

# [Supplementary Materials for] AdapMTL: Adaptive Pruning Framework for Multitask Learning Model

Anonymous Authors

## A THRESHOLDS UPDATING

We detail the process of updating thresholds within the AdapMTL framework in this section. The thresholds are determined by  $\theta_{\text{init}}$ , such that  $\alpha_t = \text{sigmoid}(\theta_{\text{init}})$ . Consequently, the challenge of updating the thresholds is transformed into the task of updating the  $\theta_t$  for each specific task. Considering a multitask model with  $T$  tasks, we divide the weight parameters  $W$  into  $W = \{W_B, W_1, W_2, \dots, W_T\}$ , where  $W_B$  represents the weight parameters for the shared backbone and  $W_t$  represents the weight parameters for the  $t$ -th task-specific head. We derive the gradient descent update equation at the  $n$ -th epoch for  $\theta_t$  as follows:

$$\begin{aligned}\theta_t^{n+1} &= \theta_t^n - \eta^n \frac{\partial \mathcal{L}(W, \alpha; \mathcal{D})}{\partial \theta_t^n} \\ &= \theta_t^n - \eta^n \frac{\partial \mathcal{L}(\theta, \alpha; \mathcal{D})}{\partial W_t^n} \odot \frac{\partial W_t^n}{\partial \theta_t^n} \\ &= \theta_t^n - \eta^n \cdot (-\text{sigmoid}(\theta_t^n))' \cdot \frac{\partial \mathcal{L}(\theta, \alpha; \mathcal{D})}{\partial W_t^n} \\ &= \theta_t^n - \eta^n \cdot (-\text{sigmoid}(\theta_t^n))' \cdot \frac{\partial \mathcal{L}(W, \alpha; \mathcal{D})}{\partial S(W_t^n, \alpha_t^n)} \odot \frac{\partial S(W_t^n, \alpha_t^n)}{\partial W_t^n} \\ &= W_t^n + \eta^n \cdot (\text{sigmoid}(\theta_t^n))' \cdot \frac{\partial \mathcal{L}(W, \alpha; \mathcal{D})}{\partial S(W_t^n, \alpha_t^n)} \odot \mathcal{B}_t^n,\end{aligned}\quad (1)$$

where  $\eta^n$  is the learning rate at the  $n$ -th epoch. We use the partial derivative to calculate the gradients. Although  $\frac{\partial S(W_t^n, \alpha_t^n)}{\partial W_t^n}$  is non-differentiable, we can approximate the gradients using the sub-gradient method. In this case, we introduce  $\mathcal{B}_t^n$ , an indicator function that acts like a binary mask. The value of  $\mathcal{B}_t^n$  should be 0 if the sparse version of the weight  $S(W_t^n, \alpha_t^n)$  is equal to 0. This indicator function facilitates the approximation of gradients and the update of the sparse weights and soft thresholds during the backpropagation process. Mathematically, the indicator function is:

$$\mathcal{B}_t^n = \begin{cases} 0, & \text{if } S(W_t^n, \alpha_t^n) = 0, \\ 1, & \text{otherwise.} \end{cases} \quad (2)$$

## B TRAINING DETAILS

We adopt the same training configurations as those used in DiSparse [4], which is the latest multitask pruning work, for fair comparisons. We conduct all our experiments using PyTorch and RTX 8000 GPUs, and we employ the Adam optimizer with a batch size of 16. For the NYUV2 dataset, we run 20K iterations with an initial learning rate of 1e-3, decaying by 0.5 every 4,000 iterations. For the Tiny-Taskonomy dataset, we train for 100K iterations with an initial learning rate of 1e-4, decaying by 0.3 every 12K iterations. The size of the sliding window in our experiments is set to 400 to smooth loss deviations. We utilized cross-entropy loss for Semantic Segmentation, negative cosine similarity between the normalized prediction and ground truth for Surface Normal Prediction, and L1

**Table 1: Hyper-parameters for training on NYUv2 and Tiny-taskonomy datasets**

Dataset	lr	lr decay	epoch
NYUv2	0.001	0.5/ 4,000 ters	20,000
Tiny-Taskonomy	0.0001	0.3/ 10,000 iters	50,000

loss for the remaining tasks. To avoid bias and diversity in different pre-trained models, we trained all models from scratch, ensuring a fair comparison among various methods. It’s noteworthy that, unlike many previous works, our method does not require any pre-training or pre-pruned models.

We use the  $\theta_{\text{init}}$  parameter, set to -20, to regulate the duration of dense training phases. A lower  $\theta_{\text{init}}$  value extends the period dedicated to dense representation, allowing for more comprehensive learning before pruning begins.

## C ADDITIONAL RESULTS

We provide additional results on the Tiny-Taskonomy dataset using Resnet34 and MobileNetV2 architecture, separately. On the Tiny-Taskonomy dataset, which comprises a total of 5 tasks, AdapMTL demonstrates a more balanced performance across tasks. As shown in Table 2, 3, our method achieved the highest  $\Delta_T$  score and the lowest absolute error for most tasks.

### C.1 Pruning Sensitivity Analysis

Different task heads may have similar amounts of model weights, but their sensitivities to pruning can vary. This observation suggests that a more discriminative pruning approach should be employed for each task, taking into account their unique sensitivities. To verify this observation, we fixed a ResNet34 backbone at a sparsity level of 95% for better visualization and pruned three task heads independently to examine their sensitivity to pruning.

As shown in Figure 1, the head of the surface normal prediction task is the least sensitive, as it maintains good accuracy even when extreme sparsity is enforced. Therefore, AdapMTL learns to keep this task head at a high level of sparsity during pruning, which is aligned with the component-wise sparsity allocation in the table of the manuscript. In contrast, the head of the semantic segmentation task is relatively more sensitive to pruning, so we strive to keep it as dense as possible throughout the training process. This tailored approach to pruning helps AdapMTL achieve better overall performance across different tasks by considering the specific pruning sensitivity of each task head.

### C.2 Adaptive weighting factor

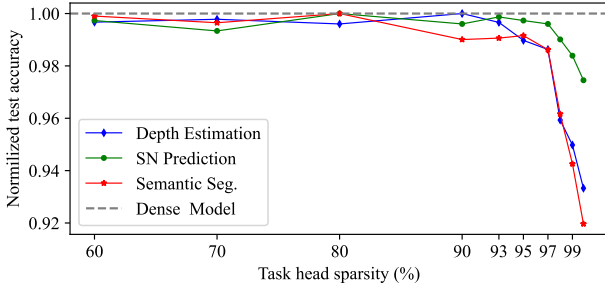
The adaptive weighting mechanism is used to decide the head sparsity allocation among different tasks based on their varying sensitivity to pruning. By adaptively learning a weighting factor, we

**Table 2: Comparison with MTL pruning methods on the Tiny-Taskonomy dataset using the Deeplab-ResNet34 backbone.**

Model	$T_1$ : Semantic Seg.		$T_2$ : Normal Pred.		$T_3$ : Depth Estimation		$T_4$ : Keypoint Det.		$T_5$ : Edge Det.		sparsity %	$\Delta T \uparrow$
	Abs. ↓	$\Delta T_1 \uparrow$	Abs. ↑	$\Delta T_2 \uparrow$	Abs. ↓	$\Delta T_3 \uparrow$	Abs. ↓	$\Delta T_4 \uparrow$	Abs. ↓	$\Delta T_5 \uparrow$		
Dense Model	0.5053	0	0.8436	0.00	0.0222	0.00	0.1961	0.00	0.2131	0.00	95	0.00
SNIP [3]	0.5659	-11.99	0.7301	-13.45	0.0246	-10.81	0.1972	-0.56	0.2221	-4.22	95	-8.21
LTH [1]	0.5345	-5.78	0.8189	-3.38	0.0234	-5.41	0.2004	-2.19	0.2187	-2.63	95	-3.88
IMP [2]	0.5163	-2.18	0.8371	-0.79	0.0221	0.45	0.1962	-0.05	0.2184	-2.49	95	-1.01
DiSparse [4]	0.5287	-4.63	0.8423	-0.16	0.0217	2.25	0.1987	-1.33	0.2089	1.97	95	-0.38
AdapMTL w/o adaptive thresholds	0.5468	-8.21	0.8059	-4.48	0.02296	-3.42	0.1937	1.22	0.2153	-1.03	95	-3.18
AdapMTL	0.5038	0.30	0.8513	0.96	0.0221	0.45	0.1923	1.94	0.2074	2.67	95	1.26

**Table 3: Comparison with MTL pruning methods on the Tiny-Taskonomy dataset using the MobileNetV2 backbone.**

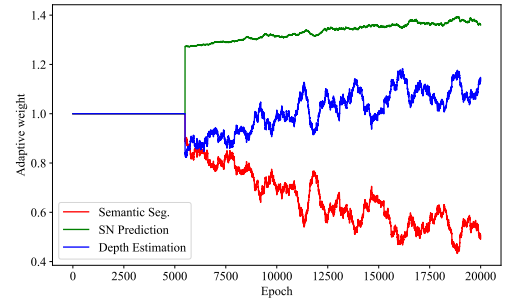
Model	$T_1$ : Semantic Seg.		$T_2$ : Normal Pred.		$T_3$ : Depth Estimation		$T_4$ : Keypoint Det.		$T_5$ : Edge Det.		sparsity %	$\Delta T \uparrow$
	Abs. ↓	$\Delta T_1 \uparrow$	Abs. ↑	$\Delta T_2 \uparrow$	Abs. ↓	$\Delta T_3 \uparrow$	Abs. ↓	$\Delta T_4 \uparrow$	Abs. ↓	$\Delta T_5 \uparrow$		
Dense Model	1.0783	0.00	0.7429	0.00	0.0318	0.00	0.203	0.00	0.2242	0.00	95	0.00
SNIP [3]	1.0901	-1.09	0.7243	-2.50	0.0321	-0.94	0.2157	-6.26	0.2364	-5.44	95	-3.25
LTH [1]	1.0869	-0.80	0.7407	-0.30	0.0325	-2.20	0.2118	-4.33	0.2328	-3.84	95	-2.29
IMP [2]	1.0795	-0.11	0.7415	-0.19	0.0327	-2.83	0.2012	0.89	0.2351	-4.86	95	-1.42
DiSparse [4]	1.0781	0.02	0.7423	-0.08	0.0322	-1.26	0.208	-2.46	0.2287	-2.01	95	-1.16
AdapMTL w/o adaptive thresholds	1.0868	-0.79	0.7329	-1.35	0.0344	-8.18	0.2043	-0.64	0.2233	0.40	95	-2.11
AdapMTL	1.0751	0.30	0.7421	-0.11	0.0305	4.09	0.2021	0.44	0.2225	0.76	95	1.10

**Figure 1: Comparison of task head sensitivities to pruning for different vision tasks. We use a 95% sparse Resnet34 backbone for better depiction. The dashed line dense model indicates the task head is dense.**

can assign different importance to each task, subsequently pruning discriminatively on different task heads.

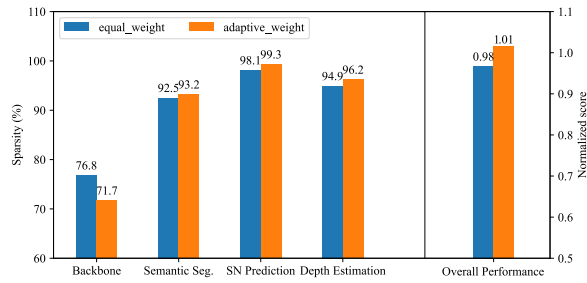
As shown in Figure 2, we initially set the weighting factor of each task equal, such as 1, to ensure sufficient training of each task for 5,000 epochs. Over time, the surface normal prediction task tends to stabilize and converge, leading to small loss fluctuations. This implies that we can prune more aggressively on that component, as the task head is robust to pruning. Consequently, the weighting factor of this task will be larger than the others. This observation is aligned with the results from Table in the main text, where the surface normal prediction achieves higher sparsity compared to other task heads. In contrast, the loss of semantic segmentation fluctuates significantly, indicating that we should consider pruning less on that component by lowering the weighting factor, as the training

is less likely to converge at higher sparsity levels. It is worth mentioning that the weighting factor is learned adaptively, eliminating the need for manual effort to fine-tune the hyper-parameters.

**Figure 2: The evolution of adaptive weighting factors during training. Equal weights are initially assigned to each task for the first 5000 epochs to ensure sufficient training.**

To validate the effectiveness of our adaptive weighting mechanism, we also carry out an experiment where we assign equal weights ( $\beta^t=1$ ) to each task and then visualize the sparsity allocation across each component under this configuration. For comparison, we also visualize the sparsity allocation with adaptive weighting, where the weighting factor for semantic segmentation, surface normal prediction, and depth estimation is set to 1.35, 1.15, and 0.5 respectively. This configuration is from the one shown in Figure 2.

As illustrated in Figure 3, the adaptive weighting factor results in a denser shared backbone compared to the equal weighting factor, while simultaneously making the task heads sparser. Notably, even though the task-specific head of semantic segmentation is assigned a lower value, it achieves higher sparsity than with equal weight.



**Figure 3: Comparison of sparsity allocation between equal weight (1) and adaptive weight configurations, where we assign weighting factors of 1.35, 1.15, and 0.5 to each task-specific head respectively. The overall sparsity is 90%**

This phenomenon arises because the shared backbone is already dense, prompting a sparser task head than before. The right-most sub-figure presents the overall performance under this sparsity allocation. It is evident that our method, with the incorporation of the adaptive weighting mechanism, outperforms the variant of our method without it.

Through this experiment, we demonstrate that the adaptive weighting mechanism plays a crucial role in maintaining high density for the shared backbone and efficiently allocating sparsity among the task-specific heads. By taking into account the sensitivity and importance of different components in the MTL model, the adaptive weighting mechanism allows for better overall performance even when high sparsity is enforced.

## D DISCUSSION

One of the noticeable aspects of AdapMTL is that the final sparsity of our model may not exactly match the requested sparsity. This discrepancy arises due to the intrinsic behavior of the soft thresholds. During pruning, these soft thresholds determine whether a specific parameter should be set to zero, thereby introducing sparsity into the model. However, the soft thresholds do not strictly enforce the exact level of sparsity but rather guide the model to approach the desired sparsity level. This level of flexibility is a design choice made to prevent any undue negative impact on model performance due to overly rigid sparsity constraints. It allows the model to strike a balance between the targeted sparsity and the necessity to preserve adequate performance levels. Our current strategy to maximally approximate the desired sparsity level involves fixing the pruning mask once the desired sparsity has been reached. Another potential approach could involve a recovery mechanism that regrows some crucial parameters that were pruned in earlier epochs. Future research could explore more precise control mechanisms over the final sparsity while ensuring that the model's performance remains robust.

## REFERENCES

- [1] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. 2020. The lottery ticket hypothesis at scale. *International Conference on Learning Representations* (2020).

- [2] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
- [3] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).
- [4] Xinglong Sun, Ali Hassani, Zhangyang Wang, Gao Huang, and Humphrey Shi. 2022. DiSparse: Disentangled Sparsification for Multitask Model Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12382–12392.