SortedNet, a Place for Every Network and Every Network in its Place

Mojtaba Valipour[1,2], **Mehdi Rezagholizadeh**[2], **Hossein Rajabzadeh**[1,2],
Marzieh Tahaei[2], **Boxing Chen**[2], **Ali Ghodsi**[1]
[1]University of Waterloo
[2]Huawei Noah's Ark Lab
{mojtaba.valipour, hossein.rajabzadeh, ali.ghodsi}@uwaterloo.ca,
{mehdi.rezagholizadeh, marzieh.tahaei, boxing.chen}@huawei.com

Abstract

As the size of deep learning models continues to grow, finding optimal models under memory and computation constraints becomes increasingly more important. Although the architecture and constituent building blocks of neural networks usually allow them to be used modularly (i.e., using the sub-networks of a given network after training), their training process is unaware of this modularity. Consequently, conventional neural network training lacks the flexibility to adapt the computational load of the model during inference. This paper proposes SortedNet, a generalized and scalable solution to harness the inherent modularity of deep neural networks across various dimensions (e.g. width, depth, blocks) for efficient dynamic inference. Our training considers a nested architecture for the sub-models with shared parameters and trains all models simultaneously to obtain many-in-one sorted models. We utilize a novel updating scheme during training that combines a random sub-model sampling with gradient accumulation to improve training efficiency. Furthermore, the sorted nature of our training leads to a search-free sub-model selection at inference time; and the nested architecture of the resulting sub-models leads to minimal storage requirement and efficient switching between sub-models at inference. Our general dynamic training approach is demonstrated across various architectures and tasks, including BERT on language understanding and ResNet on image classification. Experimental results show the efficacy of the proposed method in achieving efficient sub-models while outperforming state-of-the-art dynamic training approaches.

1 Introduction

There has been a remarkable growth in the size of deep neural networks. Nevertheless, the computation/memory resources allocated to a model at inference depend on the specific hardware availability and applications' accuracy/time requirements, whether deployed in the cloud or on edge devices.

In particular, the computational burden from concurrent processes and battery limitations can significantly impact the resources allocated to a neural network. Moreover, in the era of gigantic pre-trained models, the computational demand can vary from task to task. Therefore, a growing demand exists for models that can adapt to such dynamic conditions. Unfortunately, with its fixed architecture, conventional neural network training falls short in adaptive adjusting of the computational load at inference time.

On the other hand, deep neural networks demonstrate modular architectures along various dimensions, like layers and blocks across depth, and neurons and channels and attention heads along the width. This inherent modularity enables the extraction of sub-networks with the same shape as the original

37th Conference on Neural Information Processing Systems - ENLSP (NeurIPS 2023).

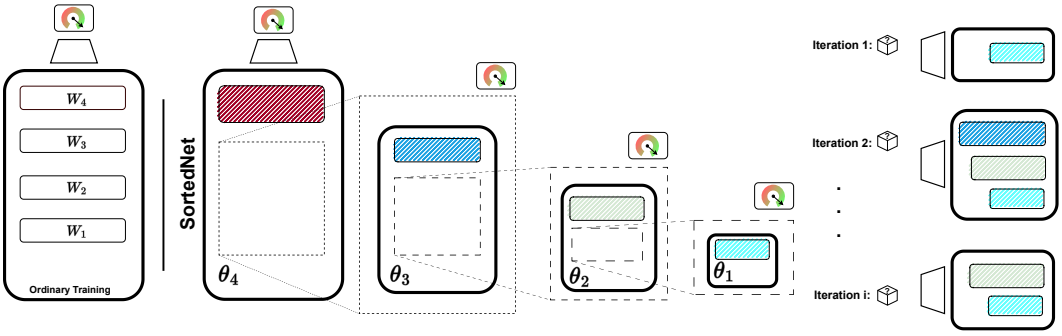

Figure 1: SortedNet: The overall diagram of our proposed method. During training, in each iteration, we sample from a pre-defined random distribution, which will help us to optimize the sub-models as well as the original model.

model. However, the current training methods fail to effectively leverage this modularity, resulting in the final product's limited practical advantages of sub-networks. Consequently, the performance of these sub-networks falls short compared to the main model, making their deployment during inference impractical.

Hence, the challenge lies in harnessing the full potential of modularity in deep neural networks, allowing for the efficient utilization of sub-networks to enhance performance and enable practical deployment in real-world scenarios.

Recent works have proposed a variety of approaches for training dynamic models. These approaches While effective often use a sophisticated training process combined with knowledge distillation Hou et al. [2020], require architecture modification Nunez et al. [2023], and involve redundant subnetwork optimization Fan et al. [2019]. Although not explicitly mentioned, an important ingredient shared by all these methods is the attempt to implicitly sort sub-networks along a specific dimension with respect to computation/accuracy. Limiting dynamicity to one or two dimensions while leaving other dimensions intact can lead to suboptimal sub-networks. Inspired by these works (Valipour et al. [2023], Rippel et al. [2014]), in this paper, we explore how sorting generalized to all dimensions can provide many in one efficient dynamic models. Our solution takes advantage of intrinsic nested sub-networks, which are sorted monotonically from bottom to top. This sorted configuration with shared parameters enforces a regular order and consistency in the knowledge learned by sub-networks. In the resulting nested architecture with shared parameters in a *sorted* manner, each smaller (shallower/narrower) model is a sub-network of a larger (deeper/wider) model. This will lead to models with sorted accuracy, latency and importance. Sorting these sub-networks based on their computation/accuracy characteristics presents the most optimal solution. By organizing the model this way, extracting the desired sub-network becomes a search-free process. Using a predefined sorting order ensures that each targeted sub-network possesses a unique computation overhead, effectively removing the optimization of redundant sub-networks from training.

To achieve this sorted architecture, during training, we propose a novel updating scheme that combines random sampling of sub-networks with gradient accumulation to reduce the cost of training further. Our proposed method can yield multiple models with different capacities with one training.

Our general dynamic training approach is applicable to any architecture without necessitating any modifications to the original model. The proposed nested architecture offers several benefits, including minimal storage requirements and efficient switching between various computation budgets during inference.

1.1 SortedNet: Towards a Generalized Solution for Training Many-in-One Networks

While many of deep neural network architectures are modular in design (using similar layers such as in Transformers Vaswani et al. [2017] or blocks such as in MobileNet Sandler et al. [2018]), this

modularity is not preserved during training. In this subsection, we present SortedNet, a generalized and scalable method for training many-in-one neural networks. In order to train many-in-one networks, we need to specify a few design choices: first, how to form the sub-networks and their configurations; second, what are the target architectures; and third, how to train the sub-networks along with the main model.

Designing the Sub-networks SortedNet imposes an inductive bias on the training based on the assumption that the parameters of sub-networks across each dimension have a concentric (or onion shape) architecture with shared parameters in a *sorted* manner. This sorted configuration with shared parameters enforces a regular order and consistency in the knowledge learned by sub-networks (see Fig. 1).

Let's consider a many-in-one neural network $f(x;\theta(n))$ with the parameters $\theta(n)$ and the input x which is comprised of n sub-networks $f(x;\theta(i))|_{i=0}^{n-1}$, where $\theta(i)$ represents the weights of the i^{th} sub-model. We define a universal set which contains all unique sub-models: $\Theta = \{\theta(0), \theta(1), ..., \theta(n)\}$.

Setting up an order Suppose that we would like to target $D = \{Dim_1, Dim_2, ..., Dim_K\}$ many-in-one dimensions in our model. Then, let's start with $\Theta = \varnothing$ and build the sub-models iteratively. In this regard, at each iteration t during training, we have sampling and truncation procedures along any of the targeted dimensions:

$$\begin{aligned} &\theta_t^* = \cap_{j=1}^{|D|} \theta_{Dim_j \downarrow b_j^t}(n) \\ &\text{where } b_j^t \sim P_{B_j} \\ &\text{IF } \theta_t^* \notin \Theta : \Theta \leftarrow \Theta \cup \{\theta_t^*\} \end{aligned} \tag{1}$$

where $Dim_j \downarrow b_j^t$ indicates that we have truncated $\theta(n)$ along the Dim_j dimension up to the index b_j^t at the iteration t. b_j^t is sampled from a distribution P_{B_j} with the support set of $B_j = \{1, 2, ..., d_j\}$ to form the i^{th} sub-model. d_j refers to the maximum index of the j^{th} dimension. This iterative process will be done during training and the set of n unique sub-models Θ will be built.

To illustrate the process better, let's see a simple case such as BERT$_{base}$ where we want to make a many-in-one network across the width and depth dimensions, $D = \{\text{Depth}, \text{Width}\}$. In this case, we have 12 layers and a hidden dimension size of 768. Suppose that $Depth$ corresponds to $j = 1$ and $Width$ corresponds to $j = 2$ in Eq. 1. For simplicity, let's use a discrete uniform distribution for sampling indices across these two dimensions. To create the first sub-network ($i = 1$), we need to sample b_1^1 uniformly from the set of natural numbers in the range of 1 to 12: $B_1 = \{1, 2, ..., 12\}$; and we need to sample b_2^1 from the range of 1 to 768: $B_2 = \{1, 2, 3, ..., 768\}$. Bear in mind that we can even choose a subset of B_1 and B_2 as the support set for sampling probability distribution. After these two samplings, we will have two truncated sets of parameters: $\theta_{Depth \downarrow b_1^1}$ and $\theta_{Width \downarrow b_2^1}$. The intersection of these two truncated parameters will give us the first sub-network: $\theta_1 = \theta_{Depth \downarrow b_1^1} \cap \theta_{Width \downarrow b_2^1}$.

Training Paradigm Regular training of neural networks concerns improving the performance of the whole model and usually this training is not aware of the performance of the sub-networks. In fact, in this scenario, if we extract and deploy the sub-models of the trained large model on a target task, we would experience a significant drop in the performance of these sub-networks compared with the main model. However in SortedNet, we propose a training method that allows for training sub-networks together with the main model in a stochastic way. The SortedNet paradigm leads to the following benefits:

- Search-free sub-model extraction: after training, by importance sorting of sub-models the best sub-model for a given budget can be selected without the need for search.
- Anytime: Each smaller sub-model is a subset of a larger one which makes switching between different sub-models efficient. This leads to an important feature of our SortedNet which is so-called *anytime* that is a network which can produce its output at any stage of its computation.
- Memory efficient: we train a many-in-one network where sub-models are all part of a single checkpoint, which minimizes storage requirement.

1.2 SortedNet Algorithm

In this subsection, we describe our proposed training algorithm. For training a SortedNet with n sub-models, at each iteration during training, a random index needs to be sampled from a pre-defined distribution: $b_j^i \sim P_{B_j}$. After finding the target sub-model θ_t^* at each iteration, we can use one of the following objectives to update the parameters of the selected sub-model:

- (Stochastic Loss) Only train the selected sub-model $f(x, \theta_t^*)$:
  $\min_{\theta_t^*} \mathcal{L} \triangleq \mathrm{L}(y, f(x, \theta_t^*))$ where L is the loss function for training the model on a given task (e.g. L can be a regular cross entropy loss) and y refers to the ground-truth labels.
- (Stochastic Summation) Train the sub-model $f(x, \theta_t^*)$ and all its targeted sub-models along each dimension. Let's assume that $\Theta^{\perp}(\theta_t^*)$ is the universal set for all targeted sub-networks of θ_t^*. Then the loss function can be defined as:
  $\min_{\Theta^{\perp}(\theta_t^*)} \mathcal{L} \triangleq \sum_{\theta \in \Theta^{\perp}(\theta_t^*)} \mathrm{L}(y, f(x, \theta))$

This way, one sub-model or a subset of sub-models are updated in each iteration. Alternatively, one can choose to train all the sub-models at each iteration which is costly in the large scale.

Experiments In this section, we discuss a set of experiments that we conducted to show the effectiveness and importance of sorting information and fixing a nested property.

Network	Width	FLOPs	NS-IN	LCS-p-IN	SortedNet-IN	NS-BN	LCS-p-BN (aka US)	SortedNet-BN
cpreresnet20 He et al. [2015] (CIFAR10)	100%	301M	88.67	87.61	**89.14**	79.84	65.87	**85.24**
	75%	209M	87.86	85.73	**88.46**	78.59	**85.67**	85.29
	50%	97M	84.46	81.54	**85.51**	69.44	65.58	**70.98**
	25%	59M	75.42	**76.17**	75.10	10.96	**15.78**	12.59
avg.	-	-	84.10	82.76	**84.55**	59.70	58.22	**63.52**

Table 1: Comparing the performance of state-of- the-art methods with Sorted-Net over CIFAR10 in terms of test accuracies.

As shown in table 1, we wanted to show our stochastic approach can outperform the state-of-the-art methods such as LCS (shown as LCS_p in table) Nunez et al. [2023], Slimmable Neural Network (NS) Yu et al. [2018], and Universally Slimmable Networks (US) Yu and Huang [2019]. To make the comparisons fair, we equalized the number of gradient updates for all models. We also tried to remove the impact of architecture design such as the choice of the normalization layers. Therefore, we tried to compare methods by different layer normalization techniques such as BatchNorm Ioffe and Szegedy [2015] and InstanceNorm Ulyanov et al. [2016]. In addition, we ensure that complementary methods such as Knowledge Distillation will not impact the results as these methods can be applied and improve the results independent of the method. As shown in the table, SortedNet demonstrates superior average performance compared to other methods, indicating its generalization across various settings such as different norms.

Model	Flops	Weights	Acc. MNLI	F1 QQP	Acc. QNLI	Acc. SST-2	Matthews Corr. CoLA	Spearman Corr. STS-B	F1 MRPC	Acc. RTE	avg.
Pre-trained Baselines											
$BERT_{BASE}(3\mathcal{L}_B)$	22.36 G	W_B	84.22 ± 0.32	87.37 ± 0.08	91.47 ± 0.21	92.61 ± 0.15	54.90 ± 0.79	88.08 ± 0.49	86.91 ± 0.82	62.96 ± 2.36	**81.07** ± 14.08
$BERT_{LARGE}(3\mathcal{L}_L)$	78.96 G	W_L	86.32 ± 0.09	88.36 ± 0.07	92.01 ± 0.29	93.21 ± 0.42	59.39 ± 1.45	88.65 ± 0.33	88.67 ± 0.75	68.23 ± 1.59	83.11 ± 12.33
Extracted Networks											
$BERT^{LARGE}_{BASE}(3\mathcal{L}_B)$	22.36 G	W_B	77.43 ± 0.08	84.88 ± 0.15	84.74 ± 0.34	84.98 ± 0.47	12.17 ± 1.62	78.33 ± 4.11	79.44 ± 0.93	55.23 ± 1.08	69.65 ± 25.18
Proposed Methods											
Sorted $BERT_{BASE}(\sim 1.5\mathcal{L}_B + 1.5\mathcal{L}_L)$	22.36 G	W_B^L	76.20 ± 0.02	83.58 ± 0.16	83.91 ± 0.18	83.26 ± 0.69	0.08 ± 0.18	70.75 ± 9.25	80.75 ± 1.29	52.85 ± 2.53	66.42 ± 28.76
Sorted $BERT_{LARGE}(\sim 1.5\mathcal{L}_B + 1.5\mathcal{L}_L)$	78.96 G	W_L^L	85.93 ± 0.33	87.28 ± 0.14	91.58 v 0.33	93.17 ± 0.26	57.08 ± 1.91	88.18 ± 0.68	87.06 ± 1.02	65.56 ± 1.41	81.98 ± 13.17
Sorted $BERT_{BASE}(\sim 3\mathcal{L}_B + 3\mathcal{L}_L)$	22.36 G	W_B^L	77.48	85.16 ± 0.02	84.96 ± 0.23	86.01 ± 0.62	12.58 ± 2.04	79.29 ± 2.80	78.96 ± 0.44	55.81 ± 1.37	70.03 ± 25.16
Sorted $BERT_{LARGE}(\sim 3\mathcal{L}_B + 3\mathcal{L}_L)$	78.96 G	W_L^L	86.12	88.26 ± 0.01	92.18 ± 0.28	93.49 ± 0.21	59.84 ± 1.35	88.85 ± 0.44	88.88 ± 1.10	68.45 ± 2.11	**83.26** ± 12.24

Table 2: The performance of BERT-base and Bert-large in the GLUE Benchmark over 5 runs for SortedNet (sharing weights across both models), pre-trained berts and different initialization.

We also investigate whether SortedNet is applicable to more complex dimensions other than width. For example, can we utilize the SortedNet for sorting the Attention Heads Vaswani et al. [2017]? To achieve this, we conducted an experiment over BERT-large Devlin et al. [2019] in which we tried to sort the information across multiple dimensions simultaneously, including the number of layers, hidden dimension, and attention heads. In other words, we tried to sort information over Bert-large and Bert-base as Bert-base can be a subset of the Bert-large. As shown in table 2, we extracted a Bert-base from a Bert-large model, and reported the performance. Additionally, we

highlighted the number of training updates with respect to each objective function in front of each model. For example, in the last row (Sorted $BERT_{LARGE}$), we approximately trained our Sorted model half of the times ($\sim 3Epochs$) over the objective function of Bert-base ($\mathcal{L}_B$) and the other half of the times over the objective function of Bert-large ($\mathcal{L}_L$) in an iterative random manner as introduced in the previous section. The learned Bert-base performance with these methods is still around 10% behind the pre-trained base, but we argue that this might be the value of pre-training. To investigate the impact, one should apply the SortedNet during pre-training, which we will leave for future research. However, the performance of the learned Bert-large is on par with an individual Bert-large, which suggests sharing the weights does not necessarily harm learning. It seems, however, that the secret sauce to achieve a similar performance is that we should keep the number of updates for each objective the same as the individual training of Bert-large and Bert-base.

2 Conclusion

In summary, this paper proposes a new approach for training dynamic neural networks that leverages the modularity of deep neural networks to switch between sub-networks during inference efficiently. Our method sorts sub-networks based on their computation/accuracy and train them using an efficient updating scheme that randomly samples sub-networks while accumulating gradients. The stochastic nature of our proposed method is helping our algorithm to generalize better and avoid greedy choices to optimize many networks at once robustly. We demonstrate through experiments that our method outperforms previous dynamic training methods and yields more accurate sub-networks across various architectures and tasks. The sorted architecture of the dynamic model proposed in this work aligns with sample-efficient inference by allowing easier samples to exit the inference process at intermediate layers. Exploring this direction could be an interesting area for future work.

References

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33: 9782–9793, 2020.

Elvis Nunez, Maxwell Horton, Anish Prabhu, Anurag Ranjan, Ali Farhadi, and Mohammad Rastegari. Lcs: Learning compressible subspaces for efficient, adaptive, real-time network compression at inference time. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3818–3827, January 2023.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. URL `https://aclanthology.org/2023.eacl-main.239`.

Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754. PMLR, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.