This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Deep Multitask Learning with Progressive Parameter Sharing

Haosen Shi[†]
The Chinese University of Hong Kong
hsshi23@cse.cuhk.edu.hk

Shen Ren
Continental Automotive Singapore
shen@shenren.org

Tianwei Zhang
Nanyang Technological University
tianwei.zhang@ntu.edu.sg

Sinno Jialin Pan
The Chinese University of Hong Kong
sinnopan@cuhk.edu.hk

Abstract

We propose a novel progressive parameter-sharing strategy (MPPS) in this paper for effectively training multitask learning models on diverse computer vision tasks simultaneously. Specifically, we propose to parameterize distributions for different tasks to control the sharings, based on the concept of Exclusive Capacity that we introduce. A scheduling mechanism following the concept of curriculum learning is also designed to progressively change the sharing strategy to increase the level of sharing during the learning process. We further propose a novel loss function to regularize the optimization of network parameters as well as the sharing probabilities of each neuron for each task. Our approach can be combined with many state-of-the-art multitask learning solutions to achieve better joint task performance. Comprehensive experiments show that it has competitive performance on three challenging datasets (Multi-CIFAR100, NYUv2, and Cityscapes) using various convolution neural network architectures.

1. Introduction

Performing multiple tasks at the same time is a fundamental ability for many intelligent agents. While the remarkable success of deep neural networks (DNNs) has been achieved in various computer vision applications such as image classification [10], semantic segmentation [38], depth estimation [31], surface normal estimation [69] and image generation [15, 21], learning and performing similar but distinctive tasks simultaneously are still a challenge for them. To address this issue, researchers proposed the Multi-Task Learning (MTL) paradigm, which usually trains one model to act as multiple distinct models. Each task in this scheme can benefit from reusing knowledge learned from others to improve its performance and reduce the model's learning time. Generally, the DNNs architecture for MTL is composed of shared parameters and task-specific parameters, which can strike a balance between shared and exclusive knowledge among different tasks [4].

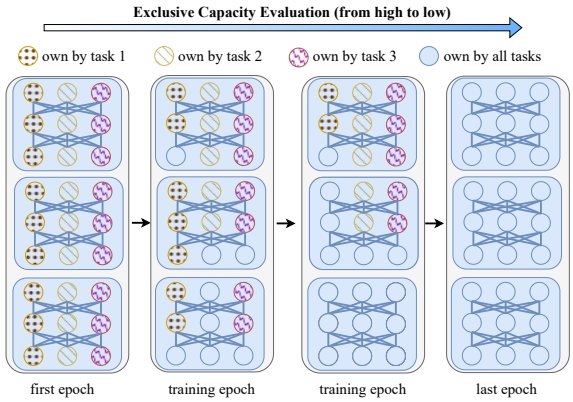

Figure 1. An illustrative diagram of proposed dynamic optimization in MPPS for MTL.

There are two problems that need to be addressed in designing effective MTL algorithms. The first one is the construction of the parameter-sharing scheme. The most common approach is hard-parameter sharing, which builds a shared feature extractor to map input data from all tasks into dense features in a common hidden representation space and then uses these features to generate different outputs via task-specific functions. Past works have demonstrated the effectiveness of this strategy in considerably reducing the model size and enhancing its overall performance across diverse settings [51]. However, it still suffers from two issues. 1) some task combinations may result in a notable performance reduction, which is known as the negative transfer [50, 61]; 2) an optimal design of the MTL models based on hard-parameter sharing still requires a high level of human expertise and rich domain-specific knowledge. To mit-

[†]This work was done when the author worked as a research assistant at Nanyang Technological University, Singapore.

igate these issues, various solutions have been introduced to complement the shared architecture [40, 37, 49]. Given the increasing complexity of DNNs (in terms of model size, design choices, and available search spaces), how to find the optimal scheme for MTL resource sharing and resource allocation is still an open problem.

The second problem in MTL is how to control the optimization process to achieve the best joint task performance. Recent works focus on balancing the task training speed via re-weighting the loss coefficients [7, 25, 36, 67], avoiding the conflict between gradients [78, 35, 8], and finding better local minima on the optimal Pareto front [65, 54]. These algorithms often clearly utilize the intuitive understanding of the training process to dynamically balance various tasks. However, some previous works [30, 73] showed that particular fixed, precisely searched loss weights can achieve the same or even better performance. This motivates us to rethink the relationship between the hard parameter-sharing design and the training dynamics optimization methods agnostic to DNNs architectures.

In this paper, we propose Multitask Learning with Progressive Parameter Sharing (MPPS), a novel progressive parameter-sharing strategy that incorporates the design of a parameter-sharing scheme with the optimization of training dynamics, aiming to achieve adaptive knowledge sharing among different tasks at different training stages. Fine-grained network architecture control is essential in this context to ensure an adequate neural network capacity for each task and support the complex mapping for all tasks with various relationships. To achieve that, we propose a dynamic resource allocation strategy at the neuron level by parameterizing the task-specific sharing probability of each neuron of the neural network, conditioning different tasks to be learned simultaneously. This design allows distinct distributions to be learned across different tasks across all neurons, potentially significantly enhancing the flexibility of deep MTL model parameter sharing to accommodate a wide range of task combinations.

However, the optimization is challenging due to the increased search space. Inspired by [1], we design a loss function to regularize the learning following an exclusive capacity scheduler, which progressively changes the sharing strategy from the highest exclusive capacity to a fully shared one during the entire training process. Figure 1 illustrates an example of our dynamic optimization. This design is inspired by the development of biological brains and dynamic functional brain connectivity. The structural modular segregation increases from 0 to 6 ages and the integration increases after then [71]. For the functional brain network, the integration increases along with higher cognitive workloads [45, 2, 56]. The biological analog and the MTL in practice also suggest that optimizing multiple single-task models may be easier than a multi-task model with shared parameters. This motivates us to design an exclusive capacity scheduler with the curriculum learning concept.

We perform extensive experiments over popular multi-task benchmarks to validate the superiority of MPPS. Evaluations show that our approach can bring substantial task performance improvement. We also perform detailed ablation studies to disclose the effectiveness and efficiency of our dynamic capacity controlling and curriculum scheduler.

We summarize our contributions as follows:

- We propose a novel MTL dynamic resource allocation strategy at the neuron level by parameterizing the task-specific sharing probability conditioning on tasks.
- We propose the "Exclusive Capacity" concept and design an Exclusive Capacity scheduler from the curriculum learning intuition.
- Our approach shows competitive results on image classification and dense prediction tasks.

2. Related Work

2.1. Multi-Task Learning

Multi-task architectures have gained significant interest due to their ability to facilitate effective information exchange between tasks and potentially avoid task conflicts [82, 66]. These architectures can be broadly categorized into two types: hard-parameter sharing [40, 37, 74, 19, 28] and soft-parameter sharing [42, 52, 14], based on the usage of shared parameters. Recent studies try to learn a good parameter partition scheme from various perspectives and technologies. We detail these strategies in Section 2.3. Our work aims to address the challenge of training shared parameters and make the developed training strategy applicable to various architectures, particularly those employing hard-parameter sharing. Conversely, methods that utilize fewer shared parameters exhibit less relevance to our work.
Multi-task optimization methods have been developed to minimize conflicts between tasks and leverage the task similarities via different techniques, such as task loss balancing or gradient correction. Previous studies have explored model-agnostic task weighting methods utilizing various properties, including task loss magnitudes [37], gradient norms [7], task uncertainty [25], meta-learning [36] and instance-level parameters [67]. However, these approaches are limited in their effectiveness and flexibility, because they linear re-weight the entire parameter update at the task level and fail to mitigate conflicts. In contrast, gradient modification methods aim to avoid gradient conflicts through techniques such as projecting the gradient onto the normal space [78], rotating gradient direction [23], gradient sign dropping [8], and others [70, 35, 44]. However, these gradient-based methods have a short-sighted view of

the optimization process. In contrast, our approach takes advantage of a long-sighted view by taking full advantage of progressive changes in architectures.

2.2. Curriculum Learning

The concept of curriculum learning (CL), originally introduced by [1], draws inspiration from the human learning process. Subsequently, numerous studies have successfully applied CL to board applications by selecting training data in an increasing level of complexity [11, 80, 81, 72, 83, 22, 79]. In addition to the widely used CL methods that adjust the scheduling of training data, several approaches employ CL through augmenting the model capacity [24, 18], reducing the penalty intensity of the regular term [43], and controlling high-level representation learning [59].

In the context of MTL, previous research has predominantly utilized this fundamental idea to design the task ordering and the data sampling processes [47, 32, 68, 53]. Different from them, our approach adopts a unique perspective that focuses on the scheduling of parameter partitions for different tasks to be learned simultaneously.

2.3. Parameter Partitioning

Enhancing the task performance of MTL models by learning to assign parameters is an attractive approach. Various works introduce new approaches to find the best task parameter partitions, based on the task similarity [17], task conflict [55], the trade-off between accuracy and efficiency [62], iterative pruning [41] and others [3, 13]. These works focus on resolving *where* to share parameters, while our method aims to resolve *how* to train parameters. The closest work to MPPS is [46], which changes the parameter partition and randomly varies the parameter participating to facilitate the training. However, the solution in [46] uses same dropout probability without progressive changes, which can pose challenges to its optimization process. More importantly, we follow a progressive scheduler, which facilitates the exchange of knowledge among tasks learned simultaneously. MPPS is also inspired by various dropout methods [60, 27, 12], which provide a theoretical framework and allow our method to work with many different distributions, and Neural Architecture Search (NAS) methods [62, 13]. MPPS differs from NAS in two aspects. 1) *Goal*: MPPS mainly focuses on training shared parameters and can be added to the NAS-based architecture, while most NAS works aim to search for the best static architectures. 2) *Motivation*: MPPS aims to compactly deploy multitask models on edge devices, maintaining inference latency and memory consumption similar to hard-parameter sharing, while NAS does not consider these requirements.

3. Methodology

3.1. Problem Statement

We aim to train an MTL model that can simultaneously perform T tasks. We use the notation $[T]$ to denote an enumerable set $\{1, 2, \ldots, T\}$. The input of the MTL model is an I-dimensional sample $x \in X \subset R^I$, and the output is $\mathbf{y} = (y_1, y_2, \ldots, y_T)$, which are the labels of T different tasks. Note that the dimensions of outputs from different tasks can be different. The training dataset include N data points $\mathcal{D} = \{(x, y_1, y_2, \ldots, y_T)_i | i \in [N]\} = \{(x, \mathbf{y})_i | i \in [N]\}$.

The MTL model consists of a shared network f_{θ_s} with parameters θ_s, and T task-specific networks $g_{\theta_1}, g_{\theta_2}, \ldots, g_{\theta_T}$ with parameters $\{\theta_i\}$. The entire parameters for this MTL model are $\theta = \theta_s \cup \left(\bigcup_{i \in [T]} \theta_i \right)$. We denote by $L_1, L_2, \ldots, L_T$ the loss functions of each task. The MTL problem can then be formulated as an optimization problem, which tries to find the parameters θ that can empirically minimize the following loss function:

$$\mathcal{L}(\theta) = \sum_{x, \mathbf{y} \in \mathcal{D}} \sum_{t \in [T]} L_t(g_{\theta_t} \circ f_{\theta_s}(x), y_t). \quad (1)$$

3.2. Overview

We introduce MPPS, a novel progressive parameter sharing strategy for MTL. The method is described based on the hard-parameter sharing MTL architecture. Instead of training all the parameters θ throughout an entire training process, we apply the concept of curriculum learning in MPPS to gradually train the neurons in different tasks according to different distributions, so that the optimization objectives of the MTL model are dynamically changed from easy to hard. In this way, the method may enhance the learning efficiency and task performance of MTL models by leveraging the knowledge learned from the easier optimization objectives to learn the harder ones.

The implementation of MPPS, however, still faces two challenges. The first is how to select and adjust the neuron activation for training in each step, and the second is how to implement a suitable initialization and scheduling of the curriculum among the tasks. We introduce several novel strategies to address the above challenges.

3.3. Exclusive Capacity

We first introduce the concept of Exclusive Capacity, which is defined as the ratio of the number of neurons used by only one single task over the number of all neurons used by multiple tasks in an MTL model. An allocation scheme with more exclusive neurons has a higher exclusive capacity, while the fully-shared scheme (hard-parameter sharing) has the lowest exclusive capacity.

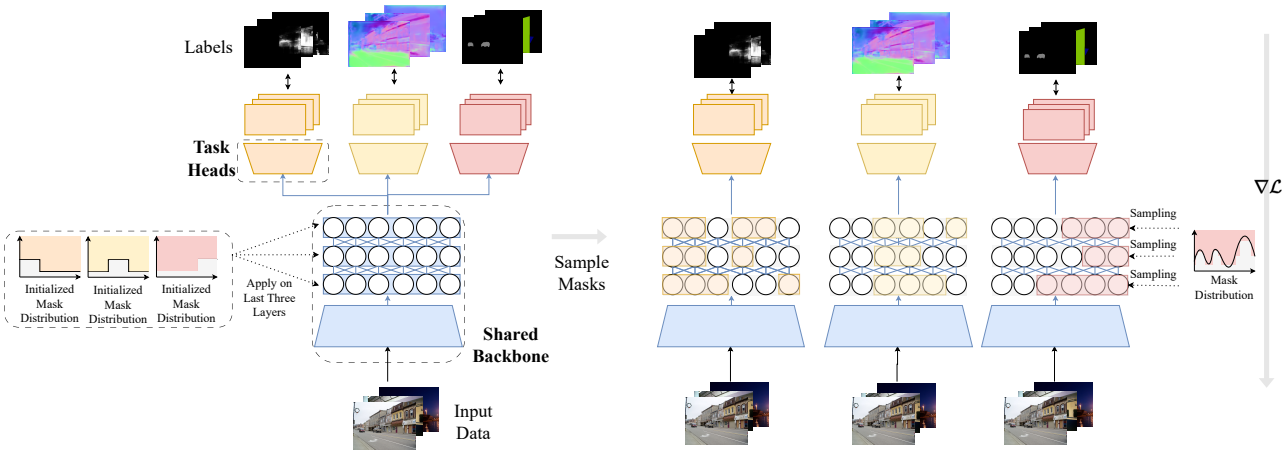

Figure 2. Workflow of MPPS

To facilitate the optimization process, it is necessary to formulate Exclusive Capacity as a continuous value over the training process (denoted by *EC*). To achieve this, we model *EC* via an expectation of resource allocation distribution. Based on a given MTL architecture, the resource allocation can be formulated as a mask m which represents the selected neurons for each task and is considered as a random variable sampled from a specific distribution. The new network after applying the mask is $f_{m\odot\theta}$, where $\odot$ denotes the element-wise product operation.

We formulate the mask distribution as a Multivariate Bernoulli distribution with learnable parameters $\Phi = \{\phi \in [0,1]^T\}$. Specifically, each neuron has a parameter vector ϕ to build a T-dimensional Multivariate Bernoulli distribution $\mathcal{B}(\phi)$. For each neuron, we assume task t with the highest value $\phi[t]$ is the owner of this neuron, while the rest $T-1$ tasks are its guests. The owner can always use this neuron for training, whereas the guests can only use it according to the sampled m from their mask distributions. Then *EC* of a single neuron can be defined as $1 - \frac{1}{T-1}(\sum_i \phi[i] - \max_i \phi[i])$. The *EC* of the mask distribution is calculated as the average of all neurons' *EC*.

Alternatively, we can also formulate the mask distribution via a Multivariate Gaussian distribution [27], where we define a translation function $r_G(v) = \sqrt{\frac{1-v}{v}}$ and parameterize the distribution as $\mathcal{N}(1, r^2(\phi))$. The calculation of *EC* for a single neuron and the mask distribution is the same as the Multivariate Bernoulli distribution. In the following, we use $\mathcal{P}(r(\phi))$ to represent both the Bernoulli distribution with $r_B(v) = v$ and the Gaussian distribution with r_G.

3.4. Optimization Objective

After formulating the parameterized mask distribution and *EC*, we can now specify the objective for scheduling and updating the mask distribution on the shared network to optimize the MTL performance. We first address a sub-problem (or a course in curriculum learning literature), which is the optimization of the neural network parameters and a parameterized mask distribution q to enhance the model's performance while maintaining a predetermined *EC* for the mask distribution. In order to solve this constrained optimization problem, we introduce a target distribution p which has the predetermined *EC*, and the loss function can be formulated as follows with two terms:

$$\begin{aligned}\mathcal{L}(\theta) \;=\; & \sum_{x,\mathbf{y}\in\mathcal{D}} \sum_{t\in[T]} \mathbb{E}_{m\sim q(\cdot|t)}[L_t(g^t_{\theta_t} \circ f_{\theta_s\odot m}(x), y_t)] \\ & + \mathbf{KL}(q(\cdot|t), p(\cdot|t)), \end{aligned} \tag{2}$$

where $\mathbf{KL}$ is the Kullback-Leibler (KL) divergence. We optimize this soft unconstrained objective to approximately solve the original constrained optimization problem.

From a variational inference view, we can view the original MTL objective as a log-likelihood function: $\log \prod_i^N p(\mathbf{y}_i|x_i)$. We can describe the objective with the mask distribution condition on task t as

$$\begin{aligned}
&\log \prod_i p(\mathbf{y}_i|x_i) = \sum_i \log \sum_t p(y_{t,i}|x_i,t)p(t) \\
&\geq \frac{1}{T}\sum_i \sum_t \log p(y_{t,i}|t,x_i) \\
&= \frac{1}{T}\sum_t \sum_i \log \int_z p(y_{t,i}|z,t,x_i)p(z|t)dz \\
&\geq \frac{1}{T}\sum_t \sum_i \int_z q(z|t) \log \frac{p(y_{t,i}|z,t,x_i)p(z|t)}{q(z|t)}dz \\
&= \frac{1}{T}\sum_t \sum_i \mathbb{E}_{z\sim q(\cdot|t)}[\log p(y_{t,i}|z,t,x_i)] \\
&\quad - \mathbf{KL}(q(\cdot|t), p(\cdot|t)).
\end{aligned}$$

Maximizing the above log-likelihood function is the same as minimizing (2) where $p(y_{t,i}|z,t,x_i)$ can be reformulated

by the loss function in both classification and regression problems.

For practical implementation, we use a parameterized distribution $\tilde{q}_\Phi(\cdot, t)$ to estimate the mask distribution $q(\cdot|t)$, and a parameterized distribution $\tilde{p}_\Pi(\cdot, t)$ to denote the target distribution $p(\cdot, t)$. Then the loss can be rewritten as

$$\mathcal{L}(\theta, \Phi, \Pi) = \sum_{x,\mathbf{y}\in\mathcal{D}} \sum_{t\in[T]} \mathbb{E}_{m\sim\tilde{q}_\Phi(\cdot,t)}[L_t(g^t_{\theta_t} \circ f_{\theta_s \odot m}(x), y_t) + \log \tilde{q}_\Phi(m, t) - \log \tilde{p}_\Pi(m, t)]. \quad (3)$$

We can use stochastic gradient descent to optimize the above loss function by both updating the network parameters θ and mask distribution parameters Φ. It is easy to update θ and calculate its gradient, but updating Φ is more difficult. When using the Gaussian distribution for q_Φ, we adopt the reparameterization trick [27] to effectively estimate the gradient: for $m \sim \tilde{q}_\Phi(\cdot, t)$, we rewrite the samples as $m = 1 + r(\Phi[t])\epsilon_r$, where $\epsilon_r \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

For the Bernoulli distribution, we adopt the REINFORCE [63], which uses the Monte Carlo method to estimate gradients: for sample $(x, \mathbf{y})$ we use

$$\nabla_\Phi \mathcal{L} = \sum_{t\in[T]} \mathbb{E}_{m\sim\tilde{q}_\Phi(\cdot,t)}[L_t(\theta \odot m, x, y_t)\nabla_\Phi \log \tilde{q}_\Phi(m, t)]$$

to estimate the first term of (3) and use an explicit analytical form to estimate the **KL** term. Note that there are many improved REINFORCE estimators [75, 64, 16], which can possibly improve the learning performance. We leave the investigations of these methods to future works.

3.5. Curriculum Learning

With the above loss as the objective (3), `MPPS` adopts the concept of curriculum learning to adjust *EC* and learn the parameters progressively. We have discussed how to update the parameters and mask distribution in each course. However, a couple of key problems remain unsolved: 1) how to initialize the parameters to fit the easiest course requirement; 2) how to build the *EC* scheduler; 3) how to update the target distribution for the next course.

Exclusive initialization. We first present the approach to initializing the entire curriculum learning process. In the beginning, there is no prior knowledge about all the tasks at all. We can initialize the target distribution p_Π to evenly assign all neurons to each task layer by layer. Also, the mask distribution q_Φ is set equal to p_Π. More specifically, we divide the neurons in each layer of the shared backbone network into T parts and allocate each part to a task to be the owner. We achieve this by setting the parameter of the task owner in ϕ as $1 - \epsilon$, while the rest parameters as ϵ, where ϵ is a small positive value for numerical stability. Such assignment can guarantee both distributions q_Φ and p_Π have the highest *EC* during initialization.

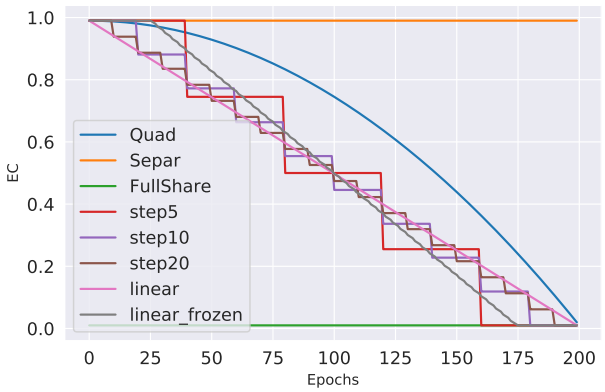

Figure 3. EC schedulers.

EC scheduler. In Figure 3, we present a compilation of various schedulers that can be categorized into three distinct groups. The first group entails schedulers that do not modify the *EC*. The second group divides the training process evenly into N courses, with each course being of identical length. The *EC* is reduced at the end of each course, and if N is equivalent to the total training epochs, this is called a Linear scheduler; otherwise, it is called a StepN scheduler. Finally, the third category employs a quadratic function to schedule the *EC*. In order to stabilize the entire training process, we increase the length of the first course and the last course respectively for better initialization and final convergence. This added length, named frozen epochs, will be discussed by experiments in Section 4.3.2. Our default scheduling approach is the Linear scheduler with frozen epochs. Our evaluations in Section 4.3.3 show that this Linear scheduler can achieve satisfactory performance.

Updating target distribution. To update the target distribution parameters Π, we reuse the loss term in (3) and build a constrained optimization problem as follows:

$$\begin{aligned} \min_{\Pi} \quad & \sum_t \mathbf{KL}(\tilde{q}_\Phi(\cdot, t), \tilde{p}_\Pi(\cdot, t)) \\ \text{s.t.} \quad & \sum_t \Pi[t] - \max_t \Pi[t] = 1 - C_i \end{aligned} \quad (4)$$

where the updated Φ is used here.

We propose a heuristic approach to updating the target distribution. We directly normalize and rescale the remaining capacity of the updated Φ to build the target distribution p_Π in the next course, as the updated Φ can greatly show the task similarity. The basic idea behind this is that if tasks A and B are more similar, the updated mask distribution q_Φ will allocate more capacity to B on the neurons whose owner is A, especially under such a limited total capacity situation, and vice visa. So using a distribution proportional to the updated $\tilde{q}_\Phi(\cdot, t)$ as the target distribution for the next course is a good choice. A more detailed analysis can be

found in Section 4.3.4

Combining all the above techniques, Algorithm 1 shows the detailed MTL training process of MPPS.

Algorithm 1 MPPS training

Require: initialized parameters: θ, ϕ, Π
  for i = $1, 2, \ldots, n_{epochs}$ **do**
    for $x_i, \{y\}_i \in \mathcal{D}$ **do**
      for $t \in [T]$ **do**
        Sampling $m \sim q_\phi(t)$
        Calculate loss $\mathcal{L}_t$ according to Eq.(3)
      end for
      Updating θ, ϕ according to the optimizer
    end for
    Get C_i from *EC* Scheduler
    $C \leftarrow 1 - (\sum_t \Phi[t] - \max_t \Phi[t])$
    $\Pi \leftarrow 1 - \frac{C_i}{C}(1 - \Phi)$
  end for

4. Experiments

We perform comprehensive experiments and ablation studies to demonstrate the superiority of MPPS.

4.1. Configurations and Implementations.

Datasets. We use three widely-used public datasets:

NYUv2 [57]. This is a challenging dataset including 1,449 indoor images recorded over 464 different scenes from Microsoft Kinect cameras. It provides three tasks, including a 13-classes semantic segmentation (SemSeg) task, a depth estimation (Depth) task, and a surface normal (Normal) estimation task. We use the pre-processed data provided by [37].

Cityscapes [9]. This dataset contains 5,000 annotated street-view images with pixel-level annotations from a car point of view. We select three tasks, including 10-class semantic segmentation (SemSeg), disparity (inverse depth, Disp) estimation, and 10-class part segmentation (PartSeg). Similarly, we use the pre-processed data provided by [37].

Multi-CIFAR100 [29]. This is a widely used image classification dataset that contains 60,000 32×32 color images in 100 different classes. This dataset is split into 20 subtasks to build a multi-task learning dataset following [36]. While this dataset is not a multi-objective multi-task learning problem that we use to derive the optimization objective (3), it should be noted that MPPS does not necessarily require a jointly labeled dataset. To facilitate comprehension, we provide concise explanations in the supplementary material.

Implementation. Based on the observation [48, 76] that higher layers of a neural network tend to learn task-specific features, we hypothesize that the challenge of optimizing shared parameters is primarily concentrated on these higher layers of the shared backbone. Therefore, we opt to implement our approach solely for the last few layers, rather than the entire network backbone, in order to minimize the overall search space. A more detailed discussion about how many layers should be selected is given in Section 4.3.1. In our experiments, we build MPPS on 2D Convolutional layers of ResNet [20] models, as well as 2D Convolutional and Pooling layers of VGG [58] models.

In the training phase, We initially sample a mask from the distribution $\mathcal{P}_\Phi$ for each task. Then we perform multiple forward propagations for each sampled shared backbone and task-specific head to produce the ultimate outputs. Following this, we calculate the loss using Eq.(3) and subsequently update the weights with the standard backpropagation. The evaluation stage of MPPS requires special considerations due to the presence of stochastic networks. Instead of using mask sampling for evaluation, we use the full shared backbone. This may potentially create an inconsistency between the training and evaluation phases, which could potentially compromise the final performance. However, we adopt this evaluation scheme for two reasons: 1) MPPS aims to optimize the parameter values rather than determining the shared parameter allocation; 2) using the complete shared backbone does not require additional inference time compared to the original model.

Without additional instructions regarding the hyperparameters introduced by our method, we use $\epsilon = 0.01$ for numerical stability, adopt the last 6 layers to build distributions, and enable the target distribution updating method for all situations. We use a Linear scheduler with 25 frozen epochs and two parameterized distributions for our main evaluations on NYUv2 and Cityscapes. For Multi-CIFAR100, we adopt a Linear scheduler with 15 frozen epochs and a parameterized Bernoulli distribution.

Model architectures. We use the Deeplabv3+ architecture [6] for NYUv2 and Cityscapes datasets. More specifically, we use a ResNet-50 network with dilated convolutions [77] as the shared backbone, and the Atrous Spatial Pyramid Pooling (ASPP) [5] module as the task-specific head for each task. We follow the optimization configurations from LibMTL [34], which uses an Adam optimizer [26] with the learning rate of 10^{-5} to train 200 epochs. The learning rate decreases by half in every 100 epochs. For Multi-CIFAR100, we use VGG16 as the shared backbone. A linear layer is used as the task-specific head. A stochastic gradient descent optimizer with momentum and a Cosine learning rate decay scheduler [39] are adopted to train Multi-CIFAR100 for 200 epochs. All experiments are trained from scratch on a single Nvidia DGX A100 GPU for all the methods included in this paper for a fair comparison.

Metrics. We evaluate the optimization method using the

relative performance improvement metric as follows:

$$\Delta(\theta) = \frac{1}{T} \sum_{t \in [T]} C_t \frac{P_t - P_{t,base}}{P_{t,base}}, \tag{5}$$

where P_t is the metric of task t under the given parameters θ, $P_{t,base}$ is the metric of task t following the same architecture but trained on the single-task learning paradigm*. C_t indicates whether a lower value ($C_t = -1$) or higher vaue ($C_t = 1$) is better.

4.2. Main Results

We run experiments to compare MPPS with the hard-parameter sharing multitask baseline on NYUv2 and Cityscapes. The results are shown in Table 1 and Table 2, respectively. We denote the performance of the single-task learning as *Single*, hard-parameter sharing baseline as *Multi*, MPPS with a parameterized Multivariate Bernoulli/Gaussian distribution as *MPPS +B* / *MPPS +G*. For each task, we record two metrics: the loss value used in training (cross entropy (CE) for SemSeg and PartSeg, L1 loss (L1) for Depth, Normal, and Disp) and a human-friendly evaluation metric (mean intersection over union (mIoU) for SemSeg and PartSeg, absolute error (AE) for Depth and Disp, and mean angle distance (MAD) for Normal). We also give an overall metric Δ. The arrows $\downarrow, \uparrow$ are added to the tables to denote we prefer a higher value ($\uparrow$) or lower value ($\downarrow$) for this metric. On both two datasets, we report the best performance on validation sets over the last 20 epochs for multitask setting and the last epoch performance for the single-task baseline.

We observe that both MPPS with Bernoulli and Gaussian distributions show improved results (higher Δ). In Table 1, MPPS performs better on Normal which gives worse performance than the single-task situation caused by the optimization of other tasks. This shows that standard multi-task optimization is susceptible to negative transfer, whereas MPPS can help. We conclude that MPPS **improves the MTL performance on two dense prediction task datasets.**

Method	**SemSeg**		**Depth**		**Normal**		$\Delta\uparrow$
Metric	CE$\downarrow$	mIoU$\uparrow$	L1$\downarrow$	AE$\downarrow$	L1$\downarrow$	MAD$\downarrow$	
Single	-	0.3768	-	0.5323	-	25.0227	+0.0%
Multi	1.291	0.407	0.469	0.469	0.651	26.384	+4.8%
MPPS +B	1.277	0.405	0.470	0.470	0.629	25.533	+5.7%
MPPS +G	1.188	0.417	0.466	0.466	0.640	25.965	+6.5%

Table 1. Comparison results on the NYUv2 dataset.

We further run experiments to compare MPPS + Multivariate Bernoulli distribution with the hard-parameter sharing baseline on Multi-CIFAR100 to show the superiority of MPPS in the non-jointly labeled dataset. We report the performance with the lowest, median, and average accuracy across all 20 domains in Table 3. The results show

*For Multi-CIFAR100, we use the multi-task learning as the baseline to avoid training 20 single-task learning models

Method	**SemSeg**		**PartSeg**		**Disp**		$\Delta\uparrow$
Metric	CE$\downarrow$	mIoU$\uparrow$	CE$\downarrow$	mIoU$\uparrow$	L1$\downarrow$	AE$\downarrow$	
Single	-	0.5620	-	0.5274	-	0.84	+0.0%
Multi	0.260	0.553	0.079	0.519	0.797	0.797	+0.7%
MPPS +B	0.244	0.571	0.074	0.522	0.800	0.800	+1.7%
MPPS +G	0.244	0.568	0.073	0.524	0.810	0.810	+1.4%

Table 2. Comparison results on the Cityscapes dataset.

the generality of MPPS on both jointly labeled and non-jointly labeled datasets. We conclude that MPPS **improves the performance on non-jointly labeled image classification datasets.**

Method	**Average Acc**$\uparrow$	**Lowest Acc**$\uparrow$	**Median Acc**$\uparrow$	$\Delta\uparrow$
Multi	59.23%	37.22%	60.52%	+0.0%
MPPS	67.62%	41.38%	71.06%	+14.4%

Table 3. Performance of 20 tasks on the Multi-CIFAR100 dataset. Results are averaged over four different random seeds.

4.3. Ablation Study

4.3.1 Network Layers

As mentioned above, considering the difficulty of optimization and based on the selected priors, we only select the last few layers of the shared backbone network to build the *EC* distribution. Table 4 shows the learning performance with a different number of layers. We can see that when the distribution is modeled on too many layers, despite using the *EC* scheduler, it is still difficult to optimize in such a large search space and can lead to performance degradation given the same number of optimization steps. The best-performed results are trained using 6 layers, so we set the number of layers as 6 by default in all other experiments.

Layers	**SemSeg**		**Depth**		**Normal**		$\Delta\uparrow$
Metric	CE$\downarrow$	mIoU$\uparrow$	CE$\downarrow$	mIoU$\uparrow$	L1$\downarrow$	AE$\downarrow$	
6	1.277	0.405	0.470	0.470	0.629	25.533	+5.7%
12	1.133	0.399	0.481	0.481	0.630	25.552	+4.5%
18	1.172	0.398	0.484	0.484	0.628	25.544	+4.2%
All	1.240	0.311	0.552	0.552	0.696	28.322	-11.4%

Table 4. Applying MPPS to different numbers of selected layers on the NYUv2 dataset.

4.3.2 Frozen Epochs

In order to demonstrate the advantages of frozen epochs, we conduct an ablation study by varying the number of frozen epochs (0, 5, 15, 25, 50, 100) on the Linear scheduler. As shown in Figure 4, freezing (15, 25) epochs is beneficial for stable training. More frozen epochs (50, 100) result in worse performance compared to training without freezing. We hypothesize that fewer epochs of scheduling lead to a more abrupt scheduler, resulting in insufficient training. Therefore, a trade-off must be made between ensuring a more gradually changing scheduler and maintaining training stability at the beginning and the end of the training process.

Method	SemSeg		Depth		Normal		Δ↑
Metric	CE↓	mIoU↑	CE↓	mIoU↑	L1↓	AE↓	
Single	-	0.377	-	0.532	-	25.023	0.0%
DWA	1.301	0.401	0.477	0.477	0.655	26.609	+3.5%
Random	1.174	0.416	0.457	0.457	0.635	25.725	**+7.2%**
Uncert	1.334	0.407	0.470	0.470	0.653	26.463	**+4.7%**
Auto-λ	1.314	0.414	0.462	0.462	0.634	25.681	**+6.8%**
HPS	1.291	0.407	0.469	0.469	0.651	26.384	+4.8%
Graddrop	1.335	0.400	0.469	0.469	0.652	26.416	+4.1%
PCGrad	1.309	0.406	0.471	0.471	0.641	25.955	**+5.2%**
Cagrad	1.291	0.406	0.457	0.457	0.629	25.492	+6.7%
MTAN	1.335	0.430	0.449	0.449	0.612	24.751	+10.3%

Method	SemSeg		Depth		Normal		Δ↑
Metric	CE↓	mIoU↑	CE↓	mIoU↑	L1↓	AE↓	
Single$*$	-	0.377	-	0.532	-	25.023	0.0%
DWA$*$	1.254	0.407	0.466	0.466	0.636	25.779	**+5.9%**
Random$*$	1.152	0.408	0.471	0.471	0.635	25.792	+5.5%
Uncert$*$	1.299	0.398	0.471	0.471	0.642	26.050	+4.3%
Auto-$\lambda*$	1.339	0.411	0.459	0.459	0.634	25.657	**+6.8%**
HPS$*$	1.277	0.405	0.470	0.470	0.629	25.533	**+5.7%**
Graddrop$*$	1.296	0.402	0.469	0.469	0.643	26.149	**+4.7%**
PCGrad$*$	1.240	0.403	0.474	0.474	0.640	25.931	+4.8%
Cagrad$*$	1.256	0.409	0.453	0.453	0.613	24.882	**+7.9%**
MTAN$*$	1.261	0.431	0.450	0.450	0.601	24.307	**+10.9%**

Table 5. Integrating MPPS with several state-of-the-art MTL methods. The left table shows the performance without adding MPPS and the right table shows the results after applying MPPS. The methods combined with our proposed MPPS are indicated by an added $*$. The better results in these two situations are highlighted in bold.

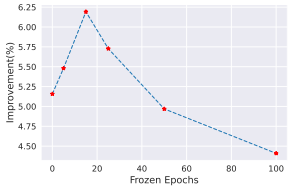

Figure 4. MPPS with different frozen epochs on NYUv2.

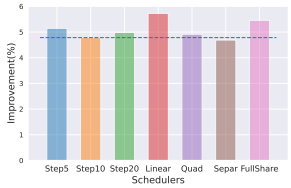

Figure 5. MPPS with different schedulers on NYUv2

4.3.3 Curriculum Scheduler

As mentioned in Section 3.5 and shown in Figure 3, we examine various *EC* schedulers, including fixed target distributions in different *EC* (Separ with the highest *EC* and FullShare with the lowest *EC*) to highlight the importance of the progressive process, as well as step functions (Step5, Step10, Step20, and Linear) and quadratic scheduling (Quad). Figure 5 shows the comparison results on the NYUv2 dataset. Almost every scheduler we test performs better than the baseline (blue dot line) except Separ because the neural network is encouraged to separate the masks almost during the entire training process, which results in less benefit from the knowledge of the related tasks.

4.3.4 Target Distribution Updating

We compare the results when the target distribution is updated in a linear manner (*without updating*) or adaptively updated following our target distribution updating method (*with updating*) in Table 6. We observe the target distribution updating mechanism can help MPPS transfer knowledge from previous courses to get better performance. Furthermore, we believe this target distribution variation process partially reflects the task similarity in the current training stage. To give a clearer view, Figure 6 illustrates this phenomenon by recording the target distribution updating on NYUv2 with a 25-frozen Linear scheduler (see supplementary material for a more complete figure). It shows the proportion of $\Phi_2/\ (\Phi_2 + \Phi_3)$ on nodes owned by task 1 for each layer. A higher value in these lines indicates a large proportion of Task 2, and the current network is more preferred to share Task 1's nodes with Task 2 than Task 3. Note that the proportion is not equal to 0.5 at the end of the training because we use a soft constraint in (3). This reveals the potential discrepancy between the learned architecture of the training process and the fully shared backbone.

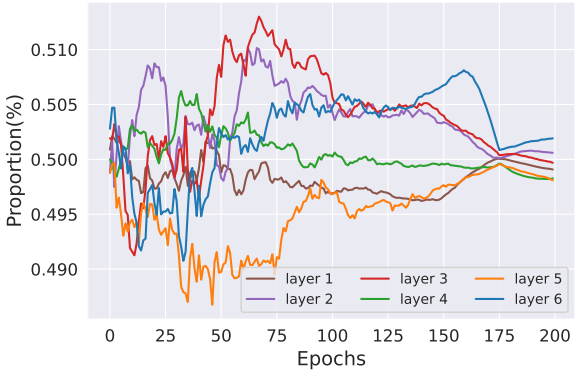

Figure 6. The proportion of task Depth relative to task Normal on neurons owned by task SemSeg.

Method	SemSeg		Depth		Normal		Δ↑
Metric	CE↓	mIoU↑	CE↓	mIoU↑	L1↓	AE↓	
w/ updating	1.277	0.405	0.470	0.470	0.629	25.533	+5.7%
w/o updating	1.250	0.401	0.475	0.475	0.635	25.731	+4.8%

Table 6. MPPS with and without target distribution updating

4.3.5 Integration with SOTA Methods

Our approach operates orthogonally from many existing MTL methods, enhancing its potential application across diverse real-world scenarios. We select several well-known MTL methods from three categories: 1) task-weighted methods, including Uncertainty [25], Random [33], DWA [37] and Auto-lambda [36]; 2) avoiding gradient conflict methods, including PCGrad [78], Cagrad [35], and Graddrop [8]; 3) multitask architecture designing methods, including hard-parameter sharing (HPS) and MTAN [37]. We integrate MPPS with each of the aforementioned methods to demonstrate the broad applicability of our approach.

As demonstrated in Table 5, our MPPS, exhibits a strong ability to complement most (6/9) state-of-the-art techniques, resulting in superior performance. This outcome serves as compelling evidence of the generalizability of MPPS. There are also some methods (i.e., Random and Uncert) that MPPS decreases slightly the performance. The potential reason may be that MPPS has a dynamic optimization objective instead of a static one, leading to increased uncertainty during the training process. Both Random and Uncert introduce a wide range of task weights to the multi-task learning objective, resulting in increased optimization difficulty. MPPS may require more training iterations or a slower-changing scheduler to potentially increase the performance of these methods.

4.3.6 Architecture Robustness

Finally, we demonstrate the effect of MPPS on different scales of neural networks to test its robustness. We choose to experiment on ResNet-18 and ResNet-34 as shared backbone networks. Table 7 show the comparison results, which demonstrate that MPPS has strong robustness.

Method	SemSeg		Depth		Normal		Δ↑
Metric	CE↓	mIoU↑	CE↓	mIoU↑	L1↓	AE↓	
ResNet18							
Single	-	0.3714	-	0.5230	-	25.0128	+0.0%
Multi	1.283	0.389	0.472	0.472	0.655	26.589	+2.7%
MPPS +B	1.249	0.387	0.472	0.472	0.629	25.529	+3.9%
MPPS +G	1.172	0.395	0.477	0.477	0.632	25.686	+4.2%
ResNet34							
Single	-	0.387	-	0.524	-	24.48	+0.0%
Multi	1.253	0.414	0.446	0.446	0.633	25.638	+5.7%
MPPS +B	1.132	0.419	0.452	0.452	0.611	24.812	+6.9%
MPPS +G	1.130	0.421	0.453	0.453	0.623	25.287	+6.4%

Table 7. MPPS with different architectures on NYUv2.

4.4. Time Complexity Analysis

There are around $T\times$ more training time or $T\times$ of memory cost when applying MPPS compared with using the standard hard-parameter sharing scheme on jointly labeled datasets. During the training stage, each task has a sampled different network architecture and needs standalone forward and backward propagation, which causes T times of training time and memory compared with the fully dense shared backbone, due to the lack of performance optimization for sparse neural network computation. It is worth noting that MPPS does not incur $T\times$ training time and memory consumption simultaneously. When sequentially sampling the mask and calculating the loss, it only adds an additional $(T-1)\times$ forward pass and backward pass through the backbone and $O(1)$ additional memory usage. When T masks are sampled parallel, the current computing platforms(like GPU) usually lead to a sublinear increase in time consumption. We build MPPS based on the forward hook provided by Pytorch on the whole shared backbone. Since we do establish MPPS for the last several layers, a better-optimized implementation could potentially save a lot of training time and GPU memory.

We perform additional experiments for efficient analysis with ResNet50 backbone: the time and memory costs of MPPS are less than $1.4\times$ and $1.3\times$ respectively compared with hard-parameter sharing(HPS). Note that MPPS with a different scheduler requires nearly the same training time and memory. MPPS also has significantly less time and memory consumption than many SOTA MTL methods discussed above. We compare one-epoch training time for various typical methods with/without MPPS, including gradient-based methods, meta-learning-based methods, and task-weighted methods in Table 8. We observe the timing cost introduced by MPPS is satisfactory.

Method	w/o MPPS	w MPPS(s)
HPS	28.69	39.34
PCGrad	41.03	41.42
Auto-λ	84.19	131.74
DWA	32.39	42.13

Table 8. Average training time (in seconds) on NYUv2 with ResNet50 backbone using 1 Nvidia A100 GPU.

5. Conclusion

In this paper, we propose MPPS, a novel methodology for training MTL models effectively. MPPS designs parameterized masks and target distributions. It also introduces a carefully-designed *EC* scheduler to progressively control parameter sharing among tasks during the training process, inspired by curriculum learning. Experiments demonstrate that our MPPS can achieve competitive performance on three challenging datasets using various neural network architectures. We also show that MPPS can be combined with current advanced methods to achieve even better results, without introducing extra parameters during inference. This makes it widely applicable to practical applications.

6. Acknowledgment

This study is supported under the RIE2020 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s). Sinno J. Pan thanks the support from HK Global STEM Professorship and the JC STEM Lab of Machine Learning and Symbolic Reasoning.

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 2, 3

[2] Urs Braun, Axel Schäfer, Henrik Walter, Susanne Erk, Nina Romanczuk-Seiferth, Leila Haddad, Janina I Schweiger, Oliver Grimm, Andreas Heinz, Heike Tost, et al. Dynamic reconfiguration of frontal brain networks during executive cognition in humans. *Proceedings of the National Academy of Sciences*, 112(37):11678–11683, 2015. 2

[3] David Bruggemann, Menelaos Kanakis, Stamatios Georgoulis, and Luc Van Gool. Automated search for resource-efficient branched multi-task networks. *arXiv preprint arXiv:2008.10292*, 2020. 3

[4] R Caruana. Multitask learning: A knowledge-based source of inductive bias1. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Citeseer, 1993. 1

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017. 6

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. 6

[7] 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. 2

[8] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. *Advances in Neural Information Processing Systems*, 33:2039–2050, 2020. 2, 8

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016. 6

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1

[11] Theresa Eimer, André Biedenkapp, Frank Hutter, and Marius Lindauer. Self-paced context evaluation for contextual reinforcement learning. In *International Conference on Machine Learning*, pages 2948–2958. PMLR, 2021. 3

[12] Xinjie Fan, Shujian Zhang, Korawat Tanwisuth, Xiaoning Qian, and Mingyuan Zhou. Contextual dropout: An efficient sample-dependent dropout module. *arXiv preprint arXiv:2103.04181*, 2021. 3

[13] Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11543–11552, 2020. 3

[14] Yuan Gao, Jiayi Ma, Mingbo Zhao, Wei Liu, and Alan L Yuille. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3205–3214, 2019. 2

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 1

[16] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoffrey Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv preprint arXiv:1711.00123*, 2017. 5

[17] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *International Conference on Machine Learning*, pages 3854–3863. PMLR, 2020. 3

[18] Yong Guo, Yaofo Chen, Yin Zheng, Peilin Zhao, Jian Chen, Junzhou Huang, and Mingkui Tan. Breaking the curse of space explosion: Towards efficient nas with curriculum search. In *International Conference on Machine Learning*, pages 3822–3831. PMLR, 2020. 3

[19] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 2

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[21] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 1

[22] Yuge Huang, Yuhan Wang, Ying Tai, Xiaoming Liu, Pengcheng Shen, Shaoxin Li, Jilin Li, and Feiyue Huang. Curricularface: adaptive curriculum learning loss for deep face recognition. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5901–5910, 2020. 3

[23] Adrián Javaloy and Isabel Valera. Rotograd: Gradient homogenization in multitask learning. *arXiv preprint arXiv:2103.02631*, 2021. 2

[24] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 3

[25] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018. 2, 8

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[27] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015. 3, 4, 5

[28] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of*

the IEEE conference on computer vision and pattern recognition, pages 6129–6138, 2017. 2

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[30] 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. 2

[31] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016. 1

[32] Changsheng Li, Junchi Yan, Fan Wei, Weishan Dong, Qingshan Liu, and Hongyuan Zha. Self-paced multi-task learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017. 3

[33] 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. 8

[34] Baijiong Lin and Yu Zhang. LibMTL: A python library for multi-task learning. *arXiv preprint arXiv:2203.14338*, 2022. 6

[35] 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, 2021. 2, 8

[36] Shikun Liu, Stephen James, Andrew J Davison, and Edward Johns. Auto-lambda: Disentangling dynamic task relationships. *arXiv preprint arXiv:2202.03091*, 2022. 2, 6, 8

[37] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019. 2, 6, 8

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 1

[39] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 6

[40] 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 *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1930–1939, 2018. 2

[41] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. 3

[42] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016. 2

[43] Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, René Vidal, and Vittorio Murino. Curriculum dropout. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3544–3552, 2017. 3

[44] Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. *arXiv preprint arXiv:2202.01017*, 2022. 2

[45] Carlo Nicolini and Angelo Bifone. Modular structure of brain functional networks: breaking the resolution limit by surprise. *Scientific reports*, 6(1):1–13, 2016. 2

[46] Lucas Pascal, Pietro Michiardi, Xavier Bost, Benoit Huet, and Maria A Zuluaga. Maximum roaming multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9331–9341, 2021. 3

[47] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. Curriculum learning of multiple tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5492–5500, 2015. 3

[48] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. *Advances in neural information processing systems*, 32, 2019. 6

[49] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017. 2

[50] Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, volume 898, 2005. 1

[51] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017. 1

[52] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 2

[53] Nikolaos Sarafianos, Theodore Giannakopoulos, Christophoros Nikou, and Ioannis A Kakadiaris. Curriculum learning for multi-task classification of visual attributes. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2608–2615, 2017. 3

[54] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018. 2

[55] Guangyuan SHI, Qimai Li, Wenlong Zhang, Jiaxin Chen, and Xiao-Ming Wu. Recon: Reducing conflicting gradients from the root for multi-task learning. In *The Eleventh International Conference on Learning Representations*, 2023. 3

[56] James M Shine, Patrick G Bissett, Peter T Bell, Oluwasanmi Koyejo, Joshua H Balsters, Krzysztof J Gorgolewski, Craig A Moodie, and Russell A Poldrack. The dynamics of functional brain networks: integrated network states during cognitive task performance. *Neuron*, 92(2):544–554, 2016. 2

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

[58] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[59] Samarth Sinha, Animesh Garg, and Hugo Larochelle. Curriculum by smoothing. *Advances in Neural Information Processing Systems*, 33:21653–21664, 2020. 3

[60] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 3

[61] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020. 1

[62] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33:8728–8740, 2020. 3

[63] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999. 5

[64] George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30, 2017. 5

[65] Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014. 2

[66] 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. 2

[67] Pavan Kumar Anasosalu Vasu, Shreyas Saxena, and Oncel Tuzel. Instance-level task parameters: A robust multi-task weighting framework. *arXiv preprint arXiv:2106.06129*, 2021. 2

[68] Cheng Wang, Qian Zhang, Chang Huang, Wenyu Liu, and Xinggang Wang. Mancs: A multi-task attentional network with curriculum sampling for person re-identification. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–381, 2018. 3

[69] Xiaolong Wang, David Fouhey, and Abhinav Gupta. Designing deep networks for surface normal estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 539–547, 2015. 1

[70] Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. *arXiv preprint arXiv:2010.05874*, 2020. 2

[71] Mackenzie Woodburn, Cheyenne L Bricken, Zhengwang Wu, Gang Li, Li Wang, Weili Lin, Margaret A Sheridan, and Jessica R Cohen. The maturation and cognitive relevance of structural brain network organization from early infancy to childhood. *NeuroImage*, 238:118232, 2021. 2

[72] Junlin Wu and Yevgeniy Vorobeychik. Robust deep reinforcement learning through bootstrapped opportunistic curriculum. In *International Conference on Machine Learning*, pages 24177–24211. PMLR, 2022. 3

[73] Derrick Xin, Behrooz Ghorbani, Ankush Garg, Orhan Firat, and Justin Gilmer. Do current multi-task optimization methods in deep learning even help? *arXiv preprint arXiv:2209.11379*, 2022. 2

[74] Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research*, 8(1), 2007. 2

[75] Mingzhang Yin and Mingyuan Zhou. Arm: Augment-reinforce-merge gradient for discrete latent variable models. *arXiv preprint arXiv:1807.11143*, 2018. 5

[76] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014. 6

[77] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 6

[78] 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. 2, 8

[79] Bowen Zhang, Yidong Wang, Wenxin Hou, Hao Wu, Jindong Wang, Manabu Okumura, and Takahiro Shinozaki. Flexmatch: Boosting semi-supervised learning with curriculum pseudo labeling. *Advances in Neural Information Processing Systems*, 34:18408–18419, 2021. 3

[80] Jiwen Zhang, Jianqing Fan, Jiajie Peng, et al. Curriculum learning for vision-and-language navigation. *Advances in Neural Information Processing Systems*, 34:13328–13339, 2021. 3

[81] Tianjun Zhang, Benjamin Eysenbach, Ruslan Salakhutdinov, Sergey Levine, and Joseph E Gonzalez. C-planning: An automatic curriculum for learning goal-reaching tasks. *arXiv preprint arXiv:2110.12080*, 2021. 3

[82] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018. 2

[83] Zixuan Zhou, Xuefei Ning, Yi Cai, Jiashu Han, Yiping Deng, Yuhan Dong, Huazhong Yang, and Yu Wang. Close: Curriculum learning on the sharing extent towards better one-shot nas. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XX*, pages 578–594. Springer, 2022. 3

7. Appendix

7.1. Non-jointly Labeled Datasets

The non-jointly labeled dataset is widely used in multi-task learning problems. The training datasets consist of T subset of data which includes N_t data points for a certain task t, $\mathcal{D}_t = \{(x_t, y_t)_i | i \in [N_t]\}$. The jointly labeled dataset can be viewed as a special case of the above definition.

The objective function defined on the non-jointly labeled dataset is similar to (6) as

$$\mathcal{L}(\theta) = \sum_{t \in [T]} \sum_{x_t, y_t \in \mathcal{D}} L_t(g_{\theta_t} \circ f_{\theta_s}(x_t), y_t). \tag{6}$$

From a variational inference perspective, the only difference is that each data point has an extra task-specific coefficient which is proportional to the number of samples for that task. We can rewrite (2) as

$$\begin{aligned}
&\log \prod_i p(y_i|x_i) = \sum_i \log \sum_t p(y_{t,i}|x_{i,t})p(t) \\
&\geq \sum_i \sum_t \frac{N_t}{N} \log p(y_{t,i}|t, x_i) \\
&= \sum_t \frac{N_t}{N} \sum_i \log \int_z p(y_{t,i}|z, t, x_i)p(z|t)dz \\
&\geq \sum_t \frac{N_t}{N} \sum_i \int_z q(z|t) \log \frac{p(y_{t,i}|z, t, x_i)p(z|t)}{q(z|t)} dz \\
&= \sum_t \frac{N_t}{N} \sum_i \mathbb{E}_{z \sim q(\cdot|t)}[\log p(y_{t,i}|z, t, x_i)] \\
&\quad - \mathbf{KL}(q(\cdot|t), p(\cdot|t)),
\end{aligned}$$

where $N = \sum_t N_t$.

7.2. Target Distribution Updating

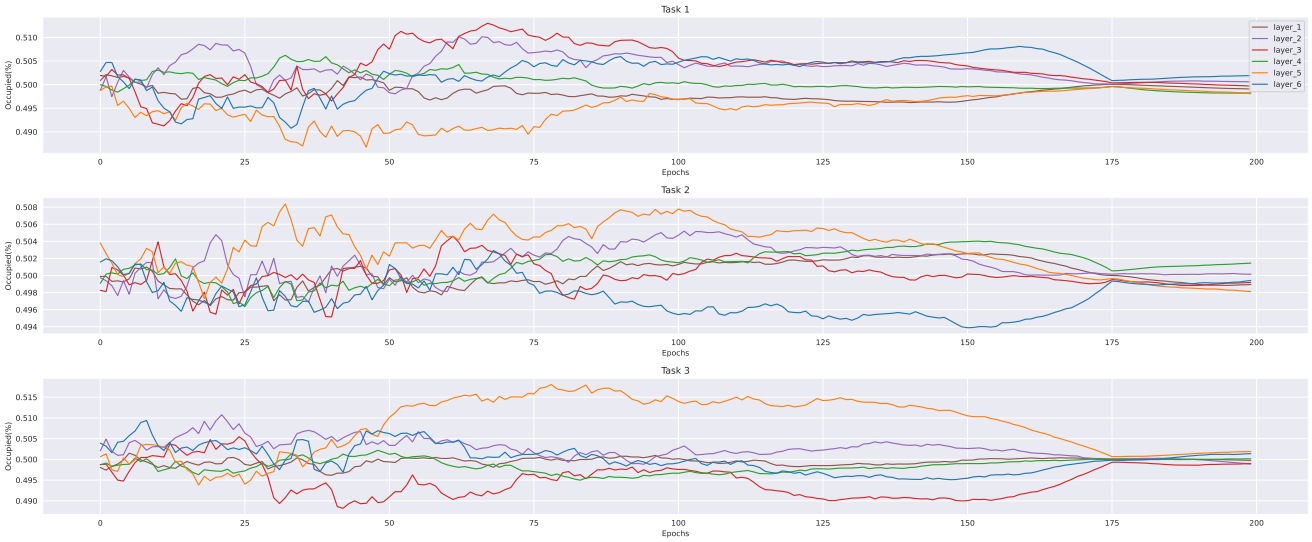

Figure 7. The proportion of tasks SemSeg, Depth, and Normal during the training process.

As mentioned in Section 4.3.4, Figure 7 illustrates the similarities among different tasks by showing the proportion of $\Phi_2/(\Phi_2 + \Phi_3)$, $\Phi_1/(\Phi_1 + \Phi_3)$ and $\Phi_1/(\Phi_1 + \Phi_2)$ respectively. This also reveals the potential structural discrepancy between the learned architecture from the training process and the fully shared backbone.

7.3. Ablation Study on the Multi-CIFAR100

In addition to the main body of the paper, the ablation studies on the Multi-CIFAR100 dataset are shown here as strong complementary evidence. The results shown in the tables below are averaged over 4 experiments with different random seeds.

7.3.1 The Parameterized Distributions

We first test MPPS with different parameterized distributions in Table 9. Note that MPPS with Gaussian distribution shows a worse performance than the baseline. From our observation of the training loss experimental logs and the mask distribution logs, the mask distribution is early converged to a fixed high EC distribution even though the target distribution has a low EC value. The reason we guess is that there may be some numerical optimization problems in calculating KL divergence between Gaussian distributions. The optimization process may easily enter into local optimum when the learning rate is not carefully fine-tuned.

Layers	**Average Acc** ↑	**Lowest Acc** ↑	**Median Acc** ↑	Δ↑
multi	59.23%	37.22%	60.52%	0.00%
MPPS +B	67.62%	41.38%	71.06%	14.36%
MPPS +G	43.66%	27.07%	42.77%	-26.32%

Table 9. Applying MPPS with different parameterized distributions on the Multi-CIFAR100 dataset

7.3.2 Applying on Different Numbers of Network Layers

Table 10 shows the learning performance with a different number of layers on the Multi-CIFAR100 dataset. We recommend our audience to search this hyperparameter for different datasets. Finding the best parameterized scheme is important for MPPS and we guess it depends both on the architecture of the shared backbone and downstream tasks. At this point, we leave this problem as future work.

Layers	**Average Acc** ↑	**Lowest Acc** ↑	**Median Acc** ↑	Δ↑
6	67.62%	41.38%	71.06%	14.36%
12	59.14%	32.43%	61.17%	-0.32%
All	48.86%	29.50%	50.09%	-17.51%

Table 10. Applying MPPS to different numbers of selected layers on the Multi-CIFAR100 dataset

7.3.3 Exclusive Capacity Scheduler

Table 11 shows the comparison results on the Multi-CIFAR100 dataset. Every scheduler we test performs better than the baseline. Separ scheduler has the lowest performance compared with others for the same reason mentioned in Section 4.3.3 at most time of the training process the network is encouraged to learn a separated pattern but use a full share scheme when testing.

Method	**Average Acc** ↑	**Lowest Acc** ↑	**Median Acc** ↑	Δ↑
multi	59.23%	37.22%	60.52%	0.00%
Linear	67.62%	41.38%	71.06%	14.36%
Separ	66.95%	39.43%	69.40%	13.12%
FullShare	67.08%	40.93%	70.21%	13.42%
Step5	67.38%	40.80%	70.23%	13.94%
Step10	67.52%	40.80%	70.87%	14.16%
Step20	67.52%	39.78%	70.72%	14.07%
Quad	67.70%	40.78%	70.45%	14.47%

Table 11. MPPS with different schedulers on the Multi-CIFAR100

7.3.4 Target Distribution Updates

In Table 12, we show the benefits of the target distribution updating mechanism where following the configuration mentioned in Section 4.3.4.

Method	Average Acc ↑	Lowest Acc ↑	Median Acc ↑	Δ↑
multi	59.23%	37.22%	60.52%	0.00%
w/ updating	67.62%	41.38%	71.06%	14.36%
w/o updating	66.51%	46.82%	63.28%	12.16%

Table 12. MPPS with and without target distribution updating

7.4. Limitations

In this section, we analyze the limitations of MPPS.

1. Extra training time and memory cost as shown in Section 4.4.

2. There is a theoretical gap between the final learned stochastic network in training and the fully shared backbone in testing. This may potentially make our approach less effective in some datasets or task combinations.

3. MPPS can hardly be used to fine-tune the backbone models pre-trained on other datasets because MPPS is trained from separated network structures, while a pre-trained model has an entirely shared backbone.

Under the Gaussian distribution parameterized situation, we can give a simple analysis of the magnitude of the theoretical gap described above. Assuming that we have θ', Φ' as the minimization of (2) in the last course where the target distribution is given as $\mathcal{N}(\mathbf{1}, (\epsilon\mathbf{1})^2)$ which approximates a fully shared target distribution when ϵ is a very small value, i.e.

$$\theta', \Phi' = \arg\min_{\theta,\Phi} \sum_t \mathbb{E}_{m\sim p_\Phi(t)}[L(\theta \odot m)] + \mathbf{KL}(p_\Phi, \mathcal{N}(\mathbf{1}, (\epsilon\mathbf{1})^2)).$$

The expected error with the stochastic network on the test dataset is given by

$$\sum_t \mathbb{E}_{m\sim p'_\Phi(t)}[L(\theta' \odot m)], \tag{7}$$

and when we use a fully shared backbone instead of stochastic networks the error is given by

$$\sum_t L(\theta'). \tag{8}$$

The gap(|(7)-(8)|) is

$$
\begin{aligned}
&|\sum_t \mathbb{E}_{m\sim p'_\Phi(t)}[L(\theta'\odot m)] - \sum_t L(\theta')| \\
&\le \sum_t |\mathbb{E}_{m\sim p'_\Phi(t)}[L(\theta'\odot m)] - \mathbb{E}_{m\sim\mathcal{N}(\mathbf{1},(\epsilon\mathbf{1})^2)}[L(\theta'\odot m)] + \mathbb{E}_{m\sim\mathcal{N}(\mathbf{1},(\epsilon\mathbf{1})^2)}[L(\theta'\odot m)] - L(\theta')| \\
&\le \sum_t |\mathbb{E}_{m\sim p'_\Phi(t)}[L(\theta'\odot m)] - \mathbb{E}_{m\sim\mathcal{N}(\mathbf{1},(\epsilon\mathbf{1})^2)}[L(\theta'\odot m)]| + \sum_t |\mathbb{E}_{m\sim\mathcal{N}(\mathbf{1},(\epsilon\mathbf{1})^2)}[L(\theta'\odot m)] - L(\theta')| \\
&\le \sum_t |\mathbb{E}_{m\sim p'_\Phi(t)}[L(\theta'\odot m)] - \mathbb{E}_{m\sim\mathcal{N}(\mathbf{1},(\epsilon\mathbf{1})^2)}[L(\theta'\odot m)]| + C \qquad (9) \\
&\le \sum_t |\mathbb{E}_{\epsilon_a\sim\mathcal{N}(0,1)}[L(\theta'+\theta'\epsilon_a\sigma_{t,\Phi})] - \mathbb{E}_{\epsilon_b\sim\mathcal{N}(0,1)}[L(\theta'+\theta'\epsilon_b\epsilon)]| + C \qquad (10) \\
&= \sum_t |\mathbb{E}_{\epsilon_a\sim\mathcal{N}(0,1)}[L(\theta'+\theta'\epsilon_a\sigma_{t,\Phi}) - L(\theta') + L(\theta') - L(\theta'+\theta'\epsilon_a\epsilon)]| + C \\
&= \sum_t |\mathbb{E}_{\epsilon_a\sim\mathcal{N}(0,1)}[\nabla L(\theta')\epsilon_a\sigma_{t,\Phi}\theta' + O(||\epsilon\sigma_{t,\Phi}\theta'||_2^2) - (\nabla L(\theta')\epsilon_a\epsilon\theta' + O(||\epsilon_a\epsilon\theta'||_2^2))]| + C \\
&\le \sum_t |\mathbb{E}_{\epsilon_a\sim\mathcal{N}(0,1)}[\nabla L(\theta')\epsilon_a\sigma_{t,\Phi}\theta' + O(||\epsilon\sigma_{t,\Phi}\theta'||_2^2)| + \sum_t |\mathbb{E}_{\epsilon_a\sim\mathcal{N}(0,1)}[\nabla L(\theta')\epsilon_a\epsilon\theta' + O(||\epsilon_a\epsilon\theta'||_2^2)] + C \\
&= \sum_t O(||\sigma_{t,\Phi}\theta'||_2^2) + \sum_t O(||\epsilon\theta'||_2^2) + C \\
&\le ||\theta'||_2^2(\sum_t ||\sigma_{\Phi,t}||_2^2 + C') + C, \qquad (11)
\end{aligned}
$$

where (9) is due to the assumption that the $\mathcal{N}(\mathbf{1},(\epsilon\mathbf{1})^2)$ is a good approximation of the fully shared target distribution and causes a bounded error gap C and (10) is due to reparameterization trick, where $m\sim p_\Phi(t)\to m=1+\sigma_{t,\Phi}\epsilon_a, \epsilon_a\sim\mathcal{N}(0,1)$ and $\sigma_{t,\Phi}=\sqrt{\frac{1-\Phi_t}{\Phi_t}}$.

Recall the KL divergence between two Gaussian distributions is $\mathbf{KL}(p,q)=\log\frac{\sigma_2}{\sigma_1}+\frac{\sigma_1^2+(\mu_1-\mu_2)^2}{2\sigma_2^2}-\frac{1}{2}$, where $p=\mathcal{N}(\mu_1,\sigma_1), q=\mathcal{N}(\mu_2,\sigma_2)$, so $\arg\min_\Phi \mathbf{KL}(p_\Phi,\mathcal{N}(\mathbf{1},(\epsilon\mathbf{1})^2)) = \arg\min_\Phi \sum_t\sum_i \log\frac{\epsilon}{\sigma_{t,\Phi,i}}+\frac{\sigma_{t,\Phi,i}^2}{2\epsilon^2}-\frac{1}{2}$. Let $\delta_{t,i}=\sigma_{t,\Phi,i}-\epsilon>0$, we can rewrite it as $\sum_t\sum_i\frac{1}{2}(\frac{\delta_{t,i}}{\epsilon})^2+\frac{\delta_{t,i}}{\epsilon}-\log(1+\frac{\delta_{t,i}}{\epsilon})\approx\sum_t\sum_i(\frac{\delta_{t,i}}{\epsilon})^2\le\sum_t\sum_i\frac{1}{\epsilon^2}\sigma_{t,\Phi,i}^2\approx\sum_t||\sigma_{\Phi,t}||_2^2$ when $\frac{\delta_t}{\epsilon}\to 0$.

So the (11) is proportional to two terms, the $\mathbf{KL}$ term and the model weight term. We propose to add a control coefficient $\beta>1$ in (2) before the KL term as

$$
\sum_{x,\mathbf{y}\in\mathcal{D}}\sum_{t\in[T]}\mathbb{E}_{m\sim\tilde{q}_\Phi(\cdot,t)}[L_t(g_{\theta_t}^t\circ f_{\theta_s\odot m}(x),y_t)+\beta(\log\tilde{q}_\Phi(m,t)-\log\tilde{p}_\Pi(m,t))]
$$

to reduce the gap and use a large weight decay coefficient to reduce the weight norm for future works.