Improvable Gap Balancing for Multi-Task Learning

Yanqi Dai[1,3] **Nanyi Fei**[2,3] **Zhiwu Lu**[1,3]

[1]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
[2]School of Information, Renmin University of China, Beijing, China
[3] Beijing Key Laboratory of Big Data Management and Analysis Methods, Beijing, China

Abstract

In multi-task learning (MTL), gradient balancing has recently attracted more research interest than loss balancing since it often leads to better performance. However, loss balancing is much more efficient than gradient balancing, and thus it is still worth further exploration in MTL. Note that prior studies typically ignore that there exist varying improvable gaps across multiple tasks, where the improvable gap per task is defined as the distance between the current training progress and desired final training progress. Therefore, after loss balancing, the performance imbalance still arises in many cases. In this paper, following the loss balancing framework, we propose two novel improvable gap balancing (IGB) algorithms for MTL: one takes a simple heuristic, and the other (for the first time) deploys deep reinforcement learning for MTL. Particularly, instead of directly balancing the losses in MTL, both algorithms choose to dynamically assign task weights for improvable gap balancing. Moreover, we combine IGB and gradient balancing to show the complementarity between the two types of algorithms. Extensive experiments on two benchmark datasets demonstrate that our IGB algorithms lead to the best results in MTL via loss balancing and achieve further improvements when combined with gradient balancing. Code is available at https://github.com/YanqiDai/IGB4MTL.

1 INTRODUCTION

Multi-task learning (MTL) is to jointly train a single model that can perform multiple tasks [Caruana, 1998, Ruder, 2017, Zhang and Yang, 2021, Vandenhende et al., 2021]. Compared with single-task learning (STL), MTL has two remarkable advantages: 1) the model typically has a smaller size and higher learning efficiency by sharing parameters across tasks [Misra et al., 2016, Yang et al., 2020, Vandenhende et al., 2021], and 2) the performance on some tasks can be further improved due to the correlation between different tasks [Swersky et al., 2013]. Therefore, MTL has been widely used in real-world application scenarios such as recommendation systems [Zhao et al., 2019] and automatic driving [Chowdhuri et al., 2019].

Note that the largest challenge in MTL is the seesaw phenomenon [Tang et al., 2020]: joint training often results in better performance on some tasks but worse performance on others. To overcome this challenge, many optimization methods have been proposed for MTL: one is loss balancing that directly assigns different weights to the losses of multiple tasks according to a variety of criteria [Kendall et al., 2018, Liu et al., 2019, Lin et al., 2022]; the other is gradient balancing that first calculates the task gradients and then aggregates them in different ways [Sener and Koltun, 2018, Liu et al., 2021a, Navon et al., 2022]. Since gradient balancing typically performs better than loss balancing in MTL, it has attracted more research interest recently. However, the training cost of gradient balancing algorithms is significantly higher than that of most loss balancing algorithms, especially when there are significantly more tasks in MTL [Kurin et al., 2022].

Although most of prior studies focus on improving the performance of MTL by proposing more advancing methods, we emphasize that **improving the learning efficiency is also of the primary research significance in MTL**. Therefore, loss balancing and gradient balancing are both worth further study, and the trade-off between efficiency and performance can be taken according to practical requirements. In this paper, we focus on further improving the performance of existing loss balancing methods. Specifically, within the loss balancing framework, we propose two novel improvable gap balancing algorithms, where the improvable gap per task is defined as the distance between the current training progress and desired final training progress. To define the improvable gap, we have to first represent the training

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

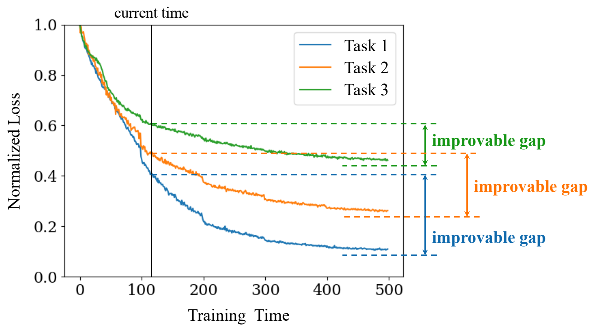

Figure 1: Schematic illustration of the improvable gaps for three different tasks in MTL. The improvable gap per task is defined as the distance between the current training progress and desired final training progress.

progress as the loss decline normalized by the average loss across the current training time (from the beginning of training). As shown in Figure 1, the normalized task losses tend to converge to nonzero values due to the limited training data and model capacity. Importantly, different tasks in MTL tend to have different convergence patterns (thus have different improvable gaps at the current training time). However, existing algorithms rarely notice that tasks trained together usually have different improvable gaps, which results in that some tasks are fully trained (or even making MTL outperform STL on these tasks), while others are still underfitted. Therefore, our main idea is to **dynamically assign task weights for improvable gap balancing (IGB)**, instead of loss balancing that has been widely used in MTL.

We propose the first algorithm IGBv1 to balance improvable gaps through a simple heuristic by uniformly defining the ideal loss as 0 for each task. Moreover, loss balancing can be interpreted as the maximum cumulative loss decline in the future by assigning weights through sequential decisions, which is consistent with the application scenario of reinforcement learning [Kaelbling et al., 1996]. Therefore, we propose another algorithm IGBv2 to jointly minimize all improvable gaps through deep reinforcement learning (DRL) [Arulkumaran et al., 2017]. Additionally, we combine IGB and gradient balancing to show the complementarity between the two types of algorithms.

In summary, our main contributions are three-fold:
(1) We propose two novel loss balancing algorithms to dynamically balancing the improvable gaps in MTL. To our best knowledge, we are the first to apply DRL to MTL.
(2) Extensive experiments on two benchmark datasets demonstrate that our IGB algorithms perform the best in loss balancing and yield further improvements when combined with gradient balancing.
(3) We rethink the significance of loss balancing in terms of learning efficiency as well as its complementarity with gradient balancing.

2 RELATED WORK

Multi-Task Learning. Multi-task learning (MTL) research is broadly divided into two categories: one is to learn the correlation between tasks through model structures [Misra et al., 2016, Ma et al., 2018, Liu et al., 2019], and the other is to balance the joint training process of all tasks through optimization algorithms [Kendall et al., 2018, Lin et al., 2022, Sener and Koltun, 2018, Liu et al., 2021b, Navon et al., 2022]. Our research is primarily concerned with the latter, which can be categorized into two types: loss balancing and gradient balancing.

Loss balancing directly updates the model by adding up or averaging the weighted losses after assigning task weights according to a variety of criteria. The training time of most loss balancing algorithms is nearly the same as that of STL, since the input data, such as losses of the current batch, is low-dimensional. The most common method for loss balancing is Equal Weighting (EW), which directly minimizes the sum of task losses. Besides, various criteria are considered in prior studies. For example, Kendall et al. [2018] measured task uncertainty through learnable parameters; Guo et al. [2018] estimated task difficulty based on key performance; Liu et al. [2019] considered change rate of loss; Lin et al. [2022] randomly assigned task weights; Ye et al. [2021] focused on metamodel validation performance.

Gradient balancing, on the other hand, first calculates task gradients separately and then updates the model by aggregating task gradients in different ways. The performance of gradient balancing is typically better than that of loss balancing, since it can deal with gradient conflicts [Yu et al., 2020] directly at the gradient level. For example, Chen et al. [2018] normalized task gradients to learn each task at a similar rate; Sener and Koltun [2018] regarded gradient aggregation as a multi-objective optimization problem; Liu et al. [2021a] added the condition of minimum average loss to Sener and Koltun [2018]; Yu et al. [2020] projected task gradients onto the normal planes of conflicting gradients; Chen et al. [2020] randomly dropped some task gradients; Liu et al. [2021b] aimed to make the aggregate gradient contribute equally to each task; Navon et al. [2022] considered gradient aggregation as a Nash bargaining game. However, due to multiple backpropagations and high-dimensional gradient aggregation, gradient balancing often requires greater training time than STL, which severely limits its learning efficiency in practice. Additionally, some works also explored the combination of loss balancing and gradient balancing, using loss balancing to update the task-specific parameters and combining loss balancing with gradient balancing to update the task-shared parameters [Liu et al., 2021b, Lin et al., 2022, Liu et al., 2022].

Reinforcement Learning. Reinforcement learning (RL) is an interactive machine learning decision-making method for streaming data to maximize the expected return [Kaelbling et al., 1996]. It is commonly modeled using a Markov decision process, which assumes that the future state is independent of the past state given the present state [Watkins, 1989]. In other words, let s_i be the state at time i, the state s_t is Markovian if and only if

$$\Pr(s_{t+1}|s_t) = \Pr(s_{t+1}|s_1, s_2, \cdots, s_t). \tag{1}$$

Soft Actor-Critic (SAC) [Haarnoja et al., 2018] is an off-policy actor-critic algorithm based on the maximum entropy RL framework, which is chosen as the loss weighting component in our IGBv2 algorithm. The Actor-Critic algorithm [Peters and Schaal, 2008] combines policy-based RL with value-based RL. The actor generates actions and interacts with the environment, while the critic evaluates the performance of the actor and directs the actions in the subsequent stage. This algorithm is more efficient than policy gradient methods because it can be updated at each step. The replay buffer [Mnih et al., 2013] in SAC is a classic off-policy mechanism where the agent learned and the agent interacting with the environment is different. It stores historical data for training, which is composed of four elements: the state s_t, the action a_t, the reward r_t and the next state s_{t+1}. The off-policy mechanism is beneficial to improve sample efficiency and reduce training instability caused by time series data. The maximum entropy reinforcement learning [Ziebart et al., 2008] changes the optimization objective to maximize both the expected return and the expected entropy of the policy. It increases the randomness of the policy, indicating that the probability distribution of the action is much wider. Note that more randomness is proved beneficial to improve performance in MTL [Lin et al., 2022, Chen et al., 2020].

In the context of RL, MTL can facilitate the transfer of knowledge between tasks, which has been shown to improve the performance of RL agents in various domains. For example, Chen et al. [2021] trained a MTL agent for autonomic optical networks, effectively expediting the training processes and improving the overall service throughput. Our work differs from this approach in that we, for the first time, apply reinforcement learning to solve general MTL optimization problems.

3 METHODOLOGY

In this section, we first provide a scale-invariant loss balancing paradigm. Based on this paradigm, we then describe our IGB algorithms. Finally, we give the fusion paradigm to combine loss balancing and gradient balancing.

3.1 SCALE-INVARIANT LOSS BALANCING

The objective of existing loss balancing algorithms is to minimize the weighted sum or average of task losses [Lin et al., 2021]. However, if task losses are on different scales, the model update is probably dominated by a specific task.

Algorithm 1 Training iteration of IGBv1

```
Input: task number $n$, current batch losses $\boldsymbol{L}$, current
    epoch $ce$, current batch $cb$, batch number of one epoch
    $bn$, learning rate $\eta$, task-shared parameters $\theta$, task-
    specific parameters $\{\psi_i\}_{i=1}^n$
Output: updated task-shared parameters $\theta'$, updated task-
    specific parameters $\{\psi'_i\}_{i=1}^n$
 1: if $ce = 2$ and $cb = bn$ then
 2:     Assign $\boldsymbol{L_{base}}$ to the average losses across every
        batch in the current epoch of all tasks;
 3: end if
 4: if $ce \leq 2$ then
 5:     $\boldsymbol{\lambda_{v1}} = \mathbf{1} \in \mathbb{R}^n$;
 6: else
 7:     $\boldsymbol{\lambda_{v1}} = n \times \mathrm{softmax}(\boldsymbol{L}/\boldsymbol{L_{base}})$;
 8: end if
 9: $\theta' = \theta - \eta \nabla_\theta \sum_{i=1}^n \lambda_{v1,i} \log(L_i)$;
10: for i = 1 to n do
11:     $\psi'_i = \psi_i - \eta \nabla_{\psi_i} \lambda_{v1,i} \log(L_i)$;
12: end for
```

To solve the scale difference problem of task losses, Navon et al. [2022] introduces a scale-invariant objective $\sum_{i=1}^n \log(L_i)$, where n is the number of tasks and L_i is the ith task loss. Inspired by this, we propose a scale-invariant loss balancing paradigm referred to as SI. The total loss of SI is defined as:

$$L_{total} = \sum_{i=1}^{n} \lambda_{l,i} \log(L_i), \tag{2}$$

where $\lambda_{l,i}$ is the ith task weight assigned by a specific loss balancing algorithm. In this paper, both of our IGB algorithms are designed based on the SI paradigm.

3.2 IMPROVABLE GAP BALANCING

We propose two loss balancing algorithms: IGBv1 directly balances improvable gaps across tasks estimated by a simple heuristic, while IGBv2 jointly minimizes all improvable gaps through a DRL model.

3.2.1 IGBv1

In the ideal situation, the MTL model can completely fit each training data, which satisfies $L_i = 0, \forall i$. Therefore, we assume that the desired final training losses of all tasks are 0, so that the improvable gaps can be directly represented by the normalized losses of the current batch, where the normalization is to deal with different task loss scales.

To normalize the training losses, we calculate the average losses across every batch in the second epoch of all tasks as $\boldsymbol{L_{base}} = [L_{base,1}, L_{base,2}, \ldots, L_{base,n}]$, since the losses of the first epoch may be too large to accurately represent the

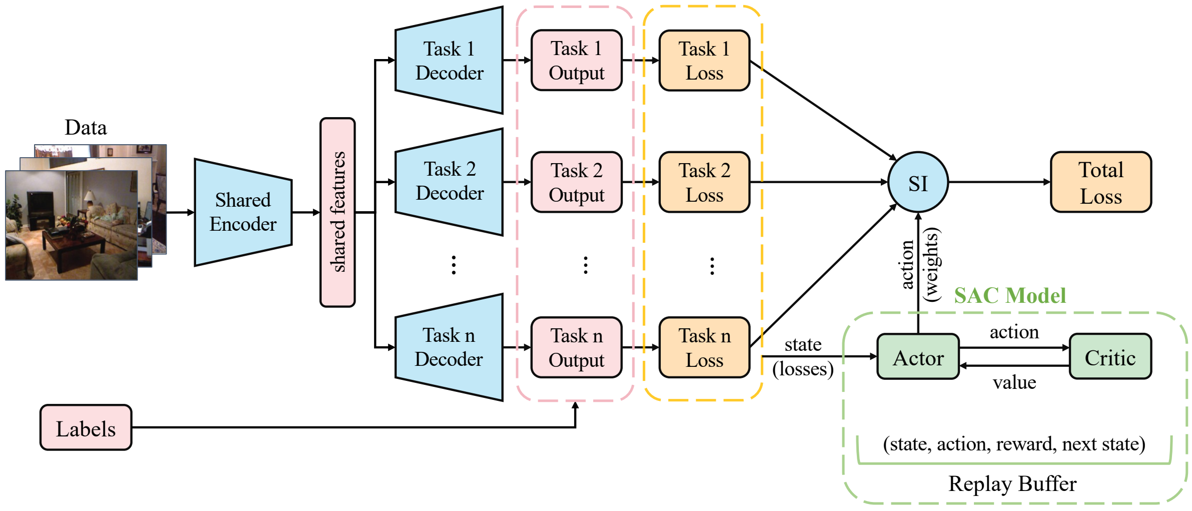

Figure 2: The architecture overview of our proposed IGBv2 algorithm by applying DRL to MTL.

loss scales if the MTL model is randomly initialized. In the first two epochs of training, the task weights are assigned to constant 1. Then in the subsequent training process, the task weights are calculated as:

$$\boldsymbol{\lambda_{v1}} = n \times \mathrm{softmax}\left(\frac{\boldsymbol{L}}{\boldsymbol{L_{base}}}\right), \tag{3}$$

where $\boldsymbol{L} = [L_1, \cdots, L_n]$ is the current batch losses of all tasks, and $\boldsymbol{L}/\boldsymbol{L_{base}}$ denotes the element-wise division of the two vectors. By this way, we dynamically assign more weights to tasks which have more improvable gaps. The training process of IGBv1 is summarized in Algorithm 1, where we initially set both the current epoch cp and the current batch cb to 1.

3.2.2 IGBv2

In practice, it is difficult for the MTL model to satisfy $L_i = 0, \forall i$ at the end of training. The reason is that existing deep learning methods cannot fully fit the training data distribution, and the minimal loss on the training data usually indicates overfitting. Therefore the estimation of the improvable gap in IGBv1 is inaccurate, and we present IGBv2 which can adaptively balance improvable gaps through DRL.

Note that assigning task weights for loss balancing can be interpreted as sequential decisions for maximizing the cumulative loss declines of tasks in the future, while the loss declines can be regarded as the feedback reward for task weights, which is consistent with the application scenario of RL. Inspired by this, we propose deploying a DRL model to assign task weights, with the loss declines of the MTL model as the reward for DRL.

We choose Soft Actor-Critic (SAC) [Haarnoja et al., 2018] as the DRL model to guide the training of the MTL model, which is an off-policy actor-critic algorithm based on the maximum entropy RL framework. As interpreted in Figure 2, the task weights are assigned by the SAC model with the losses of the MTL model as input, and the MTL model and the SAC model are alternately trained. The reasons for choosing SAC are as follows: 1) The actor-critic structure [Peters and Schaal, 2008] allows the SAC model to be updated at each step, enabling IGBv2 to assign the current optimal weights in time; 2) The replay buffer [Mnih et al., 2013] in SAC is a commonly used off-policy mechanism, which can improve sample efficiency by training the SAC model on random historical data. Because the MTL model can only be trained once, the training data available for our SAC model is much less than that for typical DRL applications, making sample efficiency crucial; 3) The maximum entropy mechanism [Ziebart et al., 2008] can increase the randomness of the action, thereby increasing the likelihood of discovering the global optimum of the MTL model.

Unlike deploying DRL to play games [Mnih et al., 2013] which requires millions of episodes, only one complete training is allowed to obtain the optimal model in MTL. Therefore, in this work, many details are redesigned to accommodate the differences between these two different domains. First, we consider the fundamental elements of DRL including environment, state, action, and reward:

- **Environment:** We regard the entire MTL model and the training data for MTL as the environment.
- **State:** The state is required to properly describe the current environment and not be too high-dimensional for learning efficiency. Therefore we choose the losses of the current batch as the state s_t, which is determined by both the current parameter situation of the MTL model and the input training data of the current batch.

Algorithm 2 Training iteration of IGBv2

```
Input: task number $n$, current batch losses $\boldsymbol{L}$, current epoch $ce$, current batch $cb$, batch number of one epoch $bn$, SAC model $sac$, replay buffer $buffer$, start epoch to update the SAC model $update\_e$, start epoch to use the SAC model $use\_e$, learning rate $\eta$, task-shared parameters $\theta$, task-specific parameters $\{\psi_i\}_{i=1}^n$
Output: updated task-shared parameters $\theta'$, updated task-specific parameters $\{\psi'_i\}_{i=1}^n$
 1: if $ce = 2$ and $cb = bn$ then
 2:     Assign $\boldsymbol{L_{base}}$ to the average losses across every batch in the current epoch of all tasks;
 3: end if
 4: if $ce > 2$ then
 5:     Add $(s_{t-1}, a_{t-1}, r_{t-1}, s_t)$ into $buffer$;
 6: end if
 7: if $ce \geq update\_e$ then
 8:     Train the SAC model $sac.\text{train}(buffer)$;
 9: end if
10: if $ce < use\_e$ then
11:     $\boldsymbol{\lambda_{v2}} = n \times \text{softmax}(\text{random\_normal}(n))$;
12: else
13:     $\boldsymbol{\lambda_{v2}} = sac.\text{select\_action}(\boldsymbol{L})$;
14: end if
15: $\theta' = \theta - \eta\nabla_\theta \sum_{i=1}^n \lambda_{v2,i} \log(L_i)$;
16: for i = 1 to n do
17:     $\psi'_i = \psi_i - \eta\nabla_{\psi_i}\lambda_{v2,i} \log(L_i)$;
18: end for
```

In this way, the past training process of the MTL model can be fully represented by the current state, which can be formulated as a standard Markov decision process.

- **Action:** We regard the task weights of the current batch as the action a_t, and limit it to positive numbers that sum to n for a fair comparison with other algorithms.
- **Reward:** Inspired by maximizing the minimum improvement of all tasks in Sener and Koltun [2018] and Liu et al. [2021a], we regard the minimum loss decline of all tasks in the current batch as the reward, where the losses are also normalized by the average losses of the second epoch $\boldsymbol{L_{base}}$. Additionally, since reducing the losses of the MTL model is more challenging when the learning rate is lower than it was at the start of training, we add a multiplier factor α to the reward, which is calculated as the ratio of the initial learning rate to the current learning rate. Finally, the reward is defined as:

$$r_t = \alpha \times \min\left(\frac{\boldsymbol{L_t} - \boldsymbol{L_{t+1}}}{\boldsymbol{L_{base}}}\right), \quad (4)$$

where $\boldsymbol{L_t}$ is the vector of the current batch losses and $\boldsymbol{L_{t+1}}$ is that of the next batch losses.

The SAC model adaptively assigns task weights to maximize the cumulative reward, so that the cumulative loss decline

Algorithm 3 Training iteration of combining loss balancing and gradient balancing

```
Input: loss balancing algorithm $LB$, gradient balancing algorithm $GB$, task number $n$, current batch losses $\boldsymbol{L}$, learning rate $\eta$, task-shared parameters $\theta$, task-specific parameters $\{\psi_i\}_{i=1}^n$
Output: updated task-shared parameters $\theta'$, updated task-specific parameters $\{\psi'_i\}_{i=1}^n$
 1: Compute the task weights assigned by loss balancing $\boldsymbol{\lambda_l} = LB(\boldsymbol{L})$;
 2: for i = 1 to n do
 3:     $g_i = \nabla_\theta \lambda_{l,i} \log(L_i)$;
 4: end for
 5: $\theta' = \theta - \eta GB(g_1, \cdots, g_n)$;
 6: for i = 1 to n do
 7:     $\psi'_i = \psi_i - \eta\nabla_{\psi_i}\lambda_{l,i} \log(L_i)$;
 8: end for
```

of each task can be maximized to the improvable gap. In other words, IGBv2 jointly minimizes the improvable gaps of all tasks by gradually minimizing the training losses to the desired final training losses.

Furthermore, we present the training process of IGBv2 in Algorithm 2, where we initially set both the current epoch *cp* and the current batch *cb* to 1. At the beginning of training, since the SAC model is not sufficiently trained to be used, RLW [Lin et al., 2022] is deployed to randomly assign task weights for MTL, which is also beneficial for the training of the SAC model through random exploration. After obtaining $\boldsymbol{L_{base}}$, we can get the state s_t, the action a_t, and the reward r_{t-1} in each training batch of the MTL model. These data are continually added into the replay buffer for training the SAC model. Once the SAC model is trained well enough, it is deployed to assign task weights instead of RLW.

Additionally, we carefully redesign the replay buffer size, which is significantly smaller than typical SAC applications. When the buffer size is too large, since the performance of the MTL model is gradually improved and the training losses are reduced gradually, training the SAC model with too earlier historical data is not beneficial to the current training of the MTL model. Conversely, when the buffer size is too small, the training instability caused by time series data and low sample efficiency may make the training of the SAC model unsatisfactory, thereby also leading to poor performance of the MTL model.

3.3 COMBINATION OF LOSS BALANCING AND GRADIENT BALANCING

Our IGB algorithms dynamically provide varying importance for each task to balance the improvable gaps, while gradient balancing algorithms deal more directly with gradient conflicts at the gradient level. These two types of algorithms are complementary and can be combined together for further improvements.

We first assign task weights with the loss balancing algorithm and then input the weighted losses to the gradient balancing algorithm to obtain the final update gradient. In this way, the performance can be further improved while keeping the training time almost the same as that of gradient balancing alone, since most loss balancing algorithms hardly add extra training time.

Typically, updating the model with existing gradient balancing algorithms can be divided into two ways: one assigns weights to gradients of both task-shared and task-specific parameters, while the other only aggregates gradients of task-shared parameters. As illustrated in Algorithm 3, when combining loss balancing and gradient balancing in the latter case, which is more common, task-shared parameters are updated by both loss balancing and gradient balancing, while task-specific parameters are updated independently by loss balancing.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. We train and evaluate our model on the NYUv2 dataset [Silberman et al., 2012] for multi-task scene understanding and on the QM9 dataset [Ramakrishnan et al., 2014] for multi-task regression prediction.

NYUv2 is an indoor scene image dataset with dense pixel-level 13-class labeling. It comprises 795 training samples and 654 test samples. Prior studies [Liu et al., 2021a, Navon et al., 2022] typically regard the average performance of the last 10 epochs on the test set as the final result, which is unreasonable in machine learning research. Therefore, we randomly divide the 654 original test samples into 197 validation samples (the validation set) and 457 test samples (the test set).

QM9 is a chemical molecule dataset widely used for graph neural networks (GNNs) [Wu et al., 2020], with approximately 130K molecular samples (which are represented as graphs annotated with node and edge features). Following Navon et al. [2022], we use this dataset for regression prediction on 11 properties of chemical molecules, and we use 110K samples for training, 10K samples for validation, and 10K samples for testing.

Compared Methods. We compare our methods with the following classic algorithms that have been described in Section 2: (1) Equal Weighting (EW) which minimizes $\sum_{i=1}^{n} L_i$; (2) Random Loss Weighting (RLW) [Lin et al., 2022]; (3) Dynamic Weight Average (DWA) [Liu et al., 2019]; (4) Uncertainty Weighting (UW) [Kendall et al., 2018]; (5) Multiple Gradient Descent Algorithm (MGDA) [Sener and Koltun, 2018]; (6) Projecting Conflicting Gradient (PCGrad) [Yu et al., 2020]; (7) Conflict-Averse Gradient (CAGrad) [Liu et al., 2021a]; (8) Impartial Multi-Task Learning (IMTL-G) [Liu et al., 2021b]; (9) Nash-MTL (Nash) [Navon et al., 2022].

Evaluation Metrics. In the experiments, we first report the common evaluation metrics for each task over each dataset. Moreover, since the significance of MTL optimization is to improve both model performance and learning efficiency, we report two overall metrics to comprehensively evaluate MTL optimization methods:

(1) Δm: the average per-task performance drop compared with STL, which is defined as:

$$\Delta m = \frac{1}{K} \sum_{k=1}^{K} (-1)^{\delta_k} \frac{M_{m,k} - M_{b,k}}{M_{b,k}}, \tag{5}$$

where K is the total number of common evaluation metrics over all tasks, $M_{m,k}$ is the value on the k-th metric of the evaluated method and $M_{b,k}$ is that of the STL baseline, $\delta_k = 1$ if a higher value is better for the k-th metric and 0 otherwise [Liu et al., 2021a, Navon et al., 2022].

(2) T: the relative training time compared with EW, which is calculated as the ratio of the training time of the evaluated method to that of the EW baseline.

Implementation Details. In the experiments on the NYUv2 dataset, we train a SegNet model from Badrinarayanan et al. [2017] for 500 epochs with the Adam optimizer [Kingma and Ba, 2014], while the learning rate is initially set to $1e$-4 and decays by half for every 100 epochs. Then we test the model which has the best *validation* metric Δm to obtain the final algorithm performance.

In the experiments on the QM9 dataset, following Navon et al. [2022], we normalize each task target to have zero mean and unit standard deviation, and use the code implemented by Fey and Lenssen [2019], which contains a popular GNN from Gilmer et al. [2017] and a pooling operator from Vinyals et al. [2015]. We train the model for 300 epochs with the Adam optimizer [Kingma and Ba, 2014], while the learning rate is searched in $\{1e\text{-}3, 5e\text{-}4, 1e\text{-}4\}$ and decreased by the ReduceOnPlateau scheduler according to the *validation* metric Δm.

For the IGBv2 algorithm, the SAC model is updated once every 50 batches during the MTL model training, where the learning rate of the SAC model is adjusted according to the situation: $1e$-4 when combining IGBv2 with MGDA [Sener and Koltun, 2018], PCGrad [Yu et al., 2020] or Nash [Navon et al., 2022], and $3e$-4 when combining IGBv2 with other gradient balancing algorithms or using IGBv2 alone. The replay buffer size is set to $1e4$. The discount factor γ is set to 0.99. The start epoch to update the SAC model $update\_e$ is set to 4, and the start epoch to use the SAC model use_e is set to 6.

Table 1: Comparative results on the NYUv2 dataset for multi-task scene understanding. $\uparrow$ ($\downarrow$) indicates that the higher (lower) the result, the better the performance. $+$ represents the combination of loss balancing and gradient balancing. In each group, the best results are **bolded** and the second-best results are underlined.

Methods	Segmentation		Depth		Surface Normal					$\boldsymbol{\Delta m}\downarrow$	$\boldsymbol{T}\downarrow$
	mIoU↑	Pix Acc↑	Abs Err↓	Rel Err↓	Angle Distance↓		Within t°↑				
					Mean	Median	11.25	22.5	30		
STL	41.16	65.70	0.6074	0.2400	24.49	18.24	31.92	59.16	70.56		
EW	40.12	66.16	0.5189	0.2039	28.30	23.58	23.07	48.35	61.39	8.45	1.00
RLW [Lin et al., 2022]	39.72	65.40	0.5252	0.2092	28.85	24.38	21.59	46.74	59.98	10.82	1.00
DWA [Liu et al., 2019]	41.60	66.52	0.5041	**0.2000**	28.11	23.32	23.32	48.71	61.83	7.08	1.00
UW [Kendall et al., 2018]	40.29	64.60	0.5081	0.2058	26.69	21.47	26.18	52.43	65.21	4.09	1.00
IGBv1 (ours)	39.91	66.03	**0.4961**	0.2009	26.08	20.69	27.52	54.11	66.57	1.76	1.00
IGBv2 (ours)	**41.66**	**66.98**	0.5392	0.2090	**25.47**	**19.88**	**28.79**	**55.86**	**68.14**	**0.50**	1.09
MGDA [Sener and Koltun, 2018]	30.55	60.20	0.5728	0.2179	**24.16**	**18.12**	**32.34**	**59.51**	**71.04**	1.63	2.88
PCGrad [Yu et al., 2020]	42.08	67.18	0.5098	0.1994	27.48	22.46	24.79	50.46	63.42	5.01	2.52
CAGrad [Liu et al., 2021a]	41.43	67.35	0.5042	0.1976	25.25	19.58	29.31	56.43	68.58	-1.31	2.85
IMTL-G [Liu et al., 2021b]	41.67	67.18	0.5019	0.1977	25.03	19.47	29.61	56.70	68.95	-1.76	2.85
Nash [Navon et al., 2022]	42.96	68.30	0.4966	0.1986	24.79	18.97	30.50	57.74	69.67	-3.39	3.16
IGBv1 + MGDA	40.93	66.66	0.5238	0.2049	24.38	18.44	31.52	58.97	70.66	-3.03	2.90
IGBv2 + MGDA	40.91	67.01	0.5359	0.2060	25.11	19.41	29.65	56.77	68.89	-0.54	2.96
IGBv1 + PCGrad	40.57	66.98	0.4989	0.1990	25.76	20.33	28.12	54.80	67.30	0.56	2.52
IGBv2 + PCGrad	36.00	63.04	0.5286	0.2079	26.00	20.41	28.11	54.65	66.96	3.66	2.60
IGBv1 + CAGrad	40.98	66.84	0.5050	0.1995	25.15	19.49	29.54	56.57	68.74	-1.23	2.86
IGBv2 + CAGrad	40.12	66.23	0.5190	0.2004	24.69	18.77	30.72	58.19	70.00	-2.16	2.93
IGBv1 + IMTL-G	42.01	67.69	0.5009	0.1995	24.97	19.21	30.01	57.20	69.28	-2.35	2.85
IGBv2 + IMTL-G	41.25	66.43	0.5134	0.2011	25.18	19.58	29.17	56.53	68.77	-0.81	2.94
IGBv1 + Nash	42.83	67.99	**0.4892**	**0.1958**	24.70	18.92	30.64	57.79	69.78	-3.71	3.17
IGBv2 + Nash	**43.97**	**68.53**	0.4928	0.1967	24.71	18.83	30.83	57.90	69.86	**-4.15**	3.23

4.2 RESULTS OF MULTI-TASK SCENE UNDERSTANDING ON NYUV2

Scene understanding on the NYUv2 dataset is the most common evaluation scenario in the MTL research, which contains three tasks: semantic segmentation, depth estimation, and surface normal prediction.

The comparative results are shown in Table 1. To verify the complementarity between IGB and gradient balancing, we combine our IGB algorithms with all compared gradient balancing algorithms. We categorize all methods into two groups according to efficiency and compare the performance in each group: one is the loss balancing algorithms; the other is the gradient balancing algorithms and also the combined algorithms by IGB and gradient balancing.

First, we focus on the overall performance and learning efficiency of MTL. **In terms of overall performance ($\boldsymbol{\Delta m}$):** Our IGB algorithms are the best in loss balancing and also competitive with some gradient balancing algorithms. Combining IGB and gradient balancing is better than using gradient balancing alone. In particular, IGBv1 + Nash and IGBv2 + Nash achieve better performance than Nash [Navon et al., 2022], yielding the SOTA performance of all algorithms. **In terms of learning efficiency ($\boldsymbol{T}$):** The training time of gradient balancing is about three times that of loss balancing. Either used alone or combined with gradient balancing, IGBv1 hardly leads to extra training time, and IGBv2 slightly leads to extra training time but can achieve better performance than IGBv1 in most situations.

Notably, with our IGB algorithms, users now have more freedom to take the trade-off between efficiency and performance in real-world MTL applications: our IGB algorithms could be chosen if efficiency is more important, and IGB + gradient balancing algorithms could be chosen if performance is more important. Specifically, IGBv1 is more efficient and simpler to implement, while IGBv2 achieves better performance in most situations.

Next, we analyze the superiority of our IGB algorithms through the specific task performance. Compared with STL, the performance of most existing algorithms is close to or even better on semantic segmentation and depth estimation, but significantly worse on surface normal prediction. To some extent, it means that in the NYUv2 MTL scenario, semantic segmentation and depth estimation are simpler, while surface normal prediction is more difficult. Such performance imbalance is undesirable in most MTL scenarios. As expected, our IGB algorithms can effectively alleviate this imbalance problem. That is, compared with other loss balancing algorithms, the performance of our IGBv1 and IGBv2 is competitive on semantic segmentation and depth

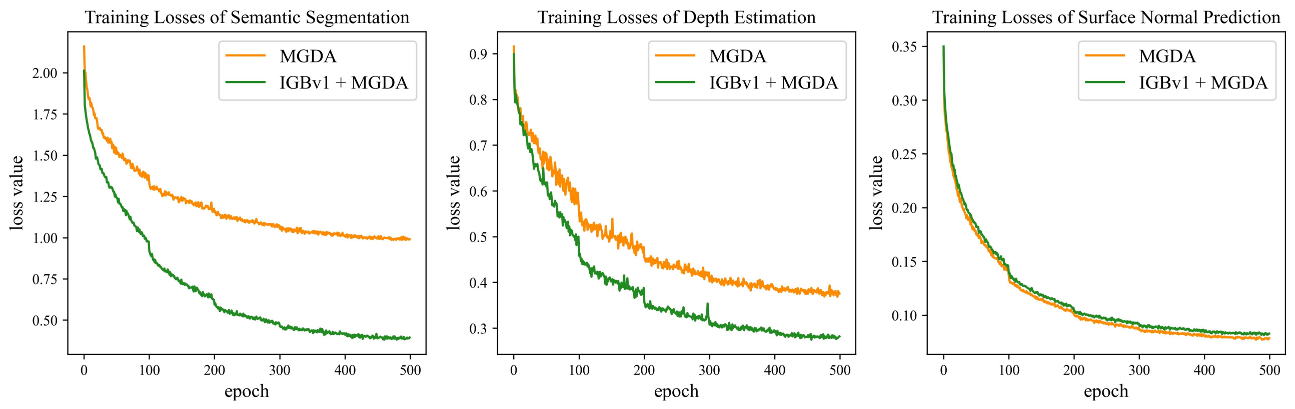

Figure 3: Comparison of loss decline curves for individual tasks of MGDA and IGBv1 + MGDA.

estimation, and significantly better on surface normal prediction, which is close to that of STL.

Moreover, the performance of MGDA [Sener and Koltun, 2018] is the best on surface normal prediction among all algorithms, but significantly worse than other algorithms on semantic segmentation and depth estimation, leading to its poor overall performance. By combining MGDA and our IGB algorithms, we significantly alleviate the performance imbalance problem and greatly improve its overall performance. As shown in Figure 3, when using IGBv1, the training losses of semantic segmentation and depth estimation decrease faster and are significantly lower. Meanwhile, for surface normal prediction, the training loss of IGBv1 + MGDA is comparable to that of MGDA during the training process. The main reason is that our improvable gap balancing algorithms can automatically focus on tasks that have not been properly trained.

4.3 RESULTS OF MULTI-TASK REGRESSION PREDICTION ON QM9

Predicting 11 properties of chemical molecules on the QM9 dataset is a more challenging MTL problem because of the larger task number, lower task relevance, and more varied task difficulty. Prior studies have found that the performance of STL on 11 property prediction is significantly better than that of MTL [Maron et al., 2019, Gasteiger et al., 2020], but driven by the need for higher learning efficiency and smaller model size, MTL methods are still worth studying.

The comparative results on the QM9 dataset are shown in Table 2. We can see that our IGB algorithms beat all compared algorithms except Nash [Navon et al., 2022] in performance, and perform much better than all gradient balancing algorithms in learning efficiency. For example, the training time of Nash is about seven times that of our IGB algorithms with close performance, and the training time of MGDA [Sener and Koltun, 2018] is twenty-two times that of our IGB algorithms due to aggregating the high-dimensional gradients by multi-objective optimization. As the number of tasks increases, the learning efficiency gap between loss balancing and gradient balancing becomes larger and larger. In a scenario like QM9, loss balancing (especially IGB) is thus more valuable. Unfortunately, the weighting or aggregating decisions made by existing loss balancing and gradient balancing algorithms are both unsatisfactory on this problem (much worse than STL). Combining loss balancing and gradient balancing is equivalent to combining the decisions made by the two types of algorithms, which has not yielded better performance.

4.4 ABLATION STUDY

As shown in Table 3, we analyze the contributions of improvable gap balancing in our algorithms and the effects of different settings in the IGBv2 algorithm on the NYUv2 dataset. The compared methods include: (1) EW which minimizes $\sum_{i=1}^{n} L_i$; (2) SI which minimizes $\sum_{i=1}^{n} \log(L_i)$ [Navon et al., 2022]; (3) IGBv1 (full); (4) IGBv2 (reward / min, buffer $1e4$) where the reward is calculated by the average loss decline instead of the minimum; (5) IGBv2 (reward / α, buffer $1e4$) where the reward is calculated without the multiplier factor α; (6) IGBv2 (full reward, buffer $5e3/1e4/5e4/1e6$) where the replay buffer size is set to $5e3$, $1e4$, $5e4$, or $1e6$, and IGBv2 (full reward, buffer $1e4$) is our full IGBv2.

We can observe from Table 3 that: (1) SI brings significant performance improvement compared with EW, indicating that the scale-invariant loss balancing paradigm is superior to the traditional loss balancing paradigm. (2) Our full IGBv1 and IGBv2 further improve the performance compared with SI, which clearly validates the effectiveness of improvable gap balancing. (3) IGBv2 (reward / min, buffer $1e4$), IGBv2 (reward / α, buffer $1e4$), and IGBv2 (full reward, buffer $5e3/5e4/1e6$) all produce performance degra-

Table 2: Comparative results on the QM9 dataset for multi-task regression prediction. $\uparrow$ ($\downarrow$) indicates that the higher (lower) the result, the better the performance. In each group, the best results are **bolded** and the second-best results are underlined.

Methods	μ	α	ϵ_{HOMO}	ϵ_{LUMO}	$\langle R^2 \rangle$	ZPVE	U_0	U	H	G	c_v	$\boldsymbol{\Delta m}\downarrow$	$\boldsymbol{T}\downarrow$
					MAE$\downarrow$								
STL	0.067	0.181	60.57	53.91	0.502	4.53	58.8	64.2	63.8	66.2	0.072		
EW	**0.106**	**0.325**	**73.57**	**89.67**	5.19	14.06	143.4	144.2	144.6	140.3	0.128	177.6	1.00
RLW [Lin et al., 2022]	0.113	0.340	76.95	92.76	5.86	15.46	156.3	157.1	157.6	153.0	0.137	203.8	1.00
DWA [Liu et al., 2019]	0.107	**0.325**	74.06	90.61	5.09	13.99	142.3	143.0	143.4	139.3	0.125	175.3	1.00
UW [Kendall et al., 2018]	0.386	0.425	166.2	155.8	1.06	4.99	66.40	66.78	66.80	66.24	0.122	108.0	1.02
IGBv1 (ours)	0.271	0.351	143.1	132.4	1.07	4.53	**56.74**	**57.13**	**57.16**	**56.73**	0.115	73.9	1.01
IGBv2 (ours)	0.251	0.333	149.1	130.2	**0.956**	**4.39**	56.75	57.19	57.25	**56.73**	**0.110**	**67.7**	1.16
MGDA [Sener and Koltun, 2018]	0.217	0.368	126.8	104.6	3.22	5.69	88.37	89.40	89.32	88.01	0.120	120.5	22.90
PCGrad [Yu et al., 2020]	0.106	0.293	**75.85**	88.33	3.94	9.15	116.4	116.8	117.2	114.5	0.110	125.7	5.85
CAGrad [Liu et al., 2021a]	0.118	0.321	83.51	94.81	3.21	6.93	114.0	114.3	114.5	112.3	0.116	112.8	4.97
IMTL-G [Liu et al., 2021b]	0.136	0.287	98.31	93.96	**1.75**	5.69	101.4	102.4	102.0	100.1	0.096	77.2	4.96
Nash [Navon et al., 2022]	**0.102**	**0.248**	82.95	**81.89**	2.42	**5.38**	**74.5**	**75.02**	**75.10**	**74.16**	**0.093**	**62.0**	7.20

Table 3: Results of ablation study on the NYUv2 dataset. 'reward / min' (or 'reward / α') denotes that min (or α) is removed from the definition of the full reward.

Methods	$\boldsymbol{\Delta m}\downarrow$	$\boldsymbol{T}\downarrow$
EW	8.45	1.00
SI [Navon et al., 2022]	2.36	1.00
IGBv1 (full)	1.76	1.00
IGBv2 (reward / min, buffer $1e4$)	5.97	1.09
IGBv2 (reward / α, buffer $1e4$)	0.91	1.09
IGBv2 (full reward, buffer $5e3$)	4.78	1.09
IGBv2 (full reward, buffer $1e4$)	**0.50**	1.09
IGBv2 (full reward, buffer $5e4$)	2.72	1.09
IGBv2 (full reward, buffer $1e6$)	3.48	1.06

Table 4: Results of loss balancing using the SI paradigm on the NYUv2 dataset. $+$ represents the combination of a loss balancing method and the SI paradigm.

Methods	$\boldsymbol{\Delta m}\downarrow$	$\boldsymbol{T}\downarrow$
SI [Navon et al., 2022]	2.36	1.00
RLW [Lin et al., 2022]	10.82	1.00
RLW + SI	6.24	1.00
DWA [Liu et al., 2019]	7.08	1.00
DWA + SI	3.25	1.00
UW [Kendall et al., 2018]	4.09	1.00
UW + SI	-	-
IGBv1	1.76	1.00
IGBv2	**0.50**	1.09

dation compared with the full IGBv2, indicating that our redesign of the reward and the replay buffer size is necessary when deploying DRL for MTL optimization.

Moreover, as shown in Table 4, we combine existing loss balancing methods with the SI paradigm (UW + SI method does not work since UW may not be compatible with SI), and their performance on NYUv2 is much lower than that of our IGB methods. It provides further evidence of the effectiveness of our IGB methods, i.e., the task weighting of IGB and the SI paradigm are complementary (combining them leads to better performance).

5 CONCLUSION

In this paper, we propose two novel loss balancing algorithms, IGBv1 and IGBv2. We dynamically assign task weights to balance the improvable gaps: IGBv1 takes a simple heuristic, and IGBv2 (for the first time) applies DRL to MTL. We analyze the complementarity between IGB and gradient balancing. Extensive experiments show that our IGB algorithms outperform all existing loss balancing algorithms and bring further performance improvements when combined with gradient balancing. More importantly, we reemphasize the significance of loss balancing in MTL in terms of learning efficiency and give users more freedom to take the trade-off between efficiency and performance in real-world applications.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (61976220), and the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China (23XNH026). Zhiwu Lu is the corresponding author.

References

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning:

A brief survey. *IEEE Signal Processing Magazine*, 34(6): 26–38, 2017.

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.

Rich Caruana. *Multitask learning*. Springer, 1998.

Xiaoliang Chen, Roberto Proietti, Che-Yu Liu, and SJ Ben Yoo. A multi-task-learning-based transfer deep reinforcement learning design for autonomic optical networks. *IEEE Journal on Selected Areas in Communications*, 39 (9):2878–2889, 2021.

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.

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.

Sauhaarda Chowdhuri, Tushar Pankaj, and Karl Zipser. Multinet: Multi-modal multi-task learning for autonomous driving. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1496–1504. IEEE, 2019.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *ArXiv Preprint ArXiv:1903.02428*, 2019.

Johannes Gasteiger, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *ArXiv Preprint ArXiv:2003.03123*, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. Dynamic task prioritization for multitask learning. In *European Conference on Computer Vision*, pages 270–287, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*, 2014.

Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M Pawan Kumar. In defense of the unitary scalarization for deep multi-task learning. *ArXiv Preprint ArXiv:2201.04122*, 2022.

Baijiong Lin, Feiyang Ye, and Yu Zhang. A closer look at loss weighting in multi-task learning. *ArXiv Preprint ArXiv:2111.10603*, 2021.

Baijiong Lin, YE Feiyang, Yu Zhang, and Ivor Tsang. Reasonable effectiveness of random weighting: A litmus test for multi-task learning. *Transactions on Machine Learning Research*, 2022.

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*, 34:18878–18890, 2021a.

Liyang Liu, Yi Li, Zhanghui Kuang, J Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. In *International Conference on Learning Representations*, 2021b.

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*, pages 1871–1880, 2019.

Shikun Liu, Stephen James, Andrew J Davison, and Edward Johns. Auto-lambda: Disentangling dynamic task relationships. *ArXiv Preprint ArXiv:2202.03091*, 2022.

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 Conference on Knowledge Discovery and Data Mining*, pages 1930–1939, 2018.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3994–4003, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *ArXiv Preprint ArXiv:1312.5602*, 2013.

Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. In *International Conference on Machine Learning*, pages 16428–16446. PMLR, 2022.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21 (4):682–697, 2008.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1(1):1–7, 2014.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *ArXiv Preprint ArXiv:1706.05098*, 2017.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. *European Conference on Computer Vision*, 7576:746–760, 2012.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. *Advances in Neural Information Processing Systems*, 26, 2013.

Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *ACM Conference on Recommender Systems*, pages 269–278, 2020.

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*, 44(7):3614–3633, 2021.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *ArXiv Preprint ArXiv:1511.06391*, 2015.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. King's College, Cambridge United Kingdom, 1989.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. *Advances in Neural Information Processing Systems*, 33:4767–4777, 2020.

Feiyang Ye, Baijiong Lin, Zhixiong Yue, Pengxin Guo, Qiao Xiao, and Yu Zhang. Multi-objective meta learning. *Advances in Neural Information Processing Systems*, 34: 21338–21351, 2021.

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.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.

Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. Recommending what video to watch next: a multitask ranking system. In *ACM Conference on Recommender Systems*, pages 43–51, 2019.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.