

529 Appendices

530 A Proofs

531 **Proposition A.1** (Proposition 3.1 in the main text). *Suppose $f_k := L_k \circ h_k$ is an infinitely differen-*
532 *tiabile real-valued function, and let us call $\mathbf{G}_k = \nabla_{\mathbf{Z}} f_k(\mathbf{Z})$ its derivative with respect to \mathbf{Z} , for every*
533 *$k = 1, 2, \dots, K$. If $\cos_sim(\mathbf{G}_i, \mathbf{G}_j) > -1/(K-1)$ pairwise; then there exists a small-enough step*
534 *size $\varepsilon > 0$ such that, for all k , we have that $L_k(h_k(\mathbf{Z} - \varepsilon \cdot C \sum_k \mathbf{U}_k; \phi_k); \mathbf{Y}_k) < L_k(h_k(\mathbf{Z}; \phi_k); \mathbf{Y}_k)$,*
535 *where $\mathbf{U}_k := \mathbf{G}_k / \|\mathbf{G}_k\|$ and $C \geq 0$.*

536 **Proof.** Since f_k is infinitely differentiable, we can take the first-order Taylor expansion of f_k around
537 \mathbf{Z} , for any k , evaluated at $\mathbf{Z} - \varepsilon \mathbf{V}$ for a given vector \mathbf{V} :

$$f_k(\mathbf{Z} - \varepsilon \mathbf{V}) = f_k(\mathbf{Z}) - \varepsilon \langle \mathbf{G}_k, \mathbf{V} \rangle + o(\varepsilon). \quad (7)$$

538 In our case, $\mathbf{V} = C \sum_k \mathbf{U}_k$ with $C \geq 0$, then:

$$f_k(\mathbf{Z} - \varepsilon \mathbf{V}) - f_k(\mathbf{Z}) = -\varepsilon \cdot C \|\mathbf{G}_k\| \sum_i \langle \mathbf{U}_k, \mathbf{U}_i \rangle + o(\varepsilon) \quad (8)$$

$$= -\varepsilon \cdot C \|\mathbf{G}_k\| \left(1 + \sum_{i \neq j} \langle \mathbf{U}_k, \mathbf{U}_i \rangle \right) + o(\varepsilon). \quad (9)$$

539 Since $\|\mathbf{U}_k\| = 1$ for all $k = 1, 2, \dots, K$, it holds that $-1 \leq \cos_sim(\mathbf{U}_k, \mathbf{U}_i) = \langle \mathbf{U}_k, \mathbf{U}_i \rangle \leq 1$.

540 If $\cos_sim(\mathbf{G}_k, \mathbf{G}_i) > -1/(K-1)$ for all $i \neq k$, then we have that $1 + \sum_{i \neq j} \langle \mathbf{U}_k, \mathbf{U}_i \rangle > 0$ and
541 $f_k(\mathbf{Z} - \varepsilon \mathbf{V}) < f_k(\mathbf{V})$ for a small enough $\varepsilon > 0$.

542 Q.E.D.

543 B Stackelberg games and RotoGrad

544 In game theory, a Stackelberg game [10] is an *asymmetric game* where two players play alternately.
545 One of the players—whose objective is to blindly minimize their loss function—is known as the
546 follower, \mathcal{F} . The other player is known as the leader, \mathcal{L} . In contrast to the follower, the leader
547 attempts to minimize their own loss function, but it does so with the advantage of knowing which
548 will be the best response to their move by the follower. The problem can be written as

$$\begin{aligned} \text{Leader: } \min_{x_l \in X_l} \{ \mathcal{L}(x_l, x_f) \mid x_f \in \operatorname{argmin}_{y \in X_f} \mathcal{F}(x_l, y) \}, \\ \text{Follower: } \min_{x_f \in X_f} \mathcal{F}(x_l, x_f), \end{aligned} \quad (10)$$

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

550 While traditionally one assumes that players make perfect alternate moves in each step of problem 10,
551 *gradient-play Stackelberg games* assume instead that players perform simultaneous gradient updates,

$$\begin{aligned} x_l^{t+1} &= x_l^t - \alpha_l^t \nabla_{x_l} \mathcal{L}(x_l, x_f), \\ x_f^{t+1} &= x_f^t - \alpha_f^t \nabla_{x_f} \mathcal{L}(x_l, x_f), \end{aligned} \quad (11)$$

552 where α_l and α_f are the learning rates of the leader and follower, respectively.

553 An important concept in game theory is that of an equilibrium point, that is, a point in which
554 both players are satisfied with their situation, meaning that there is no available move immediately
555 improving any of the players' scores, so that none of the players is willing to perform additional
556 actions/updates. Specifically, we focus on the following definition introduced by Fiez et al. [10]:

Definition B.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 $\nabla_{x_f} \mathcal{F}(x_l^*, x_f^*) = 0$, is a differential Stackelberg equilibrium point if $\nabla_{x_l} \mathcal{L}(x_l^*, r(x_l^*)) = 0$, and $\nabla_{x_l}^2 \mathcal{L}(x_l^*, r(x_l^*))$ 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 of the following result, introduced by Fiez et al. [10], to provide theoretical convergence guarantees to an equilibrium point:

Proposition B.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. [10] extended this result to the finite-time case, showing that the game will end close to an equilibrium point with high probability.

As we can observe, the Stackelberg formulation in Equation (10) is really similar to RotoGrad’s formulation in Equation (4). Even more, the update rule in Equation (11) is quite similar to that one shown in Algorithm 1. Therefore, it is sensible to cast RotoGrad as a Stackelberg game. One important but subtle bit about this link regards the extra information used by the leader. In our case, this extra knowledge explicitly appears in Equation 3 in the form of the follower’s gradient, $\mathbf{g}_{i,k}$, which is the direction the follower will follow and, as it is performing first-order optimization by assumption, this gradient directly encodes the follower’s response.

Thanks to the Stackelberg formulation in Equation 4 we can make use of Prop. B.1 and, thus, draw theoretical guarantees on the training stability and convergence. In other words, we can say that performing training steps as those described in Algorithm 1 will stably converge as long as the leader is asymptotically the slow learner, that is $\alpha_l^t = o(\alpha_f^t)$, where o denotes the little-o notation.

C Experiments

C.1 Experimental setups

Here we discuss common settings across all experiments. Refer to specific sections further below for details concerning single experiments.

Computational resources. All experiments were performed on a shared cluster system with two NVIDIA DGX-A100. Therefore, all experiments were run with (up to) 4 cores of AMD EPYC 7742 CPUs and, for those trained on GPU (CIFAR10, CelebA, and NYUv2), a single NVIDIA A100 GPU. All experiments were restricted to 12 GB of RAM.

Loss normalization. Similar as in the gradient case studied in this work, magnitude differences between losses can make the model overlook some tasks. To overcome this issue, here we perform loss normalization, that is, we normalize all losses by their value at the first training iteration (so that they are all 1 in the first iteration). To account for some losses that may quickly decrease at the start, after the 20th iteration we instead normalize losses dividing by their value at that iteration.

Checkpoints. For the single training of a model, we select the parameters of the model by taking those that obtained the best validation error after each training epoch. That is, after each epoch we evaluate the linearly-scalarized validation loss, $\sum_k L_k$, and use the parameters that obtained the best error during training. This can be seen as an extension of early-stopping where, instead of stopping, we keep training until reaching the maximum number of epochs hoping to keep improving.

Hyperparameter tuning. Model-specific hyperparameters were mostly selected by a combination of: i) using values described in prior works; and ii) empirical validation on the vanilla case (without any gradient-modifiers) to verify that the combinations of hyperparameters work. To select method-specific hyper-parameters we performed a grid search, choosing those combinations of values that performed the best with respect to validation error.

Specifically, we took $\alpha \in \{0, 0.5, 1, 2\}$ and $\mathbf{R}_k \in \mathbb{R}^{m \times m}$ with $m \in \{0.25d, 0.5d, 0.75d, d\}$ (restricting ourselves to $m \leq 1500$) for RotoGrad. Regarding the learning rate of RotoGrad (GradNorm), we performed a grid search considering $\eta_{\text{roto}} \in \{0.05\eta, 0.1\eta, 0.5\eta, \eta, 2\eta\}$, where η_{roto} and η are the learning rates of RotoGrad (GradNorm) and the network, respectively.

Notation. Let us also use along this section the following notation to describe different architectures: [CONV- F - C] denotes a convolutional layer with filter size F and C number of channels; [MAX] denotes a max-pool layer of filter size and stride 2, and [DENSE- H] a dense layer with output size H .

C.1.1 Illustrative examples

Losses and metrics. Both illustrative experiments use MSE as both loss and metric. Regarding the specific form of φ in Equation (5), the avocado-shaped experiments uses

$$\varphi((x, y), s) = (x - s)^2 + 25y^2, \quad (12)$$

while the non-convex second experiment uses

$$\varphi((x, y), s) = -\frac{\sin(3x + 4.5s)}{x + 1.5s} - \frac{\sin(3y + 4.5s)}{y + 1.5s} + |x + 1.5s| + |y + 1.5s| \quad (13)$$

Model. As described in the main manuscript, we use a single input $\mathbf{x} \in \mathbb{R}^2$ picked at random from a standard normal distribution, and drop all task-specific network parameters (that is, $h_k(\mathbf{r}_k) = \mathbf{r}_k$). As backbone, we take a simple network of the form $\mathbf{z} = \mathbf{W}_2 \max(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1, 0) + \mathbf{b}_2$ with $\mathbf{b}_1 \in \mathbb{R}^{10}$, $\mathbf{b}_2 \in \mathbb{R}^2$, and $\mathbf{W}_1, \mathbf{W}_2^\top \in \mathbb{R}^{10 \times 2}$.

Hyper-parameters, convex-case. We train the model for one hundred epochs. As network optimizer we use stochastic gradient descent (SGD) with a learning rate of 0.01. For the rotations we use RAdam [21] with a learning rate of 0.5 (for visualization purposes we need a high learning rate, in such a simple scenario it still converges) and exponential decay with decaying factor 0.99999.

Hyper-parameter, non-convex case. For the second experiment, we train the model for 400 epochs and, once again, use SGD as the network optimizer with a learning rate of 0.015. For the rotations, we use RAdam [21] with a learning rate of 0.1 and an exponential decay of 0.99999.

C.1.2 MNIST/SVHN

Datasets. We use two modified versions of MNIST [18] and SVHN [29] in which each image has two digits, one on each side of the image. In the case of MNIST, both of them are merged such that they form an overlapped image of 28×28 , as shown in Figure 5a. Since SVHN contains backgrounds, we simply paste two images together without overlapping, obtaining images of size 32×64 , as shown in Figure 5b. Moreover, we transform all SVHN samples to grayscale.

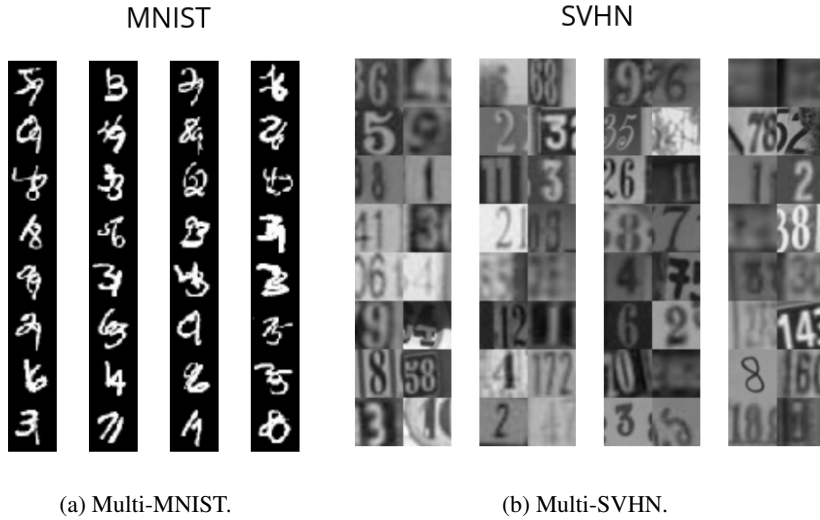


Figure 5: Samples extracted from the modified MNIST and SVHN datasets.

Tasks, losses, and metrics. In order to further clarify the setup used, here we describe in detail each task. Specifically, we have:

- **Left digit** classification. Loss: negative log-likelihood (NLL). Metric: accuracy (ACC).
- **Right digit** classification. Loss: NLL. Metric: ACC.
- **Parity** of the product of digits, that is, whether the product of both digits gives an odd number (binary-classification). Loss: binary cross entropy (BCE). Metric: f1-score (F1).
- **Sum** of both digits (regression). Loss: MSE. Metric: MSE.
- **Active pixels** in the image, that is, predict the number of pixels with values higher than 0.5, where we use pixels lying in the unit interval (regression). Loss: MSE. Metric: MSE.

Model. Our backbone is an adaption from the original LeNet [19] model. Specifically, we use:

- **MNIST.** [CONV-5-10][MAX][RELU][CONV-5-20][MAX][DENSE-50][RELU][BN],
- **SVHN.** [CONV-5-10][MAX][RELU][CONV-5-20][MAX][CONV-5-20][DENSE-50][RELU][BN],

where [BN] refers to Batch Normalization [14]. Depending on the type of task, we use a different head. Specifically, we use:

- **Regression.** [DENSE-50][RELU][DENSE-1],
- **Classification.** [DENSE-50][RELU][DENSE-10][LOG-SOFTMAX],
- **Binary-classification.** [DENSE-1][SIGMOID].

Model hyper-parameters. For both datasets, we train the model for 300 epochs using a batch size of 1024. For the network parameters, we use RAdam [21] with a learning rate of $1e-3$.

Methods hyper-parameters. In Tables 2 and 5 we show the results of GradNorm with $\alpha = 1$ and $\alpha = 0$ for MNIST and SVHN, respectively. We train RotoGrad with full-size rotation matrices ($m = d$). Both methods use RAdam with learning rate $5e-4$ and exponential decay of 0.9999.

C.1.3 CIFAR10

Dataset. We use CIFAR10 [17] as dataset, with 40 000 instances as training data and the rest as testing data. Additionally, every time we get a sample from the dataset we: i) crop the image by a randomly selected square of size $3 \times 32 \times 32$; 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. We take as backbone ResNet18 [13] without pre-training, where we remove the last linear and pool layers. In addition, we add a Batch Normalization layer. For each task-specific head, we simply use a linear layer followed by a sigmoid function, that is, [DENSE-1][SIGMOID].

Losses and metrics. We treat each class (out of ten) as a binary-classification task where we use BCE and F1 as loss and metric, respectively.

Model hyper-parameters. We use a batch size of 128 and train the model for 500 epochs. For the network parameters, we use as optimizer SGD with learning rate of 0.01, Nesterov momentum of 0.9, and a weight decay of $5e-4$. Additionally, we use for the network parameters a cosine learning-rate scheduler with a period of 200 iterations.

Methods hyper-parameters. Results shown in Tables 1 and 6 use $\alpha = 2$ for GradNorm and, as optimizer, we use RAdam [21] with learning rate 0.001 and an exponential decay factor of 0.99995 for both GradNorm and RotoGrad.

C.1.4 NYUv2

Setup. In contrast with the rest of experiments, for the NYUv2 experiments shown in Section 6, instead of writing our own implementation, we slightly modified the open-source code provided by Liu et al. [22] at <https://github.com/lorenmt/mtan> (commit 268c5c1). We therefore use the exact same setting as Liu et al. [22]—and refer to their paper and code for further details, with the addition of using data augmentation for the experiments which, although not described in the paper, is included in the repository as a command-line argument. We will provide along this work a diff file to include all gradient-modifier methods into the aforementioned code.

679 **Methods hyper-parameters.** For the results shown in Table 3 we use GradNorm with $\alpha = 1$ and
680 RotoGrad with rotations \mathbf{R}_k of size 1024. We use a similar optimization strategy as the rest of
681 parameters, using Adam [16] with learning rate $5e-5$ (half the one of the network parameters) and
682 where we halve this learning rate every 100 iterations.

683 C.1.5 CelebA

684 **Dataset.** We use CelebA [23] as dataset with usual splits. We resize each sample image so that they
685 have size $3 \times 64 \times 64$.

686 **Losses and metrics.** We treat each class (out of forty) as a binary-classification task where we use
687 BCE and F1 as loss and metric, respectively.

688 **ResNet model.** As with CIFAR10, we use as backbone ResNet18 [13] without pre-training, where
689 we remove the last linear and pool layers. In addition, we add a Batch Normalization layer. For each
690 task-specific head, we use a linear layer followed by a sigmoid function, that is, [DENSE-1][SIGMOID].

691 **ResNet hyper-parameters.** We use a batch size of 256 and train the model for 100 epochs. For the
692 network parameters, we use Radam [21] as optimizer with learning rate 0.001 and exponential decay
693 of 0.999 95 applied every 2400 iterations.

694 **Convolutional model.** For the second architecture we use a convolutional network as back-
695 bone, [CONV-3-64][BN][MAX][CONV-3-128][BN][CONV-3-128][BN][MAX][CONV-3-256][BN][CONV-
696 3-256][BN][MAX][CONV-3-512][BN][DENSE-512][BN]. For the task-specific heads, we take a simple
697 network of the form [DENSE-128][BN][DENSE-1][SIGMOID].

698 **Convolutional hyper-parameters.** We use a batch size of 8 and train the model for 20 epochs. For
699 the network parameters, we use Radam [21] as optimizer with learning rate 0.001 and exponential
700 decay of 0.96 applied every 2400 iterations.

701 **Methods hyper-parameters.** Results shown in Tables 4 and 7 use GradNorm with $\alpha = 1$ and $\alpha = 2$
702 for the convolutional and residual models, respectively. For RotoGrad, we rotate 256 and 1536
703 elements of \mathbf{z} for the convolutional and residual networks. As optimizer, we use RADam [21] with
704 learning rate $5e-6$ and an exponential decay factor of 0.999 95 for both GradNorm and RotoGrad.

705 C.2 Additional results

706 C.2.1 Illustrative examples

707 We complement the illustrative figures
708 shown in Figure 1 by providing, for each
709 example, an illustration of the effect of
710 RotoGrad shown as an active and passive
711 transformation. In an active transformation
712 (Figure 6 left), points in the space are the
713 ones modified. In our case, this means that
714 we rotate feature \mathbf{z} , obtaining \mathbf{r}_1 and \mathbf{r}_2 ,
715 while the loss functions remain the same.
716 In other words, for each \mathbf{z} we obtain a task-
717 specific feature \mathbf{r}_k that optimizes its loss
718 function. In contrast, a passive transforma-
719 tion (Figure 6 right) keeps the points unal-
720 tered while applying the transformation to
721 the space itself. In our case, this translates
722 to rotating the optimization landscape of
723 each loss function (now we have $L_k \circ \mathbf{R}_k$
724 instead of L_K), so that our single feature \mathbf{z}
725 has an easier job at optimizing both tasks.
726 In the case of RotoGrad, we can observe
727 in both right figures that both optima lie
728 in the same point, as we are aligning task
729 gradients.

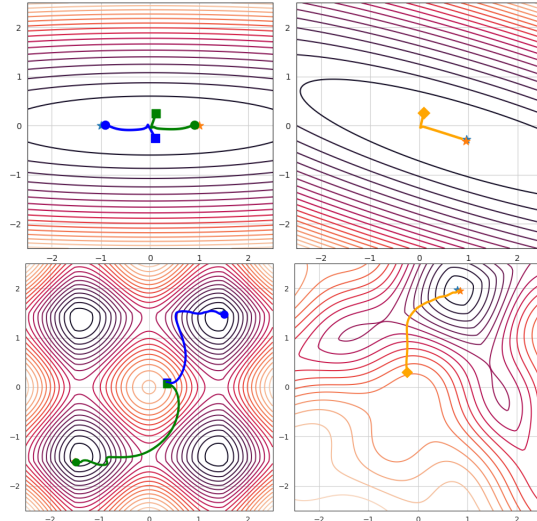


Figure 6: Level plots showing the illustrative examples of Figure 1 for RotoGrad. Top: Convex case. Bottom: Non-convex case. Left: Active transformation (trajectories of \mathbf{r}_k and the level plot of $L_1 + L_2$. Right: Passive transformation (trajectory of \mathbf{z} and level plot of $(L_1 \circ \mathbf{R}_1) + (L_2 \circ \mathbf{R}_2)$).

Besides the two regression experiments shown in Section 4, we include here an additional experiment where we test RotoGrad in the worst-case scenario of gradient conflict, that is, one in which task gradients are opposite to each other. To this end, we solve a 2-task binary classification problem where, as dataset, we take 1000 samples from a 2D Gaussian mixture model with two clusters; $y_{n,k} = 1$ if x_n was sampled from cluster k ; and $y_{n,k} = 0$ otherwise. We use as model a logistic regression model of the form $y_k = \mathbf{W}_2 \max(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1, 0) + \mathbf{b}_2$ with $\mathbf{b}_1 \in \mathbb{R}^2$, $\mathbf{b}_2 \in \mathbb{R}$, $\mathbf{W}_1 \in \mathbb{R}^{2 \times 2}$, and $\mathbf{W}_2 \in \mathbb{R}^{1 \times 2}$. Because rotations in 1D are ill-posed (there is a unique proper rotation), here we add task parameters to increase the dimensionality of z and make *all parameters shared*, so that there is still no task-specific parameters. To avoid a complete conflict where $\nabla_z L_1 + \nabla_z L_2 = 0$, we randomly flip the labels for the second tasks with 5 % probability. Figure 7 shows that, in this extreme scenario, RotoGrad is able to learn both tasks by aligning gradients, that is, by learning that one rotation is the inverse of the other $\mathbf{R}_1 = \mathbf{R}_2^\top$.

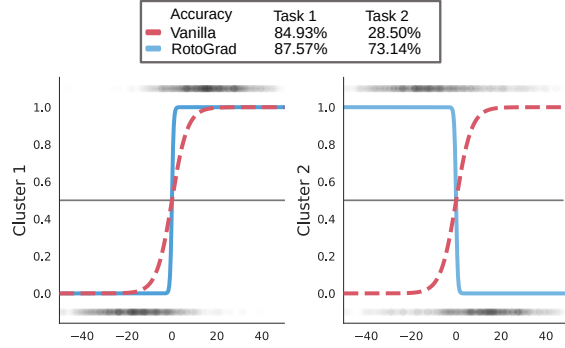


Figure 7: Logistic regression for opposite classification tasks. Test data is plotted scattered as gray dots. RotoGrad learns both opposite rotations $\mathbf{R}_1 = \mathbf{R}_2^\top$.

C.2.2 Training stability

While we showed in Section 6.1 only the results for the sum-of-digits task as they were nice and clear, here we show in Figure 8 the results of those same experiments in Section 6.1 for all the different tasks. The same discussion from the main manuscript can be carried out for all metrics. Additionally, we can observe that the vanilla case (learning rate zero) completely overlooks the image-related task (*Active pixels*) while performing the best in the parity task.

Additionally, let us clarify what we mean here with stability, as in the main manuscript we mainly talked about convergence guarantees. In these experiments we measure the convergence guarantees of the experiments in terms of ‘training stability’, meaning the variance of the obtained results across different runs. The intuition here is that, since the model does not converge, we should expect some wriggling learning curves during training and, as we take the model with the best validation error, the individual task metrics should have bigger variance (that is, less stability) across runs.

C.2.3 MNIST/SVHN

Since we present in Section 6 grouped results in MNIST and SVHN in terms of digit-related and image-related tasks, here we provide the complete results table for all metrics in Table 5. In the case of MNIST, we can observe that both regression tasks tend to be quite disruptive. GradNorm, IMTL-G, and RotoGrad manage to improve over all tasks while maintaining good performance on the rest of tasks. MGDA, however, focuses on the image-related task too much and overlooks other tasks. In SVHN we observe a similar behavior. This time, all methods are able to leverage positive transfer and improve their results on the parity and sum tasks, obtaining similar task improvement results. Yet, the image-related task is more disruptive than before, showing bigger differences between methods. As before, MGDA completely focuses on this task, but now is able to not overlook any task while doing so. Regarding the rest of the methods, all of them improved their results with respect to the vanilla case, with RotoGrad and GradNorm obtaining the second-best results.

C.2.4 CIFAR10 and CelebA

For the sake of completeness, we present in Table 6 and Table 7 the same tables as in Section 6, but with more statistics of the results. For Cifar10, we now included in Table 6 the minimum task improvement across tasks and, while noisier, we can still observe that RotoGrad also improve this statistic. The standard deviation of the task improvement across tasks is, however, not too informative. In the case of CelebA, we added in Table 7 the maximum f1-score across tasks and, similar to the last case, it is not too informative, as all methods achieve almost perfect f1-score in one of the classes.

Table 5: Complete results (median and standard deviation) of different competing methods on MNIST/SVHN on all tasks, see Section 6 and Appendix C.2.

		Left digit Acc. \uparrow	Right digit Acc. \uparrow	Product parity f1 \uparrow	Sum digits MSE. \downarrow	$\text{avg}_k \Delta_k \uparrow$	Act. Pix. MSE \downarrow
MNIST	Single	95.77 \pm 0.19	94.04 \pm 0.18	92.30 \pm 0.72	1.91 \pm 0.13	-	0.01 \pm 0.01
	Vanilla	94.94 \pm 0.19	93.27 \pm 0.26	93.08 \pm 0.45	2.18 \pm 0.16	-2.51 \pm 3.01	0.11 \pm 0.01
	GradDrop	95.44 \pm 0.37	93.59 \pm 0.27	93.33 \pm 0.53	2.17 \pm 0.08	-2.51 \pm 1.73	0.13 \pm 0.02
	PCGrad	95.03 \pm 0.36	93.30 \pm 0.17	93.25 \pm 0.49	2.13 \pm 0.18	-3.12 \pm 3.88	0.12 \pm 0.02
	MGDA	94.31 \pm 1.11	92.02 \pm 2.06	83.85 \pm 2.04	2.52 \pm 0.61	-12.57 \pm 9.97	0.06 \pm 0.02
	GradNorm	95.45 \pm 0.29	93.84 \pm 0.34	93.51 \pm 0.59	1.85 \pm 0.12	0.75 \pm 2.85	0.08 \pm 0.01
	IMTL-G	95.25 \pm 0.33	93.88 \pm 0.21	93.28 \pm 0.48	1.85 \pm 0.09	1.17 \pm 2.77	0.07 \pm 0.01
	RotoGrad	95.56 \pm 0.21	93.76 \pm 0.29	93.32 \pm 0.28	1.81 \pm 0.10	2.20 \pm 1.92	0.07 \pm 0.03
SVHN	Single	85.05 \pm 0.45	84.58 \pm 0.24	77.47 \pm 1.13	5.84 \pm 0.14	-	0.17 \pm 0.06
	Vanilla	84.18 \pm 0.30	84.18 \pm 0.38	80.11 \pm 0.85	4.81 \pm 0.06	5.14 \pm 0.83	2.75 \pm 3.17
	GradDrop	84.38 \pm 0.29	84.48 \pm 0.41	80.11 \pm 0.69	4.69 \pm 0.12	5.68 \pm 1.05	1.91 \pm 0.86
	PCGrad	84.22 \pm 0.31	84.23 \pm 0.21	79.92 \pm 0.79	4.69 \pm 0.09	5.50 \pm 0.75	2.26 \pm 0.85
	MGDA	84.61 \pm 0.75	84.38 \pm 0.45	77.44 \pm 1.44	4.47 \pm 0.18	5.99 \pm 1.48	0.66 \pm 0.75
	GradNorm	84.61 \pm 0.42	84.45 \pm 0.40	80.03 \pm 0.75	4.42 \pm 0.15	6.67 \pm 1.02	1.41 \pm 0.74
	IMTL-G	84.60 \pm 0.45	84.39 \pm 0.37	79.63 \pm 1.10	4.57 \pm 0.13	5.81 \pm 0.85	2.47 \pm 1.65
	RotoGrad	83.89 \pm 0.40	83.91 \pm 0.47	79.43 \pm 0.79	4.54 \pm 0.12	5.61 \pm 0.84	1.44 \pm 0.68

Table 6: Complete task-improvement statistics in CIFAR10 for all competing methods and RotoGrad with different dimensionality for \mathbf{R}_k , see Section 6.

Method	$\min_k \Delta_k \uparrow$	$\text{med}_k \Delta_k \uparrow$	$\text{avg}_k \Delta_k \uparrow$	$\text{std}_k \Delta_k \downarrow$	$\max_k \Delta_k \uparrow$
Vanilla	-0.81 \pm 0.37	2.73 \pm 1.37	2.58 \pm 0.54	3.38 \pm 0.94	11.14 \pm 3.35
GradDrop	-0.81 \pm 0.54	3.18 \pm 1.07	3.07 \pm 0.48	4.00 \pm 0.65	14.03 \pm 2.83
PCGrad	-1.52 \pm 0.98	3.33 \pm 1.68	2.86 \pm 0.81	3.74 \pm 0.69	12.01 \pm 3.19
MGDA	-7.27 \pm 1.36	-4.48 \pm 2.35	-1.75 \pm 0.43	3.24 \pm 0.55	3.67 \pm 0.98
GradNorm	-3.75 \pm 0.67	0.09 \pm 2.23	-0.08 \pm 0.95	3.78 \pm 0.80	8.82 \pm 3.41
IMTL-G	-0.39 \pm 0.82	1.95 \pm 2.21	2.73 \pm 0.27	3.25 \pm 0.75	10.20 \pm 2.98
IMTL-G+ \mathbf{R}_k	-1.29 \pm 0.52	4.38 \pm 1.11	3.02 \pm 0.69	3.81 \pm 0.21	12.76 \pm 1.77
RotoGrad 64	-1.70 \pm 0.81	3.44 \pm 1.51	2.90 \pm 0.49	3.98 \pm 0.62	13.16 \pm 2.40
RotoGrad 128	-1.12 \pm 0.36	3.73 \pm 2.14	2.97 \pm 1.08	3.84 \pm 0.87	12.64 \pm 3.56
RotoGrad 256	0.17 \pm 1.01	3.29 \pm 2.18	3.68 \pm 0.68	3.83 \pm 0.74	14.01 \pm 3.22
RotoGrad 512	-0.43 \pm 0.76	4.72 \pm 2.84	4.48 \pm 0.99	4.23 \pm 0.82	15.57 \pm 3.99

Table 7: Complete f1-score statistics and training hours in CelebA for all competing methods and two different architectures/settings, see Section 6.

Method	Convolutional ($d = 512$) task f1-scores (%) \uparrow						ResNet18 ($d = 2048$) task f1-scores (%) \uparrow					
	\min_k	med_k	avg_k	$\text{std}_k \downarrow$	\max_k	Hours	\min_k	med_k	avg_k	$\text{std}_k \downarrow$	\max_k	Hours
Vanilla	1.62	54.74	58.69	24.18	97.04	4.06	15.45	61.52	61.25	22.09	96.61	1.49
GradDrop	3.94	55.80	58.62	23.98	97.19	4.42	4.46	63.52	63.61	21.79	96.59	1.60
PCGrad	2.69	60.30	59.83	23.85	97.04	17.03	17.23	61.82	62.74	20.84	96.43	5.90
GradNorm	1.83	52.17	54.68	24.94	96.88	11.02	14.43	64.10	63.51	21.20	96.50	3.59
IMTL-G	3.31	53.05	56.05	26.92	96.50	4.90	21.52	62.12	61.98	21.62	96.46	1.72
RotoGrad	9.11	62.31	62.45	22.14	96.83	11.00	25.72	63.84	65.17	18.99	96.49	6.90

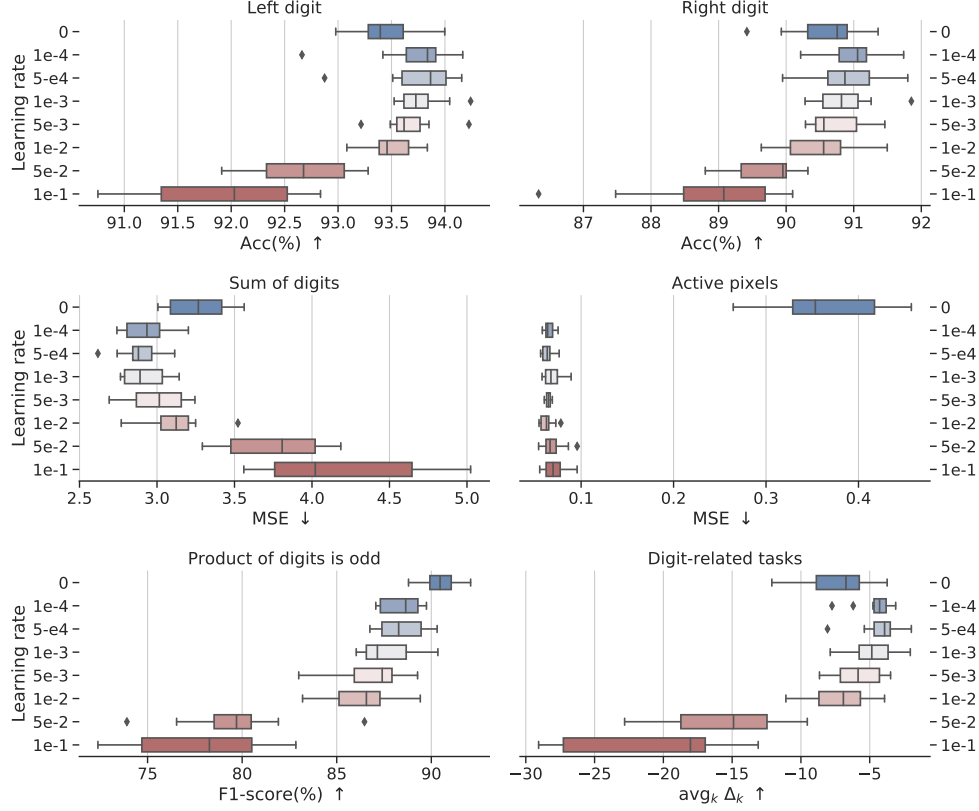


Figure 8: RotoGrad’s performance on all tasks for the experiments in Section 6.1 for all metrics. We can observe training instabilities/stiffness on all tasks as we highly increase/decrease RotoGrad’s learning rate, as discussed in the main manuscript.

783 C.2.5 NYUv2

784 Complementing the results shown in Section 6, here we show in Table 8 and Table 9 the results
 785 obtained for two other different random initializations, showing that the results discussed in the main
 786 text are consistent across runs. In other words, results in these two new tables further demonstrate
 787 that RotoGrad is capable to perform as well as the best of the competing methods while staying on
 788 par in training time.

Table 8: Results for different methods on the NYUv2 dataset with a SegNet model (with random seed two), see Section 6. RotoGrad obtains top performance in all tasks.

Method	Semantic Segmentation \uparrow			Depth Estimation \downarrow			Angle Distance \downarrow		Surface Normal Within $t^\circ \uparrow$				Hours
	mIoU	Pix Acc	$\text{avg}_k \Delta_k \uparrow$	Abs Err	Rel Err	$\text{avg}_k \Delta_k \uparrow$	Mean	Median	11.25	22.5	30	$\text{avg}_k \Delta_k \uparrow$	
Single	0.39	0.65	-	0.74	0.30	-	25.09	19.18	30.01	57.33	69.30	-	8.90
Vanilla	0.37	0.64	-3.58	0.58	0.22	24.40	29.93	25.89	19.85	43.92	57.39	-25.74	3.48
GradDrop	0.39	0.65	0.60	0.56	0.22	24.39	29.64	25.50	20.31	44.62	58.09	-24.36	3.56
PCGrad	0.40	0.66	2.41	0.53	0.22	28.02	29.17	24.94	21.14	45.72	59.14	-22.15	3.51
MGDA	0.21	0.51	-33.78	0.73	0.27	4.94	24.48	18.72	30.77	58.51	70.39	2.20	3.52
GradNorm	0.40	0.67	2.25	0.54	0.21	27.58	25.91	20.75	27.36	54.09	66.71	-5.94	3.50
IMTL-G	0.40	0.66	2.68	0.52	0.20	30.42	26.00	20.92	26.77	53.79	66.59	-6.71	3.62
RotoGrad	0.41	0.66	3.79	0.53	0.20	29.77	25.83	20.73	27.30	54.14	66.90	-5.82	3.89

Table 9: Results for different methods on the NYUv2 dataset with a SegNet model (with random seed three), see Section 6. RotoGrad obtains top performance in all tasks.

Method	Semantic Segmentation \uparrow			Depth Estimation \downarrow			Angle Distance \downarrow		Surface Normal Within $t^\circ \uparrow$					Hours
	mIoU	Pix Acc	$\text{avg}_k \Delta_k \uparrow$	Abs Err	Rel Err	$\text{avg}_k \Delta_k \uparrow$	Mean	Median	11.25	22.5	30	$\text{avg}_k \Delta_k \uparrow$		
Single	0.37	0.63	-	0.71	0.28	-	25.14	19.25	29.92	57.23	69.12	-	8.91	
Vanilla	0.38	0.64	2.20	0.63	0.23	15.28	29.40	25.31	20.94	45.02	58.35	-23.08	3.47	
GradDrop	0.37	0.62	-0.63	0.61	0.23	15.38	30.47	26.63	19.01	42.61	56.06	-28.10	3.53	
PCGrad	0.37	0.64	-0.51	0.57	0.22	19.97	29.79	25.77	20.61	44.22	57.47	-24.63	3.50	
MGDA	0.20	0.51	-31.94	0.81	0.28	-8.22	24.70	18.92	30.38	57.95	69.99	1.50	3.51	
GradNorm	0.40	0.65	6.09	0.57	0.22	20.03	26.12	20.92	27.08	53.69	66.33	-6.46	3.50	
IMTL-G	0.40	0.66	5.63	0.57	0.21	22.48	26.23	21.14	26.38	53.25	66.22	-7.43	3.60	
RotoGrad	0.39	0.66	5.45	0.53	0.21	24.04	25.90	20.59	27.44	54.35	67.02	-5.28	3.83	