Multi-task Curriculum Learning Based on Gradient Similarity

Hiroaki Igarashi\textsuperscript{1}
hiroaki.igarashi.j3m@jp.denso.com
Kenichi Yoneji\textsuperscript{1}
kenichi.yoneji.j3v@jp.denso.com
Kohta Ishikawa\textsuperscript{2}
ishikawa.kohta@core.d-itlab.co.jp
Rei Kawakami\textsuperscript{3*}
reikawa@sc.e.titech.ac.jp
Teppei Suzuki\textsuperscript{2}
suzuki.teppei@core.d-itlab.co.jp
Shingo Yashima\textsuperscript{2}
yashima.shingo@core.d-itlab.co.jp
Ikuro Sato\textsuperscript{2,3}
sato.ikuro@core.d-itlab.co.jp

\textsuperscript{1} DENSO CORPORATION
Tokyo, Japan
\textsuperscript{2} DENSO IT Laboratory
Tokyo, Japan
\textsuperscript{3} Tokyo Institute of Technology
Tokyo, Japan

Abstract

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 weights to minibatches 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.

1 Introduction

Robotics applications, such as autonomous driving (AD) and advanced driver-assistance systems (ADAS), require multiple perceptual tasks \cite{5, 7, 28}, for example, object detection

© 2022. The copyright of this document resides with its authors.
It may be distributed unchanged freely in print or electronic forms.
* Work done while at DENSO IT Laboratory.
and semantic segmentation. If these tasks are implemented as separate models, the system will be complex and can involve redundant computations between each models. 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.

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 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 Figure 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 [8], and CAGrad [16], on NYUv2 [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 Figure 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 [30, 32]. 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 [9] or the norm of the gradient [8]. Similar to MCLGS, some studies [8, 9, 15] 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 $t$

1: for $i = 0, \ldots, (N - 1)$ do
2: $g_i \leftarrow \nabla_{[\theta^\top, \theta_i^\top]} \mathcal{L}_i(\tilde{\theta}, \theta_i)$
3: $\tilde{g}_i \leftarrow \nabla_{\tilde{\theta}} \mathcal{L}_i(\tilde{\theta}, \theta_i)$
4: end for
5: for $i = 0, \ldots, (N - 2)$ do
6: for $j = (i + 1), \ldots, (N - 1)$ do
7: $w_{i,j}^t \leftarrow f(\tilde{g}_i, \tilde{g}_j, t)$
8: end for
9: end for
10: return update $\Delta \theta = \frac{1}{N(N-1)} \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} w_{i,j}^t (g_i + g_j)$

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:

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

(1)

where $\mathcal{L}(\tilde{\theta}, \theta_i)$ denotes the loss function of the $i$-th task, and $\tilde{\theta}$ and $\theta_i$ 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, $\theta := (\tilde{\theta}, \theta_0, \ldots, \theta_{N-1})$ 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 $\eta$ is the learning rate. Thus, the update rule in MTL is as follows:

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

(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 \cite{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 \cite{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 $w_{i,j}^t$, 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 \( w_{i,j}^t \) as follows:
\[
w_{i,j}^t := f(\tilde{g}_i, \tilde{g}_j, t).
\]
(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 \( w_{i,j}^t \), and the parameter \( \theta \) is updated. The update rule of MCLGS is given as follows:
\[
\theta' = \theta - \eta \frac{1}{N(N-1)} \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} w_{i,j}^t (g_i + g_j).
\]
(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:
\[
f(\tilde{g}_i, \tilde{g}_j, t) = \tanh(s(\tilde{g}_i, \tilde{g}_j) p(t)) + 1,
\]
(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:
\[
s(\tilde{g}_i, \tilde{g}_j) = \frac{\tilde{g}_i \cdot \tilde{g}_j}{\|\tilde{g}_i\| \|\tilde{g}_j\|}.
\]
(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:
\[
p(t) = \max(a_0 - t \Delta a, 0).
\]
(7)

Conversely, it could be an exponential decay, given as follows:
\[
p(t) = a_0 r^t_a.
\]
(8)

An example of the weighting function \( f \) is shown in Figure 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 (\( w_{i,j}^t = 1 \)). Additionally, \( a_0 \) and \( \Delta a \) for the linear decay and \( r^t_a \) 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.
Figure 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 $\Delta a = 3e - 4$.

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 \cite{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 \cite{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 \cite{16}. The single-task learning (STL) baseline model is SegNet \cite{1}, as described in \cite{16}, and the MTL baseline model is SegNet with MTAN \cite{17}. Conventional methods and MCLGS are applied to the MTL baseline model. For the conventional methods, we evaluated MGDA \cite{21}, PCGrad \cite{29}, GradDrop \cite{4}, and CAGrad \cite{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 \cite{16}, is reported in the supplementary material. Since MGDA \cite{21} and CAGrad \cite{16} changed the balance between task objectives, we applied uncertainty weigh loss \cite{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 \cite{16}, we also used the average per-task performance drop $\Delta m$. While $\Delta m$ is directly calculated for all metrics in \cite{16}, we first calculated the average per-metric performance drop in task $i$. Thus, $\Delta m_i = \frac{1}{K} \sum_{j=1}^{K} (-1)^{l_i,j} (M_{m,i,j} - M_{b,i,j})/M_{b,i,j}$, 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 $\Delta m$. We used the STL baseline with the Adam optimizer as a baseline for $\Delta m$ 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 FCOS-RT [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 outputs 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:

\[ L_{\text{mtl}} = w_{gt}L_{\text{gt,od}} + w_{gt}L_{\text{gt,ss}} + w_{\text{pseudo}}L_{\text{pseudo,od}} + w_{\text{pseudo}}L_{\text{pseudo,ss}}, \]  

(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 \(w_{gt} = 0.8\), \(w_{\text{pseudo}} = 0.2\), \(w_{od} = 1.7\) and \(w_{ss} = 0.3\). Note that \(L_{\text{gt,od}}\) and \(L_{\text{gt,ss}}\) could be missing, but \(L_{\text{pseudo,od}}\) and \(L_{\text{pseudo,ss}}\) 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 warm up 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 \(\Delta a\). Since \(\Delta a\) should be correlated with \(a_0\), we introduced the following definition of \(\Delta a\):

\[ \Delta a = \frac{a_0}{r_{it}t_{total}}, \]  

(10)
Table 1: A parametric sensitivity study of the pacing function $p$ for $\Delta m\%$ (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).

<table>
<thead>
<tr>
<th>$a_0$</th>
<th>2</th>
<th>1</th>
<th>1/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.50 ± 1.03</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>-0.04 ± 0.31</td>
<td>-1.51 ± 0.67</td>
<td>-1.45 ± 0.61</td>
</tr>
<tr>
<td>40</td>
<td>-0.75 ± 0.59</td>
<td>-2.12 ± 0.48</td>
<td>-1.03 ± 0.18</td>
</tr>
<tr>
<td>80</td>
<td>-1.18 ± 0.46</td>
<td>-1.33 ± 0.31</td>
<td>-0.78 ± 0.89</td>
</tr>
</tbody>
</table>

Table 2: A parametric sensitivity study of the pacing function $p$ for $\Delta m\%$ (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).

<table>
<thead>
<tr>
<th>$a_0$</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>1/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-3.88 ± 0.26</td>
<td>-3.94 ± 0.12</td>
<td>-3.79 ± 0.28</td>
<td>-3.91 ± 0.27</td>
<td>-3.36 ± 0.27</td>
</tr>
<tr>
<td>40</td>
<td>-3.96 ± 0.29</td>
<td>-3.79 ± 0.35</td>
<td>-3.86 ± 0.36</td>
<td>-3.76 ± 0.18</td>
<td>-3.58 ± 0.21</td>
</tr>
<tr>
<td>80</td>
<td>-3.76 ± 0.36</td>
<td>-3.93 ± 0.24</td>
<td>-4.03 ± 0.27</td>
<td>-3.83 ± 0.30</td>
<td>-3.38 ± 0.19</td>
</tr>
<tr>
<td>120</td>
<td>-3.80 ± 0.20</td>
<td>-4.00 ± 0.23</td>
<td>-3.78 ± 0.19</td>
<td>-4.02 ± 0.29</td>
<td>-3.72 ± 0.21</td>
</tr>
<tr>
<td>160</td>
<td>-3.88 ± 0.22</td>
<td>NaN</td>
<td>NaN</td>
<td>-3.78 ± 0.19</td>
<td>-3.43 ± 0.18</td>
</tr>
<tr>
<td>200</td>
<td>-3.88 ± 0.22</td>
<td>NaN</td>
<td>NaN</td>
<td>-3.78 ± 0.19</td>
<td>-3.43 ± 0.18</td>
</tr>
</tbody>
</table>

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.

Tables 1 and 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 $\Delta a$ 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.

### 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 $\Delta m$ 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 method can not 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 $\Delta a$ to 60 and $5e^-4$, respectively, based on the parametric sensitivity study. Moreover, as shown in Tables 1 and 3, MCLGS with several hyperparameter settings
Table 3: Multi-task learning results for the NYUv2 dataset: MCLGS with uncertainty weight 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.

<table>
<thead>
<tr>
<th>#P. Method</th>
<th>OD</th>
<th>SS</th>
<th>Δm%↓ (mean ± stderr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STL Baseline</td>
<td>25.37 ± 7.31e-4</td>
<td>49.66 ± 1.53e-3</td>
<td>-</td>
</tr>
<tr>
<td>1.15 MTL Baseline (MTAN)</td>
<td>26.16 ± 1.48e-4</td>
<td>51.43 ± 4.53e-4</td>
<td>-3.33 ± 0.17</td>
</tr>
<tr>
<td>1.15 MGDA [21]</td>
<td>15.33 ± 6.24e-4</td>
<td>46.61 ± 1.37e-3</td>
<td>22.85 ± 0.40</td>
</tr>
<tr>
<td>1.15 PCGrad [29]</td>
<td>26.14 ± 1.56e-4</td>
<td>51.37 ± 3.16e-4</td>
<td>-3.25 ± 0.23</td>
</tr>
<tr>
<td>1.15 GradDrop [4]</td>
<td>26.09 ± 4.78e-5</td>
<td>51.42 ± 5.66e-4</td>
<td>-3.20 ± 0.21</td>
</tr>
<tr>
<td>1.15 CAGrad [16]</td>
<td>25.37 ± 5.64e-5</td>
<td>50.70 ± 7.30e-4</td>
<td>-1.05 ± 0.23</td>
</tr>
<tr>
<td>1.15 MCLGS (ours)</td>
<td>26.31 ± 1.99e-4</td>
<td>51.82 ± 8.13e-4</td>
<td>-4.03 ± 0.27</td>
</tr>
</tbody>
</table>

Table 4: Multi-task learning results for 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.

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

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 δ₀ and Δα to 120 and 4.3e – 4, respectively, based on the parametric sensitivity study. Moreover,
as shown in Tables 2 and 4, MCLGS with all hyperparameter settings achieves the better average per-task performance $\Delta m$ 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 $\Delta m$ on the NYUv2 [22] and BDD100K datasets [28].

## References


