Multi-Task Curriculum Learning Based on Gradient Similarity *

Hiroaki IGARASHI Kenichi YONEJI Kohta ISHIKAWA
Rei KAWAKAMI Teppei SUZUKI Shingo YASHIMA
Ikuro SATO

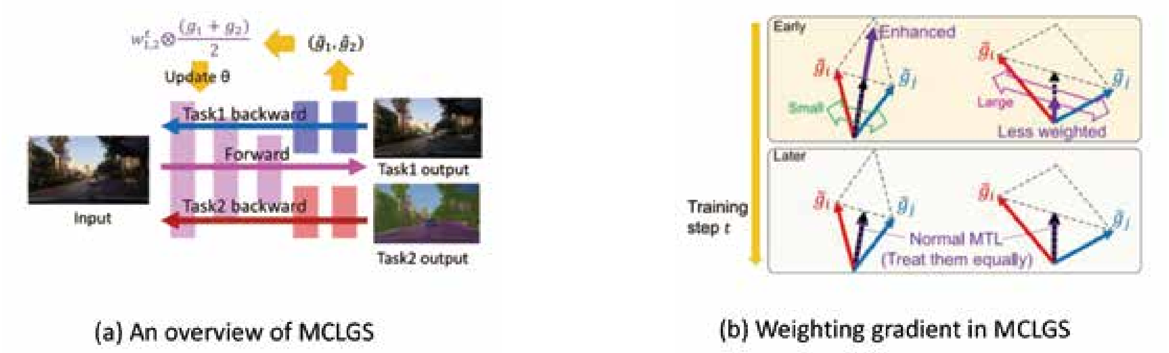
Intensive studies on multi-task learning (MTL) with deep neural networks have shown cases where both test error and computational cost can be reduced compared to single-task learning. However, several studies have argued that a naive implementation of MTL often degrades test performance due to gradient conflict, in which task-wise gradients have a negative inner product. These studies also invented ways to modify the gradients and eliminate the conflict. One concern about these methods is that the obtained solution is no longer optimal for the original objective due to the modification. In this paper, we propose a multi-task curriculum learning based on gradient similarity (MCLGS) to mitigate the negative impact of gradient conflicts while retaining the original objective toward the end of the training. We adopt a simple curriculum strategy that gives more weight to mini-batches exhibiting fewer gradient conflicts in the early stage of training. We experimentally confirmed that MCLGS outperforms existing MTL methods, such as MGDA, PCGrad, GradDrop, and CAGrad, on BDD100K and NYUv2 datasets.

Key words :
Machine Learning, Deep Neural Networks, Multi-task Learning

1 Introduction

Robotics applications, such as autonomous driving (AD) and advanced driver-assistance systems (ADAS), require multiple perceptional tasks [5, 7, 28], for example, object detection and semantic segmentation. If these tasks are implemented as separate models, the system will be complex and can involve redundant computations between each model. Multi-task learning (MTL) [6, 26, 30], which shares a portion of the network between multiple tasks, is a possible solution to simplify the system and reduce the complexity.

In MTL, if the magnitude of task-wise gradients is unbalanced during training, the trained model could be biased toward specific tasks. Therefore, several methods for balancing loss function have been proposed [3, 12]. However, if only the balancing loss function is applied, each task may have adverse effects on the other in MTL. One possible cause of this is gradient conflict, in which the inner product between the gradients of tasks is negative. In this case, since the parameter updates of each task are oriented in different directions, conflicting gradients sometimes lead to insufficient solution for each task.



(a) An overview of MCLGS

(b) Weighting gradient in MCLGS

Fig. 1 (a) An overview of MCLGS; $\tilde{g}_i$ and $\tilde{g}_j$ denote the i-th and j-th task-wise gradients on a shared network among tasks (shown as pink blocks). In MCLGS, a pair of task-wise gradients $(\tilde{g}_i, \tilde{g}_j)$ are weighted by the weight $w^t_{i,j}$ (shown in purple) and used to update the model parameters θ. (b) The design of the weight $w^t_{i,j}$. This depends on the amount of gradient conflict measured by the cosine similarity between task-wise gradients and the training step t. It is designed such that the weight becomes higher than 1 to encourage training if the cosine similarity is high; otherwise, it will be lower than 1 to suppress training in the early stage of training (top in (b)). As learning progresses, $w^t_{i,j}$ will not depend on cosine similarity and will always be around 1, which is consistent with the naive MTL update rule (bottom in (b))

Several studies [4, 16, 21, 29] have tackled this problem. For example, PCGrad [29] manipulated gradients such that the conflicting components were removed, and only the orthogonal components of each gradient were extracted and used for the update. Although such methods can remove gradient conflicts, a converged solution is no longer optimal for the original objective due to gradient manipulation. Specifically, if the conflicting components of gradients have a large difference in magnitude, then these components contain significant information. However, conventional methods simply discard this information by removing these components.

In this paper, we propose a multi-task curriculum learning based on gradient similarity (MCLGS), which mitigates the negative impact of gradient conflicts between tasks. MCLGS introduces a curriculum learning strategy [2] that removes hard samples in the early stage of training in multi-task learning. In single-task learning (STL), the difficulty of samples are generally determined by how hard input is to classify. For MTL, we redefine the difficulty by amount of gradient conflicts. Thus, in MCLGS, samples which generates gradient conflicts, are considered as hard to train in multi-task learning, and are downweighted in the early stage of training. However, these samples are gradually included as training progresses. Specifically, to validate our idea, we present a simple and somewhat heuristic function that determines the weights for each gradient given the gradient similarity and the training step. As shown in Fig. 1, the function is designed such that the conflicting (aligned) gradients are downweighted (more weighted) in the early stages, and any type of gradients is treated equally at later stages; that is, the weights for each gradient approach a fixed value. By applying these strategies, MCLGS mitigates the negative impact of gradient conflicts without gradient manipulation, and the update rule of MCLGS is consistent with that of naive MTL at the end. Thus, MCLGS retains the original objective toward the end of the training and helps to converge on a better solution than conventional methods. We confirmed experimentally that MCLGS outperforms existing MTL methods, such as MGDA [21], PCGrad [29], GradDrop [4], and CAGrad [16], on NYUv2

* Proceedings of the British Machine Vision Conference (BMVC), 2022.

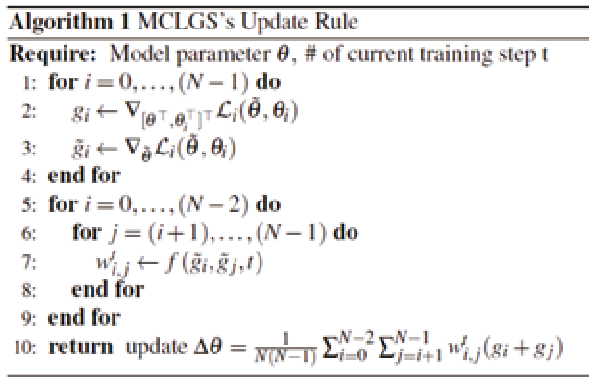
[22] and BDD100K datasets [28]. Although existing methods do not improve the performance from the baseline on the BDD100K dataset, MCLGS performs even better than the baseline.

2 Related Work

As categorized in [26], existing approaches in MTL belong to either architectural methods or optimization strategy methods. An example of architectural methods is a self-attention mechanism adopted to obtain better features shared among tasks [17]. In MTL, a backbone, which works as a feature extractor, is generally shared among multiple tasks as shown in Fig. 1 (a). However in [8, 18, 19], task-wise backbones were implemented individually and connected via connection layers for sharing features. For a head, such as the classifier or regressor, a cascaded structure was proposed to share features of the early stages [27, 31]. These methods manually introduced new connections between task-specific networks. To automatically find such connections during training, neural architecture search has also been utilized [10, 23]. Since MCLGS does not depend on a specific architecture, it can be combined with these methods.

In contrast, several approaches focus on the optimization strategy of MTL. For example, a loss balancing scheme was proposed based on homoscedastic uncertainty [12] or the norm of the gradient [3]. Similar to MCLGS, some studies [9, 15, 20] introduced a learning strategy inspired by the curriculum [2] that orders training data from easy ones to hard ones. For example, [9] and [15] prioritized tasks during training depending on the difficulty of the task. [20] divided tasks into strongly and weakly correlated groups, and applied transfer learning from the former to the latter. Since these methods are not motivated by the reduction of the gradient conflict, MCLGS can also be combined with them.

Several gradient manipulation methods [4, 16, 21, 29] have been proposed to remove gradient conflicts. For example, PCGrad [29] removed the conflicting gradient components of two tasks by simply selecting one task and subtracting the conflicting component for the other task. This was repeated for random combinations of the tasks. Only the orthogonal component for the other task is used for the parameter update. CAGrad [16] modified gradients to be a Pareto-optimal point around the original objective. However, if we manipulate the gradients, the retained solution may no longer be optimal since the objective function will be deviated from the original one. Meanwhile, MCLGS retains the original objective toward the end of the training and helps to converge on a better solution than these methods.



```
Algorithm 1 MCLGS's Update Rule
Require: Model parameter $\theta$, # of current training step t
for $i = 0,\ldots,(N-1)$ do
    $g_i \leftarrow \nabla_{[\theta^\top,\theta_i^\top]^\top} \mathcal{L}_i(\tilde{\theta},\theta_i)$
    $\tilde{g}_i \leftarrow \nabla_{\tilde{\theta}} \mathcal{L}_i(\tilde{\theta},\theta_i)$
end for
for $i = 0,\ldots,(N-2)$ do
    for $j = (i+1),\ldots,(N-1)$ do
        $w^t_{i,j} \leftarrow f(\tilde{g}_i,\tilde{g}_j,t)$
    end for
end for
return update $\Delta\theta = \frac{1}{N(N-1)} \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} w^t_{i,j}(g_i + g_j)$
```

3 Method

MTL aims to train a partially shared network to minimize the objectives of all tasks simultaneously. Considering that we have N tasks, then the loss function for MTL is given as follows:

$$\mathcal{L}(\tilde{\theta},\theta_0,\ldots,\theta_{N-1}) = \sum_{i=0}^{N-1} \mathcal{L}_i(\tilde{\theta},\theta_i), \tag{1}$$

where $\mathcal{L}(\tilde{\theta},\theta_i)$ denotes the loss function of the i-th task, and $\tilde{\theta}$ and θ_i represent the shared parameters and specific parameter of the i-th task, respectively (*i.e.*, the so-called head, such as the classifier and regressor). Hence, $\theta{:=}(\tilde{\theta}\ \theta_0,\ldots,\theta_{N\text{-}1})$ refers to all parameters in the network. Here, $g_i := \nabla_\theta\ \mathcal{L}_i\ (\tilde{\theta},\theta_i)$ denotes a batch gradient of the i-th task, and η is the learning rate. Thus, the update rule in MTL is as follows:

$$\theta' = \theta - \eta \frac{1}{N} \sum_{i=0}^{N-1} g_i. \tag{2}$$

MCLGS focuses on $\tilde{g}_i := \nabla_{\tilde{\theta}}\ \mathcal{L}_i\ (\tilde{\theta},\theta_i)$, which is a batch gradient of the shared parameter $\tilde{\theta}$. While conventional methods [16, 21, 29] removed gradient conflicts by manipulating the gradients, MCLGS introduces a curriculum learning strategy based on directional similarity into multi-task learning. Curriculum learning [2] is a training paradigm that orders training data from easy to hard, like a human learning strategy. By introducing this strategy, the learner can update parameters toward better local minima in the early stages and can reach a better solution.

To introduce this strategy in an MTL setting, MCLGS considers samples that produce more gradient conflicts as harder samples. In addition, we ignore these hard samples in the early stages of training by downweighting their gradients. In MCLGS, the curriculum is controlled by the weighting function f, and the batch gradient is weighted by $w^t_{i,j}$, which is an output of the weighting function f. An entire process of MCLGS's update rule is represented in Algorithm 1. First, task-wise gradients of shared parameters are extracted. Second, a relative weight among tasks is calculated by the weighting function f based on the number of gradient conflicts between the i-th and j-th tasks and the training step t. More details of the weighting function f will be described later. We define the relative weight $w^t_{i,j}$ as follows:

$$w^t_{i,j} := f(\tilde{g}_i,\tilde{g}_j,t). \tag{3}$$

Here, note that a curriculum is defined by f based on not only gradients but also the training step t. Specifically, f is designed such that the output increases according to the similarity in the beginning part of training. In the later training stages, the output will be independent of the similarity (*i.e.*, the output approaches a fixed value). Finally, the batch gradient is weighted by $w^t_{i,j}$, and the parameter θ is updated. The update rule of MCLGS is given as follows:

$$\theta' = \theta - \eta \frac{1}{N(N-1)} \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} w^t_{i,j}(g_i + g_j). \tag{4}$$

3.1 Weighting Function *f*

As formulated in [11], curriculum learning comprises scoring and pacing functions. The scoring function defines how hard the fed sample is, while the pacing function denotes how many hard samples are accepted in the current training step. These two functions also need to be introduced in the weighting function f. Additionally, following the definition of curriculum learning, the weighting function f should be defined as a monotonically increasing function. In this paper, we use the following function for the weighting function:

$$f(\tilde{g}_i,\tilde{g}_j,t) = \tanh(s(\tilde{g}_i,\tilde{g}_j)\,p(t)) + 1, \tag{5}$$

where s denotes the scoring function, and p represents the pacing function. Moreover, similar to [24], we use a cosine similarity given in the following equation as the scoring function:

$$s(\tilde{g}_i, \tilde{g}_j) = \frac{\tilde{g}_i \cdot \tilde{g}_j}{\|\tilde{g}_i\| \|\tilde{g}_j\|}. \tag{6}$$

The pacing function p is designed to approach 0 according to the increasing t. For example, it could be a linear decay, which is given as follows:

$$p(t) = \max(a_0 - t\Delta a, 0). \tag{7}$$

Conversely, it could be an exponential decay, given as follows:

$$p(t) = a_0 r_a^t. \tag{8}$$

An example of the weighting function f is shown in Fig. 2. The angle of f will be smoother according to the increasing t, and finally, it will converge at 0, which signifies a fixed weight ($w^t_{i,j}$ = 1). Additionally, a_0 and Δa for the linear decay and r_a^t for the exponential decay are hyperparameters, and should be tuned to the target model architecture or the target dataset appropriately. These hyperparameters could be one of the limitations of MCLGS, but it is a common problem in many MTL methods [3, 4, 16]. Regarding how to select the hyperparameters, we will experimentally show the parametric sensitivity studies in Section 4.2.1. Additionally, since this paper focuses on introducing curriculum learning based on gradient similarity in MTL, we selected the simple and somewhat heuristic weighting function f. Thus, the f in this paper might not be best solution and finding better f might be future work.

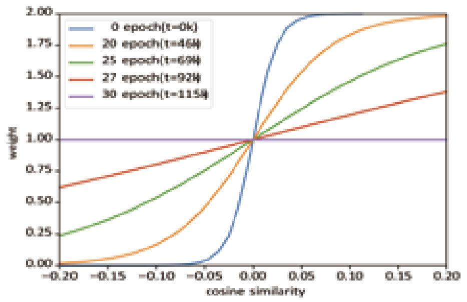


Fig. 2 An example of weighting function f on the BDD100K dataset. The angle of f will be smoother according to the increasing t, and finally, it will converge at 0, signifying a fixed weight ($w^t_{i,j}$ = 1). Note that linear decay (Eq. 7) is used for the pacing function, where a_0 = 40, and Δa = 3e-4

4 Evaluation

We evaluated MCLGS in a common MTL setting and the AD/ADAS MTL setting. For the common MTL setting, we chose the NYUv2 [22] dataset, which consists of three computer vision tasks: semantic segmentation, depth estimation, and surface normal prediction. For the AD/ADAS setting, we selected the BDD100K [28] dataset, which contains two computer vision tasks: object detection and semantic segmentation.

4.1 Setup

4.1.1 The NYUv2 Dataset

We followed the evaluation setup in [16]. The single-task learning (STL) baseline model is SegNet [1], as described in [16], and the MTL baseline model is SegNet with MTAN [17]. Conventional methods and MCLGS are applied to the MTL baseline model. For the conventional methods, we evaluated MGDA [21], PCGrad [29], GradDrop [4], and CAGrad [16]. Additionally, the combination of MCLGS and CAGrad shows the compatibility of MCLGS. For the training setup, we apply the SGD optimizer with a learning rate of 0.007, a momentum of 0.9 and a weight decay of 0.0001 because the adaptive learning rate on the Adam optimizer could be incompatible with MCLGS. The results with the Adam optimizer, which is used in the evaluation setup [16], is reported in the supplementary material. Since MGDA [21] and CAGrad [16] changed the balance between task objectives, we applied uncertainty weigh loss [12] for loss balancing on all methods to achieve a fair comparison. We trained the model three times using each different random seed and calculated the average accuracy. Similar to [16], we also used the average per-task performance drop Δm. While Δm is directly calculated for all metrics in [16], we first calculated the average per-metric performance drop in task i. Thus, $\Delta m_i = \frac{1}{K}(-1)^{l_{i,j}}(M_{m,i,j} - M_{b,i,j})/M_{b,i,j}$, where m and b represent the target method and the STL baseline, respectively; K denotes the number of metrics on task i; and l_i = 1 if a higher value satisfies a criterion $M_{i,j}$ for the metric j of task i better; otherwise, l_i = 0. We calculated the average of m_i for all the tasks to get Δm. We used the STL baseline with the Adam optimizer as a baseline for Δm calculation.

4.1.2 The BDD100K Dataset

For the STL baseline model, we used FCOS-RT [25] for object detection. For semantic segmentation, we combined ResNet50, the feature pyramid network (FPN) used in FCOSRT [25], and a segmentation head in [13]. For the MTL models, we applied an FCOS head and a segmentation head in [13] to ResNet50 and an FPN used in FCOS-RT [25], which were shared among tasks. For the loss function, we followed [25] for object detection and used the cross entropy and dice loss weighted by 0.5 for semantic segmentation.

Although the BDD100K dataset comprises two tasks, each dataset is separated, and labels are not annotated on the same image (this setting is more closer to the actual operation than the common MTL setting). Therefore, if each sample could have a ground truth of only one task, then the loss function for the other sample would be missing. A missing loss function of the task means that the learner studies each task alternatively, which may cause catastrophic forgetting [14] during training. To avoid this, we pre-trained STL models for each task and utilized their outpus to train MTL model as pseudo labels for each one. Moreover, we always fed the ground truths and pseudo labels to the MTL setting. Therefore, the loss function of this setting is formulated as follows:

$$\begin{aligned} L_{mtl} = {} & w_{gt} w_{od} L_{gt,od} + w_{gt} w_{ss} L_{gt,ss} + \\ & w_{pseudo} w_{od} L_{pseudo,od} + w_{pseudo} w_{ss} L_{pseudo,ss}, \end{aligned} \tag{9}$$

where gt and $pseudo$ represent the ground truth and the pseudo labels, respectively, and od and ss denote object detection and semantic segmentation, respectively. Additionally, w denotes the weights of loss functions for each label and task type. Here, we searched these weights without any MTL methods and used w_{gt} = 0.8, w_{pseudo} = 0.2, w_{od} = 1.7 and w_{ss} = 0.3. Note that $L_{gt,od}$ and $L_{gt,ss}$ could be missing, but $L_{pseudo,od}$ and $L_{pseudo,od}$ always exist. To generate the pseudo label, we used thresholds for the teacher model output. The threshold is 0.3 for object detection, 0.2 for non-maximum suppression, and 0.8 for semantic segmentation.

Further details of the setup are as follows. We used the SGD optimizer with 0.9 momentum, 0.0001 weight decay, and we enabled Nestrov. The batch size was 16, and the total epoch was 30. We used a multi-step learning rate schedule with 0.1 times learning decay at the 16-th, 22-th, 28-th epoch. The initial learning rate was 0.01, and we used the learning rate warmup with a 500 step. We reduced the gradient norm below 10. For the evaluation metrics, we used COCO mAP@0.5:0.95 for object detection and mIoU for semantic segmentation. Similar to NYUv2, we trained the model three times with each different random seed and calculated the average accuracy. We applied

MCLGS and conventional methods, such as MGDA [21], PCGrad [29], GradDrop [4], and CAGrad [16].

4.2 Results

4.2.1 Parametric Sensitivity Study

As mentioned in Section 3.1, the hyperparameters of the pacing function is one of the limitations in MCLGS. These hyperparameters could have a large impact regarding the performance. Thus, first, we performed a parametric sensitivity study for the pacing function. In this paper, we used linear decay (Eq. 7) for the pacing function. Therefore, the pacing function comprises two hyperparameters: a_0 and Δa. Since Δa should be correlated with a_0, we introduced the following definition of Δa:

$$\Delta a = \frac{a_0}{r_t t_{total}}, \qquad (10)$$

where t_{total} represents the total training steps, and r_t denotes the decreasing ratio, which is a hyperparameter in this definition. For example, $r_t = 1$ means that when the training is finished, the curriculum also converges at the fixed weight. Similarly, $r_t = 1/2$ means that the curriculum converges in the middle of training.

Table 1　A parametric sensitivity study of the pacing function p for Δm% (lower is better) on the NYUv2 dataset. The performance is more sensitive regarding r_t than a_0. Note that the format of table values is (mean ± stderr)

		r_t		
		2	1	1/2
a_0	0	-	-0.50 ± 1.03	-
	20	-0.04 ± 0.31	-1.51 ± 0.67	-1.45 ± 0.61
	40	-0.75 ± 0.59	**-2.12 ± 0.48**	-1.03 ± 0.18
	80	-1.18 ± 0.46	-1.33 ± 0.31	-0.78 ± 0.89

Table 1 and Table 2 show the sensitivity study results regarding a_0 and r_t on the NYUv2 and BDD100K datasets, respectively. In both cases, the performance is more sensitive regarding r_t than a_0. To maximize performance, a_0 and Δa should be appropriate values. However, most cases with the curriculum outperform cases without the curriculum, shown as $a_0 = 0$, $r_t = 1$. Additionally, a_0 and r_t should have larger values than those of the NYUv2 dataset, meaning that the pacing function should be slow.

Table 2　A parametric sensitivity study of the pacing function p for Δm% (lower is better) on the BDD100K dataset. The performance is more sensitive regarding r_t than a_0. NaN represents the diverged cases due to the large weight of the curriculum. Note that the format of table values is (mean ± stderr)

		r_t				
		4	3	2	1	1/2
a_0	0	-	-	-	-3.27 ± 0.17	-
	40	-3.88 ± 0.26	-3.94 ± 0.12	-3.79 ± 0.28	-3.91 ± 0.27	-3.36 ± 0.27
	80	-3.96 ± 0.29	-3.79 ± 0.35	-3.86 ± 0.36	-3.76 ± 0.18	-3.58 ± 0.21
	120	-3.76 ± 0.36	-3.93 ± 0.24	**-4.03 ± 0.27**	-3.83 ± 0.30	-3.38 ± 0.19
	160	-3.80 ± 0.20	-4.00 ± 0.23	-3.78 ± 0.19	-4.02 ± 0.29	-3.72 ± 0.21
	200	-3.88 ± 0.22	NaN	NaN	-3.78 ± 0.19	-3.43 ± 0.18

Table 3　Multi-task learning resuluts of the NYUv2 dataset: MCLGS with uncertainty weigh loss [12] outperforms all the other methods. MCLGS with CAGrad improves the performance even better. For the loss weighting scheme, "equal" represents no loss balancing and "uncert" denotes the uncertainty weight loss [12]. #P denotes the relative model size compared to the vanilla SegNet. The best average result for each method is marked in bold. The best average result among all multi-task methods is annotated with boxes

			Segmentation		Depth		Surface Normal					
#P.	Method	Weighting	Accuracy↑		Error↓		Angle Distance↓		Within $t°$ ↑			Δm%↓ (mean ± stderr)
			mIoU	Pix Acc	Abs Err	Rel Err	Mean	Median	11.25	22.5	30	
3	STL Baseline	-	**39.33**	**64.55**	**0.5785**	**0.2339**	26.12	20.44	28.04	54.56	66.84	-
1.77	MTL Baseline (MTAN [17])	equal	**40.88**	**66.14**	0.5489	0.2290	27.83	23.23	23.63	48.93	62.15	1.29 ± 0.60
		uncert.	38.95	64.76	**0.5423**	**0.2185**	**26.55**	**21.67**	**25.69**	**52.12**	**65.20**	**-0.50 ± 1.03**
1.77	MGDA [21]	equal	20.52	53.16	0.6635	0.2532	26.00	20.45	28.07	54.58	66.96	14.69 ± 0.17
		uncert.	**36.42**	**63.54**	**0.5912**	**0.2286**	**25.51**	$\boxed{\mathbf{19.95}}$	$\boxed{\mathbf{28.89}}$	$\boxed{\mathbf{55.70}}$	$\boxed{\mathbf{68.06}}$	**0.71 ± 0.84**
1.77	PCGrad [29]	equal	**40.73**	**66.24**	0.5558	0.2275	27.70	23.07	23.70	49.25	62.49	1.24 ± 0.66
		uncert.	39.05	65.10	$\boxed{\mathbf{0.5366}}$	$\boxed{\mathbf{0.2163}}$	**26.36**	**21.35**	**26.38**	**52.81**	**65.70**	**-1.40 ± 0.56**
1.77	GradDrop [4]	equal	**40.56**	**66.13**	0.5538	0.2251	27.90	23.35	23.26	48.70	61.99	1.47 ± 0.06
		uncert.	39.18	64.87	**0.5397**	**0.2201**	**26.42**	**21.52**	**26.04**	**52.42**	**65.44**	**-0.81 ± 0.42**
1.77	CAGrad [16]	equal	**39.39**	**65.27**	**0.5578**	**0.2270**	**25.88**	20.61	27.69	54.34	66.98	**-1.21 ± 0.82**
		uncert.	37.51	64.05	0.5722	0.2339	25.93	**20.57**	**27.77**	**54.40**	**66.99**	0.78 ± 0.28
1.77	MCLGS (ours)	equal	**40.98**	**66.43**	0.5729	0.2358	27.51	22.96	23.96	49.44	62.65	1.98 ± 0.43
		uncert.	40.20	65.63	**0.5429**	**0.2174**	**26.03**	**21.11**	**26.68**	**53.32**	**66.26**	**-2.12 ± 0.48**
1.77	CAGrad + MCLGS (ours)	equal	$\boxed{\mathbf{41.37}}$	$\boxed{\mathbf{66.47}}$	0.5513	0.2230	$\boxed{\mathbf{25.50}}$	20.22	28.26	55.16	67.75	$\boxed{\mathbf{-3.37 \pm 0.72}}$
		uncert.	39.72	65.65	**0.5470**	**0.2222**	25.51	**20.15**	**28.47**	**55.33**	**67.82**	-2.74 ± 0.29

Table 4　Multi-task learning results of the BDD100K dataset: MCLGS outperforms all the other methods. Following the evaluation format of [16], #P denotes the relative model size compared to the STL baseline. The best average result among all multi-task methods is marked in bold

#P.	Method	OD mAP@.50:.95↑ (mean ± stderr)	SS mIoU↑ (mean ± stderr)	Δm%↓ (mean ± stderr)
2	STL baseline	25.37 ± 7.31e-4	49.66 ± 1.53e-3	-
1.15	MTL baseline	26.16 ± 1.48e-4	51.43 ± 4.53e-4	-3.33 ± 0.17
1.15	MGDA [21]	15.33 ± 6.24e-4	46.61 ± 1.37e-3	22.85 ± 0.40
1.15	PCGrad [29]	26.14 ± 1.56e-4	51.37 ± 3.16e-4	-3.25 ± 0.23
1.15	GradDrop [4]	26.09 ± 4.78e-5	51.42 ± 5.66e-4	-3.20 ± 0.21
1.15	CAGrad [16]	25.37 ± 5.64e-5	50.70 ± 7.30e-4	-1.05 ± 0.23
1.15	MCLGS (ours)	**26.31 ± 1.99e-4**	**51.82 ± 8.13e-4**	**-4.03 ± 0.27**

4.2.2 Main Results

We present the results of the NYUv2 dataset in Table 3. MCLGS achieves the smallest average per-task performance drop Δm of -2.12%. If just focusing on single-task performance, PCGrad and MGDA [21] are the best for depth estimation and surface normal prediction, respectively. However, these methods cannot improve the performance of the other tasks well. This could be because they change the objectives by gradient manipulation and their retained solution is biased toward specific tasks. In contrast, MCLGS is consistent with the original objectives and achieves the better performance of all tasks. Furthermore, the uncertainty weigh loss [12] is suitable with MCLGS and improves the performance compared to equal weighting. Additionally, MCLGS with CAGrad improves the performance even better. We set a_0 and Δa to 60 and 5e−4, respectively, based on the parametric sensitivity study. Moreover, as shown in Table 1 and Table 3, MCLGS with several hyperparameter settings achieves the better average per-task performance Δm than existing methods.

We present the results of the BDD100K dataset in Table 4. MCLGS achieves the best accuracy in both object detection and semantic segmentation even including standard errors. Meanwhile, the accuracy of the conventional methods is below the naive MTL setting. This might be because of pseudo labels, which are less accurate than ground truth labels. Hence, gradients of pseudo labels could conflict with those of ground truth labels. Here, conventional methods, such as PCGrad, remove the gradient conflicts whether the gradient comes from the ground truth or pseudo label. Thus, conventional methods may update parameters in the wrong direction. Meanwhile, since MCLGS just downweights a pair of conflicted gradients, it may clean up samples that contain inaccurate labels in this setting. We set a_0 and Δa to 120 and 4.3e–4, respectively, based on the parametric sensitivity study. Moreover, as shown in Table 2 and Table 4, MCLGS with all hyperparameter settings achieves the better average per-task performance Δm than existing methods.

5 Conclusion

In this paper, we proposed MCLGS, which mitigates the negative impact of gradient conflicts between tasks. MCLGS introduces a curriculum learning strategy [2] that utilizes only easy samples in the early stages of training in multi-task learning. Conventional methods do not update parameters in the corresponding direction for the original objective because they manipulate gradients to remove the conflicts. Meanwhile, MCLGS just downweights samples that generate gradient conflicts in the early stage of training, and any type of gradient is treated equally at later stages, which is consistent with the naive MTL update rule. Therefore, MCLGS retains the original objective toward the end of the training and helps to converge at a better solution than conventional methods. As a result, we confirmed experimentally that MCLGS is superior to the conventional methods and compatible with them, and it can reduce the average per-task performance drop Δm on the NYUv2 [22] and BDD100K datasets [28].

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*, 39(12):2481–2495, 2017.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

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

[4] 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. *Advances in Neural Information Processing Systems*, 33: 2039–2050, 2020.

[5] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[6] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.

[7] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1341–1360, 2020.

[8] Yuan Gao, Jiayi Ma, Mingbo Zhao, Wei Liu, and Alan L Yuille. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3205–3214, 2019.

[9] Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. Dynamic task prioritization for multitask learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 270–287, 2018.

[10] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *International Conference on Machine Learning*, pages 3854–3863. PMLR, 2020.

[11] Guy Hacohen and Daphna Weinshall. On the power of curriculum learning in training deep networks. In *International Conference on Machine Learning*, pages 2535–2544. PMLR, 2019.

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

[13] Alexander Kirillov, Kaiming He, Ross Girshick, and Piotr Dollár. A unified architecture for instance and semantic segmentation. http://presentations. cocodataset.org/COCO17-Stuff-FAIR.pdf, 2017.

[14] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[15] Changsheng Li, Junchi Yan, Fan Wei, Weishan Dong, Qingshan Liu, and Hongyuan Zha. Self-paced multi-task learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[16] Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[17] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019.

[18] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[19] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent multi-task architecture learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4822–4829, 2019.

[20] Nikolaos Sarafianos, Theodore Giannakopoulos, Christophoros Nikou, and Ioannis A Kakadiaris. Curriculum learning for multi-task classification of visual attributes. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2608–2615, 2017.

[21] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

[22] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[23] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33:8728–8740, 2020.

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

[25] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: A simple and strong anchorfree object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[26] 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.

[27] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 675–684, 2018.

[28] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020.

[29] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

[30] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[31] Zhenyu Zhang, Zhen Cui, Chunyan Xu, Yan Yan, Nicu Sebe, and Jian Yang. Patternaffinitive propagation across depth, surface normal and semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4106–4115, 2019.

Supplementary Materials

A Additional Experimental Results

In the main paper, we reported the experimental results of the NYUv2 dataset [6], where the optimizer is changed from Adam to SGD, because the adaptive learning rate in the Adam optimizer is not suitable with MCLGS. However, this change in the evaluation setup [3] could be unfair for the other existing methods. Thus, we also evaluated each method with the Adam optimizer following the setting in [3]. Additionally, Δm is recalculated using the result of the STL baseline with the Adam optimizer.

We present the full results on the NYUv2 dataset in Table 1. The Δm of MCLGS with CAGrad using SGD reaches -5.01% and outperforms all the other methods, while that of MCLGS using the SGD optimizer is -3.71% which is better than PCGrad and GradDrop. Additionally, since the adaptive learning rate of Adam is incompatible with MCLGS, MCLGS is more compatible with SGD than Adam.

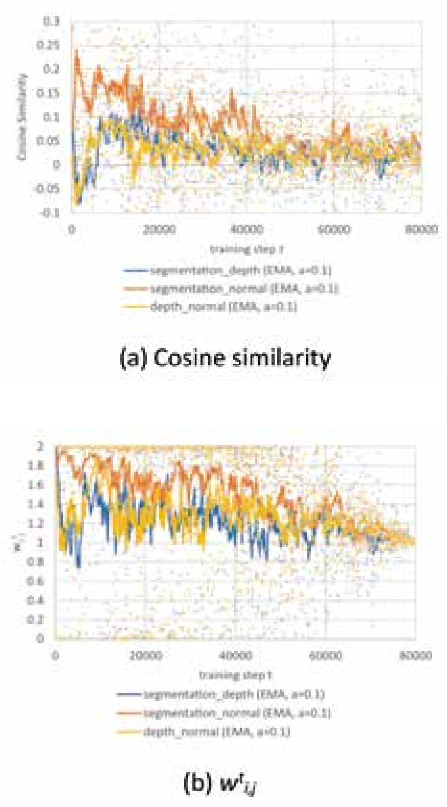
Additionally, we found that MGDA with uncertainty weigh loss performs effectively. While MGDA with equal weighting is biased against surface normal prediction, the uncertainty weigh loss improves the performance of semantic segmentation and depth estimation. This may be because MGDA breaks the balance of loss functions to maintain the Paretooptimal, while uncertainty weigh loss restores the balance. This is reasonable because CAGrad, which introduces a balance constraint of loss function into MGDA, also performs well.

Table 1 Multi-task learning resuluts of the NYUv2 dataset: MCLGS with CAGrad outperforms all the other methods. MCLGS without CAGrad is better than PCGrad and GradDrop. For the loss weighting scheme, "equal" represents no loss balancing, and "uncert" denotes the uncertainty weigh loss [2]. #P denotes the relative model size compared to the vanilla SegNet. The best average result for each method is marked in bold. The best average result among all multi-task methods is annotated with boxes

#P.	Method	Optim.	Weighting	Segmentation		Depth		Surface Normal					Δm%↓
				Accuracy↑		Error↓		Angle Distance↓		Within $t°$ ↑			(mean ± stderr)
				mIoU	Pix Acc	Abs Err	Rel Err	Mean	Median	11.25	22.5	30	
3	STL Baseline	Adam	-	37.88	63.43	0.6287	0.2532	**24.96**	**19.12**	**30.14**	**57.46**	**69.38**	-
		SGD	-	**39.33**	**64.55**	**0.5785**	**0.2339**	26.12	20.44	28.04	54.56	66.84	-1.72 ± 1.21
1.77	MTL Baseline (MTAN [4])	Adam	equal	38.49	64.68	0.5532	0.2273	28.00	23.73	22.23	47.90	61.48	1.73 ± 1.67
			uncert.	37.51	64.35	**0.5351**	0.2215	**26.51**	21.69	25.56	52.12	**65.20**	-1.31 ± 0.53
		SGD	equal	**40.88**	**66.14**	0.5489	0.2290	27.83	23.23	23.63	48.93	62.15	-0.42 ± 0.77
			uncert.	38.95	64.76	0.5423	**0.2185**	26.55	**21.67**	**25.69**	**52.12**	65.20	**-2.08 ± 0.99**
1.77	MGDA [5]	Adam	equal	30.17	60.56	0.6116	0.2419	$\boxed{\mathbf{24.66}}$	$\boxed{\mathbf{18.88}}$	$\boxed{\mathbf{30.69}}$	$\boxed{\mathbf{58.10}}$	$\boxed{\mathbf{70.06}}$	2.52 ± 0.96
			uncert.	**39.25**	**65.17**	**0.5644**	**0.2217**	24.75	19.44	29.54	56.90	69.34	**-4.58 ± 1.16**
		SGD	equal	20.52	53.16	0.6635	0.2532	26.00	20.45	28.07	54.58	66.96	13.03 ± 1.13
			uncert.	36.42	63.54	0.5912	0.2286	25.51	19.95	28.89	55.70	68.06	-0.96 ± 0.42
1.77	PCGrad [7]	Adam	equal	39.07	64.91	0.5542	0.2229	27.25	22.74	24.00	49.88	63.22	-0.18 ± 1.65
			uncert.	38.03	64.48	**0.5353**	0.2243	**26.00**	**20.97**	**26.91**	**53.56**	**66.41**	-2.35 ± 0.97
		SGD	equal	**40.73**	**66.24**	0.5558	0.2275	27.70	23.07	23.70	49.25	62.49	-0.47 ± 0.85
			uncert.	39.05	65.10	0.5366	**0.2163**	26.36	21.35	26.38	52.81	65.70	**-2.95 ± 0.64**
1.77	GradDrop [1]	Adam	equal	**40.57**	65.82	0.5413	0.2242	27.99	23.80	22.08	47.78	61.44	0.06 ± 0.82
			uncert.	38.37	64.56	**0.5369**	0.2270	**26.37**	21.57	25.62	52.36	**65.53**	-1.48 ± 0.45
		SGD	equal	40.56	**66.13**	0.5538	0.2251	27.90	23.35	23.26	48.70	61.99	-0.23 ± 1.14
			uncert.	39.18	64.87	0.5397	**0.2201**	26.42	**21.52**	**26.04**	**52.42**	65.44	**-2.39 ± 1.14**
1.77	CAGrad [3]	Adam	equal	**40.06**	**65.87**	$\boxed{\mathbf{0.5325}}$	**0.2155**	25.69	20.78	26.94	54.02	67.05	**-4.53 ± 0.63**
			uncert.	39.31	65.43	0.5394	0.2283	**25.57**	20.71	27.19	54.22	**67.30**	-3.22 ± 1.19
		SGD	equal	39.39	65.27	0.5578	0.2270	25.88	20.61	27.69	54.34	66.98	-2.85 ± 1.33
			uncert.	37.51	64.05	0.5722	0.2339	25.93	**20.57**	**27.77**	**54.40**	66.99	-0.90 ± 1.02
1.77	MCLGS (ours)	Adam	equal	37.89	63.79	0.5646	0.2288	27.97	23.68	22.74	48.06	61.46	2.47 ± 1.72
			uncert.	37.28	63.52	0.5478	0.2273	26.35	21.30	26.53	52.85	65.64	-0.78 ± 0.34
		SGD	equal	**40.98**	**66.43**	0.5729	0.2358	27.51	22.96	23.96	49.44	62.65	0.19 ± 0.83
			uncert.	40.20	65.63	**0.5429**	**0.2174**	**26.03**	**21.11**	**26.68**	**53.32**	**66.26**	**-3.71 ± 0.76**
1.77	CAGrad + MCLGS (ours)	Adam	equal	38.82	64.94	0.5436	$\boxed{\mathbf{0.2151}}$	26.24	21.56	25.80	52.37	65.59	-2.47 ± 1.48
			uncert.	38.25	64.73	**0.5364**	0.2198	25.80	20.93	26.99	53.65	66.66	-2.90 ± 0.64
		SGD	equal	$\boxed{\mathbf{41.37}}$	$\boxed{\mathbf{66.47}}$	0.5513	0.2230	**25.50**	20.22	28.26	55.16	67.75	$\boxed{\mathbf{-5.01 \pm 0.57}}$
			uncert.	39.72	65.65	0.5470	0.2222	25.51	**20.15**	**28.47**	**55.33**	**67.82**	-4.33 ± 0.96

B Visualization Results of the Cosine Similarity and Weight

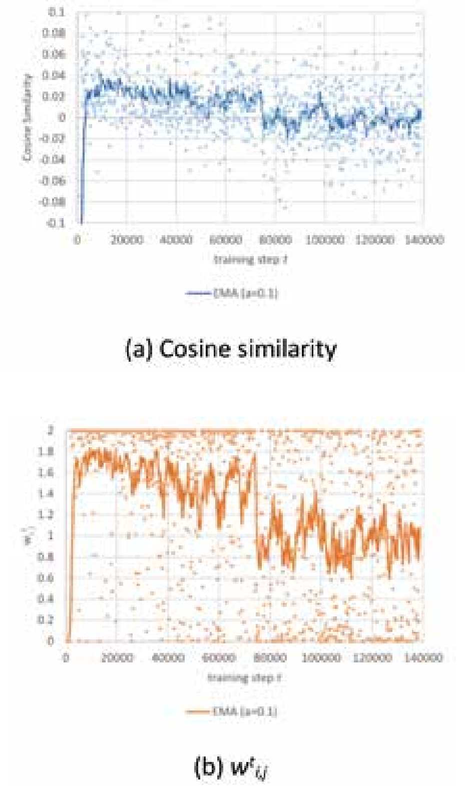
To confirm that the weight $w^t_{i,j}$ is generated depending on the cosine similarity, we visualized the cosine similarity and the weight generated by MCLGS in Fig. 1 and Fig. 2. As designed, $w^t_{i,j}$ fluctuates a lot depending on the cosine similarity but converges around 1.0 as learning progresses because MCLGS includes samples generating gradient conflicts at the end of the training. Additionally, we observed that the cosine similarity also tends to converge to 0, which means that the direction of task-wise gradients is orthogonal.



(a) Cosine similarity

(b) $w^t_{i,j}$

Fig. 1 Visualization results of cosine similarity and $w^t_{i,j}$ with increasing training step t of the NYUv2 dataset. The weight is generated depending on the cosine similarity. Segmentation, depth, and normal represent semantic segmentation, depth estimation, and surface normal prediction, respectively. Raw values are plotted as points, while lines represent the exponential mean average (EMA) of each value

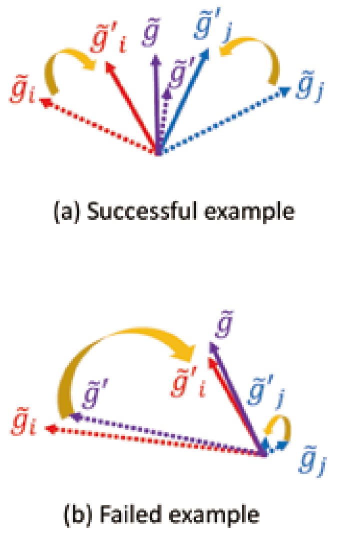


(a) Cosine similarity

(b) $w^t_{i,j}$

Fig. 2 Visualization results of cosine similarity and $w^t_{i,j}$ with increasing training step t of the BDD100K dataset. The weight is generated depending on the cosine similarity. Raw values are plotted as points while lines represent the exponential mean average (EMA) of each value

As shown in Fig. 1, the cosine similarity between semantic segmentation and surface normal estimation is higher than that at the beginning of the NYUv2 dataset. This means that the gradient of depth estimation highlights a different direction from that of the others. One possible reason could be that semantic segmentation and surface normal prediction are prone to using object boundary information, while depth estimation is not. Furthermore, the model might be trained to take this information initially.



(a) Successful example

(b) Failed example

Fig. 3 Examples that PCGrad [7] works well and not. (a) Projected average gradient $\tilde{g}$ is still similar to the original average gradient $\tilde{g}'$, while gradient conflict is removed. (b) Projected gradient $\tilde{g}$ is far from the original average gradient $\tilde{g}'$ even though gradient conflict is removed. In this case, the conflicted component of the task gradient $\tilde{g}_i$ is dominant on the average gradient because the norm of $\tilde{g}_i$ is much larger than that of $\tilde{g}_j$. However, this component is eliminated by PCGrad

C The Problem of Gradient Manipulation

In the main paper, we described that gradient manipulation leads to non-optimal solution for the original objectives. In this section, we will give the details of gradient manipulation and the case updating parameters into non-optimal direction.

For example, PCGrad [7] manipulated gradients such that the conflicting components were removed, and only the orthogonal components of each gradient were extracted and used for the update. Gradient manipulation of PCGrad is formulated as follows:

$$\tilde{g}_i = \tilde{g}_i - \frac{\tilde{g}_i \cdot \tilde{g}_j}{\|\tilde{g}_j\|^2}\tilde{g}_j, \qquad (1)$$

where $\tilde{g}_i$ and $\tilde{g}_j$ denote batch gradients of the i-th and j-th task, respectively. Note that Eq. 1 represents the manipulation for $\tilde{g}_i$, but PCGrad also applied this manipulation for $\tilde{g}_j$ as well. Fig. 3 shows examples of gradient projection by PCGrad. As shown in Fig. 3 (a), if the magnitude of task-wise gradients is similar, the projected average gradient $\tilde{g}$ is also similar to the original average gradient $\tilde{g}'$. Therefore, the retained solution is around the original objectives in this case. However, as shown in Fig. 3 (b), if the magnitude of task-wise gradient is much different, the retained solution is far from the original objectives. In this case, although conflicted component of $\tilde{g}_i$ is dominant on the average gradient $\tilde{g}$, PCGrad eliminated this component to remove gradient conflict.

References

[1] 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. *Advances in Neural Information Processing Systems*, 33: 2039–2050, 2020.

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

[3] Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[4] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019.

[5] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

[6] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[7] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

著者

五十嵐 博昭
いがらし ひろあき

AI 研究部
画像認識技術の研究開発に従事

米司 健一
よねじ けんいち

AI 研究部
画像認識関連の要素技術開発に従事

石川 康太
いしかわ こうた

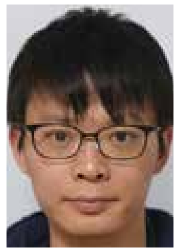
デンソーアイティーラボラトリ
機械学習，信号処理の研究開発に従事

川上 玲
かわかみ れい

東京工業大学工学院システム制御系准教授 博士（情報理工学）
コンピュータビジョン，画像処理の研究に従事

鈴木 哲平
すずき てっぺい

デンソーアイティーラボラトリ 博士（工学）
コンピュータビジョン，機械学習に関する研究開発に従事



八嶋 晋吾
やしま しんご

デンソーアイティーラボラトリ 研究開発グループ
モデル圧縮・画像認識の要素技術開発に従事

佐藤 育郎
さとう いくろう

デンソーアイティーラボラトリ 博士（理学）
機械学習関連の研究開発に従事