to a range of problem settings, including multi-task supervised learning and multi-task reinforcement learning, and can also be readily combined with other multi-task learning approaches, such as those that modify the architecture. We evaluate PCGrad on multi-task CIFAR classification, multi-objective scene understanding, a challenging multi-task RL domain, and goal-conditioned RL. Across the board, we find PCGrad leads to significant improvements in terms of data efficiency, optimization speed, and final performance compared to prior approaches. Further, on multi-task supervised learning tasks, PCGrad can be successfully combined with prior state-of-the-art methods for multi-task learning for even greater performance.

## 2 Preliminaries

The goal of multi-task learning is to find parameters $\theta$ of a model $f_\theta$ that achieve high average performance across all the training tasks drawn from a distribution of tasks $p(\mathcal{T})$. More formally, we aim to solve the problem: $\min_\theta \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})}\left[\mathcal{L}_i(f_\theta)\right]$, where $\mathcal{L}_i$ is a loss function for the $i$-th task $\mathcal{T}_i$ that we want to minimize. To obtain a model that solves a specific task from the task distribution $p(\mathcal{T})$, we define a task-conditioned model $f_\theta(y|x, z_i)$, with input $x$, output $y$, and encoding $z_i$ for task $\mathcal{T}_i$, which could be provided as a one-hot vector or in any other form.

## 3 Multi-Task Learning via Gradient Surgery

While the multi-task problem can in principle be solved by simply applying a standard single-task algorithm with a suitable task identifier provided to the model or a simple multi-head or multi-output model, a number of prior works (Parisotto et al., 2015; Rusu et al., 2016a; Sener & Koltun, 2018) have found this learning problem to be difficult, especially in the reinforcement learning setting. We hypothesize that one of the main challenges of multi-task learning can be characterized by conflicting and thrashing gradients, and find that this can significantly impede learning progress, especially when combined with iterative data collection. We identify possible causes for this problem and propose a simple and general approach to mitigate it.

### 3.1 Thrashing Gradients in Multi-Task Optimization Landscapes

We hypothesize that a key optimization issue in multi-task learning arises when gradients from multiple tasks are in conflict with one another, i.e. when gradients point away from one another. More specifically, we hypothesize that such conflict may lead to *gradient thrashing*. Concretely, gradient thrashing refers to the phenomenon where a large gradient for one task changes the parameter vectors

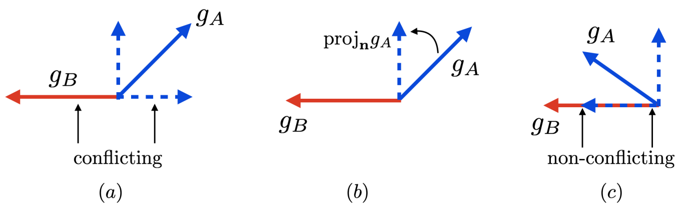

Figure 2: Visual depiction of conflicting gradients and PCGrad. In (a), we see that tasks A and B have conflicting gradient directions, which can lead to destructive interference and unstable learning. In (b), we illustrate the PCGrad algorithm in cases where gradients are conflicting. PCGrad projects the gradient of task A onto the normal vector of task B's gradient. In (c), we show that tasks with non-conflicting gradients are not altered under PCGrad, thereby keeping tasks with constructive interference.

in a way that substantially decreases performance on another task. Since worse performance typically leads to larger gradients, this results in alternating gradient directions, where, at the next iteration, the second task will have large gradients that dominate and reduce performance on the former task. This issue can be particularly pronounced for neural network optimization, since neural network loss landscapes are known to resemble long narrow valleys (Goodfellow et al., 2014), where the gradient *perpendicular* to the direction of the valley will be small.

We aim to study this hypothesis through two toy examples. First, consider the two-dimensional optimization landscape illustrated in Fig. 1a, where the landscape for each task objective corresponds to a deep and curved valley (Fig. 1b and 1c). The optima of this multi-task objective correspond to where the two valleys meet. More details on the optimization landscape are in Appendix B. We observe that the gradient thrashing hypothesis is consistent with what we observe when running Adam (Kingma & Ba, 2014) on this landscape in Fig. 1d, where we observe that Adam does not traverse one valley towards the other, preventing it from reaching an optimum.

We also aim to detect if a similar phenomenon occurs in multi-task learning with a neural network with thousands of parameters on a toy regression problem. To measure the extent of gradient thrashing, we plot the cosine similarity between the gradients of two tasks throughout the beginning of learning in Fig. 4 (left). We indeed observe a significant level of gradient thrashing at every iteration, where the cosine similarity varies between $-0.75$ and $0.75$ at a very high frequency.

Motivated by these observations, we develop an algorithm that aims to alleviate the optimization challenges caused by gradient thrashing by preventing such gradient conflict between tasks.

## 3.2 PCGrad: Projecting Conflicting Gradients

We aim to prevent gradient thrashing by directly altering the gradients themselves, i.e. through "gradient surgery." To be maximally effective and maximally applicable, we must perform surgery in a way that still allows for positive interactions between the task gradients and does not introduce any assumptions on the form of the model.

We start by first detecting whether two gradients are in conflict, by measuring whether they point away from one another. More concretely, we characterize two tasks as conflicting for the current parameter setting if they yield a negative cosine similarity between their respective gradients. The goal of PCGrad is to modify the gradients for each task so as to minimize negative conflict with other task gradients, which will in turn mitigate gradient thrashing.

To deconflict gradients during optimization, PCGrad adopts a simple procedure: if the gradients between two tasks are in conflict, i.e. their cosine similarity is negative, we project the gradient from one task onto the normal plane of the gradient of the other task. This amounts to removing the conflicting component of the gradient for the task, thereby reducing the amount of destructive gradient interference between tasks. A pictorial description of this idea is shown in Fig. 2. Suppose the gradient for task $\mathcal{T}_i$ is $\mathbf{g}_i$, and the gradient for task $\mathcal{T}_j$ is $\mathbf{g}_j$. PCGrad proceeds as follows: (1) First, it determines whether $\mathbf{g}_i$ conflicts with $\mathbf{g}_j$ by computing the cosine similarity between vectors $\mathbf{g}_i$ and $\mathbf{g}_j$, where negative values indicate conflicting gradients. (2) If the cosine similarity is negative, we replace $\mathbf{g}_i$ by its projection onto the normal plane of $\mathbf{g}_j$: $\mathbf{g}_i = \mathbf{g}_i - \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2}\mathbf{g}_j$. If the gradients are not in conflict, i.e. cosine similarity is non-negative, the original gradient $\mathbf{g}_i$ remains unaltered. (3) PCGrad repeats this process across all of the other tasks sampled in the current batch $\mathcal{T}_j \ \forall\ j \neq i$, resulting in

**Algorithm 1** PCGrad Update Rule

```
Require: Current model parameters $\theta$
 1: Sample mini-batch of tasks $\mathcal{B} = \{\mathcal{T}_k\} \sim p(\mathcal{T})$
 2: for $\mathcal{T}_i \sim \mathcal{B}$ do
 3:     Compute gradient $\mathbf{g}_i$ of $\mathcal{T}_i$ as $\mathbf{g}_i = \nabla_\theta \mathcal{L}_i(f_\theta)$
 4:     for $\mathcal{T}_j \sim \mathcal{B}$ do
 5:         Compute gradient $\mathbf{g}_j$ of task $\mathcal{T}_j$ as $\mathbf{g}_j = \nabla_\theta \mathcal{L}_j(f_\theta)$
 6:         Compute cosine similarity between $\mathbf{g}_i$ as $\mathbf{g}_j$ as $\cos(\phi_{ij}) = \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{\|\mathbf{g}_i\|\|\mathbf{g}_j\|}$.
 7:         if $\cos(\phi_{ij}) < 0$ then
 8:             Set $\mathbf{g}_i = \mathbf{g}_i - \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2}\mathbf{g}_j$        // Subtract the projection of $\mathbf{g}_i$ onto $\mathbf{g}_j$
 9:         end if
10:     end for
11:     Store $\mathbf{g}_i^{\text{proj}} = \mathbf{g}_i$
12: end for
13: return update $\Delta\theta = \sum_i \mathbf{g}_i^{\text{proj}}$
```

the gradient $\mathbf{g}_i^{\text{proj}}$ that is applied for task $\mathcal{T}_i$. We perform the same procedure for all tasks in the batch to obtain their respective gradients. The full update procedure is described in Algorithm 1.

This procedure, while simple to implement, ensures that the gradients that we apply for each task per batch interfere minimally with the other tasks in the batch, mitigating the thrashing gradient problem, producing a variant on standard first-order gradient descent in the multi-objective setting. In practice, the PCGrad gradient surgery method can be combined with any gradient-based optimizer, including commonly used methods such as SGD with momentum and Adam (Kingma & Ba, 2014), by simply passing the computed update to the respective optimizer instead of the original gradient. Our experimental results verify the hypothesis that this procedure reduces the problem of thrashing gradients, and find that, as a result, learning progress is substantially improved.

Finally, we analyze the convergence of this procedure in Theorem 1 in the two-task setting, to ensure that the procedure is sensible under the standard assumptions in optimization.

**Theorem 1.** *Consider two task loss functions $\mathcal{L}_1 : \mathbb{R}^n \to \mathbb{R}$ and $\mathcal{L}_2 : \mathbb{R}^n \to \mathbb{R}$ which are convex and differentiable. For all $\theta \in \mathbb{R}^n$, let $\mathcal{L}(\theta) = \mathcal{L}_1(\theta) + \mathcal{L}_2(\theta)$, i.e. $\mathcal{L}$ is a multi-task objective. Let $\phi$ be the angle between $\nabla\mathcal{L}_1(\theta)$ and $\nabla\mathcal{L}_2(\theta)$. Suppose $\mathcal{L}$ is differentiable and that its gradient is Lipschitz continuous with constant $L > 0$, i.e. we have $||\nabla\mathcal{L}(\theta_1) - \nabla\mathcal{L}(\theta_2)||_2 \le L||\theta_1 - \theta_2||_2$ for any $\theta_1, \theta_2$. Then, the PCGrad update rule with step size $t \le \frac{1}{L}$ will converge to either (1) a location in the optimization landscape where $\cos(\phi) = -1$ or (2) the optimal value $\mathcal{L}(\theta^*)$.*

*Proof.* See Appendix A. □

Theorem 1 states that application of the PCGrad update in the two-task setting with a convex and Lipschitz multi-task loss function $\mathcal{L}$ leads to convergence to either the minimizer of $\mathcal{L}$ or a potentially sub-optimal objective value. A sub-optimal solution occurs when the cosine similarity between the gradients of the two tasks is $-1$, i.e. the gradients directly conflict, leading to zero gradient after applying PCGrad. However, in practice, since we are using SGD, which is a noisy estimate of the true batch gradients, the cosine similarity between the gradients of two tasks in a minibatch is unlikely to be $-1$, thus avoiding this scenario.

## 4 The Practical Operations of PCGrad

We apply PCGrad to both supervised learning and reinforcement learning problem settings with multiple tasks or goals. In this section, we discuss the practical instantiations of PCGrad in those settings. Further implementation details are included in Section 6.

### 4.1 Multi-Task Supervised Learning

In multi-task supervised learning, each task $\mathcal{T}_i \sim p(\mathcal{T})$ has a corresponding training dataset $\mathcal{D}_i$ consisting of $N_i$ labeled training examples, i.e. $\mathcal{D}_i = \{(x, y)_n\}_{n=1}^{N_i}$. The objective for each task in this supervised setting is then defined as $\mathcal{L}_i(f_\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}_i}[-\log f_\theta(y|x, z_i)]$, where $z_i$ is a one-hot encoding of task $\mathcal{T}_i$.

At each training step, we randomly sample a batch of data points $\mathcal{B}$ from the whole dataset $\bigcup_i \mathcal{D}_i$ and then group the sampled data with the same task encoding into small batches denoted as $\mathcal{B}_i$ for each $\mathcal{T}_i$ represented in $\mathcal{B}$. We denote the set of tasks appearing in $\mathcal{B}$ as $\mathcal{B}_{\mathcal{T}}$. After sampling, we precompute the gradient of each task in $\mathcal{B}_{\mathcal{T}}$ as

$$\nabla_\theta \mathcal{L}_i(f_\theta) = \mathbb{E}_{(x,y)\sim\mathcal{B}_i}\left[-\nabla_\theta \log f_\theta(y|x, z_i)\right]. \tag{1}$$

Given the set of precomputed gradients $\nabla_\theta \mathcal{L}_i(f_\theta)$, we also precompute the cosine similarity between all pairs of the gradients in the set. Using the pre-computed gradients and their similarities, we can obtain the PCGrad update by following Algorithm 1, without re-computing task gradients nor backpropagating into the network.

Since the PCGrad procedure is only modifying the gradients of shared parameters in the optimization step, it is model-agnostic and can be readily applied to any architecture designed for supervised multi-task learning. In Section 6, we combine PCGrad with two state-of-the-art architectures for multi-task learning, which leads to noticeable improvement over their original performance.

## 4.2 Multi-Task and Goal-Conditioned Reinforcement Learning

For multi-task reinforcement learning, PCGrad can be readily applied to policy gradient methods by directly updating the computed policy gradient of each task, following Algorithm 1, analogous to the supervised learning setting. For actor-critic algorithms, it is also straightforward to apply PCGrad: we simply replace the task gradients for both the actor and the critic by their gradients computed via PCGrad. Hence, PCGrad can be readily incorporated into a variety of model-free RL algorithms. When applying PCGrad to goal-conditioned RL, we represent $p(\mathcal{T})$ as a distribution of goals and let $z_i$ be the encoding of a goal. Similar to the multi-task supervised learning setting discussed above, PCGrad may be combined with various architectures designed for multi-task and goal-conditioned RL (Fernando et al., 2017; Devin et al., 2016), where PCGrad operates on the gradients of shared parameters, leaving task-specific parameters untouched.

In our experiments, we apply PCGrad to the soft actor-critic (SAC) algorithm (Haarnoja et al., 2018), a recently proposed off-policy actor-critic algorithm that has shown significant gains in sample efficiency and asymptotic performance across many different domains. In SAC, we employ a Q-learning style gradient to compute the gradient of the Q-function network, $Q_\phi(s, a, z_i)$, often known as the critic, and a reparameterization-style gradient to compute the gradient of the policy network $\pi_\theta(a|s, z_i)$, often known as the actor. For sampling, we instantiate a set of replay buffers $\{\mathcal{D}_i\}_{\mathcal{T}_i\sim p(\mathcal{T})}$. Training and data collection are alternated throughout training. During a data collection step, we run the policy $\pi_\theta$ on all the tasks $\mathcal{T}_i \sim p(\mathcal{T})$ to collect an equal number of paths for each task and store the paths of each task $\mathcal{T}_i$ into the corresponding replay buffer $\mathcal{D}_i$. At each training step, we sample an equal amount of data from each replay buffer $\mathcal{D}_i$ to form a stratified batch. For each task $\mathcal{T}_i \sim p(\mathcal{T})$, the parameters of the critic $\theta$ are optimized to minimize the soft Bellman residual:

$$J_Q^{(i)}(\phi) = \mathbb{E}_{(s_t,a_t,z_i)\sim\mathcal{D}_i}\left[Q_\phi(s_t, a_t, z_i) - (r(s_t, a_t, z_i) + \gamma V_{\bar{\phi}}(s_{t+1}, z_i))\right], \tag{2}$$

$$V_{\bar{\phi}}(s_{t+1}, z_i) = \mathbb{E}_{a_{t+1}\sim\pi_\theta}\left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1}, z_i) - \alpha \log \pi_\theta(a_{t+1}|s_{t+1}, z_i)\right], \tag{3}$$

where $\gamma$ is the discount factor, $\bar{\phi}$ are the delayed parameters, and $\alpha$ is a learnable temperature that automatically adjusts the weight of the entropy term. For each task $\mathcal{T}_i \sim p(\mathcal{T})$, the parameters of the policy $\pi_\theta$ are trained to minimize the following objective

$$J_\pi^{(i)}(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}_i}\left[\mathbb{E}_{a_t\sim\pi_\theta(a_t|s_t,z_i))}\left[\alpha \log \pi_\theta(a_t|s_t, z_i) - Q_\phi(s_t, a_t, z_i)\right]\right]. \tag{4}$$

We compute $\nabla_\phi J_Q^{(i)}(\phi)$ and $\nabla_\theta J_\pi^{(i)}(\theta)$ for all $\mathcal{T}_i \sim p(\mathcal{T})$ and apply PCGrad to both following Algorithm 1.

In the context of SAC specifically, we further study how the temperature $\alpha$ should be adjusted. If we use a single learnable temperature for adjusting entropy of the multi-task policy $\pi_\theta(a|s, z_i)$, SAC may stop exploring once all easier tasks are solved, leading to poor performance on tasks that are harder or require more exploration. To address this issue, we propose to learn the temperature on a per-task basis, i.e. using a parametrized model to represent $\alpha_\psi(z_i)$ (which we abbreviate as **PA** for per-task alpha). This allows the method to control the entropy of $\pi_\theta(a|s, z_i)$ per-task. We optimize the parameters of $\alpha_\psi(z_i)$ using the same constrained optimization framework as in Haarnoja et al. (2018).

## 5 Related Work

Algorithms for multi-task learning typically consider how to train a single model that can solve a variety of different tasks (Caruana, 1997; Bakker & Heskes, 2003; Ruder, 2017). The multi-task formulation has been applied to many different settings, including supervised learning (Zhang et al., 2014; Long & Wang, 2015; Yang & Hospedales, 2016; Sener & Koltun, 2018; Zamir et al., 2018) and reinforcement-learning (Espeholt et al., 2018; Wilson et al., 2007), as well as many different domains, such as vision (Bilen & Vedaldi, 2016; Misra et al., 2016a; Kokkinos, 2017; Liu et al., 2018; Zamir et al., 2018), language (Collobert & Weston, 2008; Dong et al., 2015; McCann et al., 2018; Radford et al., 2019) and robotics (Riedmiller et al., 2018; Wulfmeier et al., 2019; Hausman et al., 2018). While multi-task learning has the promise of accelerating acquisition of large task repertoires, in practice it presents a challenging optimization problem, which has been tackled in several ways in prior work.

A number of architectural solutions have been proposed to the multi-task learning problem based on multiple modules or paths (Fernando et al., 2017; Devin et al., 2016; Misra et al., 2016b; Rusu et al., 2016b; Rosenbaum et al., 2018; Vandenhende et al., 2019; Rosenbaum et al., 2018), or using attention-based architectures (Liu et al., 2018; Maninis et al., 2019). Our work is agnostic to the model architecture and can be combined with prior architectural approaches in a complementary fashion.

A different set of multi-task learning approaches aim to decompose the problem into multiple local problems, often corresponding to each task, that are significantly easier to learn, akin to divide and conquer algorithms (Levine et al., 2016; Rusu et al., 2016a; Parisotto et al., 2015; Teh et al., 2017; Ghosh et al., 2017; Czarnecki et al., 2019). Eventually, the local models are combined into a single, multi-task policy using different distillation techniques (outlined in (Hinton et al., 2015; Czarnecki et al., 2019)). In contrast to these methods, we propose a simple and cogent scheme for multi-task learning that allows us to learn the tasks simultaneously using a single, shared model without the need for network distillation.

Similarly to our work, a number of prior approaches have observed the difficulty of optimization in the multi-task learning setting (Hessel et al., 2019; Sener & Koltun, 2018; Chen et al., 2018; Kendall et al., 2018b; Schaul et al., 2019). Our work, in contrast to many of these optimization schemes, suggests that the challenge in multi-task learning can be attributed to the problem of gradient thrashing, which we address directly by introducing a practical algorithm that de-conflicts gradients from different tasks. Prior work has also used the cosine similarity between gradients to define when an auxiliary task might be useful for single-task learning (Du et al., 2018). We similarly use cosine similarity between gradients to determine if the gradients between a pair of tasks are in conflict. Unlike (Du et al., 2018), we use this measure of gradient conflict as a part of gradient surgery in the context of multi-task learning applications. Finally, our method is distinct from and solves a different problem than the projected gradient method (Calamai & Moré, 1987), which is an approach for constrained optimization that projects gradients onto the constraint manifold.

## 6 Experiments

The goal of our experiments is to study the following questions: (1) Are conflicting gradients a major factor in making optimization for multi-task learning challenging? (2) Does PCGrad make the optimization problems easier for various multi-task learning problems including supervised, reinforcement, and goal-conditioned reinforcement learning settings across different task families? (3) Can PCGrad be combined with other multi-task learning approaches to further improve performance?

### 6.1 Experimental Setup

To evaluate our method experimentally, we consider both a multi-task supervised learning and a multi-task reinforcement learning problem setup. For supervised learning, we consider the CIFAR-100 dataset (Krizhevsky et al., 2009) where each of the 20 label superclasses are treated as distinct tasks, following Rosenbaum et al. (2018). We also conduct experiments on the NYUv2 dataset (Silberman et al., 2012), which consists of RGB-D indoor scene images. Following Liu et al. (2018), we evaluate our method on 3 tasks: 13-class semantic segmentation, depth estimation, and surface normal

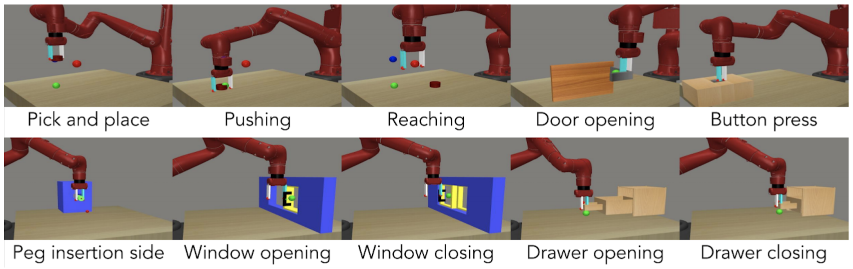

Figure 3: Visualization of 10 tasks used in MT10 from Meta-World (Yu et al., 2019), which we use for our multi-task RL experiments.

prediction. In the case of multi-task reinforcement learning, we evaluate our algorithm on the recently proposed Meta-World benchmark (Yu et al., 2019). This benchmark includes a variety of simulated robotic manipulation tasks contained in a shared, table-top environment with a simulated Sawyer arm (visualized as the "Pushing" environment in Fig. 3). In particular, we use the multi-task benchmark MT10, which consists of the 10 tasks depicted in Fig. 3 that require diverse strategies to solve them, which makes them difficult to optimize jointly with a single policy. To evaluate goal-conditioned RL scenarios, we consider goal-conditioned robotic pushing with a Sawyer robot. This domain is representative of challenges in learning goal-conditioned policies over a wide distribution of goals. For details on the experimental set-up and model architectures see Appendix C.

## 6.2 Analysis of Conflicting Gradients

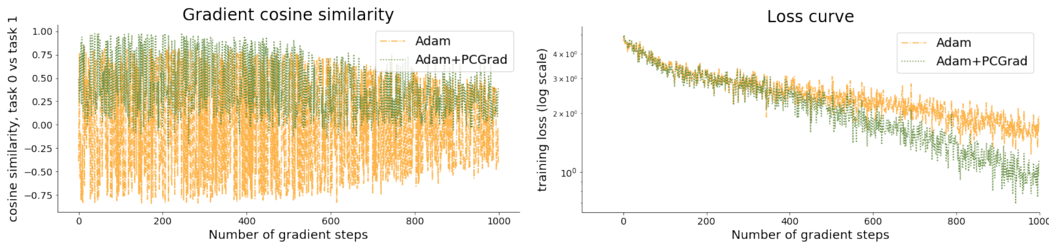

Figure 4: An analysis of the gradients during the first 1000 updates of training, on a toy 10-task sinusoid regression problem. **Left**: The cosine similarity between the gradients of 2 of the 10 tasks (selected arbitrarily, and fixed throughout this plot). We observe a substantial amount of thrashing with standard Adam training, while Adam with PCGrad reduces the thrashing and leads to more closely aligned updates. **Right**: Adam with PCGrad improves performance compared to standard Adam.

To answer question (1), we consider a simple regression problem, where each task is regressing the input to the output of a sine function. The amplitude and the phase of each task are varied. We construct 10 tasks with the amplitude uniformly sampled from the range $[0, 5]$ and the phase uniformly sampled from the range $[0, \pi]$. The input is also uniformly sampled from the range $[0, 5]$ and is concatenated with the one-hot task encoding. For training, we use a 3-layer fully-connected neural network with 100 hidden units.

We compare the performance of the network trained with Adam and the network trained with Adam with PCGrad-modified gradients while plotting the cosine similarity between a pair of tasks during training as shown in Figure 4. The plot on the left in Figure 4 demonstrates that the cosine similarity of Adam gradients between a pair of tasks has high variance, which leads to the gradient thrashing problem, while the cosine similarity of the gradient projected by PCGrad yields positive values diminishing the conflicting-gradients problem. As shown in the plot on the right in Figure 4, Adam with PCGrad leads to faster learning over Adam, which implies that gradient thrashing is indeed a problem in multi-task optimization and reducing it can result in considerable performance boost.

## 6.3 Multi-Task and Multi-Objective Supervised Learning with PCGrad

To answer question (3), we perform experiments on two standard multi-task supervised learning dataset: multi-task CIFAR-100 and NYUv2.

For CIFAR-100, we follow (Rosenbaum et al., 2018) to treat 20 coarse labels in the dataset as distinct tasks and create a dataset with 20 tasks and 2500 training instances as well as 500 test instances per task. We combine PCGrad with a powerful multi-task learning architecture, routing networks (Rosenbaum et al., 2018; 2019), by simply projecting gradients of the shared parameters in routing networks. As shown in Table 1, combining PCGrad with routing networks leads to a $4.7\%$ absolute improvement in test accuracy averaged over 3 runs, while also outperforming other approachs such as training independent networks for each task and cross-stitch (Misra et al., 2016b).

We also combine PCGrad with another state-of-art multi-task learning algorithm, MTAN (Liu et al., 2018), and evaluate the performance on a more challenging indoor scene dataset, NYUv2, which contains 3 tasks as described in Section 6.1. We compare MTAN with PCGrad to a list of methods mentioned in Section 6.1, where each method is trained with three different weighting schemes as in (Liu et al., 2018), equal weighting, weight uncertainty (Kendall et al., 2018a), and DWA (Liu et al., 2018). We only run MTAN with PCGrad with weight uncertainty as we find weight uncertainty as the most effective scheme for training MTAN. The results comparing Cross-Stitch, MTAN and MTAN + PCGrad are presented in Table 2 while the full comparison can be found in Table 3 in the Appendix C.3. MTAN with PCGrad is able to achieve the best scores in 8 out of the 9 categories where there are 3 categories per task.

Our multi-task supervised learning results demonstrate that PCGrad can be seamlessly combined with state-of-art multi-task learning architectures and further improve their results on established supervised multi-task learning benchmarks.

| | % accuracy |
|---|---|
| task specific-1-fc (Rosenbaum et al., 2018) | 42 |
| task specific-all-fc (Rosenbaum et al., 2018) | 49 |
| cross stitch-all-fc (Misra et al., 2016b) | 53 |
| routing-all-fc + WPL (Rosenbaum et al., 2019) | 64.1 |
| independent | 64.3 |
| routing-all-fc + WPL + PCGrad (ours) | $\boxed{\mathbf{69}}$ |

Table 1: CIFAR-100 multi-task results. We apply PCGrad to the routing networks and achieve a significant improvement in classfication accuracy.

| #P. | Architecture | Weighting | Segmentation | | Depth | | Surface Normal | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | (Higher Better) | | (Lower Better) | | Angle Distance (Lower Better) | | Within $t^\circ$ (Higher Better) | | |
| | | | mIoU | Pix Acc | Abs Err | Rel Err | Mean | Median | 11.25 | 22.5 | 30 |
| $\approx$3 | Cross-Stitch[‡] | Equal Weights | 14.71 | 50.23 | 0.6481 | 0.2871 | 33.56 | 28.58 | 20.08 | 40.54 | 51.97 |
| | | Uncert. Weights[*] | 15.69 | 52.60 | 0.6277 | 0.2702 | 32.69 | 27.26 | 21.63 | 42.84 | 54.45 |
| | | DWA[†], $T = 2$ | **16.11** | **53.19** | **0.5922** | **0.2611** | **32.34** | **26.91** | **21.81** | **43.14** | **54.92** |
| 1.77 | MTAN[†] | Equal Weights | **17.72** | 55.32 | **0.5906** | 0.2577 | 31.44 | **25.37** | $\boxed{23.17}$ | 45.65 | 57.48 |
| | | Uncert. Weights[*] | 17.67 | **55.61** | 0.5927 | 0.2592 | **31.25** | 25.57 | 22.99 | **45.83** | **57.67** |
| | | DWA[†], $T = 2$ | 17.15 | 54.97 | 0.5956 | **0.2569** | 31.60 | 25.46 | 22.48 | 44.86 | 57.24 |
| 1.77 | MTAN[†]+ PCGrad (ours) | Uncert. Weights[*] | $\boxed{\mathbf{20.17}}$ | $\boxed{\mathbf{56.65}}$ | $\boxed{\mathbf{0.5904}}$ | $\boxed{\mathbf{0.2467}}$ | $\boxed{\mathbf{30.01}}$ | $\boxed{\mathbf{24.83}}$ | 22.28 | $\boxed{\mathbf{46.12}}$ | $\boxed{\mathbf{58.77}}$ |

Table 2: We present the results on three tasks on the NYUv2 dataset: 13-class semantic segmentation, depth estimation, and surface normal prediction results. #P shows the total number of network parameters. We highlight the best performing combination of multi-task architecture and weighting in bold. The top validation scores for each task are annotated with boxes. The symbols indicate prior methods: [*]: (Kendall et al., 2018a), [†]: (Liu et al., 2018), [‡]: (Misra et al., 2016b). Performance of other methods as reported in (Liu et al., 2018).

## 6.4 Multi-Task Reinforcement Learning

To answer question (2), we test all methods on 10 manipulation tasks shown in Figure 3. At each data collection step, we collect 600 samples for each task, and at each training step, we sample 128 datapoints per task from corresponding replay buffers. The results are shown in the plot on the left in Figure 5. We measure success according to the metrics used in the Meta-World benchmark where the reported the success rates are averaged across tasks. For all methods, we apply **PA** as discussed in Section 4 to learn a separate alpha term per task as the task encoding in MT10 is just a one-hot encoding. PCGrad combined with SAC learns all tasks with the best data efficiency and successfully solves all of the 10 tasks. Training a single SAC policy and a multi-head policy turns out to be unable to acquire half of the skills, suggesting that eliminating gradient interference across tasks can significantly boost performance of multi-task RL. Training independent SAC agents is able

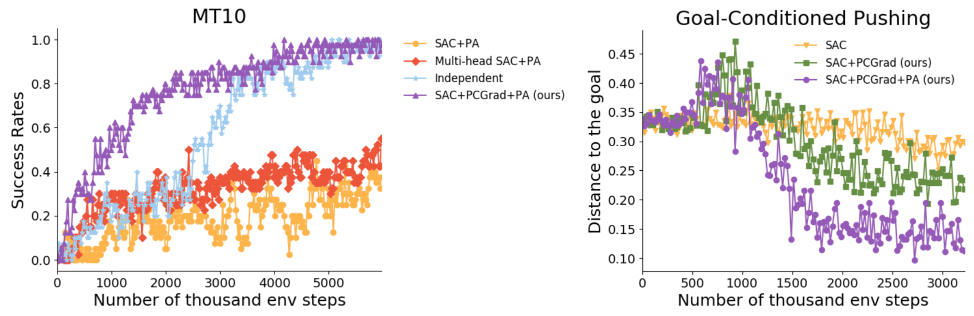

Figure 5: Learning curve on MT10 and goal-conditioned pushing. PCGrad outperforms the other methods in the two settings in terms of both success rates / average distance to the goal and data efficiency.

to eventually solve all tasks, but requires about 2 millions more samples than PCGrad with SAC, implying that applying PCGrad can result in leveraging shared structure among tasks that expedites multi-task learning.

As noted by Yu et al. (2019), these tasks involve fairly distinct behavior motions, which makes learning all of them with a single policy challenging as demonstrated by poor baseline performance. The ability to learn these tasks together opens the door for a number of interesting extensions to meta-learning, goal conditioned RL and generalization to novel task families. We present the results of PCGrad on goal-conditioned RL in the following subsection.

### 6.5 Goal-Conditioned Reinforcement Learning

For our goal-conditioned RL evaluation, we use the robot-pushing environment described in Sec. 6.1 where the goals are represented as the concatenations of the initial positions of the puck to be pushed and the its goal location, both of which are uniformly sampled (details in Appendix C.2). We also apply **PA** as discussed in Section 4 to predict the temperature for entropy term given the goal. We summarize the results in the plot on the right in Figure 5. PCGrad with SAC and PA achieves the best performance in terms of average distance to the goal position, while PCGrad with SAC improves over the baseline and a vanilla SAC agent is struggling to successfully accomplish the task. This suggests that PCGrad is able to ease the RL optimization problem also when the task distribution is continuous.

## 7 Conclusion

In this work, we identified one of the major challenges in multi-task optimization: conflicting gradients across tasks. We proposed a simple algorithm (PCGrad) to mitigate the challenge of conflicting gradients via "gradient surgery". PCGrad provides a simple way to project gradients to be orthogonal in a multi-task setting, which substantially improves optimization performance, since the task gradients are prevented from negating each other. We provide some simple didactic examples and analysis of how this procedure works in simple settings, and subsequently show significant improvement in optimization for a variety of multi-task supervised learning and reinforcement learning problems. We show that, once some of the optimization challenges of multi-task learning are alleviated by PCGrad, we can obtain the hypothesized benefits in efficiency and asymptotic performance that are believed to be possible in multi-task settings.

While we studied multi-task supervised learning and multi-task reinforcement learning in this work, we suspect the problem of conflicting gradients to be prevalent in a range of other settings and applications, such as meta-learning, continual learning, multi-goal imitation learning (Codevilla et al., 2018), and multi-task problems in natural language processing applications (McCann et al., 2018). Due to its simplicity and model-agnostic nature, we expect that applying PCGrad in these domains to be a promising avenue for future investigation. Further, the general idea of gradient surgery may be an important ingredient for alleviating a broader class of optimization challenges in deep learning, such as the challenges in the stability challenges in two-player games (Roth et al., 2017) and multi-agent optimizations (Nedic & Ozdaglar, 2009). We believe this work to be a step towards simple yet general techniques for addressing some of these challenges.

## References

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

Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *J. Mach. Learn. Res.*, 2003.

Hakan Bilen and Andrea Vedaldi. Integrated perception with recurrent multi-task neural networks. In *Advances in neural information processing systems*, pp. 235–243, 2016.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Paul H Calamai and Jorge J Moré. Projected gradient methods for linearly constrained problems. *Mathematical programming*, 39(1):93–116, 1987.

Rich Caruana. Multitask learning. *Machine Learning*, 1997.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the International Conference on Machine Learning*, 2018.

Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9. IEEE, 2018.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167. ACM, 2008.

Wojciech M. Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant M. Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS*, 2019.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1723–1732, 2015.

Yunshu Du, Wojciech M. Czarnecki, Siddhant M. Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *CoRR*, abs/1812.02224, 2018.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017. URL `http://arxiv.org/abs/1701.08734`.

Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *CoRR*, abs/1711.09874, 2017.

Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, 2018.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. 2018.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

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

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7482–7491, 2018a.

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *CVPR*, 2018b.

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

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

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Shikun Liu, Edward Johns, and Andrew J. Davison. End-to-end multi-task learning with attention. *CoRR*, abs/1803.10704, 2018.

Mingsheng Long and Jianmin Wang. Learning multiple tasks with deep relationship networks. *arXiv preprint arXiv:1506.02117*, 2, 2015.

Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. *CoRR*, abs/1904.08918, 2019.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.

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

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Conference on Computer Vision and Pattern Recognition, CVPR*, 2016b.

Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48, 2009.

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *International Conference on Learning Representations (ICLR)*, 2018.

Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in neural information processing systems*, pp. 2018–2028, 2017.

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

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *4th International Conference on Learning Representations, ICLR*, 2016a.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016b.

Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *arXiv preprint arXiv:1904.11455*, 2019.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, 2018.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pp. 746–760. Springer, 2012.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Simon Vandenhende, Bert De Brabandere, and Luc Van Gool. Branched multi-task networks: Deciding what layers to share. *CoRR*, abs/1904.02920, 2019.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022. ACM, 2007.

Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Regularized hierarchical policies for compositional transfer in robotics. *arXiv preprint arXiv:1906.11228*, 2019.

Yongxin Yang and Timothy M Hospedales. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*, 2016.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Sergey Levine, and Chelsea Finn. Meta-world: A benchmark and evaluation for multi-task and meta-reinforcement learning, 2019. URL `https://github.com/rlworkgroup/metaworld`.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pp. 94–108. Springer, 2014.

## A Proof of Theorem 1

*Proof.* We will use the shorthand $||\cdot||$ to denote the $L_2$-norm and $\nabla\mathcal{L} = \nabla_\theta\mathcal{L}$, where $\theta$ is the parameter vector. Let $\mathbf{g_1} = \nabla\mathcal{L}_1$, $\mathbf{g_2} = \nabla\mathcal{L}_2$, and $\phi$ be the angle between $\mathbf{g_1}$ and $\mathbf{g_2}$.

At each PCGrad update, we have two cases: $cos(\phi) \geq 0$ or $\cos(\phi < 0)$.

If $\cos(\phi) \geq 0$, then we apply the standard gradient descent update using $t \leq \frac{1}{L}$, which leads to a strict decrease in the objective function value $\mathcal{L}(\phi)$ unless $\nabla\mathcal{L}(\phi) = 0$, which occurs only when $\theta = \theta^*$ (Boyd & Vandenberghe, 2004).

In the case that $\cos(\phi) < 0$, we proceed as follows:

Our assumption that $\nabla\mathcal{L}$ is Lipschitz continuous with constant $L$ implies that $\nabla^2\mathcal{L}(\theta) - LI$ is a negative semidefinite matrix. Using this fact, we can perform a quadratic expansion of $\mathcal{L}$ around $\mathcal{L}(\theta)$ and obtain the following inequality:

$$\begin{aligned}\mathcal{L}(\theta^+) &\leq \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T(\theta^+ - \theta) + \frac{1}{2}\nabla^2\mathcal{L}(\theta)||\theta^+ - \theta||^2 \\ &\leq \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T(\theta^+ - \theta) + \frac{1}{2}L||\theta^+ - \theta||^2\end{aligned}$$

Now, we can plug in the PCGrad update by letting $\theta^+ = \theta - t(\nabla\mathcal{L}(\theta) - \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_1}||^2}\mathbf{g_1} - \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_2}||^2}\mathbf{g_2})$. We then get:

$$\begin{aligned}\mathcal{L}(\theta^+) \leq \mathcal{L}(\theta) &+ t(\nabla\mathcal{L}(\theta))^T(-\nabla\mathcal{L}(\theta) + \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_1}||^2}\mathbf{g_1} + \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_2}||^2}\mathbf{g_2}) \\ &+ \frac{1}{2}Lt^2||\nabla\mathcal{L}(\theta) - \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_1}||^2}\mathbf{g_1} - \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_2}||^2}\mathbf{g_2}||^2\end{aligned}$$

(Expanding, using the identity $\nabla\mathcal{L}(\theta) = \mathbf{g_1} + \mathbf{g_2}$)

$$\begin{aligned}&= \mathcal{L}(\theta) + t(-||\mathbf{g_1}||^2 - ||\mathbf{g_2}||^2 + 2\mathbf{g_1}\cdot\mathbf{g_2} + \frac{(\mathbf{g_1}\cdot\mathbf{g_2})^2}{||\mathbf{g_1}||^2} + \frac{(\mathbf{g_1}\cdot\mathbf{g_2})^2}{||\mathbf{g_2}||^2}) \\ &+ \frac{1}{2}Lt^2||\mathbf{g_1} + \mathbf{g_2} - \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_1}||^2}\mathbf{g_1} - \frac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_2}||^2}\mathbf{g_2}||^2\end{aligned}$$

(Expanding further and re-arranging terms)

$$\begin{aligned}&= \mathcal{L}(\theta) - (t - \frac{1}{2}Lt^2)(||\mathbf{g_1}||^2 + ||\mathbf{g_2}||^2 - \frac{(\mathbf{g_1}\cdot\mathbf{g_2})}{||\mathbf{g_1}||^2} - \frac{(\mathbf{g_1}\cdot\mathbf{g_2})}{||\mathbf{g_2}||^2}) \\ &- Lt^2(\mathbf{g_1}\cdot\mathbf{g_2} - \frac{(\mathbf{g_1}\cdot\mathbf{g_2})^2}{||\mathbf{g_1}||^2||\mathbf{g_2}||^2}\mathbf{g_1}\cdot\mathbf{g_2})\end{aligned}$$

(Using the identity $\cos(\phi) = \dfrac{\mathbf{g_1}\cdot\mathbf{g_2}}{||\mathbf{g_1}||||\mathbf{g_2}||}$)

$$\begin{aligned}&= \mathcal{L}(\theta) - (t - \frac{1}{2}Lt^2)[(1 - \cos^2(\phi))||\mathbf{g_1}||^2 + (1 - \cos^2(\phi))||\mathbf{g_2}||^2] \\ &- Lt^2(1 - \cos^2(\phi))||\mathbf{g_1}||||\mathbf{g_2}||\cos(\phi)\end{aligned}$$

(Note that $\cos(\phi) < 0$ so the final term is non-negative)

Using $t \leq \frac{1}{L}$, we know that $-(1 - \frac{1}{2}Lt) = \frac{1}{2}Lt - 1 \leq \frac{1}{2}L(1/L) - 1 = \frac{-1}{2}$ and $Lt^2 \leq t$.

Plugging this into the last expression above, we can conclude the following:

$$
\begin{aligned}
&\mathcal{L}(\theta^{+}) \leq \mathcal{L}(\theta) - \frac{1}{2}t[(1-\cos^2(\phi))||\mathbf{g_1}||^2 + (1-\cos^2(\phi))||\mathbf{g_2}||^2] \\
&- t(1-\cos^2(\phi))||\mathbf{g_1}||||\mathbf{g_2}||\cos(\phi) \\
&= \mathcal{L}(\theta) - \frac{1}{2}t(1-\cos^2(\phi))[||\mathbf{g_1}||^2 + 2||\mathbf{g_1}||||\mathbf{g_2}||\cos(\phi) + ||\mathbf{g_2}||^2] \\
&= \mathcal{L}(\theta) - \frac{1}{2}t(1-\cos^2(\phi))[||\mathbf{g_1}||^2 + 2\mathbf{g_1}\cdot\mathbf{g_2} + ||\mathbf{g_2}||^2] \\
&= \mathcal{L}(\theta) - \frac{1}{2}t(1-\cos^2(\phi))||\mathbf{g_1}+\mathbf{g_2}||^2 \\
&= \mathcal{L}(\theta) - \frac{1}{2}t(1-\cos^2(\phi))\|\nabla\mathcal{L}(\theta)\|^2
\end{aligned}
$$

If $\cos(\phi) > -1$, then $\frac{1}{2}t(1-\cos^2(\phi))\|\nabla\mathcal{L}(\theta)\|^2$ will always be positive unless $\nabla\mathcal{L}(\theta) = 0$. This inequality implies that the objective function value strictly decreases with each iteration where $\cos(\phi) > -1$.

Hence repeatedly applying PCGrad process can either reach the optimal value $\mathcal{L}(\theta) = \mathcal{L}(\theta^*)$ or $\cos(\phi) = -1$, in which case $\frac{1}{2}t(1-\cos^2(\phi))\|\nabla\mathcal{L}(\theta)\|^2 = 0$. Note that this result only holds when we choose $t$ to be small enough, i.e. $t \leq \frac{1}{L}$.

$\square$

## B 2D Optimization Landscape Details

To produce the 2D optimization visualizations in Figure 1, we used a parameter vector $\theta = [\theta_1, \theta_2] \in \mathbb{R}^2$ and the following task loss functions:

$$
\begin{aligned}
\mathcal{L}_1(\theta) &= 20\log(\max(|.5\theta_1 + \tanh(\theta_2)|, 0.000005)) \\
\mathcal{L}_2(\theta) &= 25\log(\max(|.5\theta_1 - \tanh(\theta_2) + 2|, 0.000005))
\end{aligned}
$$

The multi-task objective is $\mathcal{L}(\theta) = \mathcal{L}_1(\theta) + \mathcal{L}_2(\theta)$. We initialized $\theta = [0.5, -3]$ and performed 500,000 gradient updates to minimize $\mathcal{L}$ using the Adam optimizer with learning rate 0.001. We compared using Adam for each update to using Adam in conjunction with the PCGrad method presented in Section 3.2.

## C Experiment Details

### C.1 Detailed Experiment Set-up

For our CIFAR-100 multi-task experiment, we adopt the architecture used in Rosenbaum et al. (2019), which is a convolutional neural network that consists of 3 convolutional layers with 160 $3 \times 3$ filters each layer and 2 fully connected layers with 320 hidden units. As for experiments on the NYUv2 dataset, we follow Liu et al. (2018) to use SegNet (Badrinarayanan et al., 2017) as the backbone architecture.

Our reinforcement learning experiments all use the SAC (Haarnoja et al., 2018) algorithm as the base algorithm, where the actor and the critic are represented as 6-layer fully-connected feedforward neural networks for all methods. The numbers of hidden units of each layer of the neural networks are 160 and 200 for MT10 and goal-conditioned RL respectively.

We use five algorithms as baselines in the CIFAR-100 multi-task experiment: **task specific-1-fc** (Rosenbaum et al., 2018): a convolutional neural network shared across tasks except that each task has a separate last fully-connected layer, **task specific-1-fc** (Rosenbaum et al., 2018) : all the convolutional layers shared across tasks with separate fully-connected layers for each task, **cross stitch-all-fc** (Misra et al., 2016b): one convolutional neural network per task along with cross-stitch units to share features across tasks, **routing-all-fc + WPL** (Rosenbaum et al., 2019): a network that employs a trainable router trained with multi-agent RL algorithm (WPL) to select trainable functions for each task, **independent**: training separate neural networks for each task.

For comparisons on the NYUv2 dataset, we consider 5 baselines: **Single Task, One Task**: the vanilla SegNet used for single-task training, **Single Task, STAN** (Liu et al., 2018): the single-task version of MTAN as mentioned below, **Multi-Task, Split, Wide / Deep** (Liu et al., 2018): the standard SegNet shared for all three tasks except that each task has a separate last layer for final task-specific prediction with two variants **Wide** and **Deep** specified in Liu et al. (2018), **Multi-Task Dense**: a shared network followed by separate task-specific networks, **Multi-Task Cross-Stitch** (Misra et al., 2016b): similar to the baseline used in CIFAR-100 experiment but with SegNet as the backbone, **MTAN** (Liu et al., 2018): a shared network with a soft-attention module for each task.

On the multi-task and goal-conditioned RL domain, we apply PCGrad to the vanilla SAC algorithm with task encoding as part of the input to the actor and the critic as described in Section 4 and compare our method to the vanilla **SAC** without PCGrad and training actors and critics for each task individually (**Independent**).

## C.2 Goal-conditioned Experiment Details

We use the pushing environment from the Meta-World benchmark (Yu et al., 2019) as shown in Figure 3. In this environment, the table spans from $[-0.4, 0.2]$ to $[0.4, 1.0]$ in the 2D space. To construct the goals, we sample the intial positions of the puck from the range $[-0.2, 0.6]$ to $[0.2, 0.7]$ on the table and the goal positions from the range $[-0.2, 0.85]$ to $[0.2, 0.95]$ on the table. The goal is represented as a concatenation of the initial puck position and the goal position. Since in the goal-conditioned setting, the task distribution is continuous, we sample a minibatch of 9 goals and 128 samples per goal at each training iteration and also sample 600 samples per goal in the minibatch at each data collection step.

## C.3 Full NYUv2 Results

We provide the full comparison on the NYUv2 dataset in Table 3.

| | | | | Segmentation | | Depth | | Surface Normal | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | (Higher Better) | | (Lower Better) | | Angle Distance (Lower Better) | | Within $t^\circ$ (Higher Better) | | |
| Type | #P. | Architecture | Weighting | mIoU | Pix Acc | Abs Err | Rel Err | Mean | Median | 11.25 | 22.5 | 30 |
| Single Task | 3 | One Task | n.a. | 15.10 | 51.54 | 0.7508 | 0.3266 | 31.76 | 25.51 | 22.12 | 45.33 | 57.13 |
| | 4.56 | STAN$^\dagger$ | n.a. | 15.73 | 52.89 | 0.6935 | 0.2891 | 32.09 | 26.32 | 21.49 | 44.38 | 56.51 |
| Multi Task | 1.75 | Split, Wide | Equal Weights | 15.89 | 51.19 | 0.6494 | 0.2804 | 33.69 | 28.91 | 18.54 | 39.91 | 52.02 |
| | | | Uncert. Weights$^*$ | 15.86 | 51.12 | **0.6040** | 0.2570 | **32.33** | **26.62** | **21.68** | **43.59** | **55.36** |
| | | | DWA$^\dagger$, $T = 2$ | **16.92** | **53.72** | 0.6125 | **0.2546** | 32.34 | 27.10 | 20.69 | 42.73 | 54.74 |
| | 2 | Split, Deep | Equal Weights | 13.03 | 41.47 | 0.7836 | 0.3326 | 38.28 | 36.55 | 9.50 | 27.11 | 39.63 |
| | | | Uncert. Weights$^*$ | **14.53** | 43.69 | 0.7705 | 0.3340 | **35.14** | **32.13** | **14.69** | **34.52** | **46.94** |
| | | | DWA$^\dagger$, $T = 2$ | 13.63 | **44.41** | **0.7581** | **0.3227** | 36.41 | 34.12 | 12.82 | 31.12 | 43.48 |
| | 4.95 | Dense | Equal Weights | 16.06 | 52.73 | 0.6488 | 0.2871 | 33.58 | 28.01 | 20.07 | 41.50 | 53.35 |
| | | | Uncert. Weights$^*$ | **16.48** | **54.40** | 0.6282 | 0.2761 | **31.68** | **25.68** | **21.73** | **44.58** | **56.65** |
| | | | DWA$^\dagger$, $T = 2$ | 16.15 | 54.35 | **0.6059** | **0.2593** | 32.44 | 27.40 | 20.53 | 42.76 | 54.27 |
| | ≈3 | Cross-Stitch$^\ddagger$ | Equal Weights | 14.71 | 50.23 | 0.6481 | 0.2871 | 33.56 | 28.58 | 20.08 | 40.54 | 51.97 |
| | | | Uncert. Weights$^*$ | 15.69 | 52.60 | 0.6277 | 0.2702 | 32.69 | 27.26 | 21.63 | 42.84 | 54.45 |
| | | | DWA$^\dagger$, $T = 2$ | **16.11** | **53.19** | **0.5922** | **0.2611** | **32.34** | **26.91** | **21.81** | **43.14** | **54.92** |
| | 1.77 | MTAN$^\dagger$ | Equal Weights | **17.72** | 55.32 | **0.5906** | 0.2577 | 31.44 | **25.37** | $\boxed{23.17}$ | 45.65 | 57.48 |
| | | | Uncert. Weights$^*$ | 17.67 | **55.61** | 0.5927 | 0.2592 | **31.25** | 25.57 | 22.99 | **45.83** | **57.67** |
| | | | DWA$^\dagger$, $T = 2$ | 17.15 | 54.97 | 0.5956 | **0.2569** | 31.60 | 25.46 | 22.48 | 44.86 | 57.24 |
| | 1.77 | MTAN$^\dagger$ + PCGrad (ours) | Uncert. Weights$^*$ | $\boxed{\mathbf{20.17}}$ | $\boxed{\mathbf{56.65}}$ | $\boxed{\mathbf{0.5904}}$ | $\boxed{\mathbf{0.2467}}$ | $\boxed{\mathbf{30.01}}$ | $\boxed{\mathbf{24.83}}$ | 22.28 | $\boxed{\mathbf{46.12}}$ | $\boxed{\mathbf{58.77}}$ |

Table 3: We present the full results on three tasks on the NYUv2 dataset: 13-class semantic segmentation, depth estimation, and surface normal prediction results. #P shows the total number of network parameters. We highlight the best performing combination of multi-task architecture and weighting in bold. The top validation scores for each task are annotated with boxes. The symbols indicate prior methods: $^*$: (Kendall et al., 2018a), $^\dagger$: (Liu et al., 2018), $^\ddagger$: (Misra et al., 2016b). Performance of other methods taken from (Liu et al., 2018).