LibMTL: A Python Library for Deep Multi-Task Learning

Baijiong Lin
The Hong Kong University of Science and Technology (Guangzhou)

Yu Zhang
Department of Computer Science and Engineering, Southern University of Science and Technology
Peng Cheng Laboratory

Abstract
This paper presents LibMTL, an open-source Python library built on PyTorch, which provides a unified, comprehensive, reproducible, and extensible implementation framework for Multi-Task Learning (MTL). LibMTL considers different settings and approaches in MTL, and it supports a large number of state-of-the-art MTL methods, including 13 optimization strategies and 8 architectures. Moreover, the modular design in LibMTL makes it easy to use and well extensible, thus users can easily and fast develop new MTL methods, compare with existing MTL methods fairly, or apply MTL algorithms to real-world applications with the support of LibMTL. The source code and detailed documentations of LibMTL are available at https://github.com/median-research-group/LibMTL and https://libmtl.readthedocs.io, respectively.

Keywords: Multi-Task Learning, Python, PyTorch

1. Introduction
Multi-Task Learning (MTL) (Caruana, 1997, Zhang and Yang, 2022) is an important area in both machine learning and industrial communities. By learning several related tasks simultaneously, this learning paradigm could not only improve the generalization performance but also reduce the storage cost and inference time, thus it has been applied to many real-world scenarios such as autonomous driving, natural language processing, recommendation system, robotic control, bioinformation, and so on (Zhang and Yang, 2022). Although many State-Of-The-Art (SOTA) MTL models have been proposed recently, most of them are implemented in their respective frameworks with different experimental details or there is no public implementation. Therefore, it is not easy to extend existing MTL algorithms to real-world applications or make a fair comparison with them when designing new MTL models.

To remedy such situation, we develop a Python library for MTL called LibMTL, which has three key features. Firstly, LibMTL provides a unified code base to cover different MTL settings such as the single-input and multi-input problems. Hence, it allows a convenient, fair, and consistent comparison between different MTL algorithms in various application
scenarios. Secondly, built on PyTorch (Paszke et al., 2019), LibMTL has supported lots of SOTA MTL models, especially deep MTL models, including 13 optimization strategies and 8 MTL architectures. Thirdly, LibMTL follows modular design principles, which allows users to flexibly and conveniently add customized components and make personalized modifications. Therefore, users can easily and fast develop new MTL models or apply existing MTL algorithms to their own application scenarios with the support of LibMTL.

2. Settings and Approaches in MTL

Suppose there are $T$ tasks and each task $t$ has its corresponding data set $D_t = \{X_t, Y_t\}$. Let $f(\cdot; \theta, \psi_{1:T})$ denotes an MTL model with task-shared parameters $\theta$ and task-specific parameters $\psi_{1:T}$. MTL aims to train a model $f$ on all data sets $D_{1:T}$ and expects $f$ to perform well on each task. There are usually two settings in MTL: the single-input case where each task has the same input data, i.e., $X_m = X_n$ for any $m \neq n$, and the multi-input case where each task has its own input data, i.e., $X_m \neq X_n$ for any $m \neq n$. Those two settings rely on concrete application scenarios and they are different in the training implementation.

There are two main lines of research for MTL. The first line is to design the architecture in deep neural networks for MTL and it directly determines which parameters are shared and how to share. The second line is to design the optimization strategy for MTL. Since how to balance multiple training losses in MTL directly affects the update of the task-shared parameters $\theta$, several methods are proposed to balance the losses or gradients of all the tasks in different ways, which are called loss balancing methods and gradient balancing methods, respectively. Moreover, gradient balancing methods need to calculate the gradients of the task-shared parameters $\theta$ for every task, which may be computationally intensive when the number of shared parameters or tasks is large. Thus, Sener and Koltun (2018) propose to use gradients of feature representations to approximate the exact gradients of shared parameters, which significantly reduces the computational cost and is followed by other gradient balancing methods such as GradDrop (Chen et al., 2020) and IMTL (Liu et al., 2021b). Obviously, those two ways to calculate gradients are different in implementations. Noticeably, those two lines of research are almost orthogonal to each other as the optimization methods are mainly related to the objective function, while the design of the architecture is to learn relationships between tasks. Thus, optimization strategies can be seamlessly combined with architectures to further improve the performance of MTL.

To summarize, as shown in Figure 1, MTL has two settings and its learning approaches can be divided into three categories.
3. The LibMTL Library

In this section, we introduce the LibMTL library, which provides a unified and easy-to-use framework for MTL as mentioned in Section 2. In Section 3.1, we introduce MTL methods implemented in LibMTL, which enables consistent and reproducible comparisons between different MTL algorithms. In Section 3.2, we present the modular design in LibMTL, which allows flexible and extensible customization for new MTL methods or potential MTL applications. In Section 3.3, we compare different MTL models on a benchmark data set based on the LibMTL library. In Section 3.4, we show that LibMTL is more comprehensive and up-to-date than existing MTL libraries.

3.1 Supported MTL Methods

Currently, LibMTL supports 13 optimization strategies, namely, Equal Weighting (EW), Gradient Normalization (GradNorm) (Chen et al., 2018), Uncertainty Weights (UW) (Kendall et al., 2018), MGDA (Sener and Koltun, 2018), Dynamic Weight Average (DWA) (Liu et al. 2019), Geometric Loss Strategy (GLS) (Chennupati et al., 2019), Projecting Conflicting Gradient (PCGrad) (Yu et al., 2020), Gradient sign Dropout (GradDrop) (Chen et al., 2020), Impartial Multi-Task Learning (IMTL) (Liu et al., 2021b), Gradient Vaccine (GradVac) (Wang et al., 2021), Conflict-Averse Gradient descent (CAGrad) (Liu et al., 2021a), Nash-MTL (Navon et al., 2022), and Random Weighting (RW) (Lin et al., 2022). Moreover, it supports 8 MTL architectures, i.e., Hard Parameter Sharing (HPS) (Caruana, 1993), Cross-stitch Networks (Misra et al., 2016), Multi-gate Mixture-of-Experts (MMoE) (Ma et al., 2018), Multi-Task Attention Network (MTAN) (Liu et al., 2019), Customized Gate Control (CGC) (Tang et al., 2020), Progressive Layered Extraction (PLE) (Tang et al., 2020), Learning to Branch (LTB) (Guo et al., 2020), and DSelect-k (Hazimeh et al., 2021). Besides, LibMTL supports combinations of each optimization strategy and each architecture.

3.2 The Modular Design of LibMTL

Figure 2 shows the overall framework of LibMTL, which is divided into different functional modules to allow users to flexibly and conveniently add customized designs or modifications in any module.

In LibMTL, each module has different functionalities. The DataLoader module is responsible for data pre-processing and loading. The LibMTL.loss module defines loss functions for each task. The LibMTL.metrics module defines evaluation metrics for all the tasks. The above three modules are highly dependent on the MTL problem under investigation. The LibMTL.config module is responsible for all the configuration parameters involved in the training process, such as the MTL setting (i.e., the multi-input case or not), possible hyper-parameters of optimization strategies and architectures, the training configuration (e.g., the batch size, the running epoch, the random seed, and the learning rate), and so on. This module adopts command-line arguments to enable users to set those configuration parameters conveniently. The LibMTL.Trainer module provides a unified framework for the training process under different MTL settings and for different MTL approaches as introduced in Section 2. The LibMTL.utils module implements useful functionalities for
the training process such as calculating the total number of parameters in an MTL model. The `LibMTL.architecture` and `LibMTL.weighting` modules contain the implementations of various architectures and optimization strategies, respectively, as introduced in Section 3.1. The `LibMTL.model` module includes some popular backbone networks (e.g., ResNet). The last three modules are highly related to MTL models.

Noticeably, such modular design makes `LibMTL` easy to use and well-extensible. For example, when applying to new applications, users only need to prepare the new dataloaders and select (or re-define) appropriate loss and metric functions, and they can use existing MTL methods implemented in `LibMTL`. Besides, for researchers to develop new MTL methods such as new architectures, they can easily implement their new method with the support of `LibMTL`, make a fair comparison with existing models, and combine the new architecture with modern optimization methods based on `LibMTL`.

### 3.3 Performance Comparison

In Table 1, we compare different MTL methods on the NYUv2 data set (Silberman et al., 2012) and set up a benchmark for MTL. The NYUv2 data set is an indoor scene understanding data set and has been used extensively in the MTL literature. It contains 3 tasks: semantic segmentation (denoted by Segmentation), depth estimation (denoted by Depth), and surface normal prediction (denoted by Normal). The implementation details and evaluation metrics are following Lin et al. (2022).

### 3.4 Comparison with Related Libraries

There are some libraries that have been developed for MTL recently. For example, RMTL (Cao et al., 2019) is implemented in R to support shallow MTL methods such as linear regularized methods. Another library, i.e., MTLV (Rahimi et al., 2021), only provides a limited number of MTL architectures for natural language processing. Compared with them, `LibMTL` is more comprehensive and up-to-date. Firstly, `LibMTL` covers more settings and approaches as introduced in Section 2, which means that `LibMTL` can be applied to more application scenarios. Secondly, `LibMTL` implements more SOTA MTL models, especially those based on deep neural networks.
<table>
<thead>
<tr>
<th>Methods</th>
<th>Segmentation</th>
<th>Depth</th>
<th>Normal</th>
<th>Angle Distance</th>
<th>Within $t^\circ$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mIoU↑</td>
<td>PAcc↑</td>
<td>AErr↓</td>
<td>RErr↓</td>
<td>Mean↓</td>
</tr>
<tr>
<td>EW</td>
<td>53.93</td>
<td>75.53</td>
<td>0.3825</td>
<td>0.1577</td>
<td>23.57</td>
</tr>
<tr>
<td>GradNorm</td>
<td>53.91</td>
<td>75.38</td>
<td>0.3842</td>
<td>0.1571</td>
<td>23.17</td>
</tr>
<tr>
<td>UW</td>
<td>54.29</td>
<td>75.64</td>
<td>0.3815</td>
<td>0.1583</td>
<td>23.48</td>
</tr>
<tr>
<td>MGDA</td>
<td>53.52</td>
<td>74.76</td>
<td>0.3852</td>
<td>0.1566</td>
<td>22.74</td>
</tr>
<tr>
<td>DWA</td>
<td>54.06</td>
<td>75.64</td>
<td>0.3820</td>
<td>0.1564</td>
<td>23.70</td>
</tr>
<tr>
<td>GLS</td>
<td>54.59</td>
<td>76.06</td>
<td>0.3785</td>
<td>0.1555</td>
<td>22.71</td>
</tr>
<tr>
<td>PCGrad</td>
<td>53.94</td>
<td>75.62</td>
<td>0.3804</td>
<td>0.1578</td>
<td>23.52</td>
</tr>
<tr>
<td>GradDrop</td>
<td>53.73</td>
<td>75.54</td>
<td>0.3837</td>
<td>0.1580</td>
<td>23.54</td>
</tr>
<tr>
<td>IMTL</td>
<td>53.63</td>
<td>75.44</td>
<td>0.3868</td>
<td>0.1592</td>
<td>22.58</td>
</tr>
<tr>
<td>GradVac</td>
<td>54.21</td>
<td>75.67</td>
<td>0.3859</td>
<td>0.1583</td>
<td>23.58</td>
</tr>
<tr>
<td>CAGrad</td>
<td>53.97</td>
<td>75.54</td>
<td>0.3885</td>
<td>0.1588</td>
<td>22.47</td>
</tr>
<tr>
<td>Nash-MTL</td>
<td>53.41</td>
<td>74.95</td>
<td>0.3867</td>
<td>0.1612</td>
<td>22.57</td>
</tr>
<tr>
<td>RLW</td>
<td>54.04</td>
<td>75.58</td>
<td>0.3827</td>
<td>0.1588</td>
<td>23.07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Different architectures with EW strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPS</td>
</tr>
<tr>
<td>Cross-stitch</td>
</tr>
<tr>
<td>MMoE</td>
</tr>
<tr>
<td>MTAN</td>
</tr>
<tr>
<td>CGC</td>
</tr>
<tr>
<td>PLE</td>
</tr>
<tr>
<td>LTB</td>
</tr>
<tr>
<td>DSelect-k</td>
</tr>
</tbody>
</table>

Table 1: Performance comparison on the *NYUv2* data set with three tasks. Each experiment is repeated over 3 random seeds and the average performance is reported. ↑ (↓) indicates that the higher (lower) the value, the better the performance.

4. Conclusion

We present LibMTL, a comprehensive and extensible library for MTL. Built on PyTorch, it provides a unified training framework for different settings in MTL and possesses many SOTA MTL algorithms. In our future work, we will continuously maintain this library to incorporate newly proposed MTL models, update the documentation, add more applications from different areas, and provide more backbone models such as vision transformer (Dosovitskiy et al., 2021) and N-grams (Brown et al., 1992).

Acknowledgments

This work is supported by NSFC key grant 62136005 and NSFC general grant 62076118.
References


Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.


