ROTOGRAD: DYNAMIC GRADIENT HOMOGENIZATION FOR MULTITASK LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

GradNorm (Chen et al., 2018) is a broadly used gradient-based approach for training multitask networks, where different tasks share, and thus compete during learning, for the network parameters. GradNorm eases the fitting of all individual tasks by dynamically equalizing the contribution of each task to the overall gradient magnitude. However, it does not prevent the individual tasks' gradients from *conflicting*, i.e., pointing towards opposite directions, and thus resulting in a poor multitask performance. In this work we propose Rotograd, an extension to GradNorm that addresses this problem by dynamically homogenizing not only the gradient magnitudes but also their directions across tasks. For this purpose, Rotograd adds a layer of task-specific rotation matrices that aligns all the task gradients. Importantly, we then analyze Rotograd (and its predecessor) through the lens of game theory, providing theoretical guarantees on the algorithm stability and convergence. Finally, our experiments on several real-world datasets and network architectures show that Rotograd outperforms previous approaches for multitask learning.

1 INTRODUCTION

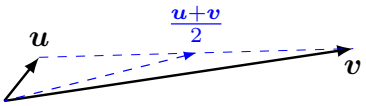
While single-task learning is broadly-used and keeps achieving state-of-the-art results in different domains —sometimes beating human performance—, there are countless scenarios where we would like to solve several related tasks, we encounter overfitting problems, or the available data is scarce. Multitask learning is a promising field of machine learning that appeared to solve —or, at least, palliate— the aforementioned problems that single-task networks suffer from (Caruana, 1993).

Many multitask architectures have emerged throughout the past years (Kokkinos, 2017; Maninis et al., 2019; Dong et al., 2015; He et al., 2017), yet most of them fall under the umbrella of hard parameter sharing (Caruana, 1993). This architecture is characterized by having a shared backbone that acts as an encoder, producing based on the input an intermediate representation shared across tasks. Then, this intermediate value is fed into a set of task-specific modules that act as decoders, producing an specialized output for each task.

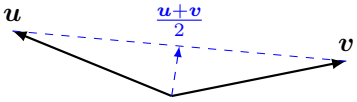
These networks have proven to be powerful, efficient, and in many occasions, capable to improve the results of their single-task counterparts. However, they can be difficult to train. Competition among tasks for the shared resources can lead to poor results where the resources are dominated by a subset of tasks. Internally, this can be traced to a sum over task gradients of the form $L = \sum_k \nabla L_k$ with respect to the backbone parameters. As depicted in Figure 1a, in this setting two undesirable scenarios may occur: a) a subset of tasks dominate the overall gradient evaluation due to magnitude differences across task gradients; and b) individual per-task gradients point towards different directions in the parameter space, cancelling each other out.

This problem can be seen as a particular case of *negative transfer*, a phenomenon used when "sharing information with unrelated tasks might actually hurt performance" (Ruder, 2017). Under the assumption that the problem comes from task unrelatedness, one solution is to use only related tasks. Several works have explored this direction (Thrun & O'Sullivan, 1996; Zamir et al., 2018), see for example Standley et al. (2019) for a recent in-depth study on task relatedness.

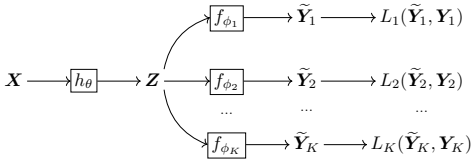
Another way of tackling the problem is by weighting the task dynamically during training, the motivation being that with this approach the task contribution task can me manipulated at will and a



(a) Magnitude disparity.



(b) Conflicting directions.



(c) Hard parameter sharing model.

Figure 1: Examples of: (a) neglecting one vector due to magnitude difference; (b) partial cancellation of two vectors with equal magnitude; (c) a hard parameter sharing architecture.

good balance between gradient magnitudes can be found. Two important works on this direction are GradNorm (Chen et al., 2018), which we will cover in detail later, and Kendall et al. (2018), which adopts a probabilistic approach to the problem. Other work, for example Guo et al. (2018), adapts the weights based on their "difficulty" to be solved to prioritize harder tasks.

Furthermore, a series of recent papers emerged trying to solve the problem of conflicting gradients in different settings. For example, Flennerhag et al. (2019) deals with a similar problem in the context of meta-learning, and Yu et al. proposes two different heuristic solutions for reinforcement learning. In the context of multitask learning, Levi & Ullman (2020) adopts an image-dependent solution that uses a top-down network, and Maninis et al. (2019) applies adversarial training to deal with conflicting gradients, making use of double-backpropagation which poorly scales.

The key contributions of this paper are as follows:

- The introduction of Rotograd, an extension of GradNorm that homogenizes both the magnitude and the direction of the task gradients during training. Different experiments support this statement and show the performance benefits that Rotograd provides.
- A novel interpretation of GradNorm (and Rotograd) as a gradient-play Stackelberg game with theoretical guarantees on the stability of the training as long as GradNorm is an asymptotically slower learner than the network's optimizer.
- Showing that GradNorm can be successfully applied to the output of the backbone instead of to a subset of its parameters, completely removing this hyperparameter. Besides, we empirically show that results are comparably good.

2 Background

2.1 Multitask learning

Let us consider K different tasks that we want to simultaneously learn using a gradient-based approach. All of them share the same input dataset $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, where each row $\boldsymbol{x}_n \in \mathbb{R}^D$ is a D-dimensional sample. Additionally, each task has its own outputs $\boldsymbol{Y}_k \in \mathbb{R}^{N \times I_k}$, and loss function $\ell^k : \mathbb{Y}_k \times \mathbb{Y}_k \to \mathbb{R}$.

As mentioned in Section 1 and illustrated in Figure 1c, the model is composed of a shared backbone and task-specific modules. Upon using the model, first the input $\boldsymbol{X} \in \mathbb{R}^{B \times D}$ is passed through the *backbone*, $h_\theta : \mathbb{X} \to \mathbb{Z}$, that produces a common intermediate representation of the input, $\boldsymbol{Z} \in \mathbb{R}^{B \times d}$ where B is the batch size and d the size of this *shared intermediate space*. Then, each task-specific model $f_{\phi_k} : \mathbb{Z} \to \mathbb{Y}_k$, based on the intermediate representation $\boldsymbol{Z}$, predicts the desired outcome for its designated task, $\widetilde{\boldsymbol{Y}}_k$.

As the focus of this work concerns only the shared parameters, we can simplify the notation regarding the task-specific pipeline —that is, the branching paths shown in Figure 1c— by writing the single-point evaluation at time t as $\ell_i^k(t)$, instead of writing $\ell^k(f_{\phi_k^t}(\boldsymbol{z}_i^t))$, and denoting the loss by $L_k(t) := \sum_i \ell_i^k(t)$.

When it comes to training, a single loss function is needed in order to backpropagate through it. In this settings, however, there is a loss function ℓ_k per task. A common approach is to combine these losses by taking a weighted sum, so that the training objective is to minimize the loss function $L(t) := \sum_k \omega_k L_k(t)$. Therefore, the common parameters θ are updated, in the case of first-order optimization methods, by following the gradient direction

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

As we spoiled in Section 1, Equation 1 turns out to be the main source of problems since *the loss functions —and thus the tasks— have been artificially coupled*. The magnitude of these task-specific gradients represent how much they contribute to the parameter update: if some of them are relatively small (large), their tasks will be overlooked (prioritized) during the training procedure, as Figure 1a exemplifies. More subtle is the case of the direction; if gradients from different tasks point to completely different regions of the parameter space, they may cancel each other out, resulting in a direction update potentially harmful for these tasks, as shown in Figure 1b.

2.2 STACKELBERG GAMES

Let us now introduce some game theoretical concepts necessary for the following sections. A Stackelberg game (Fiez et al., 2019) is a type of *asymmetric game* where two players play alternatively. The first of them is the follower $\mathcal{F}$, whose sole objective is to minimize its own loss function. Secondly, there exist a leader $\mathcal{L}$ that also tries to minimizes its own loss function, but having additional information regarding which will be the (direction of the) best move the follower could employ after the leader's move. In mathematical terms,

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

where x_l and x_f are the moves of the leader and the follower, respectively.

Another important concept in game theory is that of a local equilibrium point. In layman's terms, a local equilibrium point in a game between two players is one in which both players have reached a state that satisfies them, meaning that there is no available move improving any of the player scores and no further actions are taken. In our particular case, we are interested in:

Definition 2.1 (differential Stackelberg equilibrium)**.** The 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 if $\nabla_{x_l} \mathcal{L}(x_l^*, r(x_l^*)) = 0$, and $\nabla^f_{x_l} \mathcal{L}(x_l^*, r(x_l^*))$ is positive definite.

As a final remark, note that in this particular concept of equilibrium both players reach a critical point (local optimum). Even more, when both players optimize their objectives using a gradient-based approach and the leader is a slower learner —meaning having an asymptotically smaller learning rate—, then both players reach a local minimum on the equilibrium (Fiez et al., 2019).

3 THE GAME OF GRADNORM

GradNorm (Chen et al., 2018) was conceived with a simple idea in mind: equalize the contribution of each task to the total gradient magnitude during training such that they all are learned at the same rate. This idea, as it can be seen in Equation 1, means making the gradient magnitudes relatively equal across all tasks. GradNorm achieves this by considering the task weights ω_k as parameters that are modified during training.

These parameters are trained alongside the original network parameters via gradient descent, yet their gradients are obtained by minimizing a specific loss function L^k_{grad}:

$$L^k_{\text{grad}}(\omega_k, \theta) := \left| \omega_k ||\nabla_W L_k(t)||_2 - \frac{1}{K} ||\nabla_W L(t)||_2 \times [r_k(t)]^\alpha \right|, \tag{3}$$

where W is a subset of the shared network parameters θ, the weights are corrected such that $\sum_k \omega_k = K$ always, $[r_k(t)]^\alpha$ is a hyperparameter that balances the learning speed per task, and the average gradient magnitude (that is, the target magnitude) is considered as a constant.

Using the concepts of Section 2.2, we can reinterpret GradNorm in terms of a Stackelberg game. Namely, in this new point of view the follower would be the player whose objective is to optimize the parameters of the backbone, whereas GradNorm would play the role of the leader, optimizing the task weights in order to equalize the gradient magnitudes:

$$\mathcal{L}\text{eader: } \min_{\{\omega_k\}_k} \sum_k L^k_{\text{grad}}(\omega_k, \theta), \quad \mathcal{F}\text{ollower: } \min_\theta \sum_k \omega_k L_k(\theta). \tag{4}$$

Note that there is a fundamental difference between Equation 2 and 4: the extra information available to the leader is not explicitly written in its optimization problem. This is due to the more subtle nature of this bit of information. *The actual objective of the leader is not to equalize the magnitudes at the current step t, but at $t+1$, since that is when the effect of updating ω_k will take effect.* Therefore, the extra information that the leader possesses is that the follower updates its parameters in a smooth way and the magnitude differences between consecutive steps are almost negligible.

This is, to the best of our knowledge, the first time GradNorm has been interpreted as a (stochastic) gradient-based Stackelberg game. The main advantage of this point of view is that, based on the results presented by Fiez et al. (2019), we can provide theoretical guarantees regarding the convergence of the learning dynamics between the leader (GradNorm) and the follower (network's optimizer). Specifically:

Proposition 3.1. *Suppose a finite-time (stochastic) gradient-based Stackelberg where the learning rate of the leader tends to zero on a faster timescale than the follower, that is, $\alpha_l(t) = o(\alpha_f(t))$. Then, the game will with high probability converge to a neighborhood of a stable differential Stackelberg equilibrium.*

This previous statement means —in terms of the training dynamics of the network— that, as long as the leader is the slow learner, the parameters of the leader will stabilize at some point and the network parameters will end up in local optimum. *In other words, the training will be stable.*

4 ROTOGRAD

4.1 $\boldsymbol{Z}$ AS THE CRITICAL SPACE

As an additional remark to Section 3, we believe that it is worth mentioning that GradNorm is applied to only a subset of the network parameters $W \subset \theta$ for efficiency reasons. The selection of such subset is left to the practitioner and, to the best of our knowledge, there exist no intuition regarding which parts of the backbone are preferred to be selected as W. In this work, we propose to apply GradNorm on the last shared representation of the network, which as shown in Figure 1c, we refer to as $\boldsymbol{Z}$. In other words, we assume that $W = \boldsymbol{Z}$.

Since the size of $\boldsymbol{Z}$, $B \times d$, tends to be orders of magnitude smaller that the total number of backbone parameters θ this choice of W saves computation. Moreover, we argue that this selection is the most logical one since the intermediate representation $\boldsymbol{Z}$ is the last shared representation across all tasks, as shown in Figure 1c.

Remarkably, independently of the number of shared parameters θ, under mild assumptions we can upper and lower bound the magnitude of the gradient $\nabla_\theta L_k(t)$ by applying GradNorm on $\boldsymbol{Z}$, as shown in the following proposition (details in Appendix A.1):

Proposition 4.1. *If for every task $||\omega_k \nabla_{\boldsymbol{Z}} L_k(t)|| = C$, where C is a constant value, and $\nabla_\theta \boldsymbol{Z}$ is an invertible matrix, then:*

$$\frac{C}{||\nabla_\theta \boldsymbol{Z}^{-1}||} \leq ||\omega_k \nabla_\theta L_k(t)|| \leq ||\nabla_\theta \boldsymbol{Z}|| \cdot C \quad \textit{for all} \quad k = 1, 2, \ldots, K. \tag{5}$$

4.2 ROTOGRAD: A BETTER LEADER

We have seen in Section 3 that the objective of GradNorm is to equalize the gradient magnitude so that, in the next step, the shared parameters are updated according to the necessities of all the

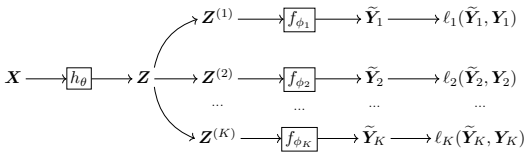


Figure 2: Extended hard parameter sharing model with task-specific representations.

tasks. GradNorm does not address the problem of conflicting gradients (Figure 1b) in the total gradient computation, which can be as detrimental as the one of magnitude disparity (Figure 1a). Fortunately, the reinterpretation of GradNorm as a Stackelberg game presented in Section 2.2 allow us to easily fix this problem. More specifically, *we introduce a new leader that aims to homogenize both the gradient magnitudes and directions for all tasks. We refer to the resulting algorithm as Rotograd.*

Rotograd performs the same operations as Gradnorm but, additionally, expands the model architecture as shown in Figure 2. $\boldsymbol{Z} \in \mathbb{R}^{B\times d}$ represents the shared intermediate representation, and $\boldsymbol{Z}^k \in \mathbb{R}^{B\times d}$ denotes task-specific intermediate representations of $\boldsymbol{Z}$, one per task. This new space $\mathbb{Z}^k$ is a shifted-and-rotated version of $\mathbb{Z}$, such that the i-th instance of $\boldsymbol{Z}$ is transformed as $\boldsymbol{z}_i^k = R_k \boldsymbol{z}_i + \boldsymbol{d}_k$, where $R_k \in SO(d)$ is a rotation matrix and $\boldsymbol{d}_k \in \mathbb{R}^d$ a translation vector.

With regard to the new objective function, for the k-task at step t, Rotograd minimizes the distance between the *the next set of shared points* $\{\boldsymbol{z}_i^{t+1}\}_{i=1}^B$ *to the next set of points we would have reached if only that task were being trained* $\{\boldsymbol{z}_i^{k,t+1}\}_{i=1}^B$. Note that the additional knowledge of the leader plays a role, as it knows the next evaluation points given the current ones, concept that we represent using two oracle functions $\mathcal{O}(\boldsymbol{z}_i^t)$ and $\mathcal{O}_k(\boldsymbol{z}_i^{k,t})$ for the multi-task and single-task next points, respectively.

We can finally write the new objective of Rotograd as

$$L_{\text{rot}}^k(R_k, \boldsymbol{d}_k, \theta) := \frac{1}{B}\sum_{i=1}^{B} ||R_k \mathcal{O}(\boldsymbol{z}_i^t) + \boldsymbol{d}_k - \mathcal{O}_k(\boldsymbol{z}_i^{k,t})||_2^2, \tag{6}$$

resulting in the Stackelberg game:

$$\begin{aligned} \mathcal{L}\text{eader: } &\underset{\{\omega_k, R_k, \boldsymbol{d}_k\}_k}{\text{minimize}} \sum_k \left[L_{\text{rot}}^k(R_k, \boldsymbol{d}_k, \theta) + L_{\text{grad}}^k(\omega_k, \theta)\right], \\ \mathcal{F}\text{ollower: } &\underset{\theta}{\text{minimize}} \sum_k \omega_k L_k(\theta). \end{aligned} \tag{7}$$

4.3 Solving the new objective function

At first glance solving the newly-formulated problem L_{rot} seems quite challenging: we cannot write down the oracle functions, and it is a constrained optimization problem since $\mathbb{R}_k$ is an orthogonal matrix. In this section we show how to approximately solve this problem in closed form.

Recall that, at the current step t, we can obtain any current intermediate point in the shared space as $\boldsymbol{z}_i^t = h_{\theta^t}(\boldsymbol{x}_i^t)$, and its task-specific equivalent as $\boldsymbol{z}_i^{k,t} = R_k^t \boldsymbol{z}_i^t + \boldsymbol{d}_k^t$. To simplify notation, let us denote the gradient of the k-th task with respect to $\boldsymbol{z}^k$ at step t by $g_i^{k,t} := \nabla_{\boldsymbol{z}^k} \ell_k(\boldsymbol{z}_i^{k,t})$, and the gradient of the weighted loss with respect to $\boldsymbol{z}$ at step t by $g_i^t := \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}_i) = \sum_k \omega_k R_k^\top g_i^{k,t}$.

As mentioned before, we cannot in general write the oracle functions in closed form. However, since we assume that the network parameters are optimized using a first-order optimization method, we can *approximate* the oracle functions by taking into account that the next point will follow the gradient direction, that is, we can approximate the oracles as:

$$\mathcal{O}(\boldsymbol{z}_i^t) \approx \boldsymbol{z}_i^t - \beta g_i^t, \qquad \text{and} \qquad \mathcal{O}_k(\boldsymbol{z}_i^{k,t}) \approx \boldsymbol{z}_i^{k,t} - \beta g_i^{k,t}, \tag{8}$$

where we treat the step-size β as an unknown variable.

By plugging this approximation to the oracles into the loss function in Equation 6 we can approximately solve $\min_{R_k, \boldsymbol{d}_k} L^k_{\text{rot}}(R_k, \boldsymbol{d}_k, \theta)$ by solving instead

$$\min_{R_k \in O(d)} \frac{1}{B} \sum_{i=1}^{B} ||R_k g_i^t - g_i^{k,t}||_2^2 \quad \text{and} \quad \min_{\boldsymbol{d}_k \in \mathbb{R}^d} \frac{1}{B} \sum_{i=1}^{B} ||R_k \boldsymbol{z}_i^t + \boldsymbol{d}_k - \boldsymbol{z}_i^{k,t}||_2^2, \tag{9}$$

which are two least-square problems with closed form solutions R_k^* and d_k^*. Details in Appendix A.2.

Note that the first problem in Equation 9 looks for the orthogonal matrix that *minimizes the gradient conflict between the weighted gradients g_i^t and the gradients of the k-th task $g_i^{k,t}$*, which was our main objective for this extension. The second problem looks for the best way of translating the current points in the common space $\boldsymbol{z}_i^t$ to the current points that were plugged into the task-specific module for the k-th task $\boldsymbol{z}_i^{k,t}$.

Similarly to what we did in Section 3 we can apply again Proposition 3.1, thus *providing theoretical guarantees on the stable behaviour of the algorithm as long as Rotograd has a stronger learning rate decay than the network's optimizer*. In order for this to hold, we should adopt a gradient-descent approach and, since we have an approximation to the solution of the problem, we set the gradient of our parameters to the difference between our solution and its current value. For example, in the case of applying SGD to the parameter R_k, the update rule would be of the form $R_k^{t+1} = R_k^t - \alpha(R_k^t - R_k^*)$ where, afterwards, we also should make sure that R_k^{t+1} is a rotation matrix.

An overview of the proposed Rotograd algorithm can be found in Algorithm 1.

Algorithm 1 Forward and update methods for the k-task using Rotograd.

procedure FORWARD_K($\boldsymbol{z}_i^t$)
  return $R^k \boldsymbol{z}_i^t + \boldsymbol{d}_k$ ▷ Transform to task-specific space $\mathbb{Z}^k$
end procedure

procedure UPDATE_LOSS_ROT_K($\{\boldsymbol{z}_i^t, g_i^t, \boldsymbol{z}_i^{k,t}, g_i^{k,t}\}_i$)
  $R^{k*} \leftarrow \text{argmin}_{R_k} \frac{1}{B} \sum_{i=1}^{B} ||R_k g_i^t - g_i^{k,t}||_2^2$ ▷ Solve conflicting gradient problem
  $\boldsymbol{d}_k^* \leftarrow \text{argmin}_{\boldsymbol{d}_k} \frac{1}{B} \sum_{i=1}^{B} ||R_k \boldsymbol{z}_i^t + \boldsymbol{d}_k - \boldsymbol{z}_i^{k,t}||_2^2$ ▷ Solve translation problem

  $\boldsymbol{d}_k^{t+1} \leftarrow \boldsymbol{d}_k^t - \alpha(\boldsymbol{d}_k^t - \boldsymbol{d}_k^*)$ ▷ Update rule for $\boldsymbol{d}_k$
  $R_k^{t+1} \leftarrow R_k^t - \alpha(R_k^t - R_k^*)$ ▷ Update rule for R_k

  $U, S, V^\top \leftarrow \text{SVD}(R_k^{t+1})$
  $R_k^{t+1} \leftarrow UV^\top$ ▷ Ensure that R_k^{t+1} is a rotation
end procedure

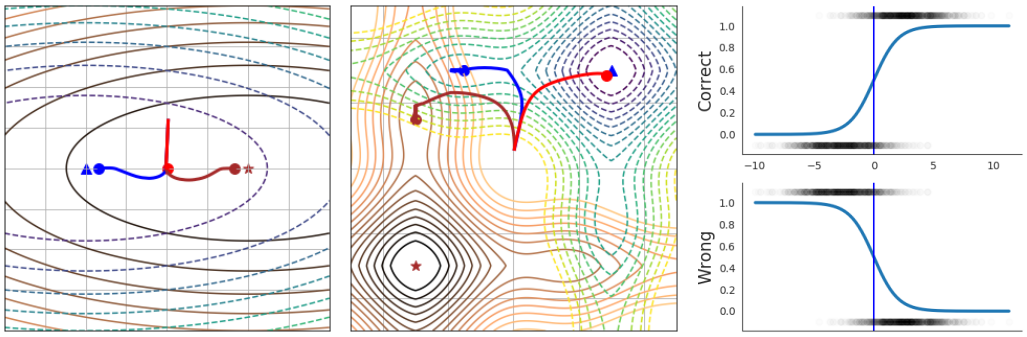
5 EXPERIMENTS

5.1 QUALITATIVE RESULTS

In order to get a better understanding of Rotograd, we provide two simple synthetic examples that show both, its effect on the shared intermediate space $\boldsymbol{Z}$, and its effect on the final predictions.

First, we consider two different settings where we use a two-layer feed-forward NN with 10 hidden units to solve two regression tasks. The first setting problem, illustrated in Figure 3a, shows an example where not correcting for conflicting gradients may completely prevent from learning (i.e., none of the tasks is optimized), while Rotograd result in converge into the global optima of both tasks. The second setting in Figure 3b shows that, due to differences in the gradient magnitudes, only one of the tasks is learned when Rotograd is not applied. It is worth-mentioning as well that Rotograd is still subject to the nature of stochastic optimization, and thus being subject to fall into local optima, as shown in Figure 3b.

As a corner example, we consider two classification tasks, which aim to predict opposite labels. As before, we use a feed-forward backbone without task-specific networks, and a logistic output



(a) Regression example 1. (b) Regression example 2. (c) Classification example.

Figure 3: Figures (a) and (b) show the trajectories followed when using Rotograd in the spaces $\boldsymbol{Z}^1$ (blue) and $\boldsymbol{Z}^2$ (brown), versus not using it (red). Circular markers mean the final point of the trajectory, whereas the triangle and star mark the global optimum of each task. Figure (c) shows the deep logistic regression classifier learned using Rotograd to predict a binary label and its (Correct) and its negated (Wrong).

for each task. In this case, the two tasks, and therefore their gradients with respect to the network parameter, are in conflict. Thus, without additional model flexibility (i.e., without our rotation layer) the network cannot learn any of the tasks. In contrast, Rotograd is able to rotate the space and learn that one function is the opposite of the other, learning an accurate classifier, as shown in Figure 3c.

5.2 Quantitative results

Dataset and tasks. Similar to Sener & Koltun (2018), we employ as dataset a multitask version of MNIST (LeCun et al., 1998) that has been expanded to contain instead two digits, one in each side of the picture. Regarding the tasks, we learn five different tasks: i) classify the left-side digit; ii) classify the right-side digit; iii) predict the sum of the two digits; iv) predict their product; and v) predict the number of pixels that are activated in the picture. The idea behind these tasks is no other than having a set of tasks highly correlated, one of correlated tasks, and an uncorrelated task.

Architecture. We use as backbone a modified version of LeNet (LeCun et al., 1998) where the number of parameters has been greatly reduced in order to not have enough parameters to learn all the tasks perfectly, and thus, try to force them to collaborate in order to improve their results. The size of $\boldsymbol{Z}$ is $d = 25$. For the task-specific parts we make use of a single layer per task, followed by a sigmoid activation function in the case of solving a classification problem. Regarding loss functions, we use mean-squared error for the regression tasks and negative log-likelihood for the classification ones. Furthermore, we divide all losses by their loss at the initial step to normalize them.

Training. We train the model for 150 epochs using a batch size of 1024 and Adam (Kingma & Ba, 2014) as the optimizer. We set the learning rate of the network parameters to 0.001, and to 0.025 for Rotograd's parameters. To ensure that the leader is asymptotically the slower learner, we put an exponential decay of 0.999. The hyperparameters of all the methods are chosen based on a grid search over 10 different runs.

Metrics. In the case of regression, we still use the mean-squared error as metric. For classification tasks we employ accuracy. Besides, as in Vandenhende et al. (2019), we include an additional metric that summarizes the overall performance of the multitask network:

$$\Delta_m = \frac{1}{K} \sum_{k=1}^{K} (-1)^{l_k} \frac{M_{m,k} - M_{s,k}}{M_{s,k}}, \tag{10}$$

where $M_{m,k}$ and $M_{s,k}$ are the performance on the k-task for the multitask and single task networks, respectively, and l_k is 1 if a lower value means better performance and 0 otherwise. This metric simply represents the average performance gained per task.

Method	Left ↑	Right ↑	Sum ↓	Multiply ↓	Density ↓	Δ_m ↑
single task	93.50 ± 00.47	90.65 ± 00.46	6.44 ± 4.63	159.08 ± 6.16	1.62 ± 1.78	-
uniform	90.15 ± 00.53	86.65 ± 00.41	5.14 ± 0.33	149.21 ± 5.97	0.51 ± 0.02	0.06 ± 0.11
rotograd	87.91 ± 01.10	83.60 ± 00.69	4.63 ± 0.11	138.68 ± 2.88	0.21 ± 0.04	0.18 ± 0.05
rotograd-sgd	87.30 ± 00.36	83.48 ± 00.69	5.08 ± 0.31	161.86 ± 8.71	0.30 ± 0.09	0.12 ± 0.04
gradnorm $\boldsymbol{Z}$	89.15 ± 00.58	85.32 ± 00.72	5.07 ± 0.30	144.91 ± 8.51	0.31 ± 0.07	0.14 ± 0.05
gradnorm W	89.60 ± 00.52	85.72 ± 00.70	5.13 ± 0.24	151.58 ± 8.01	0.44 ± 0.28	0.13 ± 0.05
d-grad	88.71 ± 00.69	84.96 ± 00.96	4.93 ± 0.38	144.54 ± 9.88	0.60 ± 0.54	0.08 ± 0.16
adversarial	89.37 ± 00.60	85.58 ± 00.53	5.60 ± 0.45	160.00 ± 6.98	1.65 ± 1.06	-0.25 ± 0.43

Table 1: Test results on the MNIST dataset for different methods. The arrow on the each column's title describes whether higher is better or the other opposite.

Methods. We compare the multitask learning performance of the following methods: i) *single task*, which learning each task independently; ii) *uniform*, which solver the multitask learning problem assuming uniform weights for all tasks; iii) *rotograd*, the proposed algorithm; iv) *rotograd-sgd*, same as *rotograd* but the rotation updates are performed using automatic differentation; v) *gradnorm* W, GradNorm with the last layer of the backbone as W, as proposed in Chen et al. (2018); vi) *gradnorm* $\boldsymbol{Z}$, GradNorm with $W = \boldsymbol{Z}$; vii) *d-grad*, which implements Yu et al., an heuristic to avoid conflicting gradients, adapted for the hard-parameter sharing setting; and viii) *adversarial* (Maninis et al., 2019), which uses adversarial training to correct the gradient directions (with a two-layer feed-forward network as discriminators).

Results. Figure 1 summarizes the results obtained averaged over ten different runs. We observe how Rotograd outperforms the rest of methods with regards to the overall improvement performance Δ_m. Rotograd is able to drastically increase the performance on the regression tasks with just a slight trade-off on the classification ones. Not only we can say that there is an improvement, but also that *approximately solving the problem in Equation 14 in closed-form is important* for it, as *rotograd-sgd* is far from Rotograd's results. When it comes to GradNorm, Table 1 shows that there is no disadvantage on applying the algorithm to $\boldsymbol{Z}$, since it actually obtains better regression performance in addition to removing the selection of the hyperparameter W. *We show in Appendix A.3 that this improvement is due to the reduction in direction conflict between gradients that Rotograd achieve.* Also in Appendix A.4 we have the preliminary results obtained on a larger dataset such as ChestX-ray14 (Rajpurkar et al., 2017).

Finally, it is worth mentioning that *d-grad*, *uniform*, and *adversarial* perform considerably worse. Our intuition for these results are as follows: *uniform* suffers from the problems mentioned throughout this work; *d-grad* adds more parameters to the task-specific modules, but since the problem is not in the per-task capacity, it does not translate into much of an improvement; and *adversarial* suffers from training instability, as it is known that this techniques can be hard to train, as it happens in Generative Adversarial Networks (Arjovsky & Bottou, 2017). Additional experiments on the ChestX-ray14 dataset are included in Appendix A.4, where we observed similar patterns in the results for the different methods, being the results of GradNorms and Rotograd comparable, as it seems that such dataset does not suffer from the conflicting gradient problem.

6 Conclusions

In this work, we have proposed a novel learning algorithm for multitask learning, Rotograd, which aims to minimize negative transfer among tasks by homogenizing the gradient magnitudes and directions across tasks. Our starting point here has been GradNorm, which we have reinterpreted as a Stackelberg game where GradNorm act as the leader and the network optimizer as the follower. With such interpretation in the hand, we have not being able to propose Rotograd, but also to derive stability and convergence guarantees for both Rotograd and GradNorm.

As future work, we plan to extend Rotograd to related scenarios, such as soft parameter sharing networks and meta-learning, which could gratefully benefit from using our algorithm. We also leave for future research trying Rotograd in heavier experiments that are affected by negative transfer, as it could happen in the field of computer vision.

REFERENCES

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862, 2017.

Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In Proceedings of the Tenth International Conference on Machine Learning, pp. 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, pp. 794–803. PMLR, 2018.

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

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), pp. 270–287, 2018.

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

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7482–7491, 2018.

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

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

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.

Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1851–1860, 2019.

Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225, 2017.

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, pp. 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.

Sebastian Thrun and Joseph O'Sullivan. Discovering structure in multiple learning tasks: The tc algorithm. In ICML, volume 96, pp. 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.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Multi-task reinforcement learning without interference.

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

A Appendix

A.1 Proof of proposition 4.1

Proposition A.1. *If for every task $||\omega_k \nabla_{\boldsymbol{Z}} L_k(t)||_2 = C$, where C is a constant value, and $\nabla_\theta \boldsymbol{Z}$ is an invertible matrix, then:*

$$\frac{C}{||\nabla_\theta \boldsymbol{Z}^{-1}||_2} \leq ||\omega_k \nabla_\theta L_k(t)||_2 \leq ||\nabla_\theta \boldsymbol{Z}||_2 \cdot C \quad \textit{for all} \quad k = 1, 2, \ldots, K. \tag{11}$$

Proof. First, we prove the upper bound, which is a direct consequence of the chain rule and the 2-norm being a submultiplicative matrix norm:

$$||\omega_k \nabla_\theta L_k(t)||_2 = ||\omega_k \nabla_\theta \boldsymbol{Z} \nabla_{\boldsymbol{Z}} L_k(t)||_2 \leq ||\nabla_\theta \boldsymbol{Z}||_2 \cdot C. \tag{12}$$

The lower bound uses again the same two methods, that $\nabla_\theta \boldsymbol{Z}$ is invertible, and a bit of simple algebra:

$$\begin{aligned} ||\omega_k \nabla_{\boldsymbol{Z}} L_k(t)||_2 &= ||\omega_k \nabla_\theta \boldsymbol{Z}^{-1} \nabla_\theta \boldsymbol{Z} \nabla_{\boldsymbol{Z}} L_k(t)||_2 \\ &\leq ||\nabla_\theta \boldsymbol{Z}^{-1}||_2 ||\omega_k \nabla_\theta \boldsymbol{Z} \nabla_{\boldsymbol{Z}} L_k(t)||_2 \\ &= ||\nabla_\theta \boldsymbol{Z}^{-1}||_2 ||\omega_k \nabla_\theta L_k(t)||_2, \end{aligned}$$

which implies that

$$\frac{C}{||\nabla_\theta \boldsymbol{Z}^{-1}||_2} \leq ||\omega_k \nabla_\theta L_k(t)||_2. \tag{13}$$

Q.E.D.

A.2 Approximation of Rotograd's loss

In Section 4.2, we had the optimization problem

$$L^k_{\text{rot}}(R_k, \boldsymbol{d}_k, \theta) := \frac{1}{B} \sum_{i=1}^{B} ||R_k \mathcal{O}(\boldsymbol{z}_i^t) + \boldsymbol{d}_k - \mathcal{O}_k(\boldsymbol{z}_i^{k,t})||_2^2, \tag{14}$$

and claim that, by plugging in the approximation to the oracles

$$\mathcal{O}(\boldsymbol{z}_i^t) \approx \boldsymbol{z}_i^t - \beta g_i^t, \qquad \text{and} \qquad \mathcal{O}_k(\boldsymbol{z}_i^{k,t}) \approx \boldsymbol{z}_i^{k,t} - \beta g_i^{k,t}, \tag{15}$$

we could get a closed form approximation for the solution of the problem in Equation 14. Here we are going to derive this approximation. First, by plugging 15 into 14 we can split the latter into three

different problems:

$$
\begin{aligned}
L^k_{\text{rot}}(R_k, \boldsymbol{d}_k, \theta) &= \frac{1}{B}\sum_{i=1}^{B} ||R_k\mathcal{O}(\boldsymbol{z}^t_i) + \boldsymbol{d}_k - \mathcal{O}_k(\boldsymbol{z}^{k,t}_i)||_2^2 \\
&= \frac{1}{B}\sum_{i=1}^{B} ||R_k(\boldsymbol{z}^t_i - \beta g^t_i) + \boldsymbol{d}_k - (\boldsymbol{z}^{k,t}_i - \beta g^{k,t}_i)||_2^2 \\
&= \frac{1}{B}\sum_{i=1}^{B} ||(R_k\boldsymbol{z}^t_i + \boldsymbol{d}_k - \boldsymbol{z}^{k,t}_i) + \beta(g^{k,t}_i - R_k g^t_i)||_2^2 \\
&= \frac{1}{B}\sum_{i=1}^{B} ||R_k\boldsymbol{z}^t_i + \boldsymbol{d}_k - \boldsymbol{z}^{k,t}_i||_2^2 && (16) \\
&+ \beta^2\frac{1}{B}\sum_{i=1}^{B} ||g^{k,t}_i - R_k g^t_i||_2^2 && (17) \\
&+ 2\beta\frac{1}{B}\sum_{i=1}^{B} \langle R_k\boldsymbol{z}^t_i + \boldsymbol{d}_k - \boldsymbol{z}^{k,t}_i, g^{k,t}_i - R_k g^t_i\rangle. && (18)
\end{aligned}
$$

Note that Equation 17 only depends on the rotation matrix R_k and it is a least-square problem, for which there is closed form solution. In particular, it can be shown that the solution for Equation 17 is $R^*_k = UV^\top$, where U and V are the two orthogonal matrices obtained in the Singular Value Decomposition of the $d \times d$ matrix obtained of multiplying the task-specific gradients $g^{k,t}_i$ with the target ones g^t_i, that is, $[g^t_1 g^t_2 \dots g^t_B] \times [g^{k,t}_1 g^{k,t}_2 \dots g^{k,t}_B]^\top = U\Sigma V^\top$. Note that in order to ensure that the matrix is a rotation, we should make sure that its determinant is one.

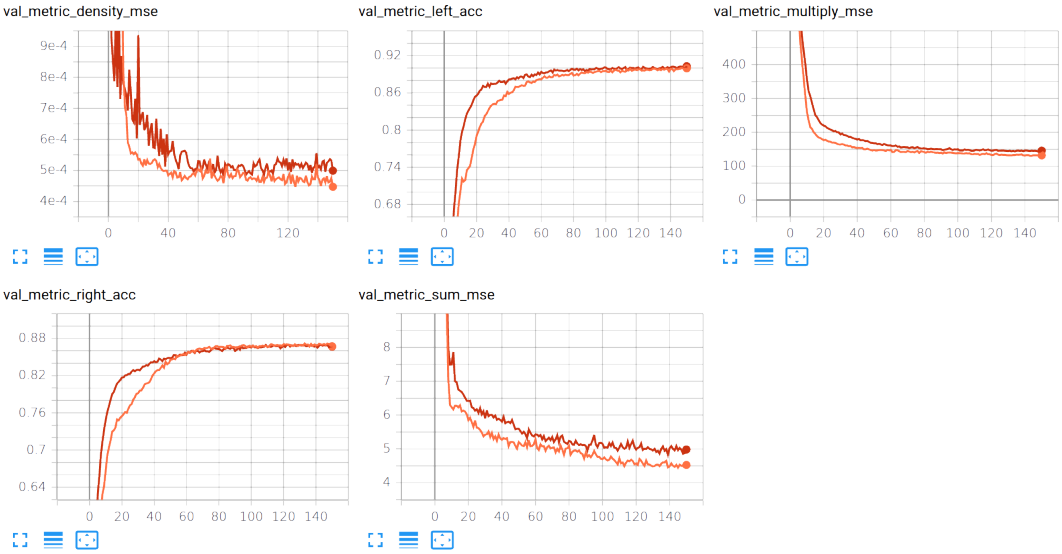
Similarly, it is straight-forward to show that the solution of the least-square problem for the translation d_k with respect to Equation 16 is given by $d^*_k = \tilde{\boldsymbol{z}}^{k,t} - R_k\tilde{\boldsymbol{z}}^t$, where $\tilde{\boldsymbol{z}}^{k,t}$ and $\tilde{\boldsymbol{z}}^t$ are the averages over the batch of $\boldsymbol{z}^{k,t}_i$ and $\boldsymbol{z}^t_i$, respectively. So, instead of having in Rotograd d^*_k as our translation parameter, we can have two parameters $\overline{\boldsymbol{z}}^k$ and $\overline{\boldsymbol{z}}$, that we update using the gradient update $\overline{\boldsymbol{z}}^k - \tilde{\boldsymbol{z}}^{k,t}$ and $\overline{\boldsymbol{z}} - \tilde{\boldsymbol{z}}^t$, respectively, and use the transformation $\boldsymbol{z}^k_i = R_k(\boldsymbol{z}_i - \overline{\boldsymbol{z}}) + \overline{\boldsymbol{z}}^k$ when doing the forward pass.

Finally, note that we can afford to ignore the error term from Equation 18 as long as our solutions in Equation 17 and 16 are good enough. The reason is that this term is nothing more but the scalar product between the residuals of the two previous problems. The error on this term will never be bigger than the maximum error committed in the other two problems.

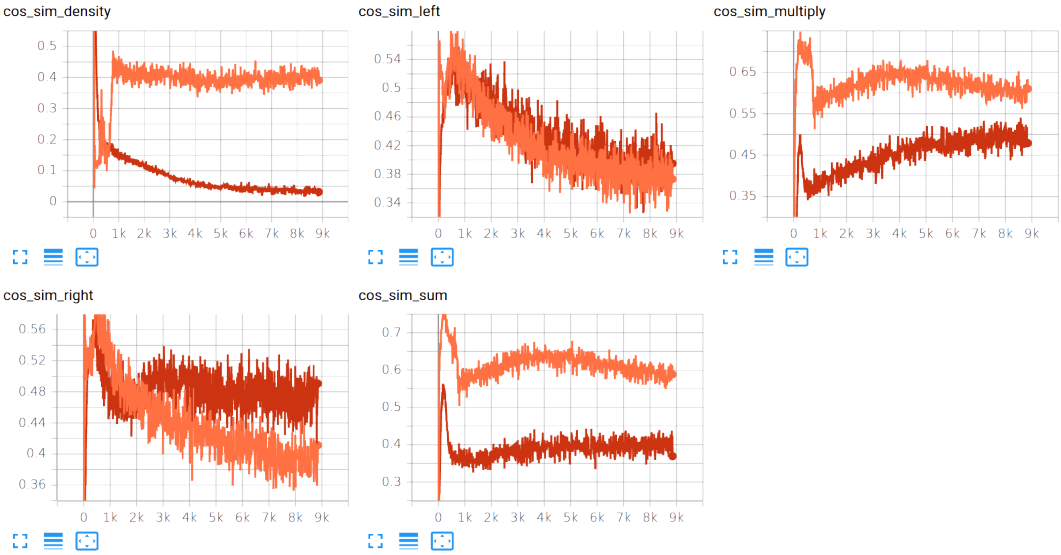
A.3 Further experimental results

In this section we present two additional plots regarding the experiments with MNIST. Specifically, Figure 4a shows the validation metrics during training for one run of Rotograd and *uniform*, showing how Rotograd outperforms *uniform* in all the regression metrics (being more stable in the density task), while having almost the same accuracy of the classification tasks. More importantly, Figure 4b shows the cosine similarity between the task gradient $\nabla_{\boldsymbol{z}}\ell_k(\boldsymbol{z})$ and the averaged gradient $\nabla_{\boldsymbol{z}}\ell(\boldsymbol{z})$ (averaged over the batch), that is, how similar is the direction these two gradients are pointing towards. This value is between -1 and 1, being -1 that they point to opposite directions, 0 that they are orthogonal (which is the most often case in high dimensional spaces), and 1 that they are pointing to the same direction.

Figure 4b clearly showcases the effectiveness of Rotograd in regards of homogenizing the direction of the task gradients. While keeping almost the same cosine similarity on the classification tasks, Rotograd highly increases the cosine similarity of the regression ones, being worth mentioning the case of the density task, which is pretty much orthogonal (i.e., *unrelated*) to the rest of tasks, achieving a cosine similarity of almost 0.5 when Rotograd is applied.



(a) Metrics over the validation set.



(b) Cosine similarity during training.

Figure 4: Tensorboard plots with the results of one run on the MNIST experiments during training for *uniform* and *rotograd* in red and orange, respectively. Figure (a) shows the validation metrics, where *rotograd* outperforms *uniform*. Figure (b) shows the cosine similarity between the task gradient $\nabla_{\boldsymbol{z}} \ell_k(\boldsymbol{z})$ and the averaged gradient $\nabla_{\boldsymbol{z}} \ell(\boldsymbol{z})$ (averaged over the batch), *clearly showing* how *rotograd* reduces the direction conflict among task gradients.

A.4 CHEST

In Table 2 we show the results obtained in a similar manner as in the MNIST case. We use the same model and dataset as in Rajpurkar et al. (2017), using a single linear layer for the task-specific modules, and we train using only a 20 % of the data.

Measure	*single task*	*uniform*	*rotograd*	*rotograd-sgd*	*gradnorm* $\boldsymbol{Z}$	*gradnorm* W	*d-grad*
atelectasis	74.58 ± 0.91	77.25 ± 0.77	76.90 ± 0.79	68.98 ± 1.87	73.39 ± 1.26	77.85 ± 0.63	72.05 ± 1.32
cardiomegaly	62.74 ± 5.23	83.00 ± 3.51	83.13 ± 1.86	62.70 ± 1.54	70.44 ± 4.79	85.00 ± 1.67	71.47 ± 4.20
effusion	84.92 ± 0.72	85.57 ± 0.29	84.88 ± 0.48	77.14 ± 1.19	80.76 ± 2.03	85.56 ± 0.64	79.07 ± 1.30
infiltration	63.89 ± 1.35	67.65 ± 0.39	67.37 ± 0.51	63.12 ± 0.68	64.24 ± 1.05	67.31 ± 0.49	64.35 ± 0.72
mass	52.43 ± 1.30	76.88 ± 1.19	76.01 ± 2.42	60.89 ± 1.51	62.78 ± 3.23	78.30 ± 1.61	63.09 ± 1.67
nodule	53.37 ± 2.10	70.03 ± 1.34	68.37 ± 2.06	56.22 ± 0.98	60.58 ± 2.88	70.76 ± 1.33	58.70 ± 1.01
pneumonia	56.47 ± 2.78	70.85 ± 1.89	70.67 ± 0.71	63.11 ± 1.75	64.99 ± 2.08	71.74 ± 1.29	66.14 ± 1.67
pneumothorax	67.08 ± 1.90	76.18 ± 2.33	76.27 ± 1.31	64.06 ± 2.54	70.19 ± 2.80	77.52 ± 2.55	69.41 ± 1.92
consolidation	70.14 ± 2.28	78.28 ± 0.53	78.24 ± 0.44	74.57 ± 0.48	76.03 ± 1.43	77.01 ± 1.27	75.47 ± 0.61
edema	82.35 ± 0.91	86.25 ± 0.46	85.44 ± 0.81	79.16 ± 1.17	82.17 ± 1.46	86.19 ± 0.66	83.09 ± 0.94
emphysema	49.35 ± 1.69	78.57 ± 2.95	78.44 ± 1.79	59.61 ± 2.18	64.82 ± 3.90	82.22 ± 1.38	65.55 ± 1.56
fibrosis	64.69 ± 2.06	76.47 ± 1.12	75.58 ± 1.68	65.10 ± 2.40	68.98 ± 2.31	77.69 ± 1.44	69.34 ± 0.55
pleural thick.	55.02 ± 1.36	72.65 ± 0.86	72.31 ± 0.99	62.79 ± 0.99	64.18 ± 2.68	72.38 ± 1.14	64.59 ± 1.06
hernia	68.58 ± 6.36	83.50 ± 4.29	88.57 ± 3.73	62.53 ± 7.81	77.64 ± 5.45	89.23 ± 1.84	70.23 ± 14.97
Δ_m	-	22.15 ± 1.37	21.98 ± 1.75	3.75 ± 1.17	9.84 ± 3.70	24.02 ± 1.28	9.30 ± 2.43

Table 2: Test results on the ChestX-ray14 dataset for different methods. All tasks are classification tasks.