Poster: Resource-Efficient Deep Subnetworks for Dynamic Resource Constraints on IoT Devices

Francesco Corti*, Christopher Hinterer*, Julian Rudolf*, Balz Maag†, Joachim Schauer‡, and Olga Saukh*
francesco.corti@tugraz.at, christopher@hinterer.at, julian.rudolf@student.tugraz.at, balz.maag@ch.abb.com, joachim.schauer@fh-joanneum.at, saukh@tugraz.at
*Graz University of Technology, †ABB Research, ‡FH JOANNEUM University of Applied Sciences

Abstract

Deep models running on edge and mobile devices typically encounter dynamic system states due to changes in available resources, fluctuating energy levels and multiple competing real-time tasks. State-of-the-art machine learning pipelines produce resource-agnostic models that cannot dynamically adjust their resource demand at runtime. We present Resource-Efficient Deep Subnetworks (REDS), deep networks that can adapt their size and inference speed at runtime by using structured sparsity to allow for further optimizations on typical embedded platforms. We extend the TFMicro framework to support REDS and present preliminary evaluation on Arduino Nano 33 BLE Sense showing linear speedups and negligible overhead at a price of minor loss in model's test set accuracy.

1 Introduction

Integration of machine learning (ML) models into Internet of Things (IoT) systems has led to rapid advancements across various domains. ML algorithms running on edge devices can extract valuable insights, identify patterns, and make predictions based on the collected data. This facilitates the development of new tools and real-time decision-making. However, machine learning models are resource-agnostic monoliths, incapable of adapting their function to dynamic resource availability. The problem is particularly severe in scenarios where resource availability depends on external factors that are difficult to predict in advance.

For example, low-power and intermittent computing systems [4, 6] make best use of their resources based on energy provisioning. Mobile agents such as drones, robots and self-driving cars may have to adjust their task priorities in response to a change in speed and thus the requirements of a different safety profile. User-driven change in task priorities, *e.g.*, triggered by an event of interest, may consume or provide more resources to a specific tasks [11]. Resource-agnostic ML models can not make use of variable resources.

Several recent research studies recognize and address the problem in specific domains. Bamusi et al. [4] propose the concept of approximate intermittent computing by designing support vector machines that can adapt to available energy while sacrificing accuracy. Kannan and Hoffman [9] propose a RNN architecture that allows early exit predictions, which prevents the input from being processed by the entire model, thus implying less energy consumption. DRESS [11] and NestDNN [2] design nested subnetworks that are increasingly more resource-efficient yet somewhat less accurate. DRESS trains nested models while making use of 2:4 sparsity pattern and thus targeting NVIDIA Ampere accelerators. NestDNN converts a pre-trained model into the nested structure by applying an iterative filter pruning, growing and fine-tuning. REDS draw inspiration from these works. However, in contrast to DRESS and NestDNN the structured sparsity pattern chosen by REDS uses insights in [8] and preserves model's structure, *i.e.*, dense layers remain dense in all lower-capacity subnetworks. This novel nested DNN approach eliminates the need for any specialized hardware and maintains low overhead for switching between different subnetworks within a single model by exploiting contiguous memory. This design decision is motivated by two observations: (1) Microcontrollers, *e.g.*, ARM Cortex-M4, often provide SIMD instructions to efficiently execute FMA and MLA operations for several inputs, yet require these to be in contiguous memory. Therefore, sparsity patterns that deviate from the original optimized model design may pay a performance penalty [1]. (2) REDS yield high-accuracy nested subnetworks of comparable quality as the models of the same architecture trained from scratch. In Table 1 all REDS subnetworks closely follow the models of the same architecture trained from scratch.

2 REDS Architecture and Training

REDS training closely follows DRESS, yet offers greater flexibility by not being limited to the 2:4 sparsity pattern. This pattern is rarely utilized on IoT devices because they lack the necessary specialized hardware. REDS applies structured sparsity (we iteratively prune neurons and convolutional filters) by enforcing each subnetwork to keep half the capacity of its parent network. The largest parent network has the largest capacity and requires the highest FLOPs

$\|\theta\|$ %	1-layer MLP / DIGITS Scratch	1-layer MLP / DIGITS REDS	3-layer MLP / DIGITS Scratch	3-layer MLP / DIGITS REDS	11-layer MLP / DIGITS Scratch	11-layer MLP / DIGITS REDS	CNN / SPEECHCMDS Scratch	CNN / SPEECHCMDS REDS
100	96.96±0.1	96.28 ±0.3	95.18 ±0.6	94.58 ±0.8	96.81±0.4	96.49 ±0.4	86.43±0.8	87.16±0.3
50	96.48±0.3	96.16 ±0.2	93.55 ±0.4	93.62 ±1.0	95.85±0.4	96.11 ±0.5	78.18±0.5	81.19±1.8
25	95.33±0.7	95.49 ±0.4	92.51 ±0.8	91.80 ±1.8	95.25±0.4	94.90 ±0.5	69.5±0.3	68.05±1.0
12.5	92.11±0.6	89.54 ±1.6	90.63 ±19.0	91.24 ±5.3	64.92±1.5	81.74 ±0.8	41.84±0.2	46.38±0.5

Table 1. Performance comparison of REDS subnetworks to networks of the same architecture trained from scratch. $|\theta|$ is the fraction of network parameters the largest network shares with the subnetwork at level i.

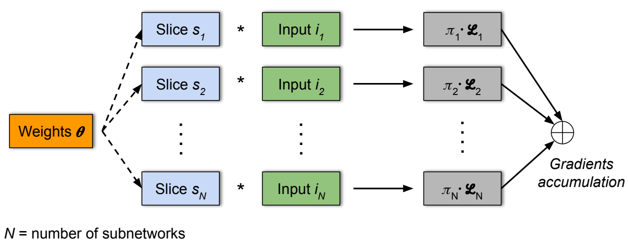

Figure 1. The tensor weights of the network are dynamically adjusted during training to decrease the number of the network's parameters θ used for inference. The parameters $\pi_i{}_{i=1}^{N} = \frac{1}{2^i}$ ensure that the contribution of the loss from individual models aligns with the fraction of the shared weights. Input and output layers are not pruned.

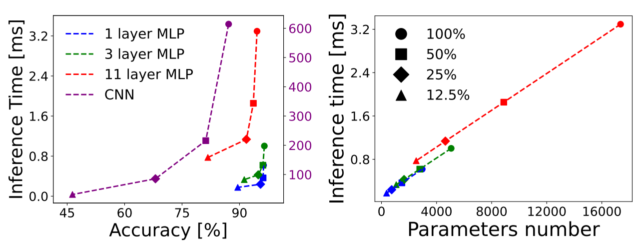

Figure 2. Evaluating REDS on three MLP architectures of varying depth trained on the 10-class DIGITS dataset, and a CNN architecture trained to classify 8 speech commands of a SPEECHCMDS reduced version. Test accuracy as a function of inference time (left), inference time as a function of model size (right). Execution time was measured on Arduino Nano 33 BLE Sense platform featuring ARM Cortex-M4 1MHz MCU and 256 kB SRAM.

to compute a prediction. The training procedure is sketched in Fig. 1. The weights of each subnetwork i are reused by all lower-capacity subnetworks $\{i+k\}_{k=1}^{N}$ by dynamically creating for each layer a tensor slice, *i.e.*, a tensor object that points to the original weight tensor, which uses half of its size. Similarly to DRESS, we balance the contribution of the loss of each individual model with parameters $\{\pi_i\}_{i=1}^{N} = \frac{1}{2^i}$, aligned with the fraction of shared weights.

3 Evaluation on Arduino Nano 33 BLE Sense

We extend the TFMicro library [7] to support REDS. We train three Multilayer Perceptron (MLP) models with 1, 3 and 11 layers, part of the TinyML Benchmarks collection [3], on DIGITS. We also train a 2-layer Convolutional Neural Network (CNN) on SPEECHCMDS [10]. REDS performance is presented in Fig. 2. The subnetworks can effectively reduce the inference time and the working memory size. MLP architectures yield high test accuracy down to a certain capacity when their performance falls of a cliff. CNN subnetworks quality degrades slowly, allowing for a more balanced trade-off. We observe a linear dependency between each subnetwork size and the required inference time on Arduino. This may however vary on other IoT devices that enjoy hardware-specific accelerations supported by TFMicro.

4 Discussion

REDS supports training resource-aware models for IoT devices by leveraging structured sparsity and preserving peculiarities of model architectures. Similarly to DRESS and NestDNN, the overhead of switching between REDS' submodels is low. REDS requires updating just one variable per layer – the active fraction of the network weights. Moreover, REDS preserves the optimizations done by the TFLite converter, thanks to its dense memory layout across all the subnetworks. Model compression methods were shown to be specific to the application domain and training regime [5]. Therefore, our future work focuses on further exploring and optimizing REDS by taking into account the dependencies between structural elements of the subnetworks.

Acknowledgements. This research was funded in part by the Austrian Science Fund (FWF) within the DENISE doctoral school (grant number DFH 5). For the purpose of open access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

5 References

[1] A. Howard et al. Mobilenets: Efficient CNNs for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[2] B. Fang et al. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *MobiCom*. ACM, 2018.

[3] B. Sudharsan et al. TinyML Benchmark: Executing fully connected neural networks on commodity microcontrollers. In *WF-IoT*, 2021.

[4] F. Bambusi et al. The case for approximate intermittent computing. In *IPSN*, pages 463–476, 2022.

[5] F. Corti et al. Studying the impact of magnitude pruning on contrastive learning methods. *arXiv:2207.00200*, 2022.

[6] M-P. Gherman et al. Towards on-demand gas sensing. In *DCOSS*, pages 72–74, 2021.

[7] R. David et al. TF Micro: Embedded ML on TinyML systems, 2021.

[8] R. Entezari et al. The role of permutation invariance in linear mode connectivity of neural networks. *arXiv:2110.06296*, 2021.

[9] T. Kannan et al. Budget RNNs: Multi-Capacity Neural Networks to Improve In-Sensor Inference Under Energy Budgets. In *RTAS*, 2021.

[10] P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR, arXiv:1804.03209*, 2018.

[11] Z. Qu et al. DRESS: Dynamic real-time sparse subnets. *arXiv preprint arXiv:2207.00670*, 2022.