Rotograd: Dynamic Gradient Homogenization for Multi-Task Learning

Adrián Javaloy[1] **Isabel Valera**[1]

[1]Computer Science Dept., Saarland University, Saarbrücken, Saarland, Germany

Abstract

While multi-task learning (MTL) has been successfully applied in several domains, it still triggers challenges. As a consequence of *negative transfer*, simultaneously learning several tasks can lead to unexpectedly poor results. A key factor contributing to this undesirable behavior is the problem of *conflicting gradients*. In this paper, we propose a novel approach for MTL, Rotograd, which homogenizes the gradient directions across all tasks by rotating their shared representation. Our algorithm is formalized as a Stackelberg game, which allows us to provide stability guarantees. Rotograd can be transparently combined with task-weighting approaches (e.g., GradNorm) to mitigate negative transfer, resulting in a robust learning process. Thorough empirical evaluation on several architectures (e.g., ResNet) and datasets (e.g., CIFAR) verifies our theoretical results, and shows that Rotograd outperforms previous approaches. A Pytorch implementation can be found in `https://github.com/adrianjav/rotograd`.

1 INTRODUCTION

Multi-task learning (MTL) [Caruana, 1993], i.e., learning a model that simultaneously solves several tasks, has proven to be a powerful and efficient alternative to the usual single-task learning in contexts such as computer vision [He et al., 2017], natural language processing [Devlin et al., 2018], and reinforcement learning [Yu et al., 2020b]. MTL has been shown to act as a regularizer during the network learning, leading to more meaningful neural representations and better generalization [Subramanian et al., 2018].

While being a promising learning paradigm for neural networks, the training process of the network in a MTL fashion often turn out difficult in practice. Since tasks share the network parameters, the competition for these resources during training may have a harmful effect on the performance of the individual tasks. The MTL process often leads to networks that accurately fit only a subset of the tasks, the performance of the rest being negatively affected by the MTL approach. This undesired effect, known as *negative transfer* [Ruder, 2017], can be traced back to the combination of task losses during training. Under the assumption that unrelated tasks cause this effect, several works have explored task-clustering solutions [Thrun and O'Sullivan, 1996, Zamir et al., 2018]. See Standley et al. [2019] for an in-depth study on task grouping.

Alternatively, related work attributed the source of the problem to the gradient computation, where the gradients with respect to the shared parameters for the different tasks are combined. In this context, we can observe that the problem is two-fold. On the one hand, differences in magnitude can make a subset of gradient tasks dominate the parameter updates. A number of different methods have been developed to address this issue under different criteria [Chen et al., 2018, Kendall et al., 2018, Guo et al., 2018], yet all of them work by tuning a set of task-weighting parameters.

On the other hand we have gradient directions which, for different tasks, could point towards opposite directions, cancelling each other out when added up. This effect, known as *gradient conflict*, is increasingly capturing the attention of the research community [Yu et al., 2020a, Suteu and Guo, 2019, Levi and Ullman, 2020]. These works provide solutions to combine the per-task gradients so that they do not cancel each other, and thus that the update of the network parameters is possible. However, in general such solutions rely on heuristics that do not guarantee that the new gradient update indeed constitutes an improvement towards the optima of the individual tasks.

Here, we argue that gradient conflict reveals a deeper problem: conflicting gradients are an indication that *the local optima for different tasks may be in completely different parts of the shared parameter space*. If this were the case, existing approaches to avoid negative transfer in MTL fall short as taking a different update direction does not alleviate the underlying problem, since the optima remain still.

In this paper, we instead propose an approach to MTL that does not only manipulate the per-task gradients, but also brings the (local) optima of different tasks closer to each other. Specifically, we introduce Rotograd, which dynamically homogenizes the gradient directions across all tasks by rotating the shared-representation space. As a result we obtain per-task representations where the local optima of the different tasks, which are pointed out by the per-task gradients, are closer to each other. As expected, such a transformation needs to be carefully performed. Fortunately, we can leverage game-theoretical results to provide theoretical guarantees on the convergence and stability of the training. Remarkably, *Rotograd is compatible with all other existing solutions in the MTL literature*, e.g., to homogenize the per-task gradient magnitudes with GradNorm [Chen et al., 2018]. Moreover, our theoretical framework allows us to re-interpret GradNorm as a Stackelberg game, and thus, to provide the same theoretical guarantees as for Rotograd.

We empirically verify this theoretical findings and provide further insights on the relation between negative transfer, model capacity, and data complexity, which sheds some light on the scenarios where Rotograd excels. Furthermore, we compare the performance of Rotograd with competing methods [Chen et al., 2018, Yu et al., 2020a, Chen et al., 2020], outperforming them in a multi-task problem on MNIST and SVHN using LeNet, as well as in a multi-label classification task on CIFAR10 using ResNet.

2 BACKGROUND

2.1 CONFLICTING GRADIENTS IN MTL

MTL setting. Let us consider K different tasks that we want to simultaneously learn. For simplicity, we assume that they share the input dataset $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, where each row $\boldsymbol{x}_i \in \mathbb{R}^D$ is a D-dimensional sample. Additionally, each task is defined by its own real-valued loss function ℓ^k. We further assume that the model can be divided into two parts: backbone and heads. The backbone h_θ contains the common parameters θ and transforms the input $\boldsymbol{X} \in \mathbb{R}^{B \times D}$ to a common intermediate representation $\boldsymbol{Z} \in \mathbb{R}^{B \times d}$, where B is the batch size, and d the size of the intermediate space (with $d << D$). Afterwards, $\boldsymbol{Z}$ is fed to each task-specific head (with non-shared parameters) producing the desired outcome. We denote by $L_k := \sum_i \ell_{i,k}$ the total loss for the k-th task,[1] where $\ell_{i,k}$ is the loss for the i-th observation.

During training, we aim to find both the set of backbone parameters θ, as well as the the head parameters, that minimize the losses of the individuals tasks, L_k. Throughout the paper we focus on first-order optimization techniques (e.g., Adam [Kingma and Ba, 2014]) to train the model. Thus, to train the model it is necessary to have a single real-valued loss function. In this setting, however, there are K different loss functions. A common approach to tackle this problem is to combine the losses by means of a weighted sum, so that the final goal is to minimize the loss function $L := \sum_k \omega_k L_k$. Unfortunately, as mentioned in the introduction, artificially coupling the losses in this way requires careful manipulation in order to accurately learn all the individual tasks.

Here, we focus on the learning of the backbone parameters, as the different tasks compete for them to minimize their individual losses, and thus, it is where *negative transfer* may occur [Ruder, 2017]. At step t of the optimization process, we denote by $\boldsymbol{z}_i^t$ the output of the backbone h_θ for the i-th input $\boldsymbol{x}_i^t$, and by $L_k(t)$ the total loss of the k-th head fed with $\boldsymbol{Z}^t$. Therefore, the common parameters θ are updated following the direction of the gradient vector

$$\nabla_\theta L(t) = \sum_{k=1}^{K} \omega_k \nabla_\theta L_k(t). \tag{1}$$

The above expression seems to be the main source of the negative transfer problem, as the linear combination of gradients may be dominated by a subset of tasks and/or point towards a direction which does not improve any of the individual tasks. This undesirable but well-known problem, that has been pointed out in the past (see, e.g. Sener and Koltun [2018] and Chen et al. [2018]), can be better understood by analyzing its two main components.

Gradient magnitudes. First, the per-task gradient magnitude, which can be interpreted as the combination of the task sensitivity to the common parameters, $||\nabla_\theta L_k(t)||$, and the importance assigned to solving that task, ω_k. Extensive literature has focused on task-weighting solutions which dynamically modify the weights during training to avoid that the overall gradient magnitude is dominated by a subset of tasks [Sener and Koltun, 2018, Chen et al., 2018, Guo et al., 2018, Kendall et al., 2018]. Although necessary, these solutions do not solve the problem of task gradients cancelling out due to them pointing towards different directions.

Gradient directions. This leads us to the second component of the problem, gradient direction. For each task, its gradient points towards the direction of the parameter space that immediately improves L_k the most, which may however *conflict* with the gradient directions of other tasks. A sensible assumption is that this can be related to task similarity and, thus, task-grouping should solve the problem, as discussed in previous work [Standley et al., 2019].

However, we argue here that the problem of conflicting gradients (and thus negative transfer) is local and tied to the optimization process (as well as to the model and the data itself [Wu et al., 2020]). As a consequence, we believe that

[1] Losses are normalized by their initial value, $L_k(0)$.

successfully addressing this problem requires local solutions during the optimization process. Previous works have explored this direction, e.g., Yu et al. [2020a] proposed an ad-hoc solution based on projections between task gradients, and Suteu and Guo [2019] included a regularization term based on the cosine similarity between gradients, involving an expensive second back-propagation call. All of these solutions have in common that they only manipulate the gradients of the individual tasks—or how they are combined—to improve the update direction to follow.

In this paper, we provide a solution to the conflicting gradient problem that instead homogenizes the gradients across tasks by rotating the last shared representation space, $\boldsymbol{Z}$, differently for each task (refer to Section 3). Our approach builds upon game-theoretical results, in particular Stackelberg games, to ensure the convergence and stability of the overall optimization process.

2.2 STACKELBERG GAMES

In game theory, a Stackelberg game [Fiez et al., 2019] is an *asymmetric game* where two players play alternately. One of the players is known as the follower $\mathscr{F}$, whose objective is simply to minimize its own loss function. The other player, known as the leader $\mathscr{L}$, has a more interesting role. While also attempting to minimize its own loss function, it does so while having and advantageous position, as it possesses additional information regarding which will be the follower's response. In mathematical terms,

$$\begin{aligned} &\mathscr{L}\text{eader: } \min_{x_l \in X_l} \{\mathscr{L}(x_l, x_f) \,|\, x_f \in \operatorname*{argmin}_{y \in X_f} \mathscr{F}(x_l, y)\}, \\ &\mathscr{F}\text{ollower: } \min_{x_f \in X_f} \mathscr{F}(x_l, x_f), \end{aligned} \tag{2}$$

where $x_l \in X_l$ and $x_f \in X_f$ are the actions taken by the leader and follower, respectively.

An important concept in game theory is that of an equilibrium point. In layman's terms, an equilibrium point is one in which both players are satisfied with their situation, meaning that there is no available move immediately improving any of the players' scores, so that none of the players is willing to perform additional actions/updates.

Specifically, as we assume a gradient-based game, meaning that their decisions are based on the local information provided by gradients, we focus on the following definition of equilibrium point introduced by Fiez et al. [2019]:

Definition 2.1 (differential Stackelberg equilibrium)**.** A pair of points $x_l^* \in X_l$, $x_f^* \in X_f$, where $x_f^* = r(x_l^*)$ is implicitly defined by $\nabla_{x_f} \mathscr{F}(x_l^*, x_f^*) = 0$, is a differential Stackelberg equilibrium point if $\nabla_{x_l} \mathscr{L}(x_l^*, r(x_l^*)) = 0$, and $\nabla^2_{x_l} \mathscr{L}(x_l^*, r(x_l^*))$ is positive definite.

Note that, when the players manage to reach such an equilibrium point, both of them are in a local optimum. Here, we make use the following result, introduced by Fiez et al. [2019], to provide theoretical convergence guarantees to an equilibrium point:

$$\begin{tikzcd}
 & & \boldsymbol{r}_1 \arrow[r, "head_1"] & \ell_1 \\
\boldsymbol{x} \arrow[r, "h_\theta"] & \boldsymbol{z} \arrow[ur, "R_1"] \arrow[r, "R_2"] \arrow[ddr, "R_K"'] & \boldsymbol{r}_2 \arrow[r, "head_2"] & \ell_2 \\
 & & \cdots & \cdots \\
 & & \boldsymbol{r}_K \arrow[r, "head_K"] & \ell_K
\end{tikzcd}$$

Figure 1: Diagram of the considered MTL model adapted for Rotograd by introducing the rotation matrices R_k.

Proposition 2.1. *In the given setting, if the leader's learning rate goes to zero at a faster rate than the follower's, that is, $\alpha_l(t) = o(\alpha_f(t))$, where $\alpha_i(t)$ denotes the learning rate of player i at step t, then they will asymptotically converge to a differential Stackelberg equilibrium point almost surely.*

In other words, as long as the follower learns faster than the leader, they will end up in a situation where both are satisfied. Even more, Fiez et al. [2019] extended this result to the finite-time case, showing that the game will end close to an equilibrium point with high probability.

3 ROTOGRAD

We are now in a position to introduce the proposed method, *Rotograd*. As previously discussed, if we merely consider the magnitude of the gradients in Eq. 1, we overlook their vectorial nature and, thus, whether they conflict. Ultimately, we would desire not to encounter conflicting gradients at all, i.e., that gradients of different tasks—with respect to the shared parameters θ—point towards a similar direction.

However, changing the update direction does not affect the position of the tasks' optima, and therefore we are likely to keep encountering conflicting gradients in the following steps. Here, we propose to *homogenize the gradient direction across all tasks by rotating the shared-representation space*, which would bring local optima closer between tasks and, thus, following the usual gradient direction would be less conflicting in the next step.

To this end, Rotograd introduces a set of task-specific rotation matrices, $R_k \in SO(d)$, which will adjust the space of the intermediate representation $\boldsymbol{Z}$ to the specific commodities of the k-th task. In other words, Rotograd extends the model architecture so that each task has its own task-specific intermediate representation, $\boldsymbol{r}_k := R_k \boldsymbol{z}$, as depicted in Fig. 1. In the absence of Rotograd these matrices are simply identity matrices, so that $\boldsymbol{r}_k = \boldsymbol{z}$.

As notation in this section can get cluttered, let us denote the gradient of the k-th task with respect to its intermediate representation $\boldsymbol{r}_k$, at step t, by $\boldsymbol{g}^t_{i,k} := \nabla_{\boldsymbol{r}_k} \ell_k(\boldsymbol{r}^t_{i,k})$; the gradient of the weighted loss with respect to $\boldsymbol{z}$, at step t, by $\boldsymbol{g}^t_i := \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}_i) = \sum_k \omega^t_k R^\top_k \boldsymbol{g}^t_{i,k}$; and by G^t_k and G^t these same gradients when considered as vectors over all the batch data, that is, as vectors of size $B \times d$.

In order to homogenize the direction of the task gradients, Rotograd adjust the rotation matrices R_k such that it minimizes the distance between the gradient of the k-th task, $\boldsymbol{g}^t_{i,k}$, and a common *reference vector* that we want them to point to, $\boldsymbol{v}^t_i$. In particular, Rotograd optimizes the following least-square problem:

$$\mathscr{L}^k_{\text{rot}}(R_k, \theta) := \frac{1}{B} \sum_{i=1}^{B} ||R^\top_k \boldsymbol{g}^t_{i,k} - \boldsymbol{v}^t_i||^2_2. \tag{3}$$

However, this optimization has to be performed carefully since it will, at each step, alter the input space of all task-specific heads. Fortunately, we can formulate Rotograd as a Stackelberg game between the leader, which optimizes R_k, and the follower, whose task is to optimize the parameters of the network θ:

$$\begin{aligned} \mathscr{L}\text{eader: } & \underset{\{R_k \in SO(d)\}_k}{\text{minimize}} \sum_k \mathscr{L}^k_{\text{rot}}(R_k, \theta), \\ \mathscr{F}\text{ollower: } & \underset{\theta}{\text{minimize}} \sum_k \omega_k L_k(R_k, \theta). \end{aligned} \tag{4}$$

One important but subtle bit about the formulation above regards the extra information used by the leader. In this case, this extra knowledge explicitly appears in Eq. 3 in the form of the follower's gradient $\boldsymbol{g}^t_{i,k}$, which is the direction the follower will follow and, as it is performing first-order optimization by assumption, the following gradients, $\boldsymbol{g}^{t+1}_{i,k}$, should be similar to the current one.

Thanks to the Stackelberg formulation in Eq. 4 we can make use of Prop. 2.1 and, thus, draw theoretical guarantees on the training stability and convergence. In other words, we can say that training according to the update rules

$$\begin{aligned} R^{t+1}_k &= R^t_k - \alpha^t_l \nabla_{R_k} \mathscr{L}^k_{\text{rot}}(R_k, \theta), \\ \theta^{t+1} &= \theta^t - \alpha^t_f \sum_i \boldsymbol{g}^t_i. \end{aligned} \tag{5}$$

will stably converge as long as the leader is asymptotically the slow learner, i.e., $\alpha^t_l = o(\alpha^t_f)$. See Appendix A for a detailed description of the algorithm.

Remark. To ensure that R_k is a rotation, i.e., that R_k holds $det(R_k) = 1$, we follow the approach of Lezcano-Casado and Martínez-Rubio [2019] and parametrize R_k as the exponential map of skew-symmetrical matrices, solving instead an unconstrained problem.

3.1 GRADNORM AND THE REFERENCE VECTOR

Two key points remain open. First, note that Rotograd preserves distances in $\boldsymbol{Z}$—and thus gradient magnitudes, so it is advisable to complement it with a task-weighting solution. Second, we need to specify the reference vector $\boldsymbol{v}^t_i$. We solve both issues by means of GradNorm [Chen et al., 2018].

GradNorm attempts to equalize the rate by which each task is learned by optimizing—along with the network parameters—the task weights such that the gradient magnitudes are similar across tasks. To this end, GradNorm optimizes ω_k to minimize

$$\mathscr{L}^k_{\text{grad}}(\omega_k, \theta) := \left| \omega_k ||G^t_k||_2 - ||G^t||_2 \left(s^t_k\right)^\tau \right|, \tag{6}$$

where G^t is treated as a constant, τ is a hyperparameter, $s^t_k := L_k(t)/[\frac{1}{K}\sum_j L_j(t)]$ the relative learning speed for the k-th task, and the weights hold $\sum_k \omega_k = K$. Therefore, we complement Rotograd with GradNorm, homogenizing both gradient magnitudes and directions across tasks.

As with Rotograd, note that we can also recast GradNorm as a Stackelberg game as in Eq. 4, but now with the leader solving Eq. 6, that is:

$$\begin{aligned} \mathscr{L}\text{eader: } & \underset{\{\omega_k\}_k}{\text{minimize}} \sum_k \mathscr{L}^k_{\text{grad}}(\omega_k, \theta), \\ \mathscr{F}\text{ollower: } & \underset{\theta}{\text{minimize}} \sum_k \omega_k L_k(\theta). \end{aligned} \tag{7}$$

This perspective of GradNorm has not been explored before and allow us to: i) easily combine GradNorm and Rotograd as a leader that solves both objectives (R_k and ω_k); and ii) provide the same stability guarantees as with Rotograd.

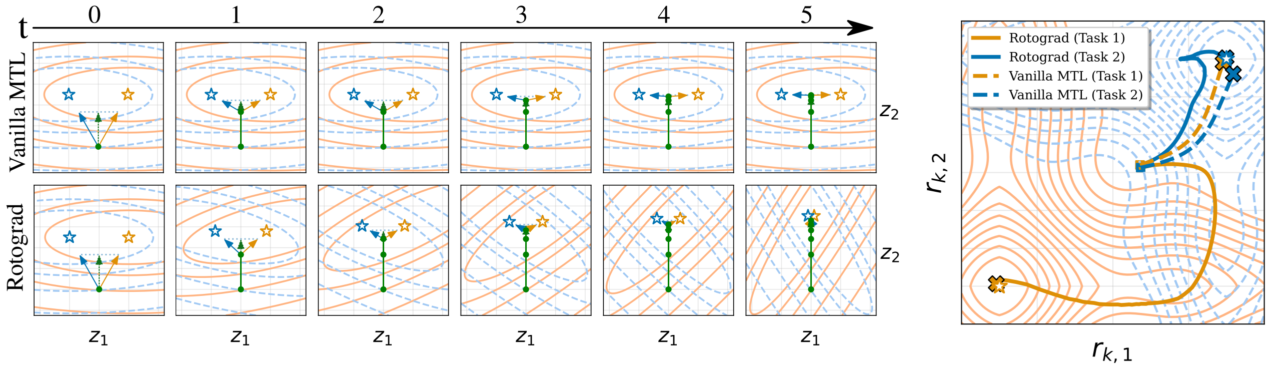
Regarding the reference vector for Eq. 3, we take the direction followed when all tasks contribute exactly the same to the gradient computation in Eq. 1, i.e., all have similar gradient magnitudes. Conveniently, this is the same final objective that GradNorm has and, as the selection of $\boldsymbol{v}^t_i$ does not directly interact with the follower, we can solve Eq. 6 in closed form. In other words, we set

$$\boldsymbol{v}^t_i := \sum_k \frac{G^t_k}{||G^t_k||_2} ||G^t||_2 (s^t_k)^\tau \tag{8}$$

where, as before, $(s^t_k)^\tau$ accounts for the differences in learning speed between tasks at step t.

3.2 COOPERATIVE ROTOGRAD

When the follower's problem, i.e., the MTL problem, is hard to solve, it may turn out complicated in practice to find learning rates (as well as schedules) for both the leader and the follower. We overcome this problem by introducing



(a) Convex avocado-shaped regression example.

(b) Non-convex regression example.

Figure 2: Evolution of the two regression examples with and without Rotograd. Fig. (a) shows the evolution in the space of $\boldsymbol{z}$ (i.e, the shared representation), whereas Fig. (b) shows the evolution in the space of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ (i.e., the per-task representations). In both examples, Rotograd is able to overcome gradient disparities and achieve the optima of all tasks. In contrast, Vanilla MTL gets stuck in a poor solution both both tasks in Fig. (a), and only solves the second task in Fig. (b).

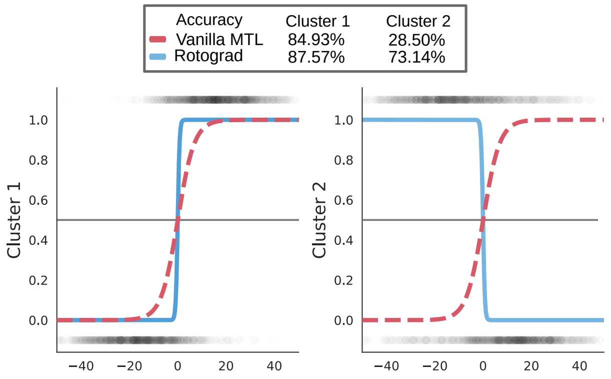


Figure 3: Decision functions of the logistic regression example. The population ground-truth values are scattered as grey dots. Rotograd is able to learn both related but opposite functions. In contrast, Vanilla MTL accurately solves only the task for cluster 1, overlooking the other task.

a cooperative variant of Rotograd which, after being selfish about optimizing its own loss functions during the first epochs, starts looking after the interests of the follower as well. That is, after T epochs, problem Eq. 4 becomes

$$\begin{aligned} \mathscr{L}\text{eader: } &\underset{\{R_k\in SO(d)\}_k}{\text{minimize}} \sum_k \left[\mathscr{L}^k_{\text{rot}}(R_k,\theta) + L_k(R_k,\theta)\right], \\ \mathscr{F}\text{ollower: } &\underset{\theta}{\text{minimize}} \sum_k \omega_k L_k(R_k,\theta), \end{aligned} \tag{9}$$

which, as updating R_k takes into account solving both problems, has a regularizing effect on the learning speed of L^k_{rot}, while maintaining the Stackelberg formulation and helping out the follower.

Furthermore, as we have introduced a secondary multi-task problem, we attempt to avoid overlooking any of the two losses when updating R_k by applying a closed-form version of GradNorm with $\tau = 0$, i.e., we update R_k according to

$$\left[\frac{\nabla_{R_k}\mathscr{L}^k_{\text{rot}}}{||\nabla_{R_k}\mathscr{L}^k_{\text{rot}}||} + \frac{\nabla_{R_k}L_k}{||\nabla_{R_k}L_k||}\right]\frac{||\nabla_{R_k}\mathscr{L}^k_{\text{rot}}|| + ||\nabla_{R_k}L_k||}{2}. \tag{10}$$

Interestingly enough, the first term of Eq. 10 resembles the formulation of Normalized Gradient Descent [Murray et al., 2017] but for the multi-objective case.

4 VISUALIZING ROTOGRAD

In this section we explore the effect of applying Rotograd in three different synthetic scenarios in order to test—as well as get a better understanding of—how Rotograd works. Refer to Appendix B for details on the experimental set-ups.

First, we solve two different multi-task regression experiments of the form

$$L(\boldsymbol{x}) = \underbrace{f(R_1\boldsymbol{z},0)}_{L_1} + \underbrace{f(R_2\boldsymbol{z},1)}_{L_2}, \tag{11}$$

where $\boldsymbol{z} = h_\theta(\boldsymbol{x})$, $r_k = R_k\boldsymbol{z}$, and f is a test function (a.k.a. artificial landscape) with a single global optimum parametrized by its second argument, i.e., both tasks are identical (and thus related) up to translations.

To easily illustrate the effect of Rotograd, we use a single input $\boldsymbol{x} \in \mathbb{R}^2$ and remove the task-specific heads. Regarding the backbone, we use a simple ReLU feed-forward network with a single ten-neuron hidden layer, and bi-dimensional output, $h_\theta(\boldsymbol{x}) = \boldsymbol{z} \in \mathbb{R}^2$. For the first experiment we choose a simple (avocado-shaped) convex objective function and, for the second experiment, we opt for a non-convex function with many local optima and a single global optimum.

Figures 2a and 2b show the trajectories followed for both tasks in the presence (and absence) of Rotograd. To provide the fairest comparison, for *uniform* we use Rotograd with fixed parameters, as Rotograd does not initialize the rotation matrices to the identity matrix. In the first experiment (Fig. 2a), Rotograd is able to find both optima, which is in stark contrast to the *uniform* case.The second experiment shows that Rotograd is able to find both global optima (while avoiding falling into local ones), instead of getting dominated by one of the tasks.

For the third experiment we test Rotograd in the worst-case scenario of gradient conflict, i.e., the case where task gradients are opposite to each other. To this end, we solve a binary multi-classification task where, as dataset, we use a simple synthetic dataset obtained from a two-component bi-dimensional Gaussian mixture model where the labels refer to whether the sample was (and was not) generated from the first component.

As each single task is easy to solve, we use as model a two-layer logistic regression model with hidden size two, where *all parameters are shared*, and where we take the bi-dimensional output of the first layer as $\boldsymbol{Z}$. Fig. 3 shows how Rotograd is able to perfectly learn both tasks, whereas the vanilla case focuses on solving only one of the tasks.

5 EXPERIMENTS

In this section, we first perform an ablation study in order to: i) better understand how model capacity and data complexity affect negative transfer; and ii) empirically verify the necessity of having a slow leader, as described in Prop. 2.1. To finalize, we compare Rotograd with existing methods in several datasets and models, showing its standalone effectiveness (combined with GradNorm), despite being also compatible with any other method. A more detailed description of the experiments can be found in Appendix C, and additional results are presented in Appendix D.

Multi-task metric. In order to summarize the performance of a model across all tasks we use a similar metric as Vandenhende et al. [2019], i.e., we use

$$\Delta_{\mathrm{MTL}} = \frac{1}{K}\sum_{k=1}^{K}(-1)^{l_k}\frac{m_k - u_k}{u_k}, \tag{12}$$

where m_k and u_k are, respectively, the performance (in terms of, e.g., accuracy or mean squared error) on the k-task for a given method and the *vanilla* case. The exponent l_k is meant to be 1 if $m_k < u_k$ means a performance improvement, and 0 otherwise. Therefore, Δ_{MTL} can be interpreted as the average relative improvement across all tasks.

5.1 ABLATION STUDY

In order to understand better the scenarios where negative transfer is prominent, we perform a small ablation study, where we attempt to solve the same set of tasks varying the dataset and network capacity. According to our intuition, the more capacity a model has, the less negative transfer should appear. Moreover, we argue that gradient conflict (and thus negative transfer) is a local problem that occurs during optimization (Eq. 1), rather than only a byproduct of tasks unrelatedness, as suggested by Ruder [2017].

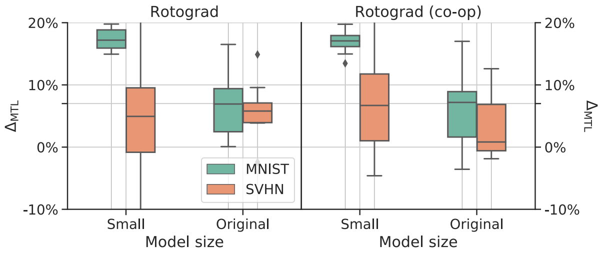


Figure 4: Relative performance improvement on two different model capacities and datasets for Rotograd and its cooperative counterpart.

Datasets and tasks. We use a modified version of MNIST [LeCun et al., 1998] and SVHN [Netzer et al., 2011], such that each sample now contains two digits, one in each side of the image. With this data we learn different correlated, highly correlated, and uncorrelated tasks: i) classify the left-side digit; ii) classify the right-side digit; iii) sum both digits; iv) predict whether the product is odd; and v) predict the number of active pixels in the image.

Architecture. Two different versions of LeNet [LeCun et al., 1998] are employed as backbone: the original one, and a modified version with an order of magnitude less parameters. These networks have $d = 50$ and $d = 25$, respectively. For the head networks we use simple two-layer ReLU networks. Regarding loss functions we use mean squared error for regression, negative log-likelihood loss for classification, and binary cross entropy for binary classification tasks. We measure performance in terms of mean squared error, accuracy, and—for the task *Odd*—f1-score.

Results. Fig. 4 summarizes the model performance in terms of Δ_{MTL}, averaged over ten different independent runs. In the case of MNIST we can observe the positive effect of applying Rotograd in the reduced model, as it forces tasks to share resources and cooperate, whereas in the original LeNet the network partly overcomes these differences with extra network capacity. On SVHN the same trend holds, yet the differences are less noticeable, probably due to the more complex nature of the dataset. Moreover, cooperative Rotograd obtains on average the best results on the reduced version of LeNet, which is the one most susceptible to negative transfer. A closer look to the tabular data in Table 1 (and Appendix D) shows that, in both cases, Rotograd specially excels at better learning the unrelated task (*Active*).

	Method	Left (%)	Right (%)	Sum	Odd (%)	Active	Δ_{MTL} (%)
MNIST	Vanilla MTL	93.48 ± 0.46	90.73 ± 0.39	3.18 ± 0.23	91.19 ± 0.55	0.35 ± 0.06	
	GradDrop	-0.18 ± 0.24	-0.02 ± 0.53	-1.36 ± 5.29	-0.30 ± 0.94	2.51 ± 15.99	0.13 ± 3.43
	PCGrad	0.04 ± 0.28	0.16 ± 0.37	0.76 ± 4.19	0.08 ± 0.66	11.59 ± 10.87	2.53 ± 2.36
	GradNorm	-0.03 ± 0.35	0.10 ± 0.19	1.21 ± 3.88	-0.21 ± 0.52	77.67 ± 4.95	15.75 ± 0.95
	Rotograd	-0.19 ± 0.38	0.53 ± 0.60	7.04 ± 4.77	-0.12 ± 0.72	79.49 ± 4.77	17.35 ± 1.71
	Rotograd (co-op)	-0.18 ± 0.39	0.43 ± 0.50	3.93 ± 4.90	0.09 ± 0.86	79.50 ± 5.50	16.76 ± 1.83
SVHN	Vanilla MTL	73.52 ± 3.80	73.75 ± 3.95	7.33 ± 0.43	69.41 ± 2.64	10.97 ± 5.12	
	GradDrop	1.20 ± 2.00	1.15 ± 3.66	1.89 ± 4.43	-4.61 ± 5.94	-17.92 ± 51.02	-3.66 ± 10.17
	PCGrad	0.96 ± 1.59	-0.11 ± 1.91	0.43 ± 2.30	-0.09 ± 2.74	-5.59 ± 38.46	-0.88 ± 8.07
	GradNorm	0.76 ± 2.11	-0.45 ± 2.59	-0.40 ± 4.38	-3.56 ± 4.34	-12.52 ± 43.55	-3.23 ± 8.66
	Rotograd	2.17 ± 5.50	2.06 ± 6.84	3.21 ± 4.92	1.51 ± 5.04	13.75 ± 40.81	4.54 ± 10.16
	Rotograd (co-op)	2.54 ± 6.53	2.31 ± 4.97	3.52 ± 6.22	1.56 ± 4.94	25.24 ± 27.97	7.04 ± 7.88

Table 1: Test results (mean and standard deviation) for different methods, averaged over ten different runs. For *Vanilla MTL* we show task performance, and the percentage of relative improvement with respect to *Vanilla MTL* for the rest of methods, i.e., each term in Eq. 12. In both datasets, Rotograd results in a positive improvement in all tasks.

5.2 LEADER'S LEARNING SPEED

Now, we test the most important result Rotograd is built upon. In layman's terms, Prop. 2.1 implies that Rotograd can rotate the space of $\boldsymbol{Z}$ safely—meaning that the training will be stable—as long as Rotograd has a smaller learning rate than that of the network's parameters.

To this end, we take the dataset and model most susceptible to being benefited by Rotograd, i.e., MNIST with the small version of LeNet, and perform an ablation study where we simply vary the learning rate of the leader. While we defer for most details to Appendix C, it is worth mentioning that the models were run enough epochs to converge, that the follower's learning rate was set to 1×10^{-3} and, to ensure the fairest results, we compute Δ_{MTL} normalizing by the results of Rotograd with fixed initial parameters.

Fig. 5 shows the relative performance improvement over ten different random starts. Here, we can observe an ascending trend in performance as we reduce the learning rate, which supports the theoretical results provided by Prop. 2.1, reaching at a learning rate of 5×10^{-4} an improvement of 19.31 %. For smaller leader's learning rates, Rotograd is not able to align the gradients fast enough and, thus, is unable to fully exploit the MTL framework.

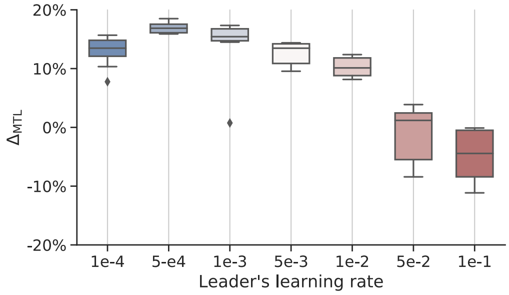


Figure 5: Performance on MNIST for different learning rates for Rotograd. The follower has a learning rate of 1×10^{-3}.

5.3 METHODS COMPARISON

Here we compare Rotograd to different methods from the MTL literature on MNIST and SVHN, including task-weighting and gradient aligning solutions. Out experiments show that Rotograd obtains comparable results in related tasks, while being the only one consistently improving the performance on the unrelated task (*Active*).

Settings. In these experiments we restrict ourselves to the smaller versions of LeNet for MNIST and SVHN, since according to Fig. 4 this is the network most susceptible to suffer from negative transfer. We attempt to simultaneously solve the same tasks as before. In short: i) *Left*; ii) *Right*; iii) *Sum*; iv) *Odd*; and v) *Active*.

Methods. We compare Rotograd and its cooperative version (combined with GradNorm) with: i) *Vanilla MTL*, the baseline; ii) *GradDrop* [Chen et al., 2020], which randomly blocks task gradients according to their "sign concordance"; iii) *PCGrad* [Yu et al., 2020a], which removes the conflicting part of the gradients by removing their projections; and iv) *GradNorm* [Chen et al., 2018], which homogenizes gradient magnitudes, as explained in Section 3.

Hyper-parameter tuning. To ensure fair comparison, all methods were compared using the same random seeds. For those methods with hyperparameters (τ in GradNorm and Rotograd), we selected for each method the best value obtained by cross-validating over identical grid searches. For GradDrop, we set its leak parameter to zero in order to not prioritize any task over the others.

Results. Table 1 shows the obtained results across ten different runs. Both versions of Rotograd obtain similar results on MNIST, obtaining an average relative improvement of 17 %. This comes as the result of keeping the good re-

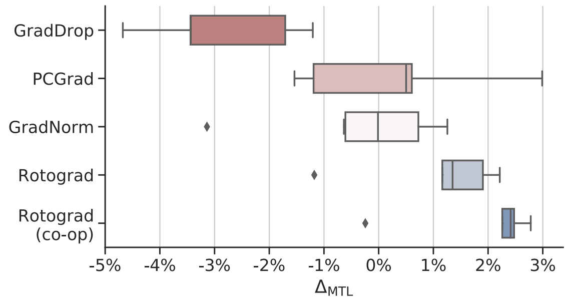


Figure 6: Performance on multi-label classification on CIFAR10 for different methods over five random seeds.

sults on the classification tasks, while improving the two regression tasks, *Sum* and *Active*, with a special boost in the latter. Yet, these results are really close to those obtained by GradNorm (16 %), which could be indicative of the absence of significant gradient conflict.

If we focus now on SVHN, we observe the same trends but with smaller overall improvement, as already observed in Fig. 4. This time, results on the individual tasks are more interesting. Specifically, we find that GradDrop exhibits the most variance, significantly improving and worsening different tasks. PCGrad and GradNorm perform similar to the vanilla case, yet they turn out specially harmful for the most unrelated task (*Active*). In contrast, both versions of Rotograd manage to obtain a significant improvement in all tasks, specially in the most unrelated task, resulting in the only two methods with a positive average improvement. Cooperative Rotograd specially shines in this case, significantly boosting the results in the *Active* task.

5.4 EXPERIMENTS ON CIFAR10

To finalize, we test all methods in a multi-label classification task, where we use CIFAR10 [Krizhevsky et al., 2009] as dataset and its ten classes as different binary classification tasks. To enforce task cooperation, we employ a small version of ResNet18 [He et al., 2016] (more details in Appendix C), with $d = 64$, and a single linear layer with a sigmoid function at the end as task-specific heads.

Fig. 6 shows the results obtained taking the best models according to validation error, averaged over five different runs. In this case, both versions of Rotograd stand out by a large margin: Rotograd obtains on average an improvement of more than 2 %, while the best competitor achieves an average improvement of 0.5 %. Cooperative Rotograd specially excels in this complex scenario where the extra-supervision on the follower's task turns out to be crucial.

Last but not least, Fig. 7 shows the cosine similarity between the update direction and task gradient (after applying all methods), averaged over tasks and five independent runs. This empirically proves the effectiveness of Rotograd in aligning the task gradients during training, consistently getting the most aligned gradient across methods. More subtly, Rotograd achieves this by following the direction of the actual gradient, as it does not directly interact with the gradients, which is in stark contrast to PCGrad, which directly modifies the gradient directions. Unsurprisingly, GradDrop obtains the same alignment as Vanilla MTL, as it only adds stochasticity to the gradients used to compute the average, while GradNorm stays in between, probably as a result of taking a better gradient direction (through proper weighting) but leaving the optimization landscape as it is.

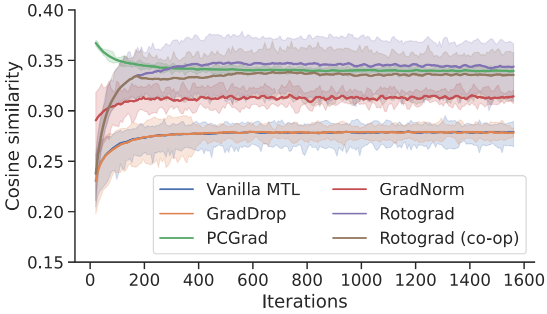


Figure 7: Cosine similarity during training on CIFAR10 for different methods, averaged over all tasks and five runs. The cosine is measured between the task gradient (after each method has been applied) and the update direction. Rotograd consistently obtains the best gradient alignment.

6 CONCLUSIONS

In this work, we have proposed Rotograd, a novel algorithm to alleviate the problem of gradient conflict (and thus negative transfer) in MTL. Rotograd homogenizes the gradient direction across all tasks by rotating, for each task, the space of shared-representations. We formulated Rotograd in terms of Stackelberg games, which allows to provide theoretical guarantees on the training stability of the model. Moreover, we reinterpreted GradNorm as a Stackelberg game for the first time, providing similar guarantees as for Rotograd. We empirically demonstrated our theoretical findings, as well as showed how Rotograd works on extreme scenarios and outperforms existing methods in usual MTL settings.

There are many interesting scenarios we would like to explore, for example, testing Rotograd in different applications such as computer vision, natural language processing, or reinforcement learning. Moreover, we believe that Rotograd can benefit other frameworks that suffer similar problems and future work could focus on adapting it to these settings. For example, problems related to conflicting gradients have been reported in meta learning [Flennerhag et al., 2019], and continual learning [Riemer et al., 2018].

References

Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In Proceedings of the Tenth International Conference on Machine Learning, pages 41–48. Morgan Kaufmann, 1993.

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

Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 2039–2050. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/16002f7a455a94aa4e91cc34ebdb9f2d-Paper.pdf`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

Tanner Fiez, Benjamin Chasnov, and Lillian J Ratliff. Convergence of learning dynamics in stackelberg games. arXiv preprint arXiv:1906.01217, 2019.

Sebastian Flennerhag, Andrei A Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. arXiv preprint arXiv:1909.00025, 2019.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pages 448–456. PMLR, 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, pages 7482–7491, 2018.

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

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

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

Hila Levi and Shimon Ullman. Multi-task learning by a top-down control network. arXiv preprint arXiv:2002.03335, 2020.

Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. arXiv preprint arXiv:1901.08428, 2019.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. arXiv preprint arXiv:1908.03265, 2019.

Ryan Murray, Brian Swenson, and Soummya Kar. Revisiting normalized gradient descent: Fast evasion of saddle points. arXiv preprint arXiv:1711.05224, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. arXiv preprint arXiv:1810.11910, 2018.

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

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In Advances in Neural Information Processing Systems, pages 527–538, 2018.

Trevor Standley, Amir R Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? arXiv preprint arXiv:1905.07553, 2019.

Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. Learning general purpose distributed sentence representations via large scale multi-task learning. arXiv preprint arXiv:1804.00079, 2018.

Mihai Suteu and Yike Guo. Regularizing deep multi-task networks using orthogonal gradients. arXiv preprint arXiv:1912.06844, 2019.

Sebastian Thrun and Joseph O'Sullivan. Discovering structure in multiple learning tasks: The tc algorithm. In ICML, volume 96, pages 489–497, 1996.

Simon Vandenhende, Stamatios Georgoulis, Bert De Brabandere, and Luc Van Gool. Branched multi-task networks: deciding what layers to share. arXiv preprint arXiv:1904.02920, 2019.

Sen Wu, Hongyang R Zhang, and Christopher Ré. Understanding and improving information transfer in multi-task learning. arXiv preprint arXiv:2005.00944, 2020.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 5824–5836. Curran Associates, Inc., 2020a. URL https://proceedings.neurips.cc/paper/2020/file/3fe78a8acf5fda99de95303940a2420c-Paper.pdf.

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

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

A ROTOGRAD: ALGORITHMIC DESCRIPTION

Here, we present a full description of Rotograd (combined with GradNorm) so that the entire algorithm can easily be found here in a concise and descriptive way. With this addition, we hope Rotograd to be adopted by the MTL community.

In short, Rotograd extend a hard-parameter sharing MTL architecture (that is, an architecture composed of a backbone and multiple task-specific heads), by including a set of rotation matrices R_k whose objective is to align the task gradients with respect to the last shared intermediate representation $\boldsymbol{Z}$. Similarly, GradNorm introduces a set of per-task weights ω_k and aims to equalize the per-task gradient magnitude during training. Specifically, each of these methods optimize:

$$\mathscr{L}^k_{\text{rot}}(R_k, \theta) := \frac{1}{B}\sum_{i=1}^{B} ||R_k^\top \boldsymbol{g}^t_{i,k} - \boldsymbol{v}^t_i||_2^2 \quad \text{with} \quad \boldsymbol{v}^t_i := \sum_k \frac{G^t_k}{||G^t_k||_2} ||G^t||_2 (s^t_k)^\tau, \tag{13}$$

$$\mathscr{L}^k_{\text{grad}}(\omega_k, \theta) := \left| \omega_k ||G^t_k||_2 - ||G^t||_2 \left(s^t_k\right)^\tau \right|, \tag{14}$$

where $\boldsymbol{g}^t_{i,k} := \nabla_{\boldsymbol{r}_k} \ell_k(\boldsymbol{r}^t_{i,k})$ is the gradient for the i-th instance for the k-th task at step t; $\boldsymbol{g}^t_i := \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}_i) = \sum_k \omega^t_k R_k^\top \boldsymbol{g}^t_{i,k}$ the average gradient for the i-th instance at step t (with respect to $\boldsymbol{z}$); G^t_k and G^t the same quantities, but considered as an entire vector over all instances (vectors of size dimension of $\boldsymbol{z}$ times batch size); $s^t_k := L_k(t)/[\frac{1}{K}\sum_j L_j(t)]$ measures whether the k-th task is being learned faster than the per-task average; and τ a hyper-parameter (α in the original formulation of GradNorm).

What this means in practice is simply that, along with the network parameters, we optimize the parameters R_k and ω_k using the gradient of the loss functions above and their own learning rates and schedulers. Regarding how these methods interact with the network (the forward pass), we simply need to compute the weighted sum of loss functions using the set of weights of GradNorm (that is, $\sum_k \omega_k L_k(t)$), and transform the intermediate representation $\boldsymbol{z}_i$ to $\boldsymbol{r}_{i,k}$ before feeding it to its task-specific head (that is, $\boldsymbol{r}_{i,k} = R_k \boldsymbol{z}_i$).

Algorithm 1 Forward and update methods using Rotograd with GradNorm.

```
procedure FORWARD(Input: $\boldsymbol{X}$)
    $\boldsymbol{Z}_i \leftarrow h_\theta(\boldsymbol{X})$
    for $k \leftarrow 1, 2, \ldots, K$ do
        $\boldsymbol{R}_k \leftarrow R_k \boldsymbol{Z}$                              ▷ Vectorized rotation (i.e., per instance $\boldsymbol{z}_i$)
        $L_k \leftarrow \sum_i \ell_{i,k}(\boldsymbol{r}_{i,k})$
    end for
    return $\sum_k \omega_k L_k$
end procedure

procedure UPDATE_LEADER(Losses: $\{L_k\}$, Gradients: $\{G_k\}_k$, step: $t$)      ▷ Gradients w.r.t. $\boldsymbol{r}_k$
    $G \leftarrow \sum_k \omega_k R_k^\top G_k$                          ▷ Assumes $G$ to be constant w.r.t. $R_k$ and $\omega_k$
    for $k \leftarrow 1, 2, \ldots, K$ do
        $s_k \leftarrow L_k / [\frac{1}{K}\sum_j L_j]$
        $\boldsymbol{v}_i \leftarrow \sum_k \frac{G_k}{||G_k||_2} ||G||_2 (s_k)^\tau$
        $\mathscr{L}^k_{\text{rot}} \leftarrow \frac{1}{B}\sum_{i=1}^{B} ||R_k^\top \boldsymbol{g}_{i,k} - \boldsymbol{v}_i||_2^2$
        $\mathscr{L}^k_{\text{grad}} \leftarrow \left| \omega_k ||G_k||_2 - ||G||_2 \left(s_k\right)^\tau \right|$
        $\omega_k \leftarrow \omega_k - \alpha_l \nabla_{\omega_k} \mathscr{L}^k_{\text{grad}}$
        if $t \leq T$ then
            $R_k \leftarrow R_k - \alpha_l \nabla_{R_k} \mathscr{L}^k_{\text{rot}}$
        else
            $R_k \leftarrow R_k - \alpha_l \left[ \frac{\nabla_{R_k}\mathscr{L}^k_{\text{rot}}}{||\nabla_{R_k}\mathscr{L}^k_{\text{rot}}||} + \frac{\nabla_{R_k} L_k}{||\nabla_{R_k} L_k||} \right] \frac{||\nabla_{R_k}\mathscr{L}^k_{\text{rot}}|| + ||\nabla_{R_k} L_k||}{2}$      ▷ Closed-form GradNorm with $\tau = 0$
        end if
    end for
end procedure
```

B DETAILS OF THE VISUALIZATION EXPERIMENTS

Here we describe all the details regarding the experiments of the Section B of the main paper.

B.1 REGRESSION TASKS

Functions. As mentioned in the main paper, we solve two different multi-task regression experiments of the form

$$L(\boldsymbol{x}) = \underbrace{f(R_1\boldsymbol{z},0)}_{L_1} + \underbrace{f(R_2\boldsymbol{z},1)}_{L_2}. \tag{15}$$

For the first experiment we choose a simple convex function of the form

$$f(\boldsymbol{x},s) := [x_1 + (-1)^s]^2 + 25x_2^2 \tag{16}$$

and, for the second experiment, we opt for a non-convex function of the form $f(\boldsymbol{z},s) := g(\boldsymbol{z} + (-1)^s(1.5,1.5))$, with

$$g(\boldsymbol{z}) := \frac{\sin(3z_1)}{z_1} + \frac{\sin(3z_2)}{z_2} + ||\boldsymbol{z}||_1. \tag{17}$$

Model. We use a simple two-layer ReLU feed-forward neural network with ten-neurons in the hidden layers, taking as input a two-dimensional vectors and output another two-dimensional vector. We assume that there is no task-specific heads, so that we evaluate the losses directly on the intermediate representations.

Training. We use as input a 256 datapoints sampled from a standard normal distribution and train the model for a total of 100 and 400 for the first and second example, respectively. For the model parameters we use SGD with a learning rate of 0.02, and for the parameters of Rotograd and GradNorm we use RAdam [Liu et al., 2019] with a learning rate of 0.5 for the first regression problem, and 0.1 for the second one. For visualization purposes and, as these examples are simple to solve, a high learning rate does not affect the stability of the training that much, and we can find the solution rapidly. Also, we use a exponential decay factor of 0.999 99.

Figures interpretation. We want to explain a bit more Figures 2a and 2b. In the former, we plot the training progress on the space of $\boldsymbol{Z}$, which is the result of applying the rotation matrices to the space rather than the points we optimize (also known as passive transformation), and therefore there is a single trajectory with varying level plots. In the latter, we combine the plot of the space of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ together, which is equivalent to applying the two different rotations to the trajectory while leaving the space untouched (this is known as an active transformation).

B.2 BINARY CLASSIFICATION TASK

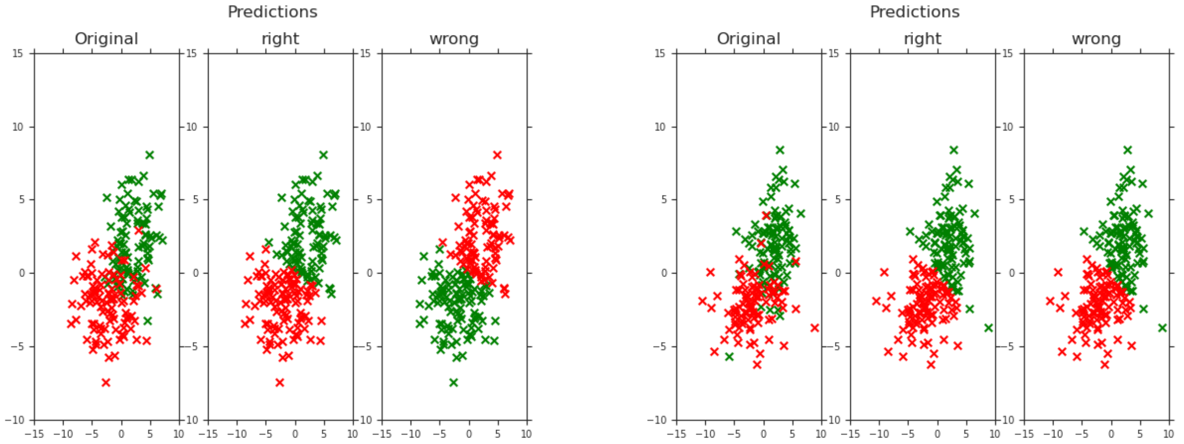
Data. We generate a synthetic dataset from a two-dimensional two-component Gaussian mixture model, using as label whether the instance was sampled from the first cluster. Specifically, we sample from:

$$X \sim \frac{1}{2}\mathcal{N}\left(\begin{pmatrix}2\\2\end{pmatrix}, \begin{pmatrix}5 & 1\\1 & 5\end{pmatrix}\right) + \frac{1}{2}\mathcal{N}\left(\begin{pmatrix}-2\\-2\end{pmatrix}, \begin{pmatrix}10 & 1\\1 & 3\end{pmatrix}\right), \tag{18}$$

and flip the labels of the second task with a probability of 10 % (to avoid completely cancelling the gradients out).

In Figure 8 we show the generated dataset, as well as the predictions for Rotograd and the vanilla case.

Model. We use as backbone a single linear layer with outputting a two-dimensional shared representation, thus, using a 2×2 matrix and a two-dimensional bias. As task-specific networks, we use another linear layer, this time outputting a single number which is passed through a sigmoid function to output a prediction. The head is shared across tasks, as otherwise the task is too easy to solve. As of excluding the heads and using a one-dimensional intermediate representation (similar as we did for the regression tasks), this would not work since the one dimensional case is ill-posed (there is a single rotation, the identity).



(a) Vanilla MTL.

(b) Rotograd.

Figure 8: Predictions for the methods. Left: original data. Center: prediction of the right label (first cluster). Right: prediction of the wrong/contrary label (second cluster). For the last two columns, green represents hits, and red misses.

Training. We use a batch size of 256 and train the model for 50 epochs. We use RAdam as optimizer, with a learning rate of 0.01 in the case of the follower, and 0.05 for the leader.

Figure. Regarding the x-axis in Fig. 3 of the main paper, we show the signed distance of each point to the point $(0,0)$, after having projected each point to the line connecting the centroids of both clusters.

C EXPERIMENTS DESCRIPTION

In this section we provide all the additional details regarding the experiments shown in the main paper. Besides, in Section D we show additional results omitted from the main manuscript due to space constraints.

As a rule of thumb for the experiments described here, all of them where trained on training data, the selected model was the best one obtained during training in terms of validation error, and results are shown with respect to test data. Also, losses are normalized with respect to their value in the first epoch but, as some losses decrease extremely fast (degrading some of the algorithms), we re-normalized using the value of the losses at the twentieth iteration.

Let us also use along this entire section the notation CONV-F-C for a convolutional layer of filter size F and number of channels C, MAX for a maxpool layer of filter size and stride 2, and DENSE-H for a dense layer with H output.

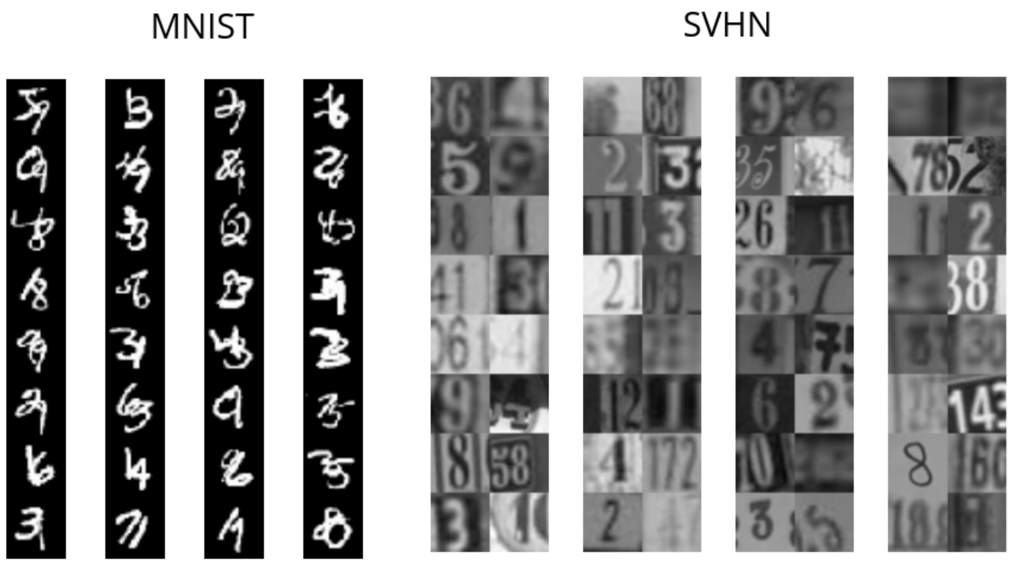
C.1 MNIST AND SVHN

Datasets. We use as datasets two modified versions of MNIST and SVHN where two digits are put together. In the case of MNIST, both of them are merged such that they form an overlapped image of 28×28. As SVHN contains backgrounds, we opted for pasting them together without overlapping, obtaining images of size $28 \times 28 \times 2$. Besides, SVHN images were transformed to gray-scale.

Tasks. The tasks, losses, and metrics we use are the following:

- *Left*. Classify the left digit. Loss: negative log-likelihodd loss. Metric: accuracy.
- *Right*. Classify the left digit. Loss: negative log-likelihodd loss. Metric: accuracy.
- *Sum*. Predict the sum of both digits. Loss: mean squared error loss. Metric: mean squared error.
- *Odd*. Classify the whether (binary) the left digit times the right one is an odd number. Loss: binary cross entropy. Metric: f1-score.

- *Active*. Predict the number of active pixels, that is, pixels with values higher than 0.5 (in the unit scale). Loss: mean squared error. Metric: mean squared error.



(a) Multi-MNIST.

(b) Multi-SVHN.

Figure 9: Some samples extracted from the modified datasets.

Architecture Our backbone is an adaption from the LeNet [LeCun et al., 1998] architecture used for MNIST. Specifically, we use the original and a reduced version for MNIST, and an adaption (due to the size differences) for SVHN. We use:

- MNIST. Normal size. [CONV-5-10][MAX][RELU][CONV-5-20][MAX][DENSE-50][RELU].
- SVHN. Normal size. [CONV-5-10][MAX][RELU][CONV-5-20][MAX][CONV-5-20][DENSE-50][RELU].
- MNIST. Reduced size. [CONV-3-3][MAX][RELU][CONV-3-5][MAX][DENSE-25][RELU].
- SVHN. Reduced size. [CONV-3-3][MAX][RELU][CONV-3-5][MAX][CONV-3-5][DENSE-25][RELU].

At the final layer of all backbones we use Batch Normalization [Ioffe and Szegedy, 2015].

Regarding the task specific heads, we use:

- Normal size. Regression. [DENSE-50][RELU][DENSE-1].
- Normal size. Classification. [DENSE-50][RELU][DENSE-10][LOG-SOFTMAX].
- Normal size. Binary classification. [DENSE-1][SIGMOID]
- Reduced size. Regression. [DENSE-25][RELU][DENSE-1].
- Reduced size. Classification. [DENSE-25][RELU][DENSE-10][LOG-SOFTMAX].
- Reduced size. Binary classification. [DENSE-1][SIGMOID]

Training. We run all models for a total of 300 epochs, using a batch size of 1024. As optimizer we use RAdam. For the follower we use a learning rate of 0.001, and for the leader a learning rate of 0.0005 and a exponential decay of 0.9999. In the case of the cooperative version of Rotograd, we set T to 20 epochs, that is, after 20 epochs the leader looks after the benefit of the follower as well.

C.2 CIFAR10

Dataset. We use CIFAR10 [Krizhevsky et al., 2009] as dataset. Additionally, every time we get a sample from the dataset we: i) crop the image by a randomly selected square of size 32×32; ii) randomly flip the image horizontally; and iii) standardize the image channel-wise using the mean and standard deviation estimators obtained on the training data.

Model. For the backbone we use a smaller version of ResNet18 [He et al., 2016] where, instead of having four layers each one with two basic building blocks, we have three layers each one with a single building block. In addition, we remove the last linear layer and add a Batch Normalization layer of size $d = 64$. For each of the task-specific heads, we simply use a linear layer followed by a sigmoid function, that is, [DENSE-1][SIGMOID].

Losses and metrics. We treat each class as a binary classification task where we use as loss the binary cross entropy loss, and we use f1-score as metric.

Training. During training, we use a batch size of 128 and train the model for 500 epochs. Regarding the network parameters, we use SGD with momentum, a learning rate of 0.01, and use a cosine scheduler with a period of 200 iterations. For the leader's parameters, we use RAdam as optimizer, with a learning rate of 1×10^{-5} and an exponential decay factor of 0.999 999. For the case of cooperative Rotograd, we set $T = 50$ epochs.

D ADDITIONAL EXPERIMENTAL RESULTS

D.1 LEADER'S LEARNING SPEED

We complement the results shown in the main paper (Fig. 5) with: i) Table 2, which shows the results from which we created the figure of the main paper (best model on validation error), and the results obtained with the final model obtained after finishing the training; ii) Fig. 10 shows the relative performance improvement for all tasks.

	Learning rate	Left (%) ↑	Right (%) ↑	Sum ↓	Odd (%) ↑	Active ↓	Δ_{MTL} (%) ↑
	Vanilla MTL	92.56 ± 2.80	89.59 ± 4.10	3.67 ± 1.66	89.48 ± 4.41	0.44 ± 0.25	
Best model	1×10^{-1}	89.64 ± 0.96	86.51 ± 1.71	5.45 ± 0.42	80.67 ± 4.19	0.11 ± 0.01	-0.02 ± 0.11
	5×10^{-2}	90.68 ± 0.57	87.80 ± 1.10	4.93 ± 0.38	80.61 ± 2.12	0.12 ± 0.04	0.02 ± 0.11
	1×10^{-2}	92.55 ± 0.54	90.12 ± 0.65	3.79 ± 0.22	86.19 ± 1.44	0.07 ± 0.01	0.13 ± 0.08
	5×10^{-3}	92.92 ± 0.42	90.31 ± 0.50	3.52 ± 0.27	87.97 ± 1.38	0.07 ± 0.00	0.15 ± 0.08
	1×10^{-3}	92.31 ± 2.80	89.55 ± 4.29	3.67 ± 1.62	88.57 ± 4.08	0.18 ± 0.33	0.14 ± 0.05
	5×10^{-4}	93.27 ± 0.42	90.97 ± 0.61	3.00 ± 0.12	90.49 ± 0.78	0.07 ± 0.00	0.19 ± 0.07
	1×10^{-4}	93.42 ± 0.20	90.98 ± 0.38	3.01 ± 0.07	90.97 ± 0.79	0.14 ± 0.05	0.15 ± 0.08
	Vanilla MTL	93.34 ± 0.34	90.94 ± 0.60	3.15 ± 0.16	91.11 ± 0.74	0.34 ± 0.05	
Latest model	1×10^{-1}	80.66 ± 14.57	77.01 ± 16.23	8.08 ± 5.43	76.74 ± 7.49	14.75 ± 43.94	-7.29 ± 21.50
	5×10^{-2}	79.24 ± 19.53	76.16 ± 19.88	7.16 ± 2.82	66.65 ± 10.72	0.10 ± 0.02	-0.23 ± 0.28
	1×10^{-2}	92.40 ± 0.65	89.87 ± 0.78	4.16 ± 0.82	85.30 ± 4.08	0.08 ± 0.01	0.07 ± 0.06
	5×10^{-3}	92.66 ± 0.47	90.03 ± 0.63	3.64 ± 0.32	87.11 ± 1.91	0.07 ± 0.01	0.11 ± 0.02
	1×10^{-3}	93.11 ± 0.46	90.64 ± 0.59	3.20 ± 0.18	89.28 ± 1.61	0.07 ± 0.00	0.15 ± 0.01
	5×10^{-4}	93.15 ± 0.49	90.82 ± 0.60	3.03 ± 0.12	90.43 ± 0.96	0.08 ± 0.01	0.16 ± 0.01
	1×10^{-4}	93.37 ± 0.20	91.01 ± 0.41	3.03 ± 0.07	90.86 ± 0.86	0.14 ± 0.05	0.13 ± 0.01

Table 2: Test results obtained for different learning rates of the leader.

D.2 MNIST AND SVHN

We show in Table 3 the complete results of relative improvements for all combinations of model sizes and datasets.

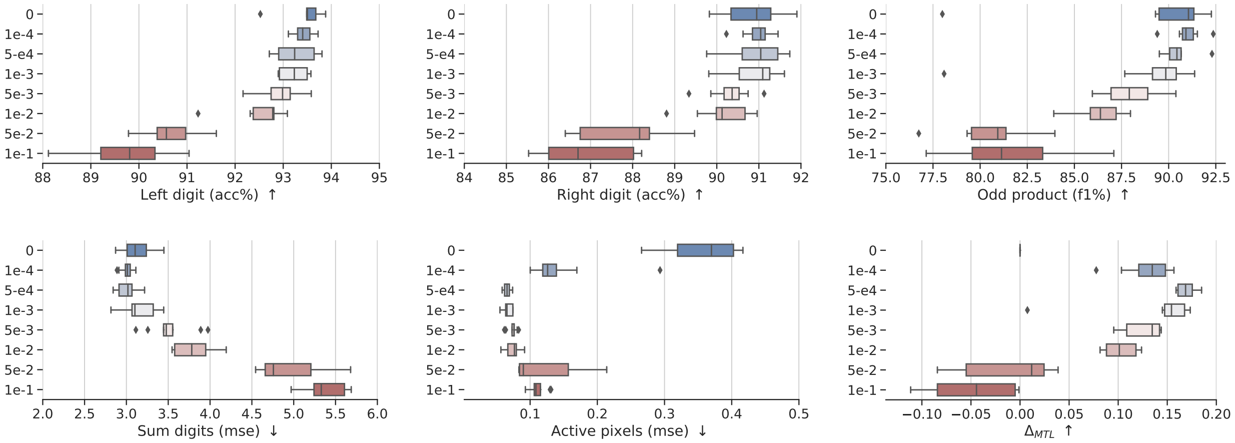
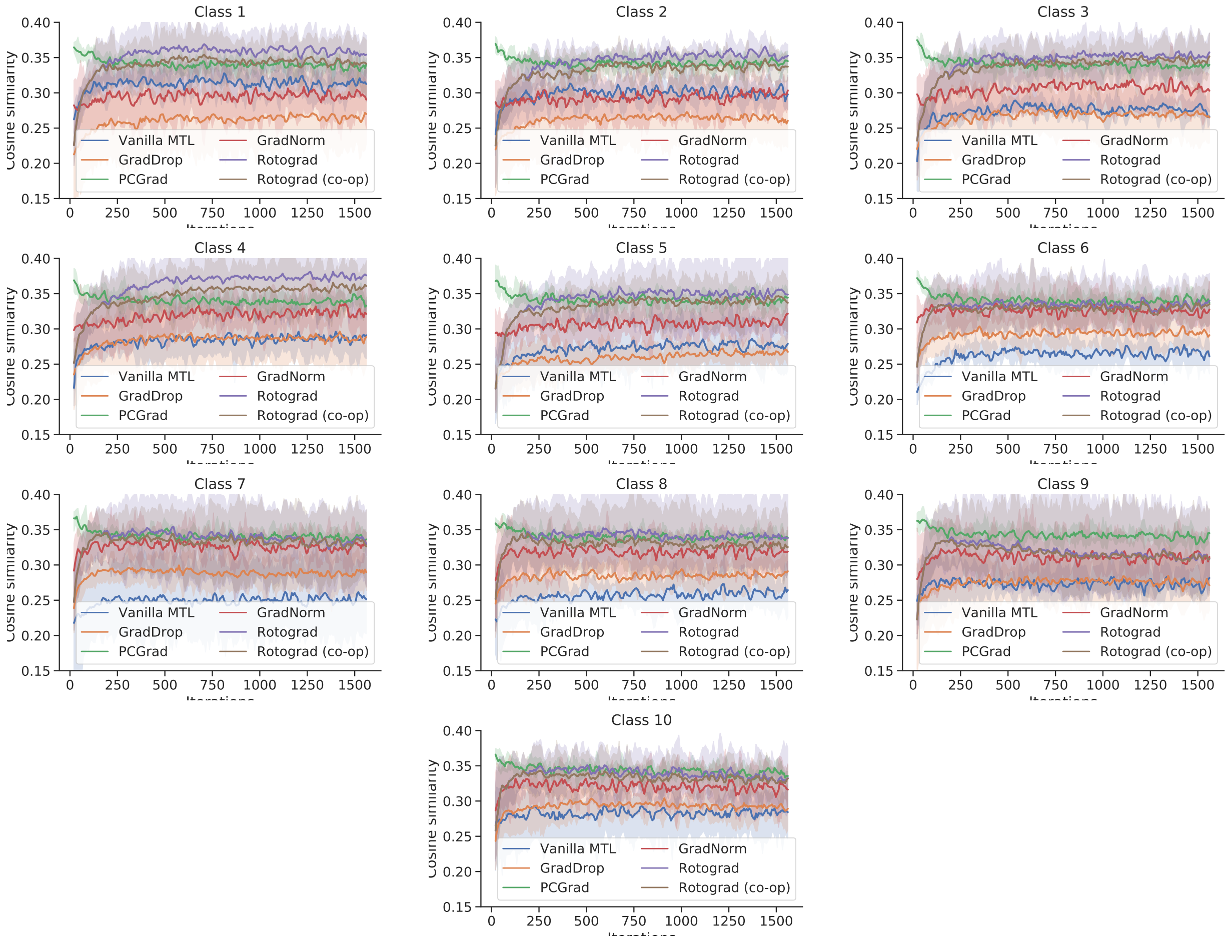


Figure 10: Box-plot with the relative improvement of Rotograd when we change the learning rate of the leader. Results are based on the model with best validation error during training.

D.3 CIFAR10

We show the cosine similarity per-class (rather than averaged over all classes) in Fig. 11. Rotograd is able to improve the cosine similarity consistently for all classes. As mentioned in the manuscript, the only similar method is PCGrad, but it achieves similar cosine similarity as Rotograd *after removing the projection of one gradient onto another*.


Class 1

Class 2

Class 3

Class 4

Class 5

Class 6

Class 7

Class 8

Class 9

Class 10

Figure 11: Cosine similarity during training on CIFAR10 for different methods, averaged over five runs. The cosine is measured between the task gradient (after each method has been applied) and the update direction. Rotograd consistently obtains the best gradient alignment.

	Method	Left (%)	Right (%)	Sum	Odd (%)	Active	Δ_{MTL} (%)
MNIST Small LeNet	Vanilla MTL	93.48 ± 0.46	90.73 ± 0.39	3.18 ± 0.23	91.19 ± 0.55	0.35 ± 0.06	
	GradDrop	−0.18 ± 0.24	−0.02 ± 0.53	−1.36 ± 5.29	−0.30 ± 0.94	2.51 ± 15.99	0.13 ± 3.43
	PCGrad	0.04 ± 0.28	0.16 ± 0.37	0.76 ± 4.19	0.08 ± 0.66	11.59 ± 10.87	2.53 ± 2.36
	GradNorm	−0.03 ± 0.35	0.10 ± 0.19	1.21 ± 3.88	−0.21 ± 0.52	77.67 ± 4.95	15.75 ± 0.95
	Rotograd	−0.19 ± 0.38	0.53 ± 0.60	7.04 ± 4.77	−0.12 ± 0.72	79.49 ± 4.77	17.35 ± 1.71
	Rotograd (co-op)	−0.18 ± 0.39	0.43 ± 0.50	3.93 ± 4.90	0.09 ± 0.86	79.50 ± 5.50	16.76 ± 1.83
SVHN Small LeNet	Vanilla MTL	73.52 ± 3.80	73.75 ± 3.95	7.33 ± 0.43	69.41 ± 2.64	10.97 ± 5.12	
	GradDrop	1.20 ± 2.00	1.15 ± 3.66	1.89 ± 4.43	−4.61 ± 5.94	−17.92 ± 51.02	−3.66 ± 10.17
	PCGrad	0.96 ± 1.59	−0.11 ± 1.91	0.43 ± 2.30	−0.09 ± 2.74	−5.59 ± 38.46	−0.88 ± 8.07
	GradNorm	0.76 ± 2.11	−0.45 ± 2.59	−0.40 ± 4.38	−3.56 ± 4.34	−12.52 ± 43.55	−3.23 ± 8.66
	Rotograd	2.17 ± 5.50	2.06 ± 6.84	3.21 ± 4.92	1.51 ± 5.04	13.75 ± 40.81	4.54 ± 10.16
	Rotograd (co-op)	2.54 ± 6.53	2.31 ± 4.97	3.52 ± 6.22	1.56 ± 4.94	25.24 ± 27.97	7.04 ± 7.88
MNIST Normal LeNet	Vanilla MTL	95.27 ± 0.21	93.50 ± 0.28	2.07 ± 0.11	93.20 ± 0.73	0.12 ± 0.03	
	GradDrop	−0.05 ± 0.34	0.08 ± 0.42	−1.94 ± 5.88	0.03 ± 0.99	−16.70 ± 26.85	−0.04 ± 0.05
	PCGrad	−0.11 ± 0.30	−0.01 ± 0.36	−0.35 ± 7.53	−0.14 ± 0.96	1.27 ± 27.47	0.00 ± 0.05
	GradNorm	−0.05 ± 0.29	0.00 ± 0.31	−3.89 ± 5.22	−0.19 ± 0.88	34.73 ± 16.38	0.06 ± 0.03
	Rotograd	−0.07 ± 0.24	0.01 ± 0.31	4.44 ± 6.08	0.37 ± 0.82	28.24 ± 22.76	0.07 ± 0.04
	Rotograd (co-op)	−0.05 ± 0.14	−0.01 ± 0.32	8.52 ± 7.28	0.60 ± 0.82	40.69 ± 14.45	0.10 ± 0.04
SVHN Normal LeNet	Vanilla MTL	84.22 ± 0.39	84.33 ± 0.49	4.80 ± 0.10	80.06 ± 1.09	4.11 ± 2.04	
	GradDrop	0.08 ± 0.71	−0.07 ± 0.83	1.96 ± 3.02	−0.15 ± 1.64	−65.12 ± 160.54	−12.66 ± 32.29
	PCGrad	−0.30 ± 0.71	−0.27 ± 0.68	−0.39 ± 3.15	−1.02 ± 1.57	−54.33 ± 118.96	−11.26 ± 24.22
	GradNorm	0.12 ± 0.39	−0.13 ± 0.49	0.46 ± 3.43	−0.18 ± 1.68	−9.69 ± 59.39	−1.89 ± 11.97
	Rotograd	−0.05 ± 0.31	−0.18 ± 0.78	0.84 ± 3.15	−0.66 ± 1.98	19.14 ± 41.71	3.82 ± 8.56
	Rotograd (co-op)	−0.31 ± 0.65	−0.53 ± 0.47	0.03 ± 3.42	−0.62 ± 1.54	6.32 ± 52.71	0.98 ± 10.95

Table 3: Test results (mean and standard deviation) for different methods, averaged over ten different runs on MNIST, SVHN, and two different model sizes for LeNet.