Challenging Common Assumptions in Multi-task Learning

Cathrin Elich[‡,1,2,3], Lukas Kirchdorfer[‡,1,4], Jan M. Köhler[*,1], Lukas Schott[*,1]

[1]Bosch Center for Artificial Intelligence, [2]MPI for Intelligent Systems, Tübingen,
[3]Max Planck ETH Center for Learning Systems, [4]Uni Mannheim

cathrin.elich@tuebingen.mpg.de, jan.koehler@bosch.com, lukas.schott@bosch.com

[‡]Work done during an internship at Bosch. [*]Joint senior authors.

Abstract

While multi-task learning (MTL) has gained significant attention in recent years, its underlying mechanisms remain poorly understood. Recent methods did not yield consistent performance improvements over single task learning (STL) baselines, underscoring the importance of gaining more profound insights about challenges specific to MTL. In our study, we challenge common assumptions in MTL in the context of STL: First, the choice of optimizer has only been mildly investigated in MTL. We show the pivotal role of common STL tools such as the Adam optimizer in MTL. We deduce the effectiveness of Adam to its partial loss-scale invariance. Second, the notion of gradient conflicts has often been phrased as a specific problem in MTL. We delve into the role of gradient conflicts in MTL and compare it to STL. For angular gradient alignment we find no evidence that this is a unique problem in MTL. We emphasize differences in gradient magnitude as the main distinguishing factor. Lastly, we compare the transferability of features learned through MTL and STL on common image corruptions, and find no conclusive evidence that MTL leads to superior transferability. Overall, we find surprising similarities between STL and MTL suggesting to consider methods from both fields in a broader context.

1. Introduction

Multi-task learning (MTL) is gaining significance in the deep learning literature and in industry applications. Especially, tasks like autonomous driving and robotics necessitate real-time execution of neural networks while obeying constraints of limited computational resources. Consequently, there is a demand for neural networks capable of simultaneously inferring multiple tasks [18, 25].

In a seminal study, Caruana [3] highlights both advantages and challenges in MTL. On the one hand, certain tasks can exhibit a symbiotic relationship, resulting in a mutual performance enhancement when trained together. It was further suggested that features learned in a MTL scenario rely less on incidental correlations and demonstrate improved transferability. On the other hand, conflicts between tasks can arise and decrease the performance when trained jointly, also known as *negative transfer*.

Several approaches have been suggested to mitigate the issue of negative transfer among tasks during network training. Our study focuses on two main branches in the literature: First, *gradient magnitude* methods which incorporate weights to scale task-specific losses to achieve an adequate balance between tasks. Second, *gradient alignment* methods which aim to resolve conflicts in gradient vectors that may arise between tasks within a shared network backbone.

The effectiveness of the proposed MTL methods remains uncertain in the literature. Upon comparing various studies, it becomes evident that there is no definitive approach that consistently performs well across different settings [44]. This observation has been reinforced in more recent studies where competitive performance was achieved through unitary scaling in combination with common regularization methods [24] or tuned task weighting [45].

The current understanding of MTL is still limited and lacks a deeper comprehension of its underlying mechanisms. To address this gap, our study aims to challenge commonly held assumptions, such as the notion of gradient alignment, gradient magnitudes, and transferability of features. Our **contributions** are:

- We evaluate the empirical effectiveness of the Adam [21] optimizer in MTL which we show to perform favorably in various experiments in comparison to SGD + momentum. We trace this back to Adam addressing differences in gradient magnitudes by tracking first and second moment estimates of gradients.
- We theoretically attribute the performance of methods in MTL to their invariance w.r.t. to different loss scalings. We show this invariance of uncertainty weighting (UW) under mild conditions. Similarly, we demonstrate Adam's a partial loss-scale invariance in MTL.

- In contrast to implicit assumptions from previous studies [6,19,28,49], we present empirical evidence that conflicts arising from *gradient alignment* are not exclusive to MTL and can even be more pronounced in single-task learning.
- Corroborating the methods proposed to balance magnitude related issues across tasks [20, 29, 31, 45], we confirm that *gradient magnitudes* pose a challenge in MTL compared to single task learning.
- We examine the presumption of increased robustness on corrupted data as a result of MTL [23, 34]. We find no evidence that an increased number of tasks would consistently result in improved transferability.

Overall, we provide a vast set of experiments and theoretical insights which challenge common assumptions and contribute to a more comprehensive understanding of MTL in computer vision to guide future research.

2. Related Work

Recent work in multi-task learning (MTL) can be roughly divided into three different fields: *Network architectures* focus on the question of how features should be shared across tasks [31, 33, 35, 46]. *Multi-task optimization (MTO)* aims to resolve imbalances and conflicts of tasks during MTL. *Task affinities* examine a grouping of tasks that should be learned together to benefit from the joint training [11,30,43]. A general overview of recent works in MTL can be found in [39,44]. Our work focuses on MTO, which we will review more thoroughly in the following.

Gradient magnitude methods prevent the dominance of individual tasks by balancing them with task-specific weights. One line of works are loss-weighting methods. Here, weights are determined before any (task-wise) gradient computation and are used for a weighted aggregation of the tasks' losses. These methods consider either the task uncertainty (**UW**) [20], rate of change of the loss (**DWA**) [31], the tasks' difficulty (**DTP**) [13], or randomly chosen task weights (**RLW**) [26]. In line with these, the geometric mean of task losses has been used to handle the different convergence rates of the tasks [7]. An advantage of theses methods is their computational efficiency as the gradient needs to be computed only once for the aggregated loss. Alternatively, other methods consider the task-specific gradients directly, e.g., by normalizing them (**GradNorm**) [5] or computing scaling factors which produce an aggregated gradient with equal projection onto each task's gradient (**IMTL**) [29]. Furthermore, there are several adaptions for the multiple-gradient descent algorithm (**MGDA**) [9], e.g. for applying it efficiently in deep learning setups [40] or by introducing a stochastic gradient correction [10]. Recently, task-wise gradient weights have been estimated by treating MTL as a bargaining problem (**Nash-MTL**) [37], or considering a stability criterion (**Aligned-MTL**) [41]. Crucially, in the context of this study, all gradient magnitude methods consider *scalar* weightings of task-wise gradients within the backbone and/or heads. They do not modify the alignment of task-specific gradient vectors.

Gradient alignment methods perform more profound vector manipulations on the task-wise gradients w.r.t. to the network weights of a shared backbone before aggregating them. The underlying assumption indicates conflicting gradients as a major problem in MTL. To address this, **GradDrop** [6] randomly drops gradient components in the case of opposing signs. **PCGrad** [49] proposes to circumvent problems of conflicting gradients by projecting them onto each other's normal plane. Following this idea, Liu et al. [28] propose **CAGrad** to converge to a minimum of the average loss instead of any point on the Pareto front. **RotoGrad** [19] rotates gradients at the intersection of the heads and backbone to improve their alignment. Shi et al. [42] propose to alter the network architecture based on the occurrence of layer-wise gradient conflicts. Lastly, [38] use separate optimizers per task.

Recent studies ***question the effectiveness of optimization-based methods*** in MTL. Xin et al. [45] execute an extensive hyperparameter search to show that simple scalar task-weighting performs equivalent or superior to many aforementioned multi-task optimization methods. Their hyperparameter search not only include the task-weights, but also common deep learning parameters such as the learning rate and regularization. Concurrently, [24] empirically show that fixed task-weights combined with regularization and stabilization techniques yield to equivalent performance compared to sophisticated multi-task optimization methods. We extend both critical studies to provide a more nuanced perspective. In particular, we theoretically and empirically demonstrate that the choice of optimizer is crucial and could potentially help to explain discrepancies found in prior studies (4.1). We further specifically distinguish between gradient alignment and gradient magnitude methods (4.2).

3. Problem Statement

Multi-task learning addresses the problem of learning a set of T tasks simultaneously. We consider supervised learning setups, use a shared backbone architecture, and learn all tasks together. Formally, given input data $\mathcal{X}$, the goal is to learn a function $f_{\boldsymbol{\theta}}(x)$ which maps a point $\mathbf{x} \in \mathcal{X}$ to each task label y_t with $t = 1, .., T$. The trainable parameters $\boldsymbol{\theta} = \{\phi, \psi_{1:T}\}$ consist of *shared* parameters ϕ and *task-specific* parameters ψ_t. Training a task t is associated with the loss $\mathcal{L}_t(f_{\boldsymbol{\theta}}(x); \boldsymbol{\theta})$, e.g., a regression or classification loss. We denote respective gradients on the shared and task-specific parameters with $\mathbf{g}_t^{\phi} = \nabla_{\phi}\mathcal{L}_t$, and $\mathbf{g}_t^{\psi} = \nabla_{\psi}\mathcal{L}_t$. When training on multiple tasks, the shared parameters ϕ needs to be updated w.r.t. all task-wise gradients $\mathbf{g}_t^{\phi}$ which requires an appropriate aggregation. A simple solution is to

uniformly sum up the task losses $\mathcal{L} = \sum_t \mathcal{L}_t$ which is referred to as *Equal Weighting* (**EW**). However, as tasks might be competing against each other, this can result in negative transfer and thus sub-optimal solutions. One way to deal with this difficulty is to adapt the *magnitude* of task-specific gradients. This can be achieved by weighting tasks during training, e.g., by scaling different losses $\mathcal{L} = \sum_t \alpha_t \mathcal{L}_t$, where often $\sum_t \alpha_t = 1$. Note, the α_t can change during training. Furthermore, the weighing can also be performed on gradient level to distinguish between shared and task-specific gradients. We refer to those approaches as *gradient magnitude* methods. Interestingly, the relationship between loss weights, network-updates and learning rate also depends on the optimizer. We show a derivation for SGD and Adam in Appendix A1.2. Additionally to adapting the gradient magnitude, one can directly adapt the *alignment* of task-wise gradient vectors within the shared backbone $\tilde{\mathbf{g}}^{\phi} = \mathbf{h}(\mathbf{g_1}^{\phi}, ..., \mathbf{g_T}^{\phi})$.

In practice, an optimum for $\boldsymbol{\theta}$ that yields best performance on all tasks often does not exist. Instead, improving performance on some task often yields a performance decrease in another task. To still enable a comparison across network instances in MTL, an instance $\boldsymbol{\theta}^*$ is called to be *Pareto optimal*, if there is no other $\boldsymbol{\theta}'$ such that $\mathcal{L}_t(\boldsymbol{\theta}') \leq \mathcal{L}_t(\boldsymbol{\theta}^*)\ \forall t$ with strict inequality in at least one task. The *Pareto front* consists of the Pareto optimal solutions.

4. Experiments and results

In this section we perform several experiments to gain a more profound understanding of multi-task learning in computer vision by questioning common assumptions. We compare the impact of Adam and SGD in multi-task learning in section 4.1, examine the process of gradient similarity in different settings in section 4.2, and evaluate the generalization performance on corrupted data in section 4.3. Throughout this evaluation, we repeatedly make use of common setups, which we will specify as follows and in more detail in appendix A3.

Datasets For our experiment, we consider three different datasets that are commonly used for evaluating multi-task learning in computer vision:

CityScapes [8] contains images of urban street scenes. In line with previous work, we consider the tasks of semantic segmentation (7 classes) and depth estimation. *NYUv2* [36] is an indoor dataset for scene understanding which was recorded over 464 different scenes across three different cities. Besides semantic segmentation (13-class) and depth estimation, it also contains the task of surface normal prediction. *CelebA* [32] consists of 200K face images which are labeled with 40 binary attributes.

Networks We use network architectures with hard-parameter sharing, which consist of a shared backbone and task-specific heads. For the dense prediction tasks on CityScapes and NYUv2, we compare SegNet [1] and DeepLabV3+ [4]. While both networks contain a similar number of shared parameters, DeepLabV3+ has relatively more of task-specific parameters. Experiments on CelebA are performed on a ResNet-18 backbone [14] with an additional single linear layer for each head.

Training For each method, we follow the loss or gradient aggregation as described in the related work, e.g., for equal weighting all task-specific losses are simply summed up to compute the joint network gradients. We also tune the learning rate for each approach. We use the validation set performance of the Δ_m metric as early stopping criteria. The Δ_m metric [33] measures the average relative task performance drop of a method m compared to the single-task baseline b using the same backbone and is computed as $\Delta_m = \frac{1}{T}\sum_{t=1}^{T}(-1)^{l_t}(M_{m,t} - M_{b,t})/M_{b,t}$ where $l_t = 1$ if a higher value means better for measure $M_{.,t}$ of some task metric t, and 0 otherwise.

4.1. The reasonable effectiveness of Adam in multi-task learning

We investigate the impact of Adam with respect to stochastic gradient descent with momentum (SGD+mom) in conjunction with common multi-task learning (MTL) methods. We identify the choice of optimizer as an important confounder in the experimental setup. Previous works reported mixed performances on dedicated multi-task weighting methods compared to a plain scalar weighting. For instance, Adam [21] was successfully used in studies to show that random/constant weighting of tasks' losses performs competitive compared to MTO methods [24, 26, 45]. In contrast, many methods proposing adaptive, task-specific weighing methods [20, 29] use the SGD optimizer with momentum (see Table A1 for an overview). Intuitively, gradient magnitude differences between tasks could require an adequate balancing which, in the linear case, could also be seen as a task-specific learning rate (derivation in Appendix A1.2). Overall, Adam's capability of estimating the learning rate per parameter based on the first and second moment estimates of the gradients might help to mitigate claimed differences in gradient magnitude.

4.1.1 Toy Task Experiment

To get a first impression of the impact of the optimizer and common hyperparameters such as the the learning rate, we investigate the impact of Adam and plain gradient descent (GD) in a simple toy task.

Approach We repeat the experiment of Liu et al. [28] using the original implementation but further tested different learning rates and optimizers. They motivate their gradient alignment method CAGrad with a simple toy optimization problem in which their method reliably converges to the

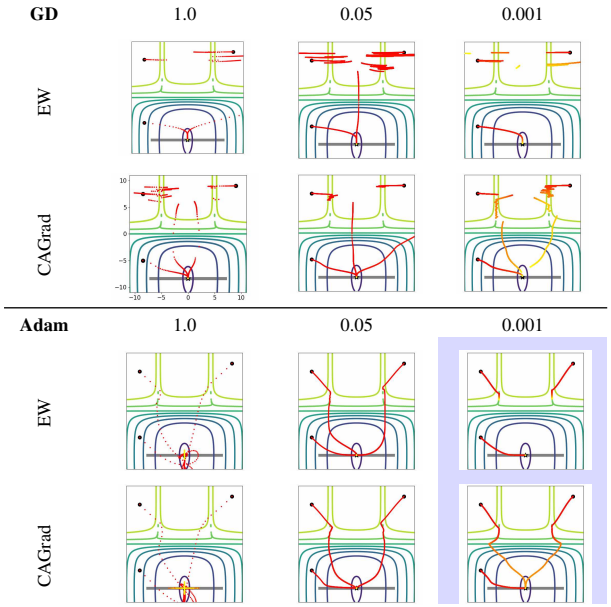


Figure 1. **Toy Task Experiment from CAGrad [28] for different learning rates and optimizers.** Consistent with results from [45], we observe that the choice of the learning rate is crucial even for this toy optimization problem. Moreover, it becomes apparent, that selecting Adam over simple gradient decent (GD) yields superior results. The contour lines depict the 2D loss landscape and the optimization trajectories are colored from red to yellow for 100k iteration steps from three different starting points (seeds). Liu et al. [28] used Adam and lr=0.001. Trajectories for further learning rates and methods can be found in Figure A5.

minimum of the average loss, while other MTO approaches would either get stuck (e.g., EW) or only convert to any point on the Pareto front (e.g., PCGrad [49], MGDA [40]).

Result and conclusion We find that for higher learning rates with the Adam optimizer, even the simple equal weighting (EW) method reaches the global optimum (cf. Figure 1, e.g., EW with Adam and lr=0.05) and often converges faster than dedicated multi-task optimization methods such as CAGrad (Table 1). Note, original results were shown for learning rate 0.001 using Adam (blue shaded column) and were, therefore, in favor of CAGrad. Furthermore, the choice of optimizer appears to be more important on the success of the outcome of the experiments than the choice of multi-task optimization method, as Adam converges considerably faster and more reliably than GD.

4.1.2 Experiments on CityScapes and NYUv2

We further extensively test the effectiveness of Adam and its role as a confounder in common MTL datasets for various multi-task optimization methods.

Approach We compare Adam and SGD+mom in combination with any MTO method from equal weighting (EW), uncertainty weighting (UW) [20], random loss weighting

		learning rate				
	method	10.0	1.0	0.1	0.01	0.001*
GD	EW	-	-	-	-	-
	PCGrad	-	-	-	-	-
	CAGrad	644	213	8,069	20,418	-
Adam	EW	*26*	**22**	**709**	**9,015**	-
	PCGrad	**25**	56	34,175	-	-
	CAGrad	27	*32*	*802*	*11,239*	**57,700**

*LR used for results in [28] with Adam

Table 1. **Number of iterations after which all seeds in the toy task experiment from CAGrad [28] reach the global minimum**. We report the maximum iteration number over all three seeds for each MTO method, learning rate, and optimizer combination. In several setups, EW+Adam shows the fastest convergence to the global minimum. If not all seeds converged to the global minimum within 100k iteration steps, we denote it as '-'. As reported in previous work, we found that PCGrad converges only to some point on the Pareto Front. The **best** and *second best* run for each learning rate over all MTO methods are indicated via font type. Results for additional learning rates are reported in Table A7.

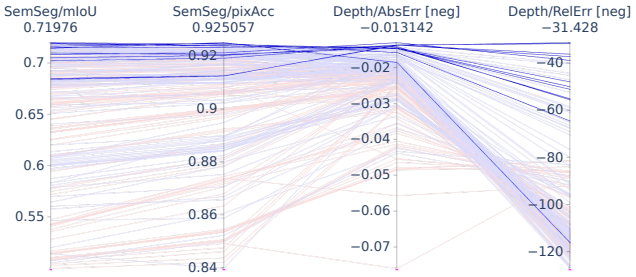


Figure 2. **Parallel coordinate plot over all experiments run on CityScapes and SegNet.** We distinguish between experiments using SGD+mom and Adam optimizer. Experiments that reached Pareto front performance are drawn with higher saturation. We observe that Adam clearly outperforms the usage of SGD+mom. Further results can be found in the Figure A4.

(RLW) [26], PCGrad [49], CAGrad [28], and IMTL [29] for which we used the implementation from [27]. We distinguish between any combination of dataset {CityScapes [8], NYUv2 [36]} and network architecture {SegNet [1], DeepLabv3 [4]}. We run experiments for ten different initial learning rates from $[0.5, 0.1, 0.05, ..., 0.00001]$. More details are described in Appendix A3.2. As we observe that some setups on Cityscapes yield unsuitable results, we exclude runs with $mIoU < 0.1$ on the segmentation task. For both datasets, we further remove diverged models with NaN output, e.g., due to inadequate learning rates.

As different models and parameter setups typically show preference towards different tasks and metrics, we are interested in those models which are Pareto optimal (PO). We report the number of models whose performance lie on the overall Pareto front for either Adam or SGD+mom.

Results We observe over all experimental setups that Adam performs favorably over SGD+mom (Table 2). This

			Adam		SGD+mom.	
		#Exp. (valid) [Adam / SGD]	PO (full)	PO w.r.t. SGD	PO (full)	PO w.r.t. Adam
CityScapes	SegNet	[151 / 149]	12	109	0	0
CityScapes	DeepLabV3	[177 / 178]	15	108	0	0
NYUv2	SegNet	[180 / 169]	9	20	2	2
NYUv2	DeepLabV3	[177 / 179]	16	31	7	7

Table 2. **Comparison of number of Pareto optimal (PO) experiments using either Adam or SGD+momentum as optimizer.** Models trained with Adam are consistently more often on the Pareto front compared to those trained with SGD+mom. The number of Adam-based runs that are not dominated by any SGD-based run (PO w.r.t. SGD) is even higher, which does not hold the other way around. We further list the number of Pareto optimal runs corresponding to each MTO method in Table A2.

especially holds true for experiments on CityScapes where the Pareto front for both network architectures only consists of Adam-based models. Moreover, an even larger number of Adam-based models is not dominated by any model trained with SGD+mom (PO w.r.t. SGD). For NYUv2, Adam still performs stronger but SGD+mom. also occasionally delivers a PO result.

For the individual metrics, the predominance of Adam on Cityscapes is further visualized in a parallel coordinate plot in Figure 2. Bold lines indicate the overall Pareto optimal experiments (PO full).

In Appendix A4 we further report best Δ_m results for every MTO method in combination with Adam or SGD+mom (Tables A3 to A6). Again, Adam boosts the overall performance across methods. Noteworthy, no multi-task optimization method seems to be Pareto dominant over plain EW with Adam which supports claims questioning the effectiveness of specific MTO methods [24,45]. Nonetheless, looking at the Δ_m-metric and individual metric, we see that sometimes with a small relative performance drop on one metric, significant gains on another metric can be achieved (e.g., Cityscapes+Semseg and depth for UW vs EW).

Conclusion Overall, we conclude that not only a well-tuned learning rate but also the optimizer is crucial for MTL performance. In a fair and extensive experimental comparison, we were able to show that Adam shows superior performance in MTL setup compared to SGD+mom.

4.1.3 The reasonable effectiveness of Adam in the context of uncertainty weighting

We attribute our observed effectiveness of Adam in MTL to its partial loss-scale invariance [21] which we show theoretically and empirically by a handcrafted loss-scaling experiment. This invariance can also be shown under mild assumptions for uncertainty weighting (UW) [20] which is the most prevalent loss weighting method in the literature.

The loss-scale invariance of UW can be shown by assuming an optimal solution for the σ values similar to [22]. This assumption is mild as this is a 1-dimensional convex optimization problem for each σ. The invariance can be demonstrated by inserting the analytical solution starting from UW. For example, assuming a Laplacian distribution for simplicity (this can be shown for other distributions as well), we have

$$\min_{\sigma_t} \frac{1}{\sigma_t}\mathcal{L}_t + \log \sigma_t \;\Rightarrow\; \sigma_t = \mathcal{L}_t \tag{1}$$

The left hand side shows the typical form of UW, as shown for a Gaussian in [20, eq.(5)] Here, $\mathcal{L}_t$ is a task-specific loss and σ_t is a scalar parameter that is usually learned. Plugging back the solution of the optimal solution into the UW we get

$$\mathcal{L} = \sum_t \frac{\mathcal{L}_t}{sg[\mathcal{L}_t]} + c, \tag{2}$$

where sg is the stop-gradient operator and c is a constant that can be omitted during optimization. Given this equation, we can directly, see the invariance w.r.t. loss-scalings. For instance, with $\mathcal{L}_1 \to \alpha_1\mathcal{L}_1$ and $\mathcal{L}_2 \to \alpha_2\mathcal{L}_2$, the derivative of the total loss $\mathcal{L}$ remains unchanged. As this invariance can be shown on the loss-level, it holds for all gradient updates w.r.t. the head and backbone. Intuitively, this explains why UW is invariant to loss scalings such as measuring depth in centimeters, meters or inches. For further details, we refer to Appendix A1.

Similarly for Adam, we can prove a scale invariance of losses in MTL that holds for the parameters of network heads. As before, we assume a hydra-like network architecture with a shared backbone and task-specific heads. We start with the parameter-update rule from Adam and scale the corresponding losses $\mathcal{L}_t \to \alpha_t\mathcal{L}_t$. When only considering the parameters of the corresponding heads ψ_t, the scalings α_t cancel out

$$\psi_{t,i} = \psi_{t,i-1} - \frac{\gamma}{\sqrt{\alpha_t^2\hat{v}'_t}}\alpha_t\hat{m}'_t. \tag{3}$$

Thus for the network heads, we see the same effect as for UW that different scalings do not impact the network update. However, this does not hold for the backbone. The full derivation is shown in Appendix A1. We confirm empirically in a handcrafted loss-scaling experiment in Appendix A2 and Figures A1 and A2 that SGD does not offer any scaling invariance, whereas Adam involves the invariance property for the heads. The optimal UW demonstrates a scaling invariance for the heads and the backbone.

We would like to note that the derivation for Adam is only valid for constant α_t, e.g., measuring depth in different units. In case of dynamic loss weights that are not constant (e.g., UW), the weights do not cancel out fully due to the accumulation of gradient histories within Adam. Nonetheless, this has profound implications for loss weighting methods that are used in conjunction with Adam. For instance, when

turning off the history within Adam and having a fixed backbone (by setting $\beta_{1,2} = 0$), all loss weighting methods, such as UW, random loss weighting and others, become equivalent to equal weighting (also for the backbone).

Conclusion During MTL network training, we derive and measure a full loss-scale invariance for an optimal UW and a partial invariance for Adam. This partial invariance does not hold for SGD+momentum and could explain the effectiveness of Adam in MTL. Thus, when comparing different loss weighting methods, it is crucial to be aware of the influence of the optimizer.

To investigate the real-world presence of differences of gradient magnitudes and the gradient alignment within the network backbone, we provide empirical investigations in the subsequent section.

4.2. Investigating gradient conflicts in multi-task learning and single-task learning

Given our prior experiments showing the effectiveness of a standard (=single-task) learning tool such as the Adam optimizer over dedicated multi-task optimization tools, we challenge notion of multi-task learning (MTL) as it is commonly interpreted in the literature.

Motivation The common interpretation of MTL is fairly vague. In computer vision, tasks are often defined to be segmentation and depth (CityScapes) or recognizing multiple attributes (CelebA). This might actually not be fully distinguished from regular single-task learning. We argue that in an extreme case, even recognizing a single cat in multiple images could be considered MTL. The cat could be hiding behind a plant and only revealing its eyes, requiring a neural network to recognize the cat solely based on the eyes. In other images, the cat might hiding under the couch only revealing its paws, or mad and curled up into a furry ball because we took so many pictures. This would require a paw or fur classifier. Overall, a neural network is required to solve multiple tasks to reliably recognize our cat.

Motivated by this example, we would like to quantify the distinction between STL and MTL in common datasets from a perspective of the multi-task optimization literature, which inspects gradient conflicts in neural networks. In particular, we challenge the view point of conflicts between different gradients being a *specific* problem in MTL [49]. While several works follow the idea of overcoming gradient conflicts in MTL [19,28,42], the appearance of gradient conflicts has only been mildly investigated so far.

Prerequisite To quantify differences between STL and MTL, we compare gradients between tasks t and samples $\mathbf{x}_i$. We can compare the alignment of two gradients $\mathbf{g}, \mathbf{g}'$ on the shared parameters, e.g., of task a and task b, with the cosine similarity

$$S_{cos}(\mathbf{g}, \mathbf{g}') = \cos(\phi) = \frac{\mathbf{g} \cdot \mathbf{g}'}{\|\mathbf{g}\| \|\mathbf{g}'\|}. \tag{4}$$

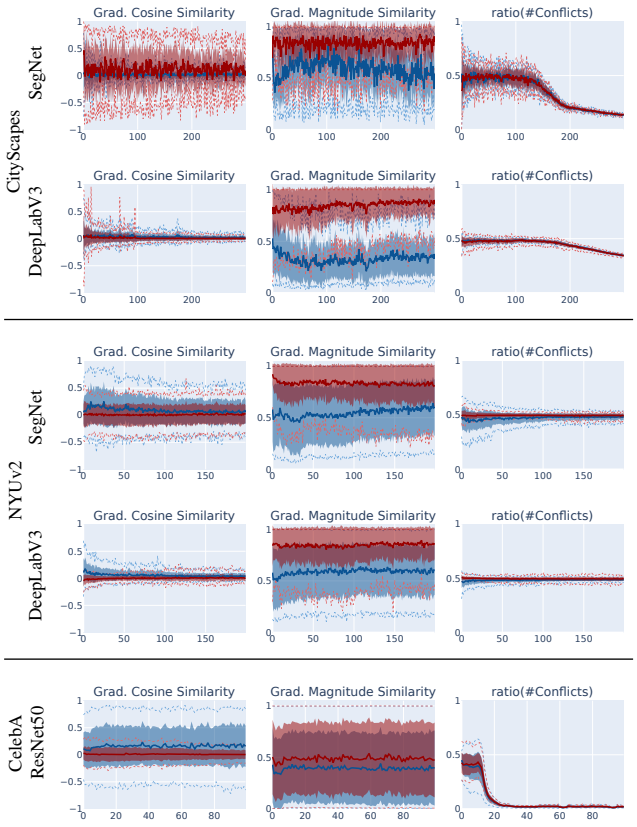


Figure 3. **Comparison of gradient similarities and conflicts** between datasets and network architectures over training epochs. For each dataset and network combination, we report (from left to right) gradient cosine similarity, gradient magnitude similarity, and the ratio of conflicting gradient parameters with respect to gradient pairs corresponding to either different samples (STL setting) or different tasks (MTL setting). We report mean (solid line), standard deviation (shaded area), upper (97.5%) and lower (2.5%) percentile (dotted line) within an epoch. Overall, the direction conflicts are similar (first / last column), whereas the magnitude differences are more pronounced in MTL (middle column).

Thus, two gradients are in conflict, if their cosine similarity is smaller than zero [49]. In particular, S_{cos} is $1/-1$ if gradients point in the same/opposite direction and 0 in case of orthogonal directions. The gradient magnitude similarity

$$S_{mag}(\mathbf{g}, \mathbf{g}') = \frac{2\|\mathbf{g}\|_2 \cdot \|\mathbf{g}'\|_2}{\|\mathbf{g}\|_2^2 + \|\mathbf{g}'\|_2^2} \tag{5}$$

as defined in [49] yields values close to 1 for gradients of similar magnitude, or close to 0 for large discrepancies in magnitude. High dissimilarity in both gradient direction and magnitude is presumed to be a common MTL problem.

Experimental setup During the training on aforementioned datasets, we examine gradient similarity across two different setups: (1) between gradients of different tasks with respect to a single sample (typical MTL setup), e.g.,

$\mathbf{g} = \nabla_\phi L_0\left(f_{\boldsymbol{\theta}}(\mathbf{x}_i)\right)$ and $\mathbf{g}' = \nabla_\phi L_1\left(f_{\boldsymbol{\theta}}(\mathbf{x}_i)\right)$; and (2) between gradients corresponding to the same task but different samples within a batch (typical gradient behavior in STL), e.g., $\mathbf{g} = \nabla_\phi L_t\left(f_{\boldsymbol{\theta}}(\mathbf{x}_0)\right)$ and $\mathbf{g}' = \nabla_\phi L_t\left(f_{\boldsymbol{\theta}}(\mathbf{x}_1)\right)$. For both setups, we compute the previously named measures, gradient cosine similarity and gradient magnitude similarity as well as the ratio of conflicting gradient parameters. We are aware that our comparison between samples and tasks is not direct. Nonetheless, it serves as a coarse indicator to estimate their impact during network training. Implementation details can be found in appendix A3.

Results We show the evolvement of different gradient similarity measures over epochs in Figure 3. Surprisingly, when comparing STL (red line) and MTL (blue line), we find no consistent evidence for gradient alignment conflicts (left column) to be an exclusive problem of MTL. For instance, for Cityscapes, the variation of gradient alignment is fully encapsulated within the spread we observe in STL. For CelebA, the converse seems to be mostly the case. Furthermore, the choice of network architecture and distribution of task-specific and shared parameters (SegNet vs. DeepLabV3) can have a large influence on the spread of the cosine-similarity. Both architectures have roughly a similar number of shared-parameters. However, DeepLabV3 has a higher number of task-specific parameters which seems to reduce the variance in conflicts for both MTL and STL (variances decrease on row one vs. two and row three vs. four). In line with these observations, we found a similar number of conflicting gradient parameters (last column) for both STL and MTL among all experiments.

For gradient magnitude similarities (middle column), we observe a clearer pattern. The similarity in magnitudes are continuously (in the mean) less pronounced in MTL compared to STL (blue line is below red one in all settings). Interestingly, the relative difference between the two setups remains similar over training which justifies the choice of fixed scalar task weightings as done in [45]. Further measures can be found in Figures A6 to A9.

Conclusion Overall, we conclude that the difficulty of MTL as opposed to STL is predominantly due to differences in gradient magnitudes. Although the problem of conflicting gradients has been typically associated with MTL [19, 28, 49], we found that gradient conflicts can actually be even more pronounced in STL. Thus, gradient-alignment methods should be considered in a wider context in deep learning. Furthermore, this finding rises the question, whether common, well developed methods from STL already address gradient conflicts in deep learning.

4.3. Robustness of multi-task representations on corrupted data

In the last part of our analysis, we investigate whether features learned for multiple tasks generalize better to corrupted data compared to those learned for single tasks only.

Motivation In his seminal paper [3], Caruana gives preliminary evidence that MTL provides stronger features and avoids spurious correlations (referred to better *attribute selection* in [3]). More recently, spurious correlations have often been directly connected with robustness [12, 17]. Results from current literature on the robustness of MTL features are mixed. While MTL is stated to increase the adversarial- and noise-robustness over STL [23,34,47], others argue features selected by MTL could be more likely to be non-causal and, therefore, less robust. [2, 16]. Here, we further challenge the assumption whether MTL features lead to better robustness. We would like to nuance that we do not challenge the transferability of representations, e.g., to new tasks, but solely focus on the claim that the MTL trained features are more robust w.r.t. different inputs.

Approach For our experiment, we use models trained on clean data and test these on corrupted data. We select the model with the best performing hyperparameter configuration from previous experiments and compare the test performance of models trained in an multi-task setup to those that learned a single-task only. Note that none of the used models were explicitly trained to handle data corruption. For testing, we use of the different perturbation modes proposed by Hendrycks et al. [15] which include different variants of noise, blur, and weather conditions and apply five levels of severity. We create a corrupted version of the test data for both CityScapes and NYUv2 for all proposed corruptions and perturbation levels (details in Appendix A3).

To quantify the robustness of single- and multi-task models, we first compute the individual task metrics M (e.g., mIoU) per task t for a STL and MTL network. Next, we compute the relative performance when each model is faced with corrupted data. Lastly, we calculate the difference of relative performances of the MTL compared to the STL model. In detail, over all corruption modes C and levels of severity S we have

$$\delta_t = \frac{1}{|C| \cdot |S|} \sum_{c \in C} \sum_{s \in S} (-1)^{p(t)} \delta_{t,c,s} \tag{6}$$

$$\text{with} \quad \delta_{t,c,s} = \frac{M_{t,c,s}^{MTL,corrupted}}{M_t^{MTL,clean}} - \frac{M_{t,c,s}^{STL,corrupted}}{M_t^{STL,clean}}$$

where $p(t) = 1$ if a higher value on task t corresponds to better performance and $p(t) = 0$ otherwise. This metric yields a negative value $\delta_t < 0$ if the MTL model was able to handle data corruption better. If the STL model is more robust, it holds that $\delta_t > 0$.

Results Figure 4 shows δ_t for different corruption types for a SegNet with EW on NYUv2 averaged over five corruption levels and three random runs. On the semantic segmentation and depth task, the STL models show a lower decrease in performance on the corrupted data than MTL ($\delta_t > 0$ more often; shaded in red), indicating that the

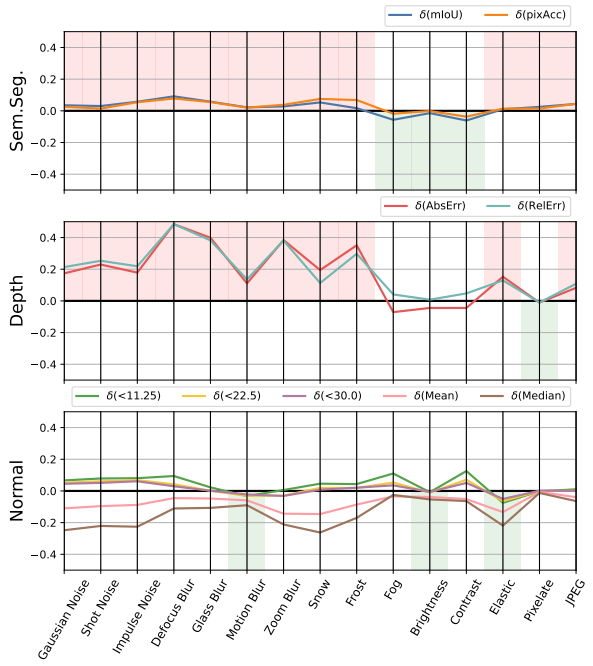


Figure 4. **Comparison of generalization performance to corrupted data for MTL and STL on NYUv2 with SegNet and EW.** We show the difference over relative performance decrease over all corruption modes averaged over five levels of severity and three runs. We color blocks in case either STL or MTL is able to handle the respective corruption better for all metrics of one task. STL is more robust compared to MTL models for the semantic and depth task In contrast, the MTL model was able to deal better with a couple of corruption modes for the normal task. Results for other setups can be found in the supplementary material.

	Sem.Seg.		Depth		
MTO	δ_{mIoU}	δ_{pixAcc}	δ_{AbsErr}	δ_{RelErr}	
EW	0.0226	0.0298	0.1713	0.1869	
UW	0.0149	0.0200	0.1691	0.1652	
RLW	0.0236	0.0243	0.1250	0.1266	
IMTL	0.0144	0.0174	0.2434	0.2721	
PCGrad	0.0206	0.0156	0.1665	0.1634	
CAGrad	0.0053	0.0147	0.2006	0.2282	
	Normal				Mean
	δ_{Mean}	δ_{Median}	$\delta_{<11.25}$	$\delta_{<22.5}$	$\delta_{<30.0}$	
EW	-0.0750	-0.1390	0.0378	0.0167	0.0129	0.0293
UW	-0.0461	-0.0895	0.0200	0.0168	0.0165	0.0319
RLW	-0.0942	-0.1717	0.0342	0.0152	0.0110	0.0104
IMTL	-0.0142	-0.0356	0.0342	0.0249	0.0217	0.0643
PCGrad	-0.0594	-0.1131	0.0364	0.0208	0.0171	0.0298
CAGrad	-0.0018	-0.0138	0.0402	0.0284	0.0240	0.0584

Table 3. **Out-Of Distribution generalization on corrupted NYUv2 [36] for different MTO methods on SegNet.** We report difference between relative performance decrease for STL and MTL averaged over all modes of corruption and severity (cf. Equation (6)). Mean of three runs is reported.

features learned for these respective tasks can better generalize to corrupted data. In contrast, the MTL model shows slightly better relative performance on the normal task ($\delta_t < 0$ more often for some of the normal metrics). To summarize, we see a slight indication that the depth and segmentation task might be more robust for STL than MTL.

The results of other multi-task optimization methods (cf. Table 3) are similar to EW. Only IMTL and CAGrad show some stronger δ_t average for the depth task. Though, results on NYUv2 with DeepLabV3 (Table A9) and for Cityscapes (Table A8) could not work out a consistent pattern, as sometimes MTL is more robust and sometimes STL.

Conclusion We cannot confirm the outcome of [23] as we do not see any indication that the segmentation task is generally more robust in the MTL setting.

Controversial to the claim of [34], our evaluation shows that none of the MTL approaches, even IMTL, PCGrad or CAGrad which adjust the gradients, yields consistent values of $\delta_t < 0$ which would have shown an advantage of certain MTO methods over STL.

Instead, it depends on the task, the type of corruption, the network, the dataset and chosen hyperparameters whether MTL or STL is superior towards corrupted data. Whether there is a general pattern, we leave to further research.

5. Conclusion and outlook

This study aims to enhance our understanding of multi-task learning (MTL) in computer vision, providing valuable insights for future research as well as guidance for implementations of real-world applications.

We show that common optimization methods from single task learning (STL) like the Adam optimizer are effective in MTL problems. We explain this with Adam's partial loss-scale invariance. Next, we compare gradient statistics during training for STL and MTL. While gradient magnitudes are a specific problem in MTL, we find the variability in gradient alignment to be similar in STL and MTL.

Thus, we encourage a more unified viewpoint in which specific multi-task optimization methods are also considered in single-task problems and vice versa. Furthermore, current methods often require exhaustive hyperparameter searches to perform well on multiple tasks due to the unknown/ non-linear behavior of the Pareto front and optimization landscape [45]. Hence, methods alleviating this extensive search and finding more robust methods are a promising direction. Lastly, our understanding of task (and sample) specific capacity allocation within a network and how best to tune it to custom requirements, is still not thoroughly understood. Often task-weights are increased to assign more importance to a task which is in contrast to tuning the learning rate per task where sometimes a smaller learning rate can be beneficial. Therefore, we require further investigations and disentanglement of these two concepts.

For the ongoing debate regarding the robustness of features from multi-task approaches, we find empirical evi-

dence that multi-task features can actually be more brittle than single-task features. Here, future work could try to disentangle the different current perspectives from causal features [16], adversarial/noise robustness [23, 34, 47] and our results on common corruption robustness.

Acknowledgments

We thank Claudia Blaiotta, Martin Rapp, Frank R. Schmidt, Leonhard Hennicke, and Bastian Bischoff for their feedback and valuable discussions. Cathrin Elich thanks her supervisors, Jörg Stückler and Marc Pollefeys, for enabling the opportunity to pursue an internship during her Ph.D. studies.

The Bosch Group is carbon neutral. Administration, manufacturing and research activities do no longer leave a carbon footprint. This also includes GPU clusters on which the experiments have been performed.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 3, 4, 10, 11

[2] Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. In *Proceedings of the European conference on computer vision (ECCV)*, pages 456–473, 2018. 7

[3] Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997. 1, 7

[4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Computer Vision – ECCV 2018*, 2018. 3, 4, 7, 10, 11

[5] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In Jennifer G. Dy and Andreas Krause, editors, *Proc. of ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 793–802. PMLR, 2018. 2

[6] 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 Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 2

[7] Sumanth Chennupati, Ganesh Sistu, Senthil Yogamani, and Samir A Rawashdeh. Multinet++: Multi-stream feature aggregation and geometric loss strategy for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019. 2

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 3213–3223. IEEE Computer Society, 2016. 3, 4, 7, 10, 15

[9] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350:313–318, 2012. 2

[10] Heshan Devaka Fernando, Han Shen, Miao Liu, Subhajit Chaudhury, Keerthiram Murugesan, and Tianyi Chen. Mitigating gradient bias in multi-objective learning: A provably convergent approach. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. 2

[11] Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27503–27516, 2021. 2

[12] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020. 7

[13] 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)*, 2018. 2

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. 3, 7

[15] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *Proc. of ICLR*. OpenReview.net, 2019. 7

[16] Ziniu Hu, Zhe Zhao, Xinyang Yi, Tiansheng Yao, Lichan Hong, Yizhou Sun, and Ed Chi. Improving multi-task generalization via regularizing spurious correlation. *Advances in Neural Information Processing Systems*, 35:11450–11466, 2022. 7, 9

[17] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 125–136, 2019. 7

[18] Keishi Ishihara, Anssi Kanervisto, Jun Miura, and Ville Hautamäki. Multi-task learning with attention for end-to-end autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2021,*

virtual, June 19-25, 2021, pages 2902–2911. Computer Vision Foundation / IEEE, 2021. 1

[19] Adrián Javaloy and Isabel Valera. Rotograd: Gradient homogenization in multitask learning. In *Proc. of ICLR*. OpenReview.net, 2022. 2, 6, 7

[20] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7482–7491. IEEE Computer Society, 2018. 2, 3, 4, 5, 1, 7

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *Proc. of ICLR*, 2015. 1, 3, 5, 2

[22] Lukas Kirchdorfer, Cathrin Elich, Simon Kutsche, Heiner Stuckenschmidt, Lukas Schott, and Köhler. Analytical uncertainty-based loss weighting in multi-task learning. *unpublished*, 2023. 5, 1

[23] Marvin Klingner, Andreas Bar, and Tim Fingscheidt. Improved noise and attack robustness for semantic segmentation by using multi-task training with self-supervised depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 320–321, 2020. 2, 7, 8, 9

[24] Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M. Pawan Kumar. In Defense of the Unitary Scalarization for Deep Multi-Task Learning. In *Neural Information Processing Systems*, 2022. 1, 2, 3, 5

[25] Dong-Gyu Lee. Fast drivable areas estimation with multi-task learning for real-time autonomous driving assistant. *Applied Sciences*, 11(22):10713, 2021. 1

[26] Baijiong Lin, Feiyang YE, Yu Zhang, and Ivor Tsang. Reasonable Effectiveness of Random Weighting: A Litmus Test for Multi-Task Learning. *Transactions on Machine Learning Research*, 2022. 2, 3, 4, 7

[27] Baijiong Lin and Yu Zhang. LibMTL: A Python Library for Multi-Task Learning. *ArXiv preprint*, abs/2203.14338, 2022. 4, 7, 8

[28] Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 18878–18890, 2021. 2, 3, 4, 6, 7, 8, 12

[29] Liyang Liu, Yi Li, Zhanghui Kuang, Jing-Hao Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. In *Proc. of ICLR*. OpenReview.net, 2021. 2, 3, 4, 7

[30] Shikun Liu, Stephen James, Andrew J Davison, and Edward Johns. Auto-Lambda: Disentangling Dynamic Task Relationships. *Transactions on Machine Learning Research*, 2022. 2

[31] Shikun Liu, Edward Johns, and Andrew J. Davison. End-to-end multi-task learning with attention. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 1871–1880. Computer Vision Foundation / IEEE, 2019. 2, 7

[32] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 3730–3738. IEEE Computer Society, 2015. 3, 7

[33] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 1851–1860. Computer Vision Foundation / IEEE, 2019. 2, 3

[34] Chengzhi Mao, Amogh Gupta, Vikram Nitin, Baishakhi Ray, Shuran Song, Junfeng Yang, and Carl Vondrick. Multitask learning strengthens adversarial robustness. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 158–174. Springer, 2020. 2, 7, 8, 9

[35] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 3994–4003. IEEE Computer Society, 2016. 2

[36] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 3, 4, 8, 7, 11, 15

[37] Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 16428–16446. PMLR, 2022. 2

[38] Lucas Pascal, Pietro Michiardi, Xavier Bost, Benoit Huet, and Maria A Zuluaga. Improved optimization strategies for deep multi-task networks. *ArXiv preprint*, abs/2109.11678, 2021. 2, 4

[39] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *ArXiv preprint*, abs/1706.05098, 2017. 2

[40] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 525–536, 2018. 2, 4

[41] Dmitry Senushkin, Nikolay Patakin, Arseny Kuznetsov, and Anton Konushin. Independent component alignment for multi-task learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 20083–20093. IEEE, 2023. 2

[42] Guangyuan Shi, Qimai Li, Wenlong Zhang, Jiaxin Chen, and Xiao-Ming Wu. Recon: Reducing Conflicting Gradients

From the Root For Multi-Task Learning. In *The Eleventh International Conference on Learning Representations*, 2023. 2, 6

[43] Trevor Standley, Amir Roshan Zamir, Dawn Chen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *Proc. of ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 9120–9132. PMLR, 2020. 2

[44] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 1, 2

[45] Derrick Xin, Behrooz Ghorbani, Ankush Garg, Orhan Firat, and Justin Gilmer. Do Current Multi-Task Optimization Methods in Deep Learning Even Help? In *Neural Information Processing Systems*, 2022. 1, 2, 3, 4, 5, 7, 8

[46] Dan Xu, Wanli Ouyang, Xiaogang Wang, and Nicu Sebe. Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 675–684. IEEE Computer Society, 2018. 2

[47] Teresa Yeo, Oguzhan Fatih Kar, and Amir Zamir. Robustness via cross-domain ensembles. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 12169–12179. IEEE, 2021. 7, 9

[48] Fisher Yu, Vladlen Koltun, and Thomas A. Funkhouser. Dilated residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 636–644. IEEE Computer Society, 2017. 7

[49] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 2, 4, 6, 7

[50] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6230–6239. IEEE Computer Society, 2017. 7

Challenging common assumptions in multi-task learning -Supplementary Material-

A1. Theoretical insights into multi-task learning dynamics

In this section, we aim to explain the success of the Adam optimizer [21] by relating it to uncertainty weighting [20]. We show partial invariances w.r.t. prior task-weights for the Adam optimizer and full invariances for the uncertainty weighting under mild assumptions. We further show that for SGD + momentum no invariance can be observed and the loss-weight can be seen as a task-specific learning rate which is not the case for the Adam optimizer. Previous literature on weighting methods in MTL did not explicitly show how task-weighting methods are affected by different optimizers.

A1.1. Uncertainty weighting (UW): Full loss-scale invariance

To demonstrate the invariance of uncertainty weighting (UW) [20], we start from the the original formula. In UW, the homoscedastic uncertainty[1] σ_t to weight task t is learned by gradient descent as done in [22]. However, we can also analytically compute the optimal uncertainty weights in each iteration instead of learning them using gradient descent. The minimization objective depends on the underlying loss function and likelihood. For simplicity, we show the derivation exemplary for the L_1 loss. It is straight-forward to derive the same for a Gaussian and other distributions. The objective of uncertainty weighting is given as

$$\min_{\sigma_t} \frac{1}{\sigma_t}\mathcal{L}_t + \log \sigma_t \tag{7}$$

with $\mathcal{L}_t = |y - f^W(x)|$ which can be derived from a log likelihood of a Laplace distribution $p(y|f^W(x), \sigma) = \frac{1}{2\sigma} exp(-\frac{|y-f^W(x)|}{\sigma})$. Taking the derivative and solving for σ_t results in an analytically optimal solution:

$$\frac{\partial}{\partial \sigma_t}\frac{1}{\sigma_t}\mathcal{L}_t + \log \sigma_t = -\frac{1}{\sigma_t^2}\mathcal{L}_t + \frac{1}{\sigma_t} \tag{8}$$

$$-\frac{1}{\sigma_t^2}\mathcal{L}_t + \frac{1}{\sigma_t} \stackrel{!}{=} 0 \tag{9}$$

$$\sigma_t = \mathcal{L}_t \tag{10}$$

with $\sigma_t > 0$. Here, we further see, that the optimization problem is convex and just one dimensional. Thus assuming an optimal log-sigma is a mild assumption. Plugging the optimal solution back into the original uncertainty weighting, we get

$$\mathcal{L} = \sum_t \frac{1}{sg[\mathcal{L}_t]}\mathcal{L}_t + \log \sqrt{sg[\mathcal{L}_t]}, \tag{11}$$

where we denote sg as the stopgradient operator. Since no gradient is computed of the second part of the loss, it can be simplified, such that

$$\mathcal{L} = \sum_t \frac{\mathcal{L}_t}{sg[\mathcal{L}_t]}. \tag{12}$$

Now considering task-specific weights α_t, the final equation does not change as task-specific weights cancel out:

$$\begin{aligned} \mathcal{L} &= \sum_t \frac{\alpha_t \mathcal{L}_t}{\alpha_t sg[\mathcal{L}_t]} \\ &= \sum_t \frac{\mathcal{L}_t}{sg[\mathcal{L}_t]} \end{aligned} \tag{13}$$

Therefore, the optimal uncertainty weighting is invariant w.r.t. task-specific loss-scalings, as each scaling cancels out.

A1.2. SGD: No loss-scale invariance and relationship of learning rate and task weights on a gradient level

Unlike for UW-O, we show that the SGD update rule itself does not show any invariances and that task-weights are essentially task-specific learning rates. We show that task-weights and learning rate are interacting hyperparameters and thus cannot be viewed in isolation. Thus, it is crucially to tune the learning rate for different loss weighting methods.

[1] In Kendall et al., this is termed the aleatoric homoscedastic uncertainty. However, as the task weights vary over the course of training and also with respect to the model capacity, it is technically not only the aleatoric uncertainty but also encapsulates further components such as model capacity and amount of data seen.

The parameter update rule in neural networks optimized with SGD is

$$\theta_i = \theta_{i-1} - \gamma \frac{\partial}{\partial \theta_{i-1}} \mathcal{L}, \tag{14}$$

where the network parameters in iteration i are defined as θ_i, γ is the learning rate and $\mathcal{L} = \sum_t \alpha \mathcal{L}_t$ for uniform task weights (EW). Therefore, in the case of EW we can rewrite Equation 14, such that

$$\begin{aligned} \theta_i &= \theta_{i-1} - \gamma \frac{\partial}{\partial \theta_{i-1}} \sum_i \alpha \mathcal{L}_t \\ &= \theta_{i-1} - \gamma \alpha \frac{\partial}{\partial \theta_{i-1}} \sum_i \mathcal{L}_t \end{aligned} \tag{15}$$

We can conclude that in the case of EW and SGD, task weight and learning rate are interchangeable. For example, increasing the weights α by a factor of 10 has the same effect as increasing the learning rate by a factor of 10. A value > 1 for α increases the parameter update while a value < 1 reduces the update.

In the case of non-uniform task weights, the parameter update is given as follows:

$$\begin{aligned} \theta_i &= \theta_{i-1} - \gamma \frac{\partial}{\partial \theta_{i-1}} \sum_i \alpha_t \mathcal{L}_t \\ &= \theta_{i-1} - \frac{\partial}{\partial \theta_{i-1}} \sum_i \gamma \alpha_t \mathcal{L}_t \end{aligned} \tag{16}$$

Therefore, for non-uniform task weights and SGD, we conclude that the learning rate can be included in the task-specific weight. This means that task weighting is nothing else than assigning task-specific learning rates. Tasks with a higher weight α_i have a proportionally higher parameter update step while tasks with smaller weights are moving slower towards the loss minimum.

Hence, task weight α acts exactly like the learning rate γ. The major difference is that task weights are task-specific while usually a single learning rate is applied to all network parameters. Note that this holds for SGD and SGD + momentum, but it does not apply to optimizers such as Adam, Adagrad, or RMSProp. In the following, we show our findings for the widely used Adam optimizer.

A1.3. Adam: Partial invariance towards loss-scales

Similarly to the invariance demonstrated for UW, we can derive a partial invariance for Adam. This invariance has already been demonstrated by Kingma and Ba [21], who showed that the magnitudes of the parameter updates using Adam are invariant to rescaling the gradients. Our novelty lies in showing this invariance property in the context of multi-task learning and its impact on different MTL optimizers. For Adam, we claim that the magnitude of task-specific weights only affects the backbone and cancels out for the heads.

Comparing Adam to SGD, the interaction between task weights and learning rate is fundamentally different. For SGD, the task weights could be viewed as task-specific learning rates. However, Adam interferes with many task-weighting methods as shown in more detail below. Thus comparisons of loss weighting methods in the multi-task learning literature based on SGD and Adam cannot be directly set side by side.

Our derivation relies on the usual MTL setting with task-specific heads and shared backbone. Here, we only look at the task-specific parameters ψ_t of task t whose loss $\mathcal{L}_t$ is scaled by α_t, such that $\mathcal{L}_t \rightarrow \alpha_t \mathcal{L}_t$ and assume a frozen backbone. We can look at the parameter update of one head independently of the other heads because the derivative of the losses w.r.t. the other tasks is 0:

$$\frac{\partial}{\partial \psi_{t,i-1}} \mathcal{L}_j = 0 \quad \text{for t} \neq \text{j}. \tag{17}$$

The general update rule for parameters psi at time step i using Adam is

$$\psi_i = \psi_{i-1} - \frac{\gamma}{\sqrt{\hat{v}_i} + \epsilon} \hat{m}_i, \tag{18}$$

where $m_i = \beta_1 m_{i-1} + (1 - \beta_1) g_i$ and $v_i = \beta_2 v_{i-1} + (1 - \beta_2) g_i^2$. To counteract the bias towards 0, the moments are corrected as $\hat{m}_i = \frac{m_i}{1-\beta_1^i}$ and $\hat{v}_i = \frac{v_i}{1-\beta_2^i}$.

For task-specific parameters ψ_t, task weights α_t linearly scale the first moment $m_{t,i}$

$$
\begin{aligned}
m_{t,i} &= \beta_1 m_{t,i-1} + (1-\beta_1) g_{t,i} \\
&= \beta_1 m_{t,i-1} + (1-\beta_1) \frac{\partial}{\partial \psi_{t,i-1}} \alpha_t \mathcal{L}_t \\
&= \beta_1 m_{t,i-1} + (1-\beta_1) \alpha_t \frac{\partial}{\partial \psi_{t,i-1}} \mathcal{L}_t \\
&= \beta_1 m_{t,i-1} + (1-\beta_1) \alpha_t g'_{t,i}
\end{aligned}
\tag{19}
$$

and quadratically scale the second moment $v_{t,i}$

$$
\begin{aligned}
v_{t,i} &= \beta_2 v_{t,i-1} + (1-\beta_2) g_{t,i}^2 \\
&= \beta_2 v_{t,i-1} + (1-\beta_2) (\frac{\partial}{\partial \psi_{t,i-1}} \alpha_t \mathcal{L}_t)^2 \\
&= \beta_2 v_{t,i-1} + (1-\beta_2) \alpha_t^2 (\frac{\partial}{\partial \psi_{t,i-1}} \mathcal{L}_t)^2 \\
&= \beta_2 v_{t,i-1} + (1-\beta_2) \alpha_t^2 {g'}_{t,i}^2,
\end{aligned}
\tag{20}
$$

where ${g'}_{t,i}$ is the gradient of the unscaled loss $\mathcal{L}_t$ w.r.t. the task-specific parameters for task t. As this holds for iteration i and because we have $m_{t,1} = \alpha {g'}_{t,1} + 0$ respectively $v_{t,1} = \alpha_t^2 {g'}_{t,1}^2 + 0$ with $m_{t,0} = 0$, $v_{t,0} = 0$ at the first iteration, this holds for any iteration step. This thus allows us to rewrite $\hat{m}_{t,i} = \alpha_t \hat{m'}_{t,i}$ and $\hat{v}_{t,i} = \alpha_t^2 \hat{v'}_{t,i}$.

Plugging, this back into the update rule, the final update rule for the task-specific parameters is given as

$$
\begin{aligned}
\psi_{t,i-1} &= \psi_{t,i-1} - \frac{\gamma}{\sqrt{\hat{v}_t i}} \hat{m}_{t,i} \\
&= \psi_{t,i-1} - \frac{\gamma}{\sqrt{\alpha_t^2 \hat{v'}_{t,i}}} \alpha_t \hat{m'}_{t,i} \\
&= \psi_{t,i-1} - \frac{\gamma}{\sqrt{\hat{v'}_{t,i}}} \hat{m'}_{t,i},
\end{aligned}
\tag{21}
$$

where the loss-scaling α_t cancels out. Therefore, the parameters of the task-specific heads are invariant to loss-scalings using Adam.

This partial invariance is a highly desired property, as there is a fundamental trade-off between tuning the learning rate and manual task weights. Given Adams invariance for the head, the weighting only affects the backbone. Thus the learning rate can be set for the parameters of the head independent of the loss weights. With the loss weights, we can prioritize tasks in the backbone and therefore walk along the Pareto front as empirically shown by [45].

The invariance, however, does not hold anymore when the backbone parameters θ are updated as well. As we have

$$
\begin{aligned}
m_i &= \beta_1 m_{i-1} + (1-\beta_1) \frac{\partial}{\partial \theta_{i-1}} \sum_t \alpha_t \mathcal{L}_t \\
&= \beta_1 m_{i-1} + (1-\beta_1) \sum_t \alpha_t g'_{t,i}
\end{aligned}
\tag{22}
$$

and

$$
\begin{aligned}
v_i &= \beta_1 v_{i-1} + (1-\beta_1) (\frac{\partial}{\partial \theta_{i-1}} \sum_t \alpha_t \mathcal{L}_t)^2 \\
&= \beta_1 v_{i-1} + (1-\beta_1) (\sum_t \alpha_t g'_{t,i})^2
\end{aligned}
\tag{23}
$$

we conclude that the task weights a_t linearly affect the first moment m_i, while having a quadratic effect on the update of the second moment v_i.

Note that for both task-heads only as well as the backbone, we have a full invariance in case of independent optimizers,

e.g., one Adam optimizer per task similar to [38]. However, naive implementations scale poorly (in terms of computational complexity) with the number of tasks here.

In the following experiments, we provide empirical evidence for our finding that a) Adam offers loss-scale invariance for the parameters of the task-specific heads, and b) Adam offers loss-scale invariance for all network parameters (backbone and heads) if $\beta_{1,2} = 0$.

A2. Empirical Confirmation of scale invariances in Adam and Optimal Uncertainty Weighting

In the prior section, we derived theoretical results for loss-scale (partial) invariance within multi-task learning for the Adam optimizer and uncertainty weighting. In this section, we confirm this invariance empirically with a toy task.

Experimental Setup We consider a two-task toy experiment in which we look at the gradient magnitudes with different combinations of Adam, SGD, EW, optimal uncertainty weighting (UW-O), and loss-scalings. To generate the data, we sample scalar input values from a uniform distribution. The outputs are just scalings of the input. As a neural network which consists of a shared backbone (two layers with LeakyReLU as non-linearity and 20 neurons per hidden layer) and two heads for the two tasks, each consisting again of two layers. Both task measure the depth but in different units. Both tasks are measured with an L_1-loss. We provide two settings, in the first one, depth is measured on the same scale. In the second setting, one depth loss is scaled by 10x (e.g., measured in cm instead of deci-meters) and one other loss is scaled by 0.1 (e.g., measured in meters instead of deci-meters). For each setting, we test various combinations of loss weighting and optimizer combinations.

The 8 different experiments are:

- Equal weighting using SGD
- Equal weighting using SGD with scalings $10 * L_{seg}$ and $0.01 * L_{dep}$

- Equal weighting using Adam
- Equal weighting using Adam with scalings $10 * L_{seg}$ and $0.01 * L_{dep}$

- Optimal uncertainty weighting using SGD
- Optimal uncertainty weighting using SGD with scalings $10 * L_{seg}$ and $0.01 * L_{dep}$

- Equal weighting using separate Adam optimizers per task
- Equal weighting using separate Adam optimizers per task with scalings $10 * L_{seg}$ and $0.01 * L_{dep}$

To better control for different factors of influence, we first perform the first 6 of the listed experiments with a fixed backbone, i.e., we do not update the parameters in the backbone but only in the heads. Afterward, we show all 8 experiments trained with a network where all parameters (including the backbone) are updated. This allows us to verify if our theoretical derivations regarding the (partial) loss-scaling invariance of Adam and UW-O also hold in practice, and compare this to the SGD optimizer.

Note that we only care about the invariance and did not tune any hyperparameters for performance.

Results for fixed backbone Figure A1 shows the losses, the scaled losses (by loss weighting method), the gradient magnitudes as well as the gradient update magnitudes for both heads along the 100 epochs of training with a fixed backbone. Regarding SGD, we can observe that the equal weighting experiment differs from its scaled variant along all 8 dimensions. This is because SGD does not offer any loss-scaling invariance. As expected, at the beginning of the training the gradient update magnitude of the first depth head parameters with the scaled loss (dotted line) is by a factor of 10 higher than the unscaled (solid line) one. The same effect applies to the gradient update magnitude of the second depth head parameters, but with a factor of 0.01.

In contrast, Adam is loss-scale invariant. We can observe that the unscaled (solid line) and the scaled version (dotted line) have equal gradient update magnitudes in the last row. Note that practically due to an $epsilon = 10^{-8}$ parameter in the denominator and float precision a slight divergence would occur with larger number of epochs. This result confirms our theoretical finding in equation 21. We skip the experiment of separated Adam optimizers per task because it would be equivalent to this version given a fixed backbone.

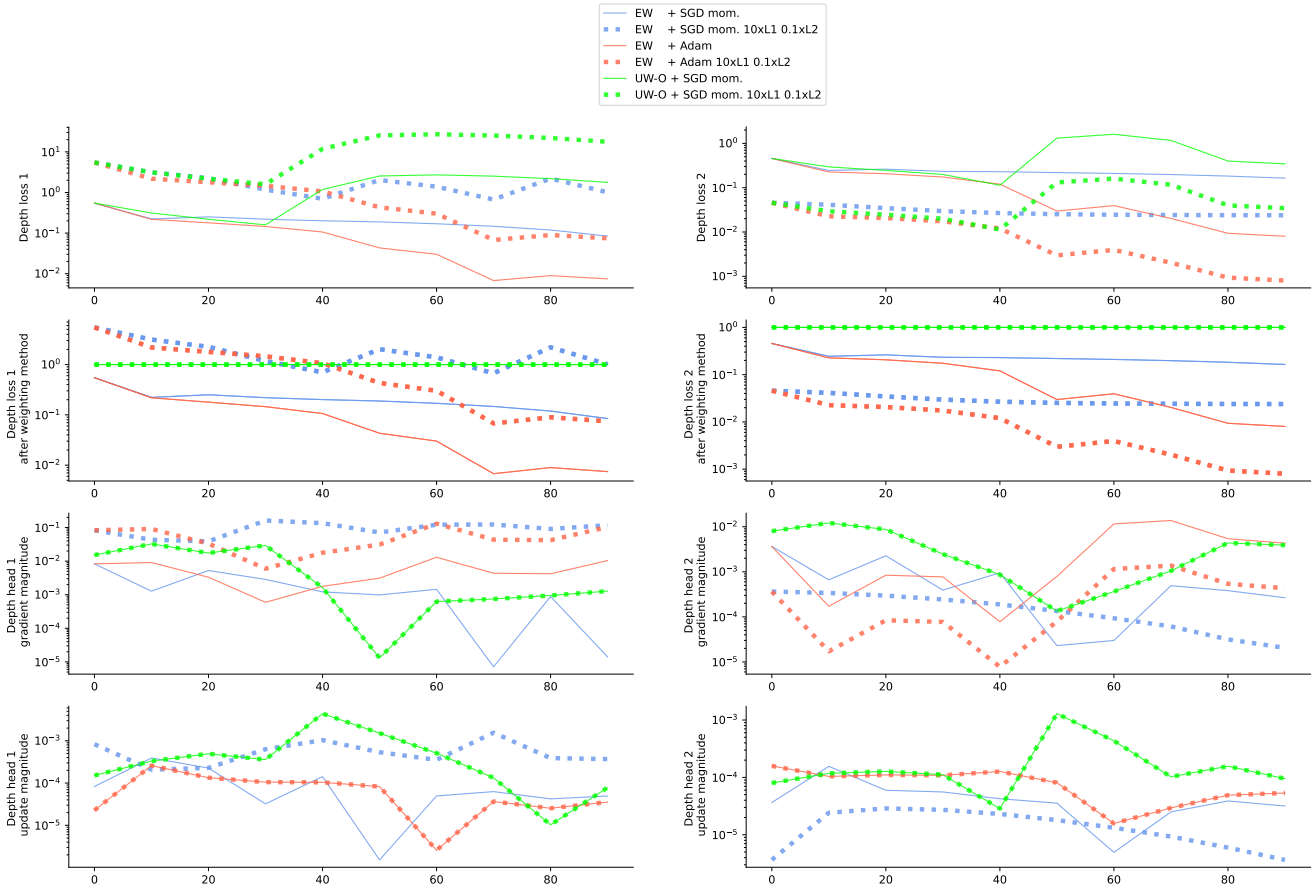


Figure A1. **Invariances within the neural network for a frozen backbone.** Comparing the effect of loss-scalings in a toy experiment with two tasks. For each optimizer and loss weighting combination, we run two settings with a) loss L1 and loss L2 are equally weighted or b) L1 is scaled by 10x and L2 by 0.1. For each setting, we measure the SGD + momentum and Adam optimizer with no post weighting (EW) and SGD + momentum with optimimal uncertainty weighting. We show the scaled losses, gradient magnitudes, and gradient update magnitudes in the the two task heads and keep the backbone frozen. While SGD does not offer any loss-scaling invariance, Adam makes the gradient updates of the head parameters invariant to scales confirming our derivation (red lines overlap in lowest row). Equivalently, for UW-O we also observe the theoretically derived invariances (green lines overlap in lowest row)

Lastly, we want to investigate the invariance properties of UW-O. We compare the scaled (dotted line) and unscaled (solid line) version of UW-O with the SGD optimizer. As expected, the gradients, as well as the gradient updates, match in both heads.

In the following, let's investigate whether the observed results still hold if we also consider the update of the backbone parameters.

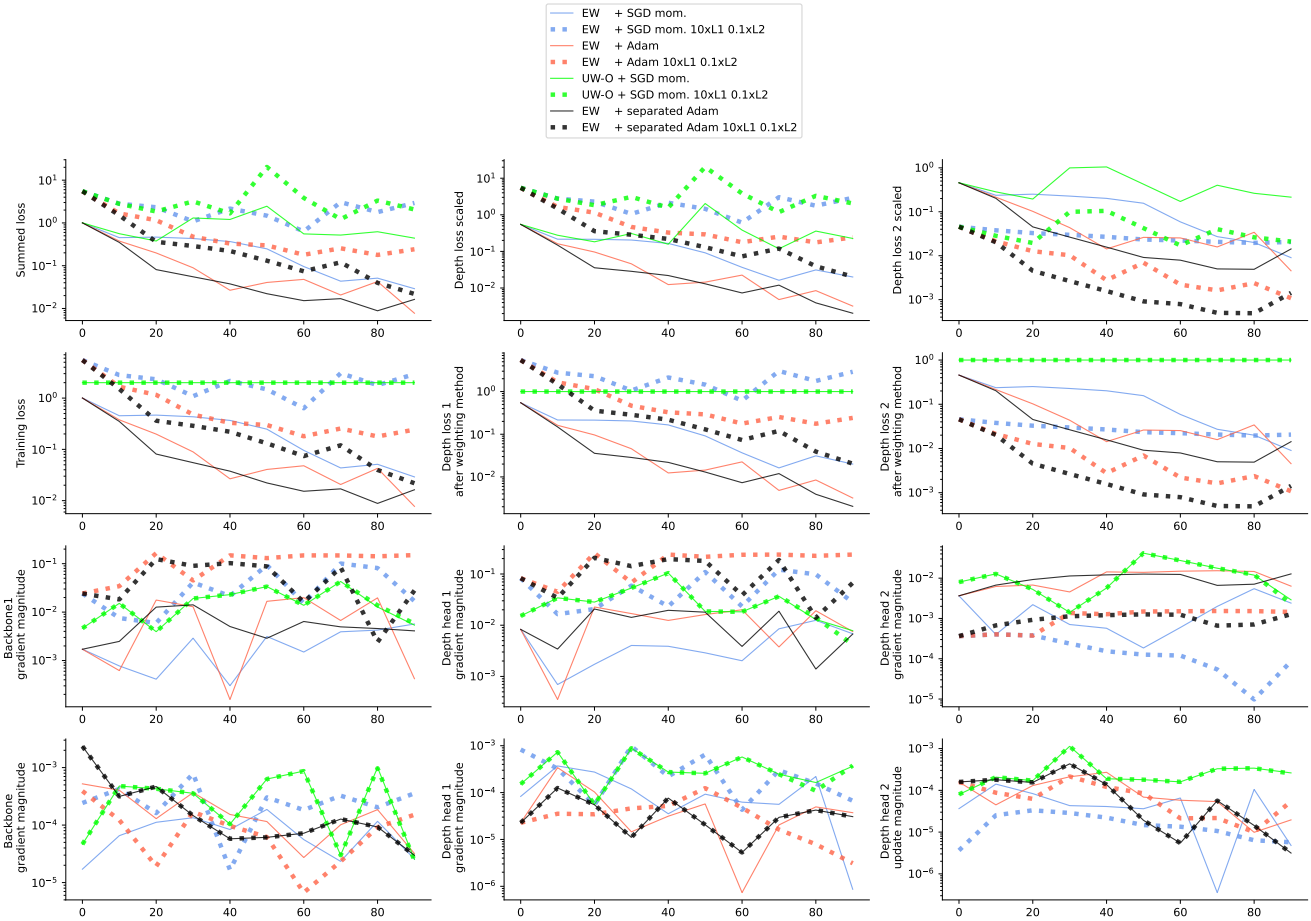


Figure A2. **Invariances within the neural network for a learnable backbone.** Comparing the effect of loss-scalings in a toy experiment with two tasks. For each optimizer and loss weighting combination, we run two settings with a) loss L1 and loss L2 are equally weighted or b) L1 is scaled by 10x and L2 by 0.1. For each setting, we measure the SGD + momentum and Adam optimizer with no post weighting (EW) and SGD + momentum with optimimal uncertainty weighting. Additionally, we implement independent Adam optimizer per task. We show the scaled losses, gradient magnitudes, and gradient update magnitudes in the backbone(first row) and the the two task heads (2nd and 3rd row). Neither Adam, nor SGD show invariances if the backbone is trained as well. UW-O is still invriant (green lines are overlapping). We revoke Adam's inveriance by implementing separate optimizers per task (lowerst black lines are overlapping).

Results for free backbone Figure A2 shows the scaled losses, the gradient magnitudes as well as the gradient update magnitudes in the backbone and the depth heads along the 100 epochs of training with a free backbone. Again, the loss-scalings affect the gradient magnitudes using SGD. This applies to both backbone and heads.

When looking at the Adam experiments, we can observe that it is partly loss-scale invariant by looking at the first iteration in the heads. However, due to different updates in the backbone, the networks behave different in both settings (scaled and unscaled loses). Furthermore, when implementing task-specific optimizers, we can observe that not only the gradient update magnitudes in the task heads, but also in the backbone match between the scaled (**dotted line**) and the unscaled (**solid line**) variant. Thus, all network parameters are invariant to loss-scalings when using separate Adam optimizers. This confirms our theoretical results.

Along the lines of our theoretical findings, we can observe that UW-O offers scaling-invariance across the whole network as the gradients as well as the gradient updates match among the two variants in the backbone and in both heads. This empirical observation matches our theoretical derivation in equation 13.

A3. Implementation Details

In this section, we explain the applied experiment settings used for the reported experiments in more detail. In particular, we describe the handling of the different datasets in appendix A3.1 and provide further information on the applied training procedures in appendix A3.2. Our chosen experimental setups are designed to follow previous work. In particular, we took inspiration from [28] and [45]. However, we found that the experimental setup would vary widely across different works in the field of multi-task learning as can be seen in Table A1. We decided for a uniform setup for each dataset independent of the choice of network and MTO.

A3.1. Datasets

CityScapes [8] We make use of the official split of the dataset which consists of 2975 training and 500 validation scenes. Similar to [45], we denote 595 random samples from the training split as validation data and report test results on the original validation split. We further follow the pre-processing scheme from [31] of re-scaling images to 128x256 pixels and use inverse depth labels. During training, we apply random scaling and cropping for data augmentation[2]. Following previous work [28] for number of epochs, and learning rate schedule, we train for 300 epochs and decrease the learning rate by a factor of 0.5 every 100 epochs. The batch size is set the batch size to 64, similar to [45]. We only consider a fixed weight decay of 10^{-5} for all datasets and experiments as we found varying this parameter had only little influence in initial experiments.

NYUv2 [36] From the 795 official training images we use 159 for our validation split as in [27] and report test performance on the official 654 test images. Similar to [31], we re-size the images to 288x384 pixels. Training is run for 200 epochs with a batch size of 8. We apply the same data augmentation and learning rate scheduler as for CityScapes.

CelebA [32] We re-size images to 64x64 pixels as done in [26] and consider the original split of 162,770/19,867/19,962 for training, validation, and testing. We set the batch size to 512, train for 100 epochs, and halve the learning rates every 30 epochs.

Corrupted variants For Cityscapes and NYUv2 we also apply common corruptions form [15]. Here, we use the original code from code https://github.com/hendrycks/robustness.

Data	MTO	Network	Optimizer	learning rate	weight decay	batch size	#train. iterations
CityScapes [8]	UW [20]	DeepLabV3 [4] with ResNet101 [48]	SGD + Nesterov updates, Mom.	init.: $2.5 \cdot 10^{-3}$; polynomial lr decay	$1 \cdot 10^{-4}$	8	100k iter.
	RLW [26]	DeepLabV3 [4] with ResNet50 [48]	Adam	$1 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	64	
	IMTL [29]	ResNet50 [48] + PSPNet [50] heads	SGD+Mom.	init.: 0.02; polynomial lr decay	$1 \cdot 10^{-4}$	32	200 epochs
	PCGrad [49]	MTAN [31]	Adam	init.: $1 \cdot 10{-}4$; halving lr after 40k iter.	-	8	80k iter
	CAGrad [28]	MTAN [31]	Adam	init.: $1 \cdot 10{-}4$; halving lr every 100 epochs	-	8	200 epochs
NYUv2 [36]	RLW [26]	DeepLabV3 [4] with ResNet50 [48]	Adam	$1 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	8	
	IMTL [29]	ResNet50 [48] + PSPNet [50] heads	SGD+Mom.	init.: 0.03	-	48	200 epochs
	PCGrad [49]	MTAN [31]	Adam	init.: $1 \cdot 10{-}4$; halving lr after 40k iter.	-	2	80k iter
	CAGrad [28]	MTAN [31]	Adam	init.: $1 \cdot 10{-}4$; halving lr after 100 epochs	-	2	200 epochs
CelebA [32]	RLW [26]	ResNet17 [14] + lin. classifier	Adam	$1 \cdot 10^{-3}$	-	512	
	IMTL [29]	ResNet17 [14] + lin. classifier	Adam	0.003	-	256	100 epochs
	PCGrad [49]	ResNet17 [14] + lin. classifier	Adam	init. from $\{10^{-4}, ..., 5 \cdot 10^{-2}\}$; halving lr every 30 epochs	-	256	100 epochs

Table A1. **Original experiment setup as reported in respective papers.** We note a high variation regarding the choice of network, optimizer, and other hyper-parameters among the different works.

[2] https://github.com/Cranial-XIX/CAGrad

A3.2. Training

Effectiveness of Adam in MTL. All presented results are based on performing early stopping wrt. Δ_m- metric on the validation set. For this, we further trained single-task learning (STL) models for each experiment combination (dataset and network) using the respective network architecture except for the missing head(s). In particular, we trained the models using any learning rate from $\{0.01, 0.005, ..., 0.00005\}$. The training was stopped early based on the validaton loss.

Our implementation for all experiments is based on the LibMTL library [27].

Gradient Similarity. Our gradient similarity experiments were conducted on the best performing hyper-parameter configuration for EW from the previous extensive evaluation. Over the full training, gradient similarity measures are computed every five iteration steps and summarized per epoch. To make the computation effort more feasible in case of settings with large batch size or high number of tasks, we randomly select eight samples or tasks respectively and consider corresponding gradients in these cases.

A4. Additional results on comparison between Adam and SGD

We present additional evaluation results for our comparison between optimizers for multi-task learning. Figure A4 shows additional parallel coordinate plots for more choices of dataset/network combinations. In Table A2, we count for each used MTO the number of experiment runs that are located on the Pareto front of each setup. We compare the Δ_m-metric performance between the usage of Adam and SGD+Momentum in Figure A3. Best performing quantitative results for all MTOs can be found in Tables A3 to A6.

We further show extended results on the toy task by Liu et al. [28] for more learning rates in Figure A5.

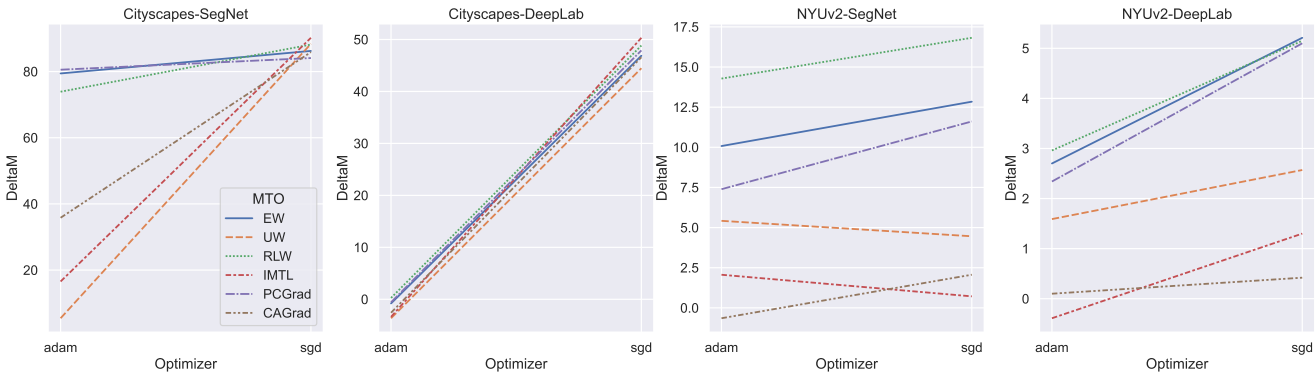
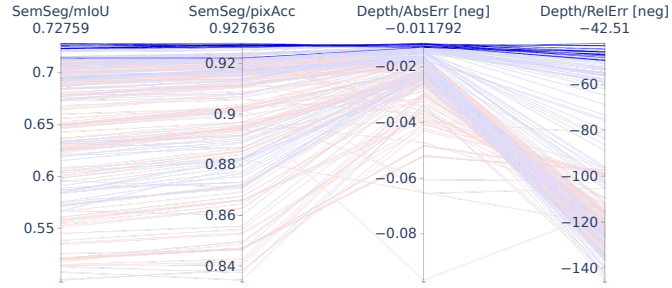
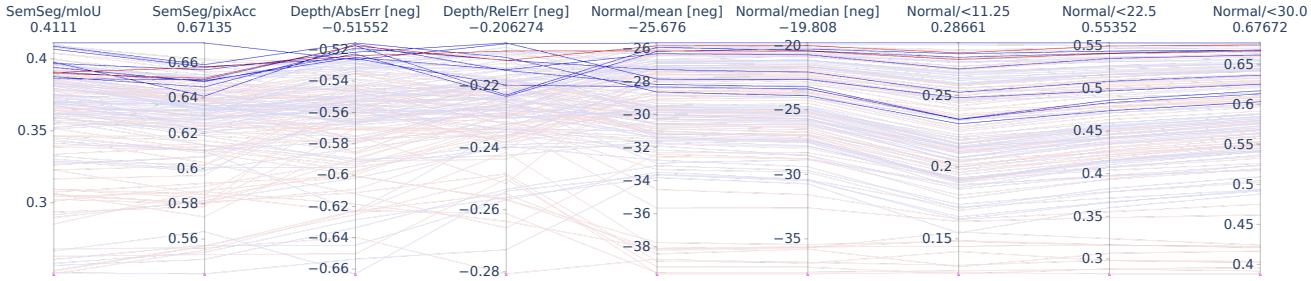


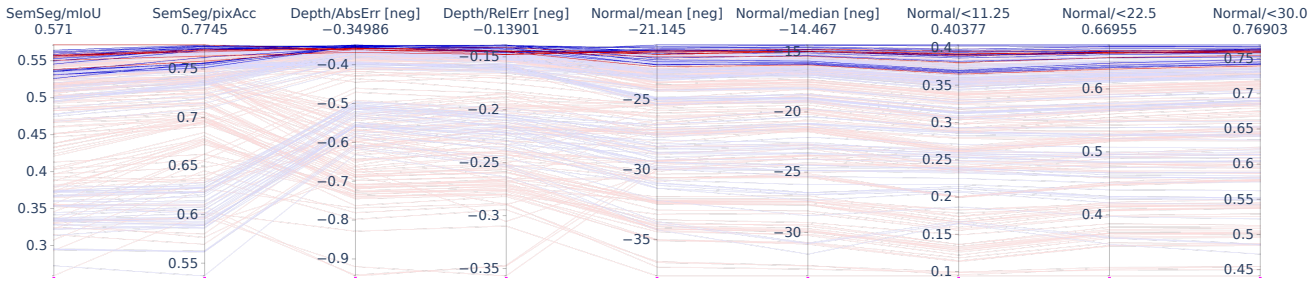
Figure A3. **Line Plot of mean of Δ_m-metric for experiments run on CityScapes and NYUv2 with SegNet and DeepLabV3.** We compare the performance of the best hyperparameter setting (lower is better) for every MTO using either Adam (left) or SGD+Momentum (right). Every MTO is associated with a different line color/style. A lower value indicates better performance. On Cityscapes there is a large difference for the Δ_m score for Adam compared to SGD+Momentum, especially for UW, IMTL, and CAGrad. Therefore, using these methods, the result depends more on the optimizer than on the MTO method. On the NYUv2 dataset this observation weakens. Adam still achieves the lowest Δ_m scores across different MTO methods (except for SegNet with UW and IMTL), though, besides chosing Adam, it is also important to select the appropriate MTO method.



(a) CityScapes, DeepLabV3



(b) NYUv2, SegNet



(c) NYUv2, DeepLabV3

Figure A4. **Parallel coordinate plot** for further results on additional combinations of CityScapes and NYUv2 with either SegNet or DeepLabV3. Every line in the respective plot represents one run from section 4.1 with a specific choice of hyperparameters. We distinguish between experiments using SGD+mom and Adam optimizer. Experiments that reached Pareto front performance are drawn with higher saturation.
We note a clear dominance of Adam on the CityScapes dataset when using DeepLabV3, similar to the results in SegNet which we presented in the main paper. For NYUv2, we observe a similar trend albeit here we also have some experiments using SGD+Mom. on the overall Pareto front.

Data	Network	Optimizer	EW	UW	RLW	IMTL	PCGrad	CAGrad	Total
CityScapes	SegNet	Adam	1	7	-	3	-	1	12
CityScapes	SegNet	SGD+Mom.	-	-	-	-	-	-	-
CityScapes	DeepLabV3	Adam	3	3	2	3	2	2	15
CityScapes	DeepLabV3	SGD+Mom.	-	-	-	-	-	-	-
NYUv2	SegNet	Adam	1	3	-	3	1	1	9
NYUv2	SegNet	SGD+Mom.	-	-	-	2	-	-	2
NYUv2	DeepLabV3	Adam	2	2	-	6	1	5	16
NYUv2	DeepLabV3	SGD+Mom.	-	1	-	2	-	4	7

Table A2. **Count of Pareto optimal experiments for each MTO method.** We found no single MTO method to be superior over all combinations of dataset and networks. When using Adam, all methods but RLW and PCGrad in one case (CityScapes+SegNet) would result in at least one model yielding Pareto optimal performance. Total numbers can be compared to Table 2

MTO	Optimizer	lr	Sem.Seg.		Depth		DeltaM ↓
			mIoU ↑	pixAcc ↑	AbsErr ↓	RelErr ↓	
STL	adam		0.7122	0.9221	0.0134	29.88	
EW	adam	0.005	0.6898	0.9165	**0.0196**	**109.84**	**79.43** ±3.68
EW	sgd	0.1	**0.6967**	**0.9179**	0.0216	113.82	86.24 ±1.97
UW	adam	0.001	**0.7052**	**0.9202**	**0.0136**	**35.69**	**5.44** ±2.38
UW	sgd	0.01	0.6750	0.9110	0.0219	114.67	88.39 ±1.35
RLW	adam	0.001	**0.7013**	**0.9196**	**0.0197**	**103.61**	**73.91** ±7.63
RLW	sgd	0.1	0.6918	0.9156	0.0227	113.59	88.16 ±0.67
IMTL	adam	0.005	**0.6963**	**0.9170**	**0.0148**	**45.63**	**16.55** ±1.52
IMTL	sgd	0.01	0.6716	0.9107	0.0230	114.38	90.21 ±0.74
PCGrad	adam	0.01	0.6770	0.9135	**0.0226**	**103.88**	**80.56** ±3.70
PCGrad	sgd	0.1	**0.6972**	**0.9176**	0.0235	107.06	84.09 ±0.93
CAGrad	adam	0.001	**0.7088**	**0.9208**	**0.0162**	**66.39**	**35.81** ±14.91
CAGrad	sgd	0.1	0.6896	0.9156	0.0205	115.52	85.88 ±0.31

Table A3. **Results for different MTO methods and optimizers on CityScapes [8] using a SegNet network [1].** We report the mean test performance over three seeds for the best learning rate w.r.t. the validation data based on the delta-M metric. The best score per metric is highlighted **for each MTO method** as well as over all methods and optimizers. While we observe that different MTO perform best over the distinct metrics, Models trained with Adam outperform those trained with SGD + Momentum in most cases. On the overall Δ_m-metric, using Adam show superior performance for all MTO, in some cases even with a high margin.

MTO	Optimizer	lr	Sem.Seg.		Depth		DeltaM ↓
			mIoU ↑	pixAcc ↑	AbsErr ↓	RelErr ↓	
STL	adam		0.7203	0.9253	0.0132	47.37	
EW	adam	0.001	**0.7247**	**0.9268**	**0.0128**	**47.73**	**-0.80** ±0.69
EW	sgd	0.05	0.7100	0.9217	0.0174	120.34	46.88 ±2.02
UW	adam	0.001	**0.7224**	**0.9259**	**0.0122**	**44.37**	**-3.65**±0.61
UW	sgd	0.005	0.7003	0.9187	0.0171	116.09	44.47 ±1.91
RLW	adam	0.001	**0.7230**	**0.9263**	**0.0133**	**47.76**	**0.26** ±1.08
RLW	sgd	0.05	0.7070	0.9205	0.0176	123.13	48.84 ±1.79
IMTL	adam	0.001	**0.7226**	**0.9259**	**0.0121**	**45.25**	**-3.38** ±0.92
IMTL	sgd	0.005	0.7027	0.9192	0.0185	122.37	50.33 ±3.30
PCGrad	adam	0.001	**0.7247**	**0.9272**	**0.0130**	**47.24**	**-0.58** ±0.56
PCGrad	sgd	0.05	0.7083	0.9212	0.0173	122.52	47.90 ±3.68
CAGrad	adam	0.001	**0.7245**	**0.9264**	**0.0124**	**45.56**	**-2.59** ±0.68
CAGrad	sgd	0.1	0.7096	0.9220	0.0172	120.24	46.58 ±2.21

Table A4. **Results for different MTO methods and optimizers on CityScapes [8] using a DeepLabV3+ network [4].** We report the mean test performance over three seeds for the best learning rate w.r.t. the validation data. The best score per metric is highlighted **for each MTO method** as well as over all methods and optimizers. While we observe that different MTO perform best over the distinct metrics, Models trained with Adam consistently outperform those trained with SGD + Momentum. We further note a clear superiority on the Δ_m-metric which indicates for all but RLW better improved performance compared to the STL baseline.

			Sem.Seg.		Depth		Normal					
MTO	Optimizer	lr	mIoU ↑	pixAcc ↑	AbsErr ↓	RelErr ↓	Mean ↓	Median ↓	$< 11.25 \uparrow$	$< 22.5 \uparrow$	$< 30.0 \uparrow$	DeltaM ↓
STL	adam		0.3922	0.6462	0.6068	0.2579	24.74	18.49	0.3084	0.5816	0.6996	
EW	adam	0.0001	**0.3979**	**0.6496**	**0.5299**	**0.2123**	**29.53**	**25.02**	**0.2162**	**0.4542**	**0.5838**	**10.08** ±2.84
EW	sgd	0.01	0.3843	0.6438	0.5511	0.2271	30.12	25.72	0.2121	0.4427	0.5712	12.84 ±0.65
UW	adam	0.0001	**0.4011**	**0.6517**	**0.5215**	**0.2132**	27.93	22.80	0.2432	0.4943	0.6231	5.42 ±1.04
UW	sgd	0.05	0.3832	0.6421	0.5508	0.2178	**27.07**	**21.66**	**0.2586**	**0.5162**	**0.6435**	**4.46** ±2.13
RLW	adam	0.0001	**0.3898**	**0.6380**	**0.5367**	**0.2179**	**30.83**	**26.78**	0.1961	**0.4243**	**0.5538**	**14.28** ±2.38
RLW	sgd	0.05	0.3690	0.6325	0.5722	0.2328	31.21	27.17	**0.1976**	0.4197	0.5459	16.82 ±1.16
IMTL	adam	0.0001	0.3796	0.6436	**0.5242**	0.2156	26.35	20.77	0.2703	0.5342	0.6603	2.06 ±1.31
IMTL	sgd	0.05	**0.3957**	**0.6557**	0.5320	**0.2154**	**26.16**	**20.38**	**0.2773**	**0.5419**	**0.6664**	**0.72** ±0.68
PCGrad	adam	0.0001	**0.4057**	**0.6540**	**0.5286**	**0.2152**	**28.60**	**23.75**	**0.2307**	**0.4768**	**0.6062**	**7.39** ±0.86
PCGrad	sgd	0.01	0.3893	0.6441	0.5473	0.2227	29.76	25.22	0.2143	0.4506	0.5800	11.61 ±0.36
CAGrad	adam	0.0001	**0.4046**	**0.6606**	**0.5273**	**0.2114**	**25.85**	**20.02**	**0.2816**	**0.5495**	**0.6735**	**-0.65** ±0.96
CAGrad	sgd	0.05	0.4003	0.6561	0.5474	0.2226	26.39	20.78	0.2682	0.5344	0.6615	2.06 ±0.65

Table A5. **Results for different MTO methods and optimizers on NYUv2 [36] using a SegNet network [1].** We report the mean test performance over three seeds for the best learning rate w.r.t. the validation data. The best score per metric is highlighted **for each MTO method** as well as over all methods and optimizers. Over all metrics, best performance is always achieved by a model trained with Adam. We note that the Δ_m is more effected by the normal task due to the higher number of corresponding metrics as can be observed in the case of UW.

			Sem.Seg.		Depth		Normal					
MTO	Optimizer	lr	mIoU ↑	pixAcc ↑	AbsErr ↓	RelErr ↓	Mean ↓	Median ↓	$< 11.25 \uparrow$	$< 22.5 \uparrow$	$< 30.0 \uparrow$	DeltaM ↓
STL	adam		0.5517	0.7668	0.3650	0.1524	21.16	14.52	0.4023	0.6679	0.7679	
EW	adam	0.0001	0.5521	**0.7697**	**0.3515**	**0.1432**	**22.54**	**16.05**	**0.3659**	**0.6340**	**0.7422**	**2.70** ±0.12
EW	sgd	0.01	**0.5558**	0.7686	0.3646	0.1473	23.35	16.76	0.3520	0.6167	0.7273	5.21 ±0.18
UW	adam	0.0005	0.5319	0.7553	**0.3579**	**0.1408**	**22.01**	**15.30**	**0.3825**	**0.6502**	**0.7533**	**1.59** ±0.14
UW	sgd	0.01	**0.5570**	**0.7692**	0.3643	0.1479	22.37	15.69	0.3742	0.6408	0.7464	2.57 ±0.56
RLW	adam	0.0001	**0.5562**	**0.7675**	**0.3604**	**0.1495**	**22.37**	**15.87**	**0.3697**	**0.6375**	**0.7446**	**2.96** ±0.55
RLW	sgd	0.05	0.5412	0.7617	0.3678	0.1508	23.00	16.44	0.3571	0.6245	0.7338	5.15 ±0.54
IMTL	adam	0.0005	0.5324	0.7557	**0.3526**	**0.1404**	**21.34**	**14.61**	**0.3990**	**0.6661**	**0.7660**	**-0.39** ±0.05
IMTL	sgd	0.05	**0.5449**	**0.7617**	0.3669	0.1499	21.63	14.98	0.3902	0.6576	0.7602	1.30 ±0.40
PCGrad	adam	0.0001	**0.5583**	**0.7716**	**0.3569**	**0.1460**	**22.33**	**15.81**	**0.3711**	**0.6388**	**0.7460**	**2.34** ±0.48
PCGrad	sgd	0.01	0.5563	0.7685	0.3609	0.1471	23.36	16.79	0.3514	0.6167	0.7279	5.10 ±0.28
CAGrad	adam	0.0005	0.5310	0.7550	**0.3568**	**0.1428**	**21.42**	**14.67**	**0.3988**	**0.6640**	**0.7639**	**0.10** ±0.55
CAGrad	sgd	0.05	**0.5555**	**0.7684**	0.3600	0.1437	21.77	14.99	0.3906	0.6573	0.7592	0.42 ±0.07

Table A6. **Results for different MTO methods and optimizers on NYUv2 [36] using a DeepLabV3+ network [4].** We report the mean test performance over three seeds for the best learning rate w.r.t. the validation data. The best score per metric is highlighted **for each MTO method** as well as over all methods and optimizers. We note a clear dominance of Adam on both the depth and normal tasks as well as on the Δ_m-metric. Overall, best results were also achieved using Adam as optimizer for all metrics.

	method	learning rate								
		10.0	5.0	1.0	0.5	0.1	0.05	0.01	0.005	0.001*
GD	EW	-	103	-	-	-	-	-	-	-
	PCGrad	-	-	-	-	-	-	-	-	-
	CAGrad	644	-	213	621	8,069	5,732	20,418	34,405	-
Adam	EW	*26*	*37*	**22**	**58**	**709**	**2,135**	**9,015**	*16,005*	-
	PCGrad	**25**	4,960	56	15,741	34,175	41,438	-	-	-
	CAGrad	27	**30**	*32*	*106*	*802*	*7,109*	*11,239*	**14,323**	**57,700**

*LR used for results in [28] with Adam

Table A7. **Number of iterations after which all seeds in toy task experiment from CAGrad [28] have reached the global minimum** for different learning rates and optimizer. We show results for additional learning rates compared to the main paper. The maximum iteration number over all three seeds for each MTO method / learning rate / optimizer combination is reported. If not all seeds converged to the global minimum within 100k iteration steps, we denote it as '-'. In several setups, EW+Adam converges fastest to the global minimum. Especially for small learning rates, CAGrad performs advantageous compared to EW. As reported in previous work, we found that PCGrad often would converge only to some point on the Pareto Front. The **best** and *second best* run for each learning rate over all MTO methods are indicated via font type. Best and second best learning rate + optimzer combination for each MTO are marked via cell background color.

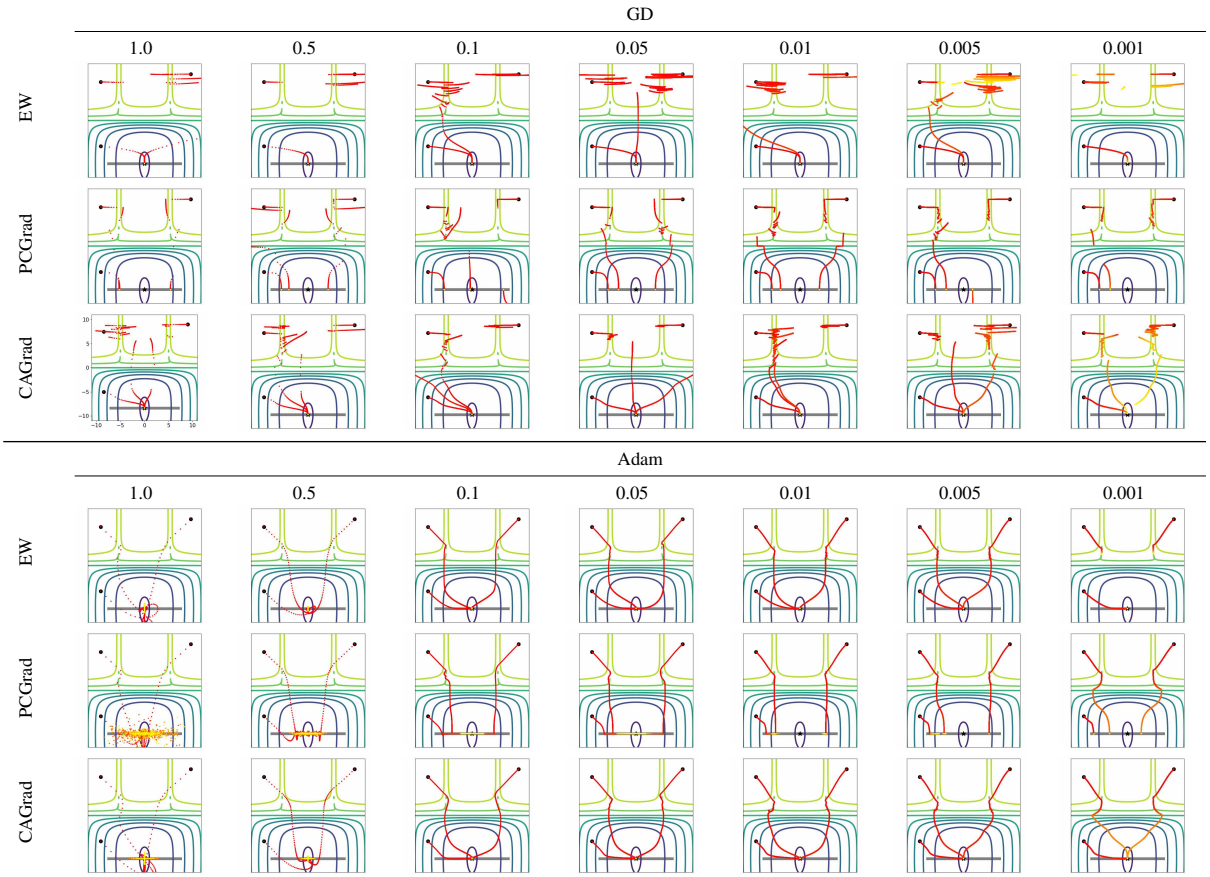
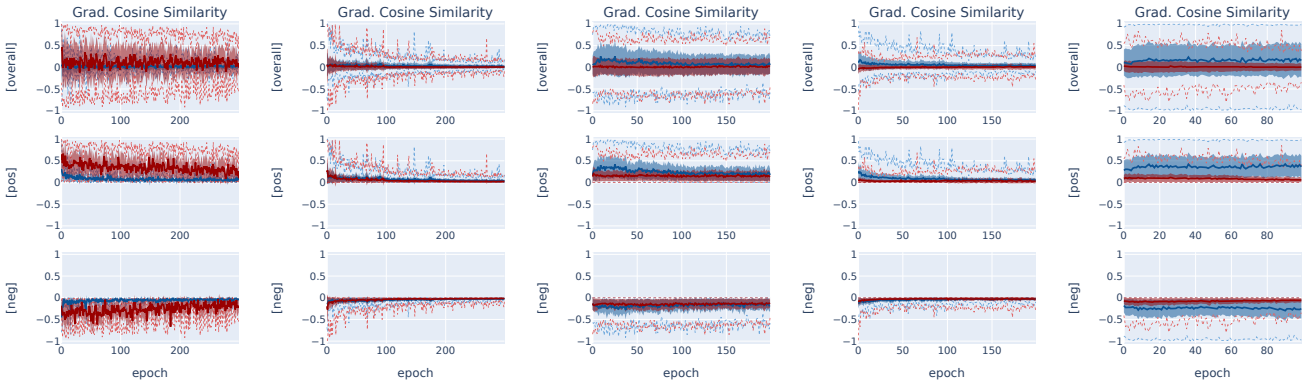


Figure A5. Extension of Figure 1 with additional learning rates. We show the optimization trajectories for three different seeds (black dots). The Optimization trajectories are colored from red to yellow for 100k iteration steps. The global optimum is depicted as asterix ($*$), the Pareto front is highlighted in gray. We note the importance of a good selection of learning rate even for simple toy examples. Moreover, using Adam yields overall faster convergence to the global optimum. We use the original implementation which can be found under https://github.com/Cranial-XIX/CAGrad .

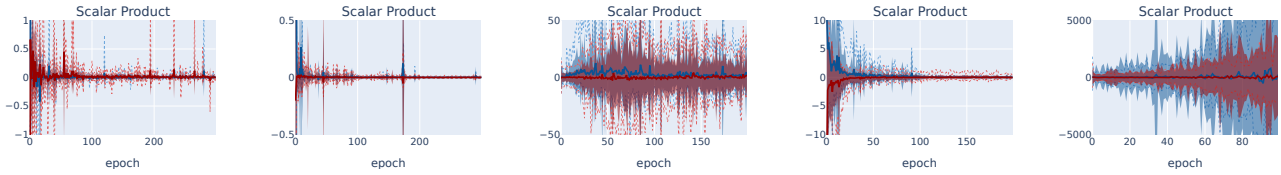
A5. Additional gradient alignment results

Besides the gradient similarity measures defined in the main paper, we provide some further insights when comparing pairs of gradient in either in the multi-task or single-task learning setup. In Figure A6, we differentiate between conflicting and supporting gradient pairs when evaluating the cosine similarity. Figure A7 shows the evaluation of the scalar product as an combined measure of similarity in gradient direction and magnitude. We report the number of gradient components which are either positive, negative or zero in Figure A8. Finally, in Figure A9, we plot the course of loss functions with respective to the different dataset splits.



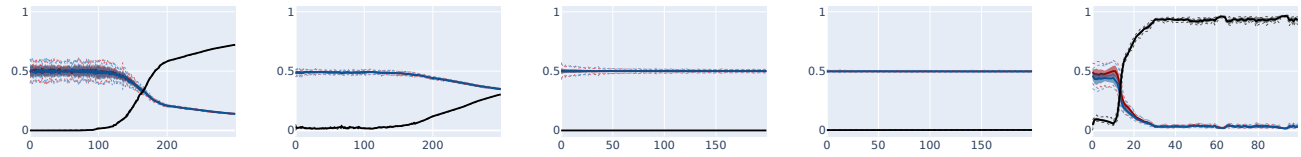
(a) CityScapes, SegNet (b) CityScapes, DeepLabV3 (c) NYUv2, SegNet (d) NYUv2, DeepLabV3 (e) CelebA, ResNet50

Figure A6. **Differentiation between conflicting and supportive gradients.** We report mean (solid line), standard deviation (shaded area), upper (97.5%) and lower (2.5%) percentile (dotted line) of the gradient cosine similarity between either gradients of different samples or different tasks within an epoch. While showing overall results over all respective gradient pairs (Top) as can be also found in Figure 3, we also show the course of cosine similarity for either gradients that are conflicting ([neg], bottom) or those which have cosine similarity greater than zero ([pos], middle).



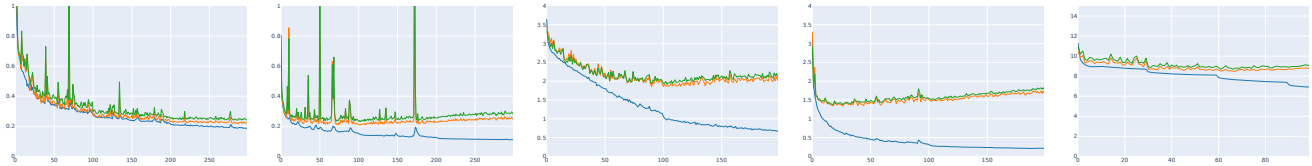
(a) CityScapes, SegNet (b) CityScapes, DeepLabV3 (c) NYUv2, SegNet (d) NYUv2, DeepLabV3 (e) CelebA, ResNet50

Figure A7. **Scalar product between pairs of gradients.** We report mean (solid line), standard deviation (shaded area), upper (97.5%) and lower (2.5%) percentile (dotted line) of the gradient cosine similarity between either gradients of different samples or different tasks within an epoch. We observe an overall decrease of the variance of the scalar product for both CityScapes setups and the NYUv2+DeepLabV3 experiment over the training which we explain with evenly smaller overall gradients. Surprisingly, this does not apply for NYUv2 with SegNet or CelebA. Similar to previous results, we do not see any indication for gradients of different samples being better aligned than gradients of different tasks.



(a) CityScapes, SegNet (b) CityScapes, DeepLabV3 (c) NYUv2, SegNet (d) NYUv2, DeepLabV3 (e) CelebA, ResNet50

Figure A8. **Count of positive/ negative/ zero components in gradients.** For each gradient corresponding to one task and one sample, we count the number of positive and negative as well as **zero-valued** scalar entries. As expected, the amount of positive and negative values is approximately equal during the entire time of training along all experiments. For experiments on CityScapes and CelebA, we found that an increasing number of network components would not receive gradient updates anymore starting at some point in training. This especially holds for CelebA, were we assume to happen because of the ReLU activation functions used in ResNet18.



(a) CityScapes, SegNet (b) CityScapes, DeepLabV3 (c) NYUv2, SegNet (d) NYUv2, DeepLabV3 (e) CelebA, ResNet50

Figure A9. **Loss over experiments for gradient similarity experiment.** We visualize the loss with respect to the training, validation, and test set during the training process of the gradient similarity experiments.

A6. Additional results for out-of distribution generalization

We present results on generalization performance to corrupted data on NYUv2 with DeepLabV3 as well as on CityScapes with either choice of network in Figure A10 and Tables A8 and A9.

Additional to the main part of the paper, we run the corruption robustness experiments for NYUv2 with a DeepLabV3 network (Table A9) and for Cityscapes (Table A8, Figure A10). Across all experiments, we could not find any consistent pattern. Segnet+Cityscapes seems to vote in favour of MTL for more robustness on the depth task, though, for DeepLabV3+Cityscapes, STL is more robust on the same task. For the Sem.Seg task, both networks seem to be slightly more robust with STL.

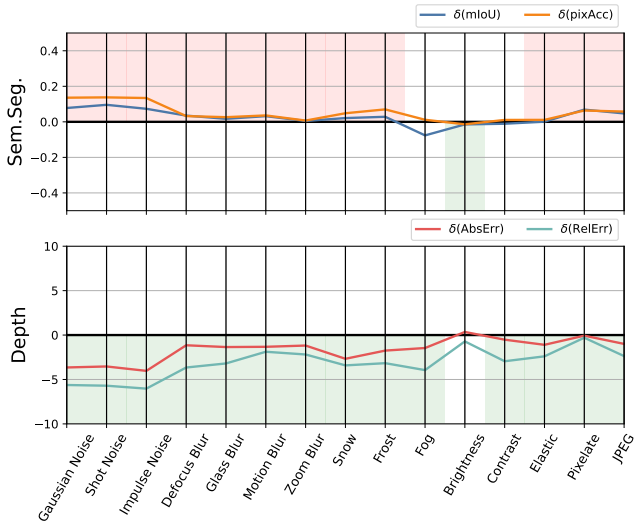
Interestingly, the δ_{Mean} and δ_{Median} for the Normal task of NYUv2 is consistently better for MTL, though, this does not hold for the other task metrics.

		Sem.Seg.		Depth		Mean
Network	MTO	δ_{mIoU}	δ_{pixAcc}	δ_{AbsErr}	δ_{RelErr}	
SegNet	EW	0.0268	0.0513	-1.6291	-3.1684	-1.1798
	UW	0.0532	0.0542	-0.1942	-1.2537	-0.3351
	RLW	0.0377	0.0483	-1.6592	-3.1739	-1.1868
	IMTL	0.0443	0.0301	-0.8538	-1.8310	-0.6526
	PCGrad	0.0385	0.0405	-1.6255	-3.1475	-1.1735
	CAGrad	0.0211	0.0221	-0.6951	-2.3448	-0.7492
DepLabV3	EW	0.0084	0.0079	0.7569	0.5861	0.3398
	UW	0.0051	0.0119	0.6217	0.2126	0.2128
	RLW	0.0178	0.0251	-0.0504	0.1443	0.0342
	IMTL	0.0261	0.0240	1.0149	0.5637	0.4072
	PCGrad	0.0227	0.0190	0.2163	0.4825	0.1851
	CAGrad	0.0135	0.0142	0.2889	0.4765	0.1983

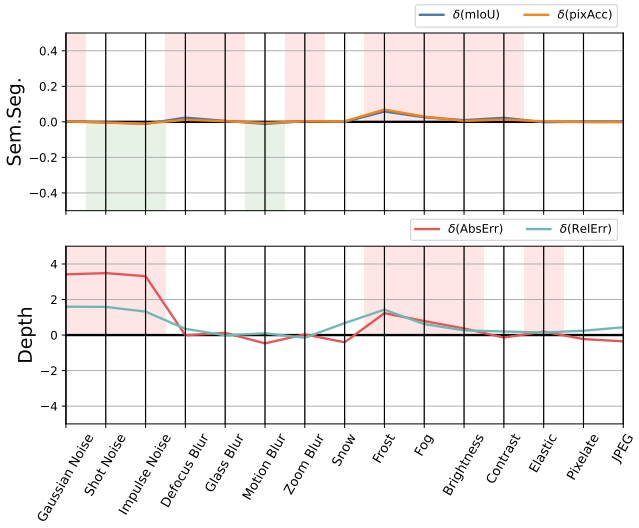
Table A8. **Out-Of Distribution generalization on corrupted CityScapes [8] dataset for different networks.** We report difference between relative performance decrease for single-task and multi-task learning averaged over all modes of corruption and all levels of severity (cf. Equation (6)). A value lower than zero indicates a better generalization capability of the MTL model, a positive value displays that the STL shows a lower decrease when evaluated on the corrupted data. Results are averaged over runs for three seeds for both multi-task and single-task models. We note a stronger performance of the MTL when using SegNet which is mainly due to a strong performance on the depth task. However, this behavior is not reinforced when using DeepLabV3, where the performance of the STL model decreases less over nearly all metrics and MTO methods.

		Sem.Seg.		Depth		Normal					Mean
Network	MTO	δ_{mIoU}	δ_{pixAcc}	δ_{AbsErr}	δ_{RelErr}	δ_{Mean}	δ_{Median}	$\delta_{<11.25}$	$\delta_{<22.5}$	$\delta_{<30.0}$	
SegNet	EW	0.0226	0.0298	0.1713	0.1869	-0.0750	-0.1390	0.0378	0.0167	0.0129	0.0293
	UW	0.0149	0.0200	0.1691	0.1652	-0.0461	-0.0895	0.0200	0.0168	0.0165	0.0319
	RLW	0.0236	0.0243	0.1250	0.1266	-0.0942	-0.1717	0.0342	0.0152	0.0110	0.0104
	IMTL	0.0144	0.0174	0.2434	0.2721	-0.0142	-0.0356	0.0342	0.0249	0.0217	0.0643
	PCGrad	0.0206	0.0156	0.1665	0.1634	-0.0594	-0.1131	0.0364	0.0208	0.0171	0.0298
	CAGrad	0.0053	0.0147	0.2006	0.2282	-0.0018	-0.0138	0.0402	0.0284	0.0240	0.0584
DeepLabV3	EW	-0.0027	-0.0131	0.1118	0.0880	-0.0328	-0.0628	-0.0056	0.0046	0.0049	0.0102
	UW	0.0050	-0.0147	0.0555	0.0336	-0.0160	-0.0328	0.0140	0.0163	0.0146	0.0084
	RLW	0.0067	-0.0040	0.0553	0.0006	-0.0125	-0.0309	0.0142	0.0170	0.0155	0.0069
	IMTL	0.0022	-0.0150	0.0511	0.0325	0.0073	-0.0086	0.0022	0.0068	0.0076	0.0096
	PCGrad	-0.0116	-0.0302	0.0505	0.0390	-0.0339	-0.0563	0.0040	0.0108	0.0102	-0.0019
	CAGrad	0.0064	0.0017	0.1699	0.1124	0.0156	0.0070	0.0097	0.0121	0.0117	0.0385

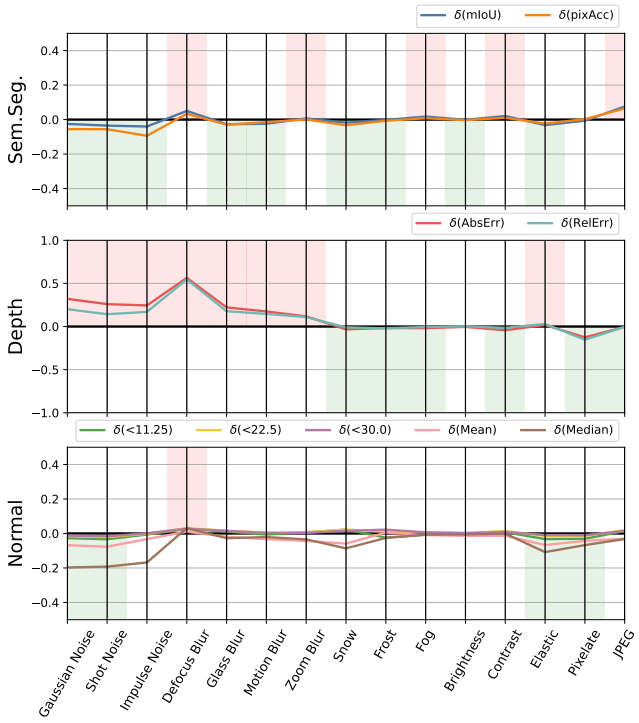
Table A9. **Out-Of Distribution generalization on corrupted NYUv2 [36] dataset for different multi-task optimization methods.** We report difference between relative performance decrease for single-task and multi-task learning averaged over all modes of corruption and all levels of severity (cf. Equation (6)). A value lower than zero indicates a better generalization capability of the MTL model, a positive value displays that the STL shows a lower decrease when evaluated on the corrupted data. Results are averaged over runs for three seeds for both multi-task and single-task models. Interestingly we found that even for different metrics corresponding to the same task, either the multi-task or single-task learning model would show lower decrease in performance on the corrupted data. There is no evidence that MTO methods would increase generalization capabilities of the trained models.



(a) CityScapes, SegNet



(b) CityScapes, DeepLabV3



(c) NYUv2, DeepLabV3

Figure A10. **Comparison of generalization performance to out-of-distribution data for MTL and STL** For every task and respective metrics, we show the difference over relative performance decrease over all corruption modes averaged over five levels of severity and three runs. EW was used to train the MTL model on uncorrupted data. We color blocks in case either STL or MTL is able to handle the respective corruption better for all metrics of one task. Regarding the CityScapes dataset, the performance on the depth task would benefit when using the SegNet network but not in case of DeepLabV3. For NYUv2+DeepLabV3, we even observe different behaviours across a single task for the different corruption modes. Overall, we do not see a clear evidence that multi-task learning would result in features that would generalize better to corrupted data.