Compact Multi-level Sparse Neural Networks with Input Independent Dynamic Rerouting

Minghai Qin[1*], Tianyun Zhang[2*], Fei Sun[3*], Yen-Kuang Chen[3], Makan Fardad[4],
Yanzhi Wang[5], Yuan Xie[3]

[1]Western Digital Research [2]Cleveland State University [3]Alibaba DAMO Academy
[4]Syracuse University [5]Northeastern University

Abstract—Deep neural networks (DNNs) have shown to provide superb performance in many real life applications, but their large computation cost and storage requirement have prevented them from being deployed to many edge and internet-of-things (IoT) devices. Sparse deep neural networks, whose majority weight parameters are zeros, can substantially reduce the computation complexity and memory consumption of the models. In real-use scenarios, devices may suffer from large fluctuations of the available computation and memory resources under different environment, and the quality of service (QoS) is difficult to maintain due to the long tail inferences with large latency. Facing the real-life challenges, we propose to train a sparse model that supports multiple sparse levels. That is, a hierarchical structure of weights are satisfied such that the locations and the values of the non-zero parameters of the more-sparse sub-model are a subset of the less-sparse sub-model. In this way, one can dynamically select the appropriate sparsity level during inference, while the storage cost is capped by the least sparse sub-model. We have verified our methodologies on a variety of DNN models and tasks, including the ResNet-50, PointNet++, GNMT, and graph attention networks. We obtain sparse sub-models with an average of 13.38% weights and 14.97% FLOPs, while the accuracies are as good as their dense counterparts. More-sparse sub-models with 5.38% weights and 4.47% of FLOPs, which are subsets of the less-sparse ones, can be obtained with only 3.25% relative accuracy loss. In addition, our proposed hierarchical model structure supports the mechanism to inference the first part of the model with less sparsity, and dynamically reroute to the more-sparse level if the real-time latency constraint is estimated to be violated. Preliminary analysis shows that we can improve the QoS by one or two nines depending on the task and the computation-memory resources of the inference engine.

Index Terms—artificial intelligence, deep neural networks, weight pruning

I. Introduction

Deep neural networks (DNNs) have been widely used in many areas and have significantly changed our lives such as object recognition, autonomous driving, and language translation. The fast development of DNNs is largely credited to the rapid growing of computation power, such as Graph Processing Units (GPUs) and Tensor Processing Units (TPUs), that enables the efficient search of the large and powerful DNN models. On the other hand, there are a variety of environments that DNNs are deployed on the edge, where the computation capacity is limited. For example, mobile phones are increasingly integrating more artificial intelligence features that require models to perform inference on a variety of tasks, such as picture beautification, image/video super resolution, and 3-D augmented reality. Internet-of-Things (IoTs) devices are becoming more popular to be customized to understand the consumers' habits. The state-of-the-art DNN models for these complicated tasks are usually extremely large, *i.e.*, containing huge amount of weights in the model, and thus not suitable to be directly deployed on mobile devices or embedded systems.

One of the widely-used approaches to cope with the complexity is to reduce the size of the models by sparsification: a technique to set the majority of the weights in the models to zero values (called *pruning*) through carefully designed algorithms, software, and hardware implementations [1], [2]. Since the pioneering work [1], there are growing academic and industrial interests in this area. The principle of DNN model sparsification lies in the ability to identify and prune the non-critical weights, while minimally reducing the accuracy loss by adjusting the values of the remaining weights.

Real world applications in the wild need to handle dramatically different environments, which challenges the mobile inference engines [3]. For example, a mobile phone may allocate more computation capacity and power budget when it is fully charged and is at low temperature. But when its thermal limit is reached, the phone is forced to reduce the processor frequency to cool down the device, which is a process called *thermal throttling*. As the result, its computation capacity is significantly reduced. Thus, the inference latency distribution of DNN models may show two peaks, one at its highest computation capacity, and one at the thermal throttled state [4]. In addition, an experiment has been performed in [5] to measure the inference latency of a layer in a model on A11 iPhone mobile chipset. Even though the mean latency is 2.02ms with a standard deviation of 1.92ms, some inferences take 16ms or longer. These inferences with large latency, though small in percentage, may be easily visible to the customers. Thus, when a model is deployed to millions of devices, it is customary to optimize the 99th or 99.9th percentile latency to improve the quality of service (QoS). The traditional remedy to the long tail latency inferences is to skip frames in a series of inferences (*e.g.* in video analysis), allocating more time to the remaining inferences. This approach has an obvious drawback of the degraded quality visible to users.

As the result, one efficient sparse neural network model

---

*Equal contribution.
[1]Work done while working at Alibaba DAMO Academy.
[2]Work done while interning at Alibaba DAMO Academy.

may still be insufficient to handle the changing environment with different optimization objectives. It is desirable to prepare two models: a relatively large model that is more accurate but cost more computation, and a relatively small model that is more efficient but less accurate. Admittedly, it is trivial to train two separate models at different sparsity levels and store them in the device. However, the storage and memory cost of the separate models are not satisfactory, which is more severe when more than two sparsity levels are desired. It is much more appealing if multiple sparse models can be stored one on top of another, consuming no extra storage and memory compared to the least-sparse model.

In this work, we explore methodologies to address large variations of the inference latency on the devices in the wild **within** neural network models. We propose a training and inference methodology that creates sparse neural network models with multiple sparsity levels. The different sparsity levels follow a **hierarchical structure** such that the locations and values of the non-zero weights of the more-sparse sub-model is a subset of the less-sparse sub-model. We call the less-sparse sub-model as the **small** sub-model and the more-sparse sub-model as the **tiny** sub-model in this work. Figure 1 shows an overview of our proposals. In the inference phase, the tiny sub-model (weights colored in orange) is a subset of the small one (weights colored in blue and orange) and thus compact representations can be designed. The appropriate sub-model may be selected based on the environment. For example, the small sub-model may be selected when the processor is at its full capacity, and the tiny sub-model is selected when the processor is at its thermal throttled state. In the training phase, we propose a "tiny-dense-small" training methodology that derives the small sub-model from the tiny sub-model. The resultant small sub-model performs as good as its dense counterpart, sometimes even better.

With multiple levels of sparsity, the preferred level may be selected before each inference to reduce the the 99th or 99.9^{th} percentile latency. For some critical applications with hard real-time constraint, making the selection before the inference may not be sufficient. We provide a mechanism to dynamically reroute the inference from the small to the tiny sub-models in the middle of the inference to further reduce the long tail latency. We also show that the accuracy after rerouting is better than the tiny sub-model. Depending on the rerouting locations, the accuracy improvement may vary.

The contributions of the paper are summarized as follows.

- We propose a methodology to create sparse sub-models of a deep neural network at multiple sparsity levels, whose weights follow a logically hierarchical structure that the values and locations of the weights in the tiny sub-model are a subset of those in the small sub-model. This weight hierarchy brings 20% to 30% savings comparing with storing two separate sparse models.
- We present a physically hierarchical and compact sparse weight representation that has almost the same cost as storing the small sub-model, meanwhile we can efficiently extract from it all weights in the tiny sub-model.

- We show that our methodology is universal and can be applied to different kinds of neural network models, such as 2-D image classification (ResNet-50), 3-D object recognition (PointNet++), machine translation (GNMT), and transductive learning (Graph attention networks). The results are reported as the geometric mean of the four models. We obtain small sub-models with 13.38% weights and 14.97% FLOPs with accuracy as good as their dense counterparts in [6]–[9]. Tiny sub-models, which are subsets of the small ones, cost 5.38% weights, 4.47% of FLOPs, and result in only 3.25% relative accuracy loss.
- We propose a dynamic inference rerouting method such that the model can partially inference the small sub-model in the beginning and switch to the tiny sub-model without throwing away the activation already computed. Experiments on ResNet-50 and GNMT have shown that the dynamic inference rerouting achieves higher accuracy than the tiny sub-model at different rerouting positions.
- Our proposed multi-level sparsity solution has the potential to improve the QoS given a fixed latency bound. Based on the latency distribution from real devices, we have created a model to estimate the QoS for the small and tiny sub-models. With the same latency that the small sub-model delivers two nines of QoS (99th percentile), the tiny sub-model is capable of delivering four nines of QoS (99.99^{th} percentile). As far as we know, this is the first work looking into the problem on sparse models.

II. RELATED WORKS

Network pruning [10], [11] can reduce the number of non-zero weights in neural network models. Some works focus on *irregular* pruning where no constraints are imposed on the locations of non-zero weights. [12] proposes a magnitude-based importance metric and iteratively prune weights with small magnitude. [13] uses ℓ_0 regularization and [14] manages to solve the non-convexity problem of ℓ_0 by ℓ_1 approximation. [15], [16] utilize an effective technique in optimization theory to improve the sparsity based on ℓ_0 regularization. On the other hand, *structured* pruning imposes constraints on the locations of non-zero weights and thus reduces irregularity. For example, filter-wise pruning and shape-wise pruning [14] can remove rows and columns in matrix-matrix multiplications, which translates to computation reduction in GPU platforms. Recent works explore some special dimensions and propose pattern pruning and kernel-wise pruning [17]. All previous works focus on a single sparse level and our work differs from them by creating multiple sparsity levels in a single model. Our work uses ADMM [18] algorithm to irregularly prune the dense model to achieve the maximum accuracy.

Some other works propose to prune and grow the DNN back and forth. Transitioning from dense to sparse then back to dense has been shown to improve the accuracy of dense models [19], [20]. In [21], a dense network composed of multiple sparse networks is proposed to accomplish multiple tasks with different sparse subsets. While their goals are to train dense
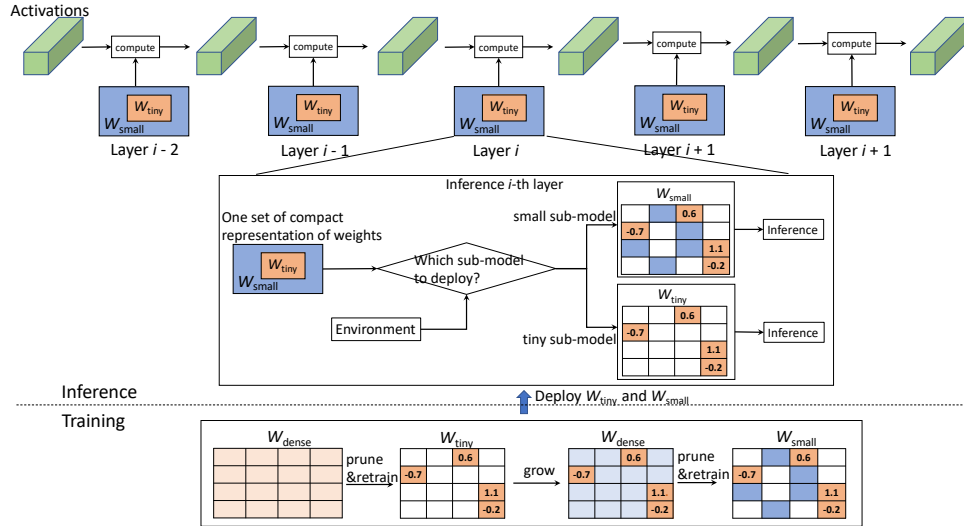
556

Fig. 1. Inference and training methodology for the proposed compact multi-level sparse neural networks with weight hierarchy. During inference, the compute engine selects the desired sparsity level based on the environment. During training, a tiny-dense-small methodology is proposed to optimize tiny and small sub-models. Empty boxes denote zero-valued weights and colored boxes denote non-zero weights. Once the tiny sub-model is acquired, their weights are never changed in the rest of training process.

networks, our work differs in that the transition from tiny to dense is an intermediate step and our final goal is to obtain multiple sparse sub-models with weight hierarchy. [22]–[24] propose to grow part of connections based on the gradients of loss and prune it back to the desired sparsity. Similar efforts with different growing criterion are made in [25]–[27]. While all the non-zero weights in the existing grow-and-prune works are free to be updated, our work fundamentally differs in that the tiny sub-model is permanently frozen during all processes to guarantee the weight hierarchy.

[28] proposes an efficient progressive shrinking method to train a large super-network based on some architectures and deploy only a portion of it depending on the platform (mobile, cloud, or edge device). Other works [29]–[33] design to early exit based on some confidence score. While they focus on searching for dense networks, our work is orthogonal to them in the search of sparse sub-models. Another difference from [28] is that our sparse weight hierarchy is a tight restriction while weights of two dense elastic kernels in [28] are related by matrix transformations and are fine-tuned.

III. METHODOLOGY

In this section, we present our training methodologies to acquire the small and tiny sub-models whose weights follow a hierarchical structure. We enforce the sparse weights in convolution, matrix-vector (MV) multiplication, and matrix-matrix (MM) multiplication operators to have multiple levels of sparsity. The weights in these operators represent the majority of the parameters in a model. The biases in those operators, if exist, remain dense and are not constrained. The weights and biases in other operators, such as batch normalization are also not constrained. The parameters in those operators only

contribute a tiny fraction of the model size and floating point operations (FLOPs).

Throughout the paper, we denote a DNN model (or a sub-model) by M, denote the set of non-zero weights of M by $W(M)$, and denote the locations of the non-zero weights of M by $L(M)$. The pair $(L(M), W(M))$ uniquely defines the weight structure of the DNN model M.

A. A tiny-dense-small training process

First, we pre-train a dense DNN model into a tiny one with good learning accuracy to start with. This is shown in the first step in the training phase of Figure 1. It is known that different layers may have different redundancies in maintaining the accuracy when they are sparsified [14]. In this paper, we try to limit this degrees of freedom for a fair comparison, such that all layers in one sparse sub-model have the same sparsity level. In order to obtain the tiny sub-model, we use the alternating direction method of multipliers (ADMM) [18] algorithm that has been shown to perform well in restraining the number of non-zero weights in each layer.

The second step is to grow the tiny sub-model back to a dense DNN model M_d by fixing those weights $W(M_t)$ in the location $L(M_t)$. It is shown in the second step in the training phase of Figure 1. Note that the non-zero weights in the orange boxes are frozen in this step (and the third step as well). All other weights are trainable and can be updated to maximally increase the accuracy of the dense DNN model. Note that the first two steps are similar to the dense-sparse-dense training method [19] except for two differences. First, $W(M_t)$ in the location $L(M_t)$ of the dense model is permanently fixed to guarantee that the dense model is a superset of the tiny sub-model. Second, the pruned weights in
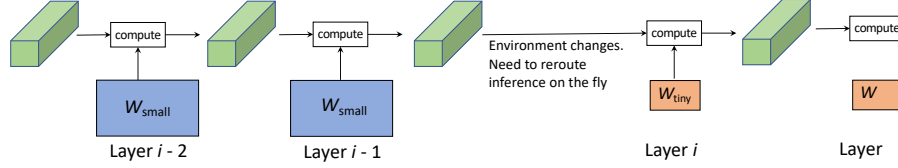
Fig. 2. Dynamic rerouting of multi-level sparse neural networks from the small to the tiny sub-models.

the tiny sub-model during re-training to the dense model are sometimes required to be re-initialized as random numbers, as opposed to zero values in [19], since some of our tiny sub-models are too sparse for many weights to be updated in the backward propagation. For example, if the entire kernel of a convolutional layer or the entire row of a weight matrix is pruned, re-training by initializing them as zeros will not update them effectively.

The last step is to prune the dense DNN model back to a small sub-model. It is shown in the third step in the training phase in Figure 1. During this pruning, all weights $W(M_t)$ in the location $L(M_t)$ are fixed. It means that the values of weights $W(M_t)$ in the location $L(M_t)$ are never changed once we obtain the tiny sub-model. This is illustrated by the orange boxes of the last three weight matrices in the training phase in Figure 1. In this way, it is guaranteed that the final small sub-model M_s satisfies $L(M_t) \subset L(M_s)$ and $W(M_t) \subset W(M_s)$. This pruning process again utilizes a modified version of the ADMM algorithm, where the projection of the dense tensor to the sparse tensor requires to keep $W(M_t)$ in $L(M_t)$ intact.

After this step, we have the tiny and small sub-models such that their weights follow a hierarchical structure, i.e., $L(M_t) \subset L(M_s)$ and $W(M_t) \subset W(M_s)$.

We use two-level sparsity to demonstrate our "tiny-dense-small" training methodology. In order to obtain more than two levels, the training method can be generalized to "micro-dense-tiny-dense-small" for triple-level sparsity, where the "micro" sub-model can be three to four times smaller than the tiny one. In practice, a three to four times difference between the adjacent sparsity levels makes a good trade-off between accuracy and efficiency.

B. Weight representations

Due to the subset relationship between the small and tiny sub-models, storing both sub-models consumes almost the same storage as storing the small sub-model along. The sparse matrices can be represented in many different formats, such as compressed sparse row (CSR), compressed sparse column (CSC), dictionary of keys (DOK), list of list (LIL), coordinate list (COO) [34]. We use CSR [35] to describe the efficiency of storing both small and tiny sub-models, but the same approach can be easily adapted to other formats.

The CSR format represents a sparse matrix A using three arrays, called $data$, $index$, and ind_ptr. Assume A has m rows and n columns, and the number of non-zero elements in A is N, the array $data$ stores all the numerical non-zero

values of A in the row-scanning order and thus the length of $data$ is N. The $index$ array stores the column indices of the non-zero elements with the same order as in $data$, and thus the length of $index$ is also N. The ind_ptr array, whose length is $m + 1$, stores the index pointer pointing to the first value in each row in $data$ and $index$. Note that state-of-the-art pruning algorithms is capable to prune modern DNNs by $2\times$ to $10\times$ with negligible accuracy loss. Hence, N is still much larger than m or n and thus the size of ind_ptr is negligible.

The CSR format can be illustrated by the following example. Suppose the sparse matrix A in Eq (1) is a weight matrix in a small sub-model, B in Eq (2) is a subset of A, referring to the weight matrix in the corresponding tiny sub-model.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 6 & 4 \end{bmatrix}, \quad (1)$$

$$B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 \end{bmatrix}, \quad (2)$$

To represent A by zero-based numbering,

$data = [1, 2, 8, 7, 3, 5, 9, 6, 4], index = [1, 0, 3, 6, 2, 5, 4, 6, 7],$
$ind\_ptr = [0, 1, 4, 6, 9].$

To decode the CSR format, ind_ptr first splits $data$ and $index$ into rows by checking the number of non-zero elements in each row $(ind\_ptr[i] - ind\_ptr[i-1]$, for $i = 1, \ldots, m)$. In the above example, the four rows have $(1, 3, 2, 3)$ non-zero elements, thus $data$ and $index$ are split into $[[1], [2, 8, 7], [3, 5], [9, 6, 4]]$ and $[[1], [0, 3, 6], [2, 5], [4, 6, 7]]$, respectively. Note that the element order in the same row can be arbitrary in standard CSR.

In order to represent a sub-matrix B in Eq (1) where $B \subset A$, we first permute the values in $data$ such that it is still in row-scanning order but the values in B appear before those values in $A \backslash B$ in each row. We also allocate an additional array end_ind_ptr of length m. The entry i in end_ind_ptr stores the index of the last non-zero value plus one in the $data$ and $index$ array for row i.

Thus, we have a compact representation of A and B as

$data = [1, 8, 7, 2, 3, 5, 6, 9, 4], index = [1, 3, 6, 0, 2, 5, 6, 4, 7],$
$ind\_ptr = [0, 1, 4, 6, 9], end\_ind\_ptr(B) = [1, 3, 5, 7],$
$end\_ind\_ptr(A) = [1, 4, 6, 9].$

558

Note that $end\_ind\_ptr(A)$ is the same as ind_ptr by removing the first element which is a constant 0. Thus it does not require a separate array, and can be stored as end_ind_ptr with an offset one. The decoding of A (or B) first splits $data$ according to ind_ptr and $end\_ind\_ptr(A)$ (or $end\_ind\_ptr(B)$). For the i-th row, the starting index in $data$ is $ind\_ptr[i]$ and the ending index is $end\_ind\_ptr\_A[i]$ (or $end\_ind\_ptr\_B[i]$), exclusively. For example, to decode B, $data$ and $index$ are split into $[data[0:1), data[1:3), data[4:5), data[6:7)] = [[1], [8,7], [3], [6]]$ and $[index[0:1), index[1:3), index[4:5), index[6:7)]] = [[1], [3,6], [2], [6]]$, respectively. Then the non-zero values of each row in B can be identified.

Since the storage cost of ind_ptr and end_ind_ptr are both negligible compared with $data$ and $index$, using the proposed method to store both A and B (both the small and tiny sub-models) has almost the same cost of storing A (the small sub-model). Also note that by adding more end_ind_ptr arrays and re-permuting $data$ and $index$, we can increase the number of sparse sub-models to an arbitrary number while the the storage cost of all sub-models is almost the same as the least sparse sub-model.

C. Dynamic inference rerouting between small and tiny sub-models

The tiny-dense-small method creates two sparse sub-models with different weight sparsity and computation complexity. However, selecting a preferred sparsity level before each inference might not be sufficient for some time-critical applications with hard real-time constraint. If the processor unexpectedly slows down after the start of inferencing the small sub-model, it is preferred to detect such situation in the first few layers and *dynamically* reroute to the tiny sub-model for the rest of the layers to reduce the latency. As being said, the rerouting can be initiated if the first part of the small sub-model suffers extra unexpected latency and hope to complete the inference in time without totally throwing away the computation already done in the small sub-model.

We propose a dynamic inference rerouting such that we can inference the first few layers in the small sub-model and use the intermediate results (activation or feature maps) to reroute to the corresponding layers in the tiny sub-model. We constrain the rerouting to be from small to tiny instead of the opposite direction, since it is preferred to inference with the small sub-model in the first place to maintain accuracy. Figure 2 illustrates our proposed dynamic rerouting. To enable this functionality during training, if a rerouting location at the i-th layer is desired, we fix the small sub-model (including all parameters such that all weights, biases, batch normalization related weights, biases, running means, and running variances) until the i-th layer and train the tiny sub-model from the $(i+1)$-st layer to the end, where only biases and batch normalization related weights can be changed, *i.e.*, the weights in convolution, MV multiplications and MM multiplications are still following the weight hierarchy from the small sub-model. Experiments show that the propose dynamic rerouting

not only enables to handle unexpected latency during the first few layers of inference, but also makes a good trade-off between the accuracy and the computation complexity (e.g., FLOPs in deployment).

IV. EXPERIMENTS

We have evaluated the proposed methodology on four different learning tasks: 2-D image recognition, 3-D object recognition, machine translation, and transductive learning. All experiments are performed on the PyTorch [36] framework. The results are shown in Table I, and they have validated the effectiveness of our methodology.

A. 2-D image recognition

We use a popular deep convolutional neural network, ResNet-50 model, on ImageNet dataset to demonstrate the results on hierarchically structured sparse deep neural networks.

The first section in Table I shows our results compared to the dense ResNet-50. The tiny-dense-small method generates a tiny sub-model and a small sub-model by keeping 5.43% and 20.36% of the weights, respectively. The resultant percentage of FLOPS are 8.20% and 22.69%, respectively. It is observed that the accuracy of the small sub-model is slightly better than the dense counterpart [6] and the tiny sub-model reduces 91.8% FLOPs with only 2.6% top-1 accuracy loss. Our tiny sub-model has 5% of weights of a dense counterpart, which achieves 1.4% higher top-1 accuracy than Slimmable neural network [37] with 27.06% of weights of a dense counterpart. The results on dynamic rerouting is shown in the next four rows. These four results correspond to four different switching locations, which are after the first, second, third, and fourth residual blocks [6]. It is observed that switching at a later point better preserves accuracy while keeping higher percentage of weights and FLOPs.

Note that the percentage of weights are not precisely equal to the FLOPs due to three main reasons. First, the first convolutional layer before the residual blocks is not pruned but its weights are shared by all sub-models. Second, the distributions of weights and FLOPs across ResNet-50 layers are very different. Third, the biases and batch normalization related parameters (0.4% of a dense model) are not pruned as they barely contribute to the FLOPs.

B. Machine translation

We use Google neural machine translation (GNMT) [8] model on WMT En $\rightarrow$ De dataset to demonstrate the proposed method on machine translation tasks. GNMT consists of eight layers of long short-term memory (LSTM) in both encoder and decoder and a classifier. GNMT contains over 127 millions of parameters excluding 33 millions word embedding, and the proportion of MV and MM multiplication related weights is 99.9%. The word embedding is not pruned because pruning it may not easily translate to FLOP savings. In our experiments, the word embedding is shared between the encoder and the decoder, as well as the small and the tiny sub-models.

Base model	Sub-models	Accuracy		Weights percentage	FLOPs percentage
ResNet-50	Dense [6]	76.10%	92.90%	100%	100%
	Slimmable NNs [37]	72.10%	-	27.06%	26.83%
	Slimmable NNs [37]	65.00%	-	7.84%	6.78%
	Small sub-model	76.45%	93.10%	20.36%	22.69%
	Tiny sub-model	73.50%	91.72%	5.43%	8.20%
	Small-tiny reroute 1	73.66%	91.71%	5.56%	10.84%
	Small-tiny reroute 2	73.84%	91.94%	6.26%	14.48%
	Small-tiny reroute 3	74.41%	92.16%	10.40%	19.99%
	Small-tiny reroute 4	76.14%	93.04%	19.16%	22.69%
GNMT	Dense [8]	24.61		100%	100%
	Small sub-model	24.47		36.53%	20.02%
	Tiny sub-model	23.06		24.63%	5.02%
	Small-tiny reroute	23.65		28.94%	10.45%
PointNet++	Dense [7]	91.9%		100%	100%
	Small sub-model	91.94% ± 0.14%		4.25%	3.00%
	Tiny sub-model	89.92% ± 0.21%		1.99%	0.72%
GAT	Dense [9]	83.00%		100%	100%
	Small sub-model	84.80%		10.14%	36.81%
	Tiny sub-model	81.00%		3.15%	13.48%

TABLE I
ACCURACY OF DENSE, SMALL, AND TINY SUB-MODELS OF RESNET-50, GNMT, POINTNET++, AND GAT. RESNET-50S SHOW TOP-1 AND TOP-5 ACCURACY, GNMTS SHOW BLEU SCORES, POINTNET++'S AND GATS SHOW TOP-1 ACCURACY.

We prune 80% of the prunable weights for the small sub-model, resulting to 36.53% of the weights. The main difference is due to the word embedding, which remains dense. The FLOPs of the small sub-model is reduced to 20.02% of the dense model. The 0.02% extra FLOPs is due to the small attention matrix multiplications, which remains dense. The small sub-model slightly decreases the BLEU score from the dense model (from 24.61 to 24.47).

Similarly, we prune 95% of the prunable weights for the tiny sub-model, resulting to 24.63% of the weights and 5.02% of the FLOPs. It reduces the BLEU score to 23.06. When the model is rerouted from the small sub-model in the encoder to the tiny sub-model in the decoder, the remaining weights are 28.94% and the remaining FLOPs are 10.45%. The FLOPs is calculated assuming the input and output sequence length of 16 with a beam size of one. Its BLEU score (23.65) falls between the tiny and small sub-models.

C. 3-D object recognition

We use PointNet++ with multi-scale grouping (MSG) [7] on ModelNet40 [38] dataset for the 3-D object recognition. PointNet++ with MSG consists of several 1×1 convolutional layers (MM multiplications) and three fully connected layers (MV multiplications).

Compared to the dense PointNet++ [7], the tiny-dense-small method generates a tiny sub-model and a small sub-model by keeping 1.99% and 4.25% of weights, respectively. The resultant percentage of FLOPS is reduced to 0.72% and 3% of their dense counterparts, respectively. Even larger difference is observed in the distribution of weights and FLOPs in PointNet++ layers. Since PointNet++ MSGs involves random sampling operations, we test the model 50 times and report

their means and standard deviations. The resultant small sub-model is again slightly better than its dense counterpart.

D. Transductive learning on graphs

We use graph attention networks (GATs) [9] on the Cora citation dataset [39] to demonstrate the propose method on transductive learning tasks. The GATs consist of eight attention heads and an output classifier, all of which are MV and MM multiplications. We follow the same setup [9] in partitioning training nodes, validation nodes, and test nodes.

We observe from Table I (last three rows) that the proposed tiny-dense-small method generates a small sub-model with roughly 10% of weights and 36% FLOPs while the accuracy is improved compared to the dense counterpart. A more-sparse tiny GAT sub-model has about 3% of weights and 13% FLOPs and the accuracy is only reduced by 2%. The difference in percentage of weights and FLOPs is due to the attention layers which remain dense.

E. Weight savings compared with two separate sparse models

A benefit of introducing weight hierarchy is to reduce the memory and storage cost of storing multiple sub-models. If there is no weight hierarchy, we need to store small and tiny models separately. In Figure 3, we present the percentage of weight savings when using hierarchical sub-models over using separate models. An average of 20% to 30% of storage reductions are expected from saving two models separately. The overall storage savings may be larger if CSR format is used and the storage for the *index* and *ind_ptr* arrays are counted. It should also be noted that the savings are amplified when more than two sparsity levels are desired.
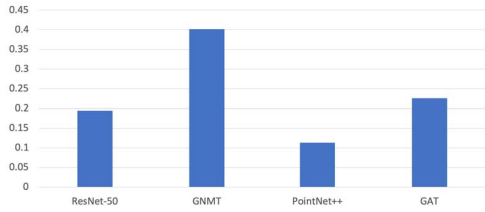
560

Fig. 3. Percentage of weights savings of the sub-models with weight hierarchy over two separate sparse models (small and tiny).
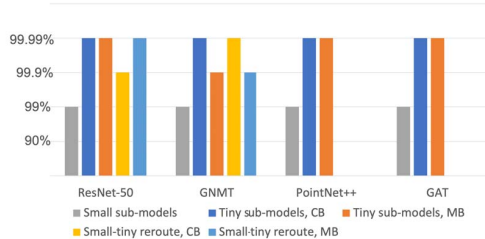

Fig. 4. QoS comparison of small sub-models, tiny sub-models, and small-to-tiny reroutes in computation-bound (CB) or memory-bound (MB) scenarios.

F. Quality of Service (QoS) improvement

In order to verify that dynamically selecting the small and tiny sub-models before and during inferences can improve the quality of service, we have collected the the latency of over one million inferences on iPhone X devices on a proprietary model [3]. The distribution of the latency is similar to the one reported in [5]. We hypothesize that the latency distributions of the models reported in Table I are similar to the collected distribution. We further linearly correlate the number of FLOPs and the number of weight parameters to the execution latency, depending on whether the inference is computation-bound (CB) or memory-bound (MB). We estimate that for the inference latency corresponding to the 99th percentile of the small sub-models, the tiny sub-models are capable to reach 99.99^{th} percentile most of the time, on either computation-bound or memory-bound inference systems, as shown in Figure 4. The quality-of-service (QoS) reflects the percentage of inferences (e.g., 99.99%) within a certain latency requirement. Our result illustrates that if the decision to inference the small or tiny sub-model is made before each inference based on the runtime condition, the QoS can be improved by two nines (*i.e.*, from 99% to 99.99%). Similarly, in the case that the runtime condition is changed during inference and the inference is rerouted from small sub-models to tiny sub-models, the QoS can still be improved by at least one nine (*i.e.*, from 99% to 99.9%). This experiment on QoS estimation has preliminarily validated the usefulness of our approach in improving the QoS in the wild.

G. Comparison to existing dynamic DNNs

Several recent works [37], [40]–[42] focus on "input dependent" dynamic inference, i.e., the path to inference

is determined by the individual input, where the "easy" inputs go through simple paths and thus reduce computation on average. Our "dynamic rerouting" is fundamentally different in that ours is input independent, i.e., any input data can be switched from small to tiny sub-models from any designated switching point (e.g., 4 points for ResNet50). The switching decision is based on the run-time status such as the latency before each switching point, independent of the input data. Input independent rerouting is much simpler and more practical than the NN based conditional rerouting in existing literatures. Second, our DNN models are much more sparse than the existing works. For example, our tiny ResNet50 has 5% of the weights of the dense counterpart, while SkipNet [40] has 88%. Slimmable neural network [37] has one result that achieves 1.4% *less* accuracy than our tiny ResNet50 with $5\times$ parameters (27.06% vs 5.43%). CondConv [41] or WeightNet [42] are even larger than the dense ResNet50.

V. DISCUSSIONS

A. Discussion on more than two levels of sparsity

We use two-level sparsity to demonstrate our "tiny-dense-small" training methodology. In practice, two-level sparsity has demonstrated a good trade-off between accuracy and DNN complexity. In order to obtain more than 2 levels, we can generalize the training method to "micro-dense-tiny-dense-small" for triple-level sparsity, where the "micro" sub-model can be 3 to 4 times smaller than the tiny one. 3 to 4 times difference between adjacent sparsity levels makes a reasonable trade-off between accuracy and complexity.

B. The reason of using ADMM on weight pruning

Any pruning method can be used such as magnitude-based, L1-regularization based, etc. We use ADMM algorithm because it enables dynamic penalty on the weights . Thus, it receives higher accuracy than the fixed regularization methods, in particular at large sparsity (i.e., tiny sub-models). As a result, we achieve the state-of-the-art accuracy for the presented sparsity levels. As a comparison, if one uses magnitude-based pruning, the accuracy of the tiny ResNet50 is only 68%. Using ADMM, we presented the tiny ResNet50 with 73.60% for the same setting.

VI. CONCLUSIONS

Realized by the large fluctuations in the runtime environment and the difficulty of maintaining QoS with one model due to the long tail inferences with large latency, we propose a compact multi-level sparse neural network with a hierarchical structure of weights such that the more-sparse sub-model is a subset of the less-sparse sub-model. This hierarchy enables us to select the sub-model of the desired sparsity level before and during each inference, while the storage cost of multiple sub-models is almost the same as the least sparse one. Experiments show that the small sub-models can reduce the number of weights and FLOPs by an average of $7\times$ without accuracy degradation and a further $3\times$ reduction can be realized at the cost of only 2-3% of relative accuracy. In this work, we have

validated our methodology from the algorithm perspective. Our next step is to translate the gain into latency reduction via structured pruning and hardware/software co-design of the inference system.

VII. ACKNOWLEDGMENTS

REFERENCES

[1] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learning Representations*, 2016.

[2] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *Proc. Int. Symp. Computer Architecture*, 2016, pp. 243–254.

[3] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu, C. Lv, and Z. Wu, "MNN: A universal and efficient inference engine," in *MLSys*, 2020.

[4] B. Gaudette, C.-J. Wu, and S. Vrudhula, "Optimizing user satisfaction of mobile workloads subject to various sources of uncertainties," *IEEE Trans. Mobile Computing*, vol. 18, no. 12, pp. 2941–2953, 2018.

[5] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *Proc. Int. Symp. High Performance Computer Architecture*, 2019, pp. 331–344.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[7] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Conf. Neural Information Processing Systems*, 2017, pp. 5099–5108.

[8] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, and M. Norouzi, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *Proc. Int. Conf. Learning Representations*, 2018.

[10] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal Brain Surgeon," in *Proc. Conf. Neural Information Processing Systems*, 1993, pp. 164–171.

[11] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Conf. Neural Information Processing Systems*, 1990, pp. 598–605.

[12] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Conf. Neural Information Processing Systems*, 2015, pp. 1135–1143.

[13] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," in *Proc. Int. Conf. Learning Representations*, 2018.

[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Conf. Neural Information Processing Systems*, 2016, pp. 2074–2082.

[15] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Proc. European Conf. Computer Vision.*, 2018, pp. 184–199.

[16] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "ADMM-NN: An algorithm-hardware co-design framework of DNNs using alternating direction methods of multipliers," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 925–938.

[17] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," in *Proc. AAAI Conf. on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5117–5124.

[18] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[19] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, B. Catanzaro, and W. J. Dally, "DSD: Dense-sparse-dense training for deep neural networks," in *Proc. Int. Conf. Learning Representations*, 2017.

[20] A. Prakash, J. Storer, D. Florencio, and C. Zhang, "Repr: Improved training of convolutional filters," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2019, pp. 10 666–10 675.

[21] A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2018, pp. 7765–7773.

[22] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," in *IEEE Trans. Computers*, vol. 68, no. 10, Oct. 2019, pp. 1487–1497.

[23] X. Dai, H. Yin, and N. Jha, "Grow and prune compact, fast, and accurate LSTMs," *IEEE Trans. Computers*, vol. 69, no. 3, pp. 441–452, Mar. 2020.

[24] X. Dai, H. Yin, and N. K. Jha, "Incremental learning using a grow-and-prune paradigm with efficient neural networks," *arXiv preprint arXiv:1905.10952*, 2019.

[25] S. Hassantabar, Z. Wang, and N. K. Jha, "SCANN: Synthesis of compact and accurate neural networks," *arXiv preprint arXiv:1904.09090*, 2019.

[26] X. Du, Z. Li, and Y. Cao, "CGaP: Continuous growth and pruning for efficient deep learning," *arXiv preprint arXiv:1905.11533*, 2019.

[27] W. Wen, F. Yan, and H. Li, "Autogrow: Automatic layer growing in deep convolutional networks," *arXiv preprint arXiv:1906.02909*, 2019.

[28] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learning Representations*, 2020.

[29] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in *Proc. Int. Conf. Machine Learning*, 2017, pp. 527–536.

[30] K. Berestizshevsky and G. Even, "Sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence," *arXiv preprint arXiv:1805.10982*, 2018.

[31] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learning Representations*, 2017.

[32] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *Proc. Int. Conf. Machine Learning*, Jun 2019.

[33] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. Int. Conf. Pattern Recognition*, 2016, pp. 2464–2469.

[34] D. Langr and P. Tvrdik, "Evaluation criteria for sparse matrix storage formats," *IEEE Transactions on Parallel & Distributed Systems*, vol. 27, no. 02, pp. 428–440, 2016.

[35] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *Proc. IEEE*, vol. 55, no. 11, pp. 1801–1809, 1967.

[36] A. Paszke, S. Gross, F. Massa *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Conf. Neural Information Processing Systems*, 2019, pp. 8026–8037.

[37] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," *Proc. Int. Conf. Learning Representations*, 2019.

[38] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2015, p. 1912–1920.

[39] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, September 2008.

[40] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proc. European Conf. Computer Vision.*, 2018, pp. 409–424.

[41] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, "Condconv: conditionally parameterized convolutions for efficient inference," in *Proc. Conf. Neural Information Processing Systems*, 2019, pp. 1307–1318.

[42] N. Ma, X. Zhang, J. Huang, and J. Sun, "Weightnet: Revisiting the design space of weight networks," in *Proc. European Conf. Computer Vision.*, 2020, pp. 776–792.