Deep Multitask Learning with Progressive Parameter Sharing

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

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

Shen Ren
Continental Automotive Singapore
shen@shenren.org

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].

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 $y = (y_1, y_2, \ldots, y_T)$, 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 $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} = \{(x, y) | i \in [N]\}$.

The MTL model consists of a shared network $f_\theta_s$ with parameters $\theta_s$, 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 $\theta$ that can empirically minimize the following loss function:

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

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 $\theta$ 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.
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 $\phi$ to build a $T$-dimensional Multivariate Bernoulli distribution $B(\phi)$. 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 $r_G(v) = \sqrt{\frac{1-n}{v}}$ 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 $P(r(\phi))$ 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:

$$\mathcal{L}(\theta) = \sum_{x,y \in D} \sum_{t \in [T]} \mathbb{E}_{m \sim q(\cdot|t)}[L_t(y_{\theta_t} \circ f_{\theta_0 \odot m}(x), y_t)]$$

$$+ \text{KL}(q(\cdot|t), p(\cdot|t)),$$

where 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: $\log \prod_t p(y_t|x_i)$. We can describe the objective with the mask distribution condition on task $t$ as

$$\log \prod_i p(y_i|x_i) = \sum_i \log \sum_t p(y_{t,i}|x_i, t)p(t)$$

$$\geq \frac{1}{T} \sum_i \sum_t \log p(y_{t,i}|t, x_i)$$

$$= \frac{1}{T} \sum_t \sum_i \log \int \mathbb{E}_{z \sim q(\cdot|t)} p(y_{t,i}|z, t, x_i)p(z|t)dz$$

$$\geq \frac{1}{T} \sum_t \sum_i \int q(z|t) \log \frac{p(y_{t,i}|z, t, x_i)p(z|t)}{q(z|t)}dz$$

$$= \frac{1}{T} \sum_t \sum_i \mathbb{E}_{z \sim q(\cdot|t)} \log p(y_{t,i}|z, t, x_i)$$

$$- \text{KL}(q(\cdot|t), p(\cdot|t)).$$

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

\[
L(\theta, \Phi, \Pi) = \sum_{x, y \in D} \sum_{t \in [T]} \mathbb{E}_{m \sim \tilde{q}_\phi(\cdot, t)} [L_I(g_{\theta_t} \circ f_{\theta_t} \circ m(x), y_t) + \log \tilde{q}_\phi(m, t) - \log \tilde{p}_\Pi(m, t)].
\]  

(3)

We can use stochastic gradient descent to optimize the above loss function by both updating the network parameters \( \theta \) and mask distribution parameters \( \Phi \). It is easy to update \( \theta \) and calculate its gradient, but updating \( \Phi \) is more difficult. When using the Gaussian distribution for \( q_\phi \), 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 \( \epsilon_r \sim \mathcal{N}(0, I) \).

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

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

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 training process. In the beginning, there is no prior knowledge about all the tasks at all. We can initialize the target distribution \( p_\Pi \) to evenly assign all neurons to each task layer by layer. Also, the mask distribution \( q_\phi \) is set equal to \( p_\Pi \). 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 \( \phi \) as \( 1 - \epsilon \), while the rest parameters as \( \epsilon \), where \( \epsilon \) is a small positive value for numerical stability. Such assignment can guarantee both distributions \( q_\phi \) and \( p_\Pi \) have the highest \( EC \) during initialization.

**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 \( \Pi \), we reuse the loss term in (3) and build a constrained optimization problem as follows:

\[
\min_{\Pi} \sum_t \text{KL}(\tilde{q}_\phi(\cdot, t), \tilde{p}_\Pi(\cdot, t))
\]

\[\text{s.t.} \quad \sum_t \Pi[t] \leq \max_t \Pi[t] = 1 - C_i \]  

where the updated \( \Phi \) is used here.

We propose a heuristic approach to updating the target distribution. We directly normalize and rescale the remaining capacity of the updated \( \Phi \) to build the target distribution \( p_\Pi \) in the next course, as the updated \( \Phi \) 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 \( q_\phi \) will allocate more capacity to \( B \) on the neurons whose owner is \( A \), especially under such a limited total capacity situation, and vice versa. 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: \( \theta, \phi, \Pi \)

for \( i = 1, 2, \ldots, n_{epochs} \) do

for \( x_i, \{ y_i \}_i \in D \) do

for \( t \in [T] \) do

Sampling \( m \sim q_{\alpha}(t) \)

Calculate loss \( L_t \) according to Eq.(3)

end for

Updating \( \theta, \phi \) according to the optimizer

end for

Get \( C_i \) from EC Scheduler

\( C \leftarrow 1 - \left( \sum_t \Phi[t] - \max_t \Phi[t] \right) \)

\( \Pi \leftarrow 1 - \frac{C}{t} (1 - \Phi) \)

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 sub-tasks 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 \( \mathcal{P}_\theta \) 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:
\[
\Delta(\theta) = \frac{1}{T} \sum_{t \in [T]} C_t \frac{P_t - P_{t,\text{base}}}{P_{t,\text{base}}},
\]
where \(P_t\) is the metric of task \(t\) under the given parameters \(\theta\), \(P_{t,\text{base}}\) 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 value \((C_t = 1)\) 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 relative performance improvement metric as follows:

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

where \(P_t\) is the metric of task \(t\) under the given parameters \(\theta\), \(P_{t,\text{base}}\) 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 value \((C_t = 1)\) is better.

We observe that both MPPS with Bernoulli and Gaussian distributions show improved results (higher \(\Delta\)). 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.

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.

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.

---

Table 1. Comparison results on the NYUv2 dataset.

<table>
<thead>
<tr>
<th>Layers</th>
<th>SemSeg</th>
<th>Depth</th>
<th>Normal</th>
<th>(\Delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CE (_{\text{mIoU}})</td>
<td>L1 (_{\text{AE}})</td>
<td>L1 (_{\text{AE}})</td>
<td>MAD (_{\text{AE}})</td>
</tr>
<tr>
<td>Single</td>
<td>-0.368</td>
<td>0.532</td>
<td>0.523</td>
<td>25.027</td>
</tr>
<tr>
<td>Multi</td>
<td>1.291</td>
<td>0.407</td>
<td>0.469</td>
<td>0.651</td>
</tr>
<tr>
<td>MPPS +B</td>
<td>1.277</td>
<td>0.405</td>
<td>0.470</td>
<td>0.629</td>
</tr>
<tr>
<td>MPPS +G</td>
<td>1.188</td>
<td>0.417</td>
<td>0.466</td>
<td>0.640</td>
</tr>
</tbody>
</table>

Table 2. Comparison results on the Cityscapes dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Acc (_{\text{AE}})</th>
<th>Lowest Acc (_{\text{CE}})</th>
<th>Median Acc (_{\text{Pred}})</th>
<th>(\Delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi</td>
<td>59.23%</td>
<td>37.22%</td>
<td>60.52%</td>
<td>+0.0%</td>
</tr>
<tr>
<td>MPPS</td>
<td>67.62%</td>
<td>41.38%</td>
<td>71.06%</td>
<td>+14.4%</td>
</tr>
</tbody>
</table>

Table 3. Performance of 20 tasks on the Multi-CIFAR100 dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>SemSeg</th>
<th>PartSeg</th>
<th>Disp</th>
<th>(\Delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CE (_{\text{mIoU}})</td>
<td>CE (_{\text{mIoU}})</td>
<td>L1 (_{\text{AE}})</td>
<td>AE (_{\text{AE}})</td>
</tr>
<tr>
<td>Single</td>
<td>-0.526</td>
<td>-0.527</td>
<td>-0.84</td>
<td>+0.0%</td>
</tr>
<tr>
<td>Multi</td>
<td>0.260</td>
<td>0.553</td>
<td>0.519</td>
<td>0.797</td>
</tr>
<tr>
<td>MPPS +B</td>
<td>0.244</td>
<td>0.571</td>
<td>0.522</td>
<td>0.800</td>
</tr>
<tr>
<td>MPPS +G</td>
<td>0.244</td>
<td>0.568</td>
<td>0.524</td>
<td>0.810</td>
</tr>
</tbody>
</table>

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

---

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

<table>
<thead>
<tr>
<th>Method</th>
<th>SemSeg</th>
<th>Depth</th>
<th>Normal</th>
<th>Δ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>CE (%)</td>
<td>mIoU (%)</td>
<td>CE (%)</td>
<td>mIoU (%)</td>
</tr>
<tr>
<td>Single</td>
<td>-</td>
<td>0.377</td>
<td>-</td>
<td>0.532</td>
</tr>
<tr>
<td>DWA</td>
<td>1.301</td>
<td>0.401</td>
<td>0.477</td>
<td>0.477</td>
</tr>
<tr>
<td>Uncert</td>
<td>1.344</td>
<td>0.407</td>
<td>0.470</td>
<td>0.470</td>
</tr>
<tr>
<td>Auto-λ</td>
<td>1.314</td>
<td>0.414</td>
<td>0.462</td>
<td>0.462</td>
</tr>
<tr>
<td>HPS</td>
<td>1.291</td>
<td>0.407</td>
<td>0.469</td>
<td>0.469</td>
</tr>
<tr>
<td>GradDrop</td>
<td>1.335</td>
<td>0.400</td>
<td>0.469</td>
<td>0.469</td>
</tr>
<tr>
<td>PCGrad</td>
<td>1.309</td>
<td>0.406</td>
<td>0.471</td>
<td>0.471</td>
</tr>
<tr>
<td>Cagrad</td>
<td>1.291</td>
<td>0.406</td>
<td>0.457</td>
<td>0.457</td>
</tr>
<tr>
<td>MTAN</td>
<td>1.335</td>
<td>0.430</td>
<td>0.449</td>
<td>0.449</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>SemSeg</th>
<th>Depth</th>
<th>Normal</th>
<th>Δ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>CE (%)</td>
<td>mIoU (%)</td>
<td>CE (%)</td>
<td>mIoU (%)</td>
</tr>
<tr>
<td>Single+</td>
<td>-</td>
<td>0.377</td>
<td>-</td>
<td>0.532</td>
</tr>
<tr>
<td>DWA+</td>
<td>1.234</td>
<td>0.407</td>
<td>0.466</td>
<td>0.466</td>
</tr>
<tr>
<td>Uncert+</td>
<td>1.299</td>
<td>0.398</td>
<td>0.471</td>
<td>0.471</td>
</tr>
<tr>
<td>Auto-λ+</td>
<td>1.339</td>
<td>0.411</td>
<td>0.459</td>
<td>0.459</td>
</tr>
<tr>
<td>HPS+</td>
<td>1.277</td>
<td>0.405</td>
<td>0.470</td>
<td>0.470</td>
</tr>
<tr>
<td>GradDrop+</td>
<td>1.296</td>
<td>0.402</td>
<td>0.469</td>
<td>0.469</td>
</tr>
<tr>
<td>PCGrad+</td>
<td>1.240</td>
<td>0.403</td>
<td>0.474</td>
<td>0.474</td>
</tr>
<tr>
<td>Cagrad+</td>
<td>1.256</td>
<td>0.409</td>
<td>0.453</td>
<td>0.453</td>
</tr>
<tr>
<td>MTAN+</td>
<td>1.261</td>
<td>0.431</td>
<td>0.450</td>
<td>0.450</td>
</tr>
</tbody>
</table>

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.

Table 6. MPPS with and without target distribution updating

<table>
<thead>
<tr>
<th>Method</th>
<th>SemSeg</th>
<th>Depth</th>
<th>Normal</th>
<th>Δ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>CE (%)</td>
<td>mIoU (%)</td>
<td>CE (%)</td>
<td>mIoU (%)</td>
</tr>
<tr>
<td>w/ updating</td>
<td>1.277</td>
<td>0.405</td>
<td>0.470</td>
<td>0.470</td>
</tr>
<tr>
<td>w/o updating</td>
<td>1.250</td>
<td>0.401</td>
<td>0.475</td>
<td>0.475</td>
</tr>
</tbody>
</table>

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 Grad-drop [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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Metric</th>
<th>SemSeg</th>
<th>Depth</th>
<th>Normal</th>
<th>∆T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L1</td>
<td>mIoU</td>
<td>L1</td>
<td>mIoU</td>
<td></td>
</tr>
<tr>
<td>ResNet18</td>
<td>Single</td>
<td>0.3714</td>
<td>0.5280</td>
<td>25.0128</td>
<td>+0.0%</td>
</tr>
<tr>
<td>Multi</td>
<td>1.283</td>
<td>0.421</td>
<td>0.453</td>
<td>0.632</td>
<td>25.686</td>
</tr>
<tr>
<td>MPPS +G</td>
<td>1.132</td>
<td>0.419</td>
<td>0.453</td>
<td>0.623</td>
<td>24.812</td>
</tr>
<tr>
<td>ResNet34</td>
<td>Single</td>
<td>0.387</td>
<td>0.524</td>
<td>24.48</td>
<td>+0.0%</td>
</tr>
<tr>
<td>Multi</td>
<td>1.253</td>
<td>0.446</td>
<td>0.446</td>
<td>0.633</td>
<td>25.638</td>
</tr>
<tr>
<td>MPPS +G</td>
<td>1.130</td>
<td>0.421</td>
<td>0.453</td>
<td>0.623</td>
<td>25.287</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>w/o MPPS</th>
<th>w MPPS(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPS</td>
<td>28.69</td>
<td>39.34</td>
</tr>
<tr>
<td>PCGrad</td>
<td>41.03</td>
<td>41.42</td>
</tr>
<tr>
<td>Auto-λ</td>
<td>84.19</td>
<td>131.74</td>
</tr>
<tr>
<td>DWA</td>
<td>32.39</td>
<td>42.13</td>
</tr>
</tbody>
</table>

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


[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


