Towards On-device Learning on the Edge: Ways to Select Neurons to Update under a Budget Constraint

Aël Quélennec Enzo Tartaglione Pavlo Mozharovskyi Van-Tam Nguyen
LTCI, Télécom Paris, Institut Polytechnique de Paris
19 Place Marguerite Perey, 91120 Palaiseau, France
{name.surname}@telecom-paris.fr

Abstract

In the realm of efficient on-device learning under extreme memory and computation constraints, a significant gap in successful approaches persists. Although considerable effort has been devoted to efficient inference, the main obstacle to efficient learning is the prohibitive cost of backpropagation. The resources required to compute gradients and update network parameters often exceed the limits of tightly constrained memory budgets. This paper challenges conventional wisdom and proposes a series of experiments that reveal the existence of superior sub-networks. Furthermore, we hint at the potential for substantial gains through a dynamic neuron selection strategy when fine-tuning a target task. Our efforts extend to the adaptation of a recent dynamic neuron selection strategy pioneered by Bragagnolo et al. (NEq), revealing its effectiveness in the most stringent scenarios. Our experiments demonstrate, in the average case, the superiority of a NEq-inspired approach over a random selection. This observation prompts a compelling avenue for further exploration in the area, highlighting the opportunity to design a new class of algorithms designed to facilitate parameter update selection. Our findings usher in a new era of possibilities in the field of on-device learning under extreme constraints and encourage the pursuit of innovative strategies for efficient, resource-friendly model fine-tuning.

1. Introduction

In recent years, the dynamic landscape of deep learning has witnessed remarkable progress across a multitude of domains, spanning from computer vision [23, 35] and speech recognition [4] to natural language processing [37]. This evolution, coupled with the escalating prowess of novel model architectures, has firmly entrenched deep learning as a pivotal technological force. The accelerated growth in hardware capabilities, alongside the

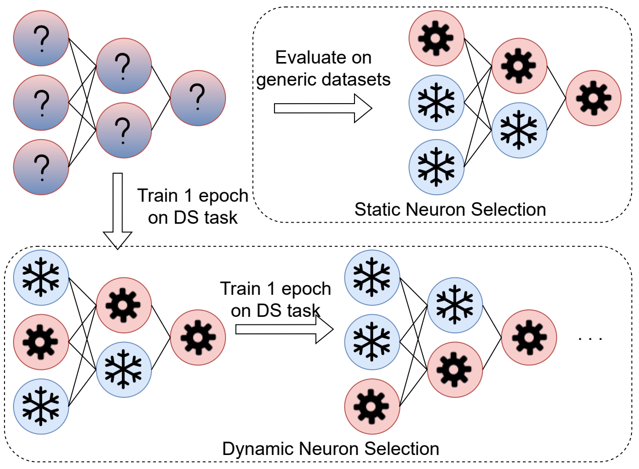


Figure 1. Comparison between static neuron update selection (upright corner) and dynamic selection (down). While the static strategy evaluates the neurons to update on generic datasets and the sub-network will remain static for the whole training on the downstream (DS) task, with the dynamic selection it can change after every epoch.

advent of AI-dedicated processing units, has enabled the training of increasingly expansive models, as indicated by prior work [1]. Yet, the intrinsic demand for substantial computational resources during training, and the consequential surge in energy consumption [32], underscore a pressing challenge. Whether it is at inference or training time, the inextricable interplay between three vital factors—computational resource utilization, energy efficiency, and inference time—requires meticulous attention. The mission at hand is twofold: to empower smaller institutions to partake in the development and training of state-of-the-art models and to conscientiously curtail the carbon footprint of the deep learning community.

On-device training stands as a pivotal advancement in the realm of artificial intelligence, carrying profound implications across various applications [14]. One of its primary merits is its ability to facilitate continuous model improvement even after deployment (when combined with online learning strategies) [12]. This becomes especially beneficial when the nature of the data evolves or when user interactions are integral to the application. Language models, for instance, can adapt to ever-changing linguistic trends, colloquialisms, and user preferences, resulting in more accurate and relevant predictions. Moreover, on-device training empowers models with lifelong learning capabilities: this means they can accumulate knowledge and adapt as they encounter new data, making them invaluable in fields such as healthcare where medical knowledge is continually evolving [28]. On-device training also ushers in the era of user customization, allowing models to adapt to individual preferences and writing styles, thus enhancing the user experience.

Although making models efficient at inference is a well-known and even currently explored challenge, efficient on-device learning under extreme memory and computation budget is a relatively new one. More specifically, compared to inference, the biggest obstacle resides in the cost of performing backpropagation (BP) (computing the gradient and updating all the parameters in the network quickly exceeds tight memory budgets). Although some approaches to perform learning on-device are alternatives and modifications to BP such as unsupervised learning for image segmentation [38] or the recently proposed Forward-Forward algorithm [13] and PEPITA [26], they are currently below the performance achieved by BP. Building on this, Lin *et al.* was finally able to successfully fine-tune a deep model with an extreme memory budget (under 256kB) [21] by statically determining which parts of a pre-trained model should be updated on any downstream task. The authors here implicitly whisper that there is essentially a sub-network providing sufficiently general features to be adapted to most of the downstream tasks, and can be kept frozen while updating a minor part of it. This hypothesis can be also seen, in a certain sense, as an extension of the lottery ticket hypothesis [8] applied on the BP graph in isolation [34], where just a sub-network needs to be updated to find a target performance.

In this paper, we challenge the vision by Lin *et al.* under extreme memory budget constraints, proposing some experiments where we observe that better sub-networks exist. We summarize our contributions as follows.

- We modify a dynamic neuron update selection strategy, NEq [3], to work under extreme memory budgets (Sec. 3.3).
- We compare the static selection strategy [21] to our dynamic approach (Fig. 1), reporting in the average case the superiority of the dynamic one (Sec. 4.1).
- We introduce a random dynamic neuron selection baseline, where neurons to update are chosen randomly. Our proposed strategy, in the majority of the tested setups, shows its superiority (Sec. 4.2). Such observation opens the road to more study in the field, where a new class of algorithms for parameter update selection should be designed.

2. Related work

On-device learning On-device learning is a growing field of research due to the increasing number of embedded devices for IoT applications. To this day, the principal modus operandi is to train a model offline and then compress and deploy it on-device for inference only. However, such methodology often yields poor performance due to real data distribution shifts from training data [30]. Research in continual learning shows that it is an efficient solution to adapt models to distribution shifts in post-deployment scenarios. Continual learning mimics human behavior in sequentially acquiring and retaining knowledge across various tasks [25, 40]. Evidently, in such a scenery catastrophic forgetting, which is the undesired loss of information acquired from previous tasks, is massively addressed by the research community [10, 16, 25] defined as the significant loss of earlier acquired knowledge during the learning of new tasks. However, embedded devices are highly constrained in computational and memory resources while training: especially backpropagation is very costly, making it a true challenge to achieve without significantly affecting model performance. For on-device learning based on backpropagation, we can distinguish two types of approaches, often combined to attain the best training accuracies under resource constraints: improving the architecture's efficiency and performing sparse updates.
Efficient architectures. The first is to design resource-efficient neural networks such as MobileNets [29], EfficientNets [33] or MCUNet [20], or even some efficient versions of transformers [7]. From a certain perspective, this method consists of the reduction of trainable parameters, impacting directly on memory and computation reduction. This method is also often paired to quantization to reduce the memory footprint on-device. More general approaches in such perspective involve automatic Neural Architecture Search (NAS) [11, 19, 22]. NAS automates the exploration of neural network architectures, optimizing factors like model size, performance in terms of accuracy, and required FLOPs, often employing multi-objective optimization. Evidently, despite the big effort in making these strategies as efficient as possible [15, 27, 39], fine-tuning a pre-trained, off-the-shelf architecture remains the least energy-consuming approach.
Sparsely update the model. The second method is to sparsely update the network. As shown in [5], the memory

footprint of backpropagation is greatly due to the loading of each layer's input tensor from which the gradient is computed. However, in the context of fine-tuning a pre-trained model given a shifted distribution, it is better to surgically select a subset of layers to train while freezing the rest of the network [18], leading to drastic savings regarding activation costs while achieving good performances. Such a concept is what led Lin *et al.* to design the Sparse Update (SU), an optimized static selection of a subset of layers to train while on-device [21]. However, the SU configurations are very costly to find as they require an evolutionary algorithm to iterate over many trainings. In the next section, we will present an approach embodying a different philosophy, challenging a static update graph allocation but rather identifying dynamically the partition of the model better to be updated.

3. Sub-network selection under constraint: from static to dynamic

In this section, we present our novel contributions, which extend upon SU and NEq selection techniques. We discuss the inherent limitations of these methods within the on-device learning domain, starting from an overview of SU schemes, and finally introducing two innovative approaches, inspired by the NEq approach. Our primary objective is to address these limitations by proposing dynamic online neuron selection mechanisms, tailored to accommodate stringent memory budgets.

3.1. Unboxing Sparse Update

In [21], Lin *et al.* present a pioneering algorithm-system co-design framework, enabling on-device training to adapt models to sensor data without compromising privacy. Overcoming the memory limitations of IoT devices, they were indeed able to achieve on-device training with just 256kB of memory. The authors were able to accomplish this thanks to a combination of four elements:

1. SU, corresponding to a selection of a subset of layers and weights to optimize during training, in order to reduce the memory footprint of gradient computation during backpropagation;
2. the usage of networks specifically conceived for tight resource environments, such as MobileNetV2 or MCUNet;
3. the introduction of a quantization-aware scaling in order to stabilize quantized gradient update;
4. the design of "Tiny Training Engine", an efficient training system transforming the actual training into slim binary codes.

The SU approach builds on top of the assumption that just a sub-network can be fine-tuned to achieve good performance. To validate this, Lee *et al.* demonstrate that when fine-tuning a pre-trained model on a target dataset, specifically selecting a subset of layers to train depending on the type of distribution shift does not affect performance negatively and instead can even improve it [18]. The reduction of training memory usage through selective layer training during fine-tuning is especially interesting when considering on-device learning where models are typically pre-trained offline on a large dataset. Lin *et al.* took this further by analyzing the impact of training the bias at different depths as well as different layer update rates, finding a positive correlation between combined training results and individual accuracy gains. Their methodology is the following: for a classification model pre-trained on ImageNet-1k, the objective is to determine the influence on accuracy gain of training different individual bias and layer configurations on the downstream task of Visual Wake Word classification [6], in comparison with only training the classifier layer.

Bias update utility. Lin *et al.* conducted a comprehensive quantitative evaluation to assess the impact of bias updates within the network. Specifically, they conducted a series of individual training sessions, where every bias from the l-th layer onward (from the l-th layer to the output) was updated. Each training iteration resulted in a relative accuracy improvement $\Delta\mathcal{A}_l^{\text{bias}}$ when compared to training only the classifier. Through their empirical analysis, the authors noted a steady increase in accuracy enhancement, eventually reaching a point of saturation at a specific depth within the network.

Layer update utility. At each network layer, Lin *et al.* performed a series of experiments to assess the impact of training with varying channel ratios $\zeta \in \{0, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$ on the relative accuracy gains $\Delta\mathcal{A}_{l,\zeta}^{\text{w}}$. This process entailed performing a set of training sessions equal to the number of network layers, each multiplied by the number of ratios tested.

Cost of individual configurations. The analysis of memory usage for each layer reveals that early layers of the network have a high activation cost $\mathcal{C}_l^{\text{act}}$ (cost of loading the layer's input tensor to compute the gradient) and a low-weight memory cost $\mathcal{C}_l^{\text{w}}$ (cost of loading the layer weights in memory). Late layers in the network have the opposite behavior and middle layers have both low activation and memory cost.

SU scheme selection. Given each bias depth and layer ratio relative accuracy gains, Lin *et al.* observe that the aggregation of different configurations leads to a final accuracy that is positively correlated with the sum of relative accuracy gains from each individual configuration. They use this empirical observation in an evolutionary algorithm to optimize the final sum of $\{\Delta\mathcal{A}_{l,\zeta}^{\text{w}}, \Delta\mathcal{A}_l^{\text{bias}}\}\forall l, \zeta$, given a memory budget constraint. This evolutionary search results in a specific subset of layers ratios and bias depth to update which is referred to as the SU scheme.

Beyond SU. The SU approach is labor-intensive and costly, requiring multiple trainings for various layers and update combinations followed by an evolutionary search to find the optimal configuration. Additionally, Lin *et al.* compute the relative accuracy gains on one downstream task and then apply the SU scheme found to many different target datasets. As demonstrated theoretically by Lee *et al.* [18] and then validated empirically by Kwon *et al.* [17], the optimal subset of layers to fine-tune depends on the target dataset, meaning that the contribution analysis should be performed for each downstream task. Additionally, the SU configuration remains unchanged throughout network fine-tuning, leading to the same neurons being updated over and over until potential over-fitting while other neurons remain frozen even though they could require some training to improve performance. This motivated us to move to dynamic update schemes - here follow the preliminaries to set up bases for a dynamic update scheme under a memory (and computation) budget.

3.2. A dynamic approach: Neurons at Equilibrium

In this section, we will present NEq, an algorithm proposed by Bragagnolo *et al.* which targets energy consumption reduction at training time with no performance loss [3]. NEq aims at reducing training time and cost without affecting performance by progressively selecting neurons to freeze throughout training.

Let us define the output of the i-th neuron when the input $\boldsymbol{x}$ is fed to the whole model trained after t epochs as $\boldsymbol{y}^t_{i,\boldsymbol{x}}$. For a given set of inputs $\boldsymbol{x} \in \mathcal{D}_{\text{val}}$ (where $\mathcal{D}_{\text{val}}$ is the validation set), we can compare each n-th output $y^t_{i,\boldsymbol{x},n}$ with $y^{t-1}_{i,\boldsymbol{x},n}$ by computing the cosine similarity ϕ^t_i between all the outputs of the i-th neuron at time t and at time $t-1$ for the whole validation set $\mathcal{D}_{\text{val}}$:

$$\phi^t_i = \sum_{\boldsymbol{x}\in\mathcal{D}_{\text{val}}} \sum_{n=1}^{N_{i,\boldsymbol{x}}} \hat{y}^t_{i,\boldsymbol{x},n} \cdot \hat{y}^{t-1}_{i,\boldsymbol{x},n}, \tag{1}$$

where $N_{i,\boldsymbol{x}}$ is the cardinality of the output for the i-th neuron when $\boldsymbol{x}$ is fed as input to the neural network. The variations of ϕ_i over different epochs inform us about the stability of the relationship between the input and the output of the i-th neuron. To quantify the amount of variation between epochs, we compute the relative variation

$$\Delta\phi^t_i = \phi^t_i - \phi^{t-1}_i. \tag{2}$$

What is defined as "equilibrium" corresponds here to the value of the similarities between epochs remaining constant, traducing the idea that the i-th neuron has learned its input-output relationship. To account for the temporal trend of $\Delta\phi_i$, a *velocity* is computed as:

$$v^t_i = \Delta\phi^t_i - \mu_{\text{eq}} v^{t-1}_i, \tag{3}$$

where μ_{eq} is a momentum term, allowing the velocity to carry a memory of its previous values.

A neuron is then considered *at equilibrium* when its velocity satisfies the following condition:

$$\left|v^t_i\right| < \varepsilon, \quad \varepsilon \geq 0. \tag{4}$$

As the learning process advances (and eventually the learning rate decreases), the neurons' velocities gradually "slow down", resulting in more neurons reaching equilibrium. When a neuron is at equilibrium, it can be frozen, allowing for computational savings (as its gradient should not be computed) without hurting the final accuracy (as it has already learned its input-output relationship). In other terms, in the early stage of the training, few neurons are typically frozen as the network is moving to a different loss subspace [9]. As the training progresses, the network stabilizes: many neurons have learned their target function and do not require further update steps.

From a very different perspective, NEq proposes an interesting alternative to SU: it does not require any prior knowledge of the target dataset and thus skips the tedious process of analyzing the contributions layers and bias as well as the heavy evolutionary search. Furthermore, it is computed dynamically, selecting the best neurons to update in the network for each epoch. However, it is not off-the-shelf ready to be applied to on-device learning, since in the first epochs many neurons have a high velocity exceeding the threshold value ε, leading to the gradient computation and backpropagation being out of memory.

3.3. Overhauling the velocity threshold: a budget constraining approach

We present here our approach to adapt NEq for enabling effective training complexity reduction satisfying a fixed maximum budget. Intuitively, the online fine-tuning of a network pre-trained on large datasets should endow the neural network with much fewer neurons to update. The pre-training leads the network to learn a latent representation that will be useful for learning the downstream task; thus, as described by Lee *et al.*, only a subset of parameters needs to be updated [18]. We thus replace the ε-threshold freezing condition with a budget for the whole neural network $\mathcal{B}^w$, expressed here in terms of the number of updatable parameters. To determine which are the parameters to update, we rank all the neurons in the network along the absolute value of their velocity, from the fastest to the slowest. Given $\mathcal{C}^{\text{w}}_i$ the number of parameters of the i-th neuron, we evaluate the total parameter's cost including the j-th fastest neurons

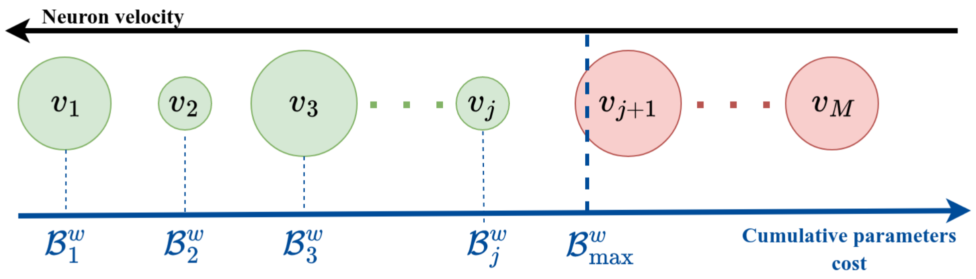


Figure 2. Selection of neurons to update given a network of M neurons and a budget $\mathcal{B}^w_{\text{max}}$. The cost $\mathcal{C}^w_i$ is proportional to the size of the circle which represents the i-th neuron. The neurons selected for the update are in green, while those frozen in red.

as

$$\mathcal{B}^{\text{w}}_j = \sum_{i=1}^{j} \mathcal{C}^w_i \Bigg| \; v_i \geq v_j, \; 1 \leq i \leq j \leq M, \tag{5}$$

where M is the total number of neurons of the neural network. We will then try to greedily solve the problem of selecting the highest j according to

$$\max_j \{\mathcal{B}^{\text{w}}_j \leq \mathcal{B}^w_{\text{max}}\}. \tag{6}$$

A graphical representation of the problem in (6) is provided in Fig. 2. In the figure, the neurons' sizes are proportional to $\mathcal{C}^{\text{w}}_i$. The neurons on the left side of the budget threshold are updated during BP, whereas the neurons on the right are frozen. The budget threshold overlaps a neuron since the cumulative sum grows incrementally with steps of size $\mathcal{C}^{\text{w}}_i$. This way, we can fit a network training under a given budget constraint and dynamically select the best neurons to update at each epoch.

Although (6) provides a formulation of the neurons that are changing more rapidly their function, we identify an unfair selection towards neurons having a higher number of parameters. To compensate for this effect, we propose a measure estimating the average velocity-per-parameter

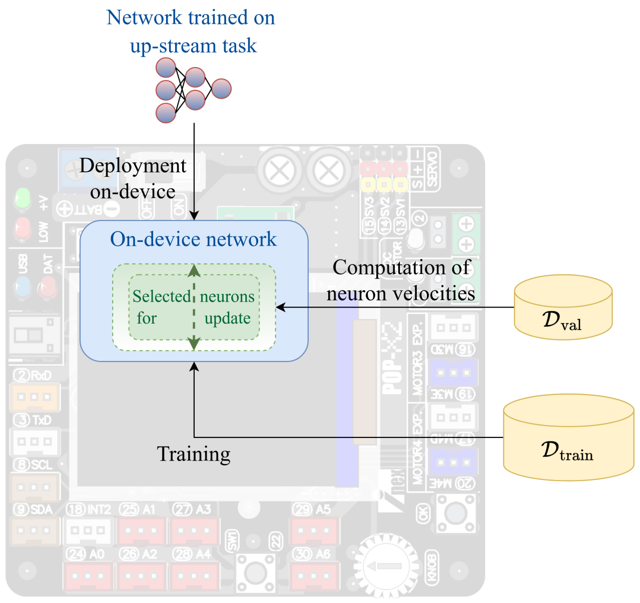
$$\tilde{v}_i = \frac{v_i}{\mathcal{C}^{\text{w}}_i}. \tag{7}$$

This re-weighted velocity allows us to focus on the neurons with the highest per-parameter average velocity instead of the neurons with the highest global velocity.

A random selection baseline. To the best of our knowledge, our proposal is the very first approach attempting to dynamically select a sub-network to update. A very intuitive baseline we can build consists of randomly selecting neurons to be updated, until the budget $\mathcal{B}^{\text{w}}_{\text{max}}$ is met.

3.4. Training algorithm for a dynamic neuron's update selection strategy

To summarize, the on-device learning strategies all involve pre-training a model on some upstream task, which is then loaded to the target device. Subsequently, either a static or a dynamic neuron selection strategy can take place - for a neuron velocity-based selection the use of a validation set $\mathcal{D}_{\text{val}}$ is required - and training on the $\mathcal{D}_{\text{train}}$ data of the target down-stream task takes place (as synthetically visualized in Fig. 3). Overall, the neuron selection strategy for on-device fine-tuning is composed of three distinct phases.



Network trained on up-stream task
Deployment on-device
On-device network
Selected neurons for update
Computation of neuron velocities
$\mathcal{D}_{\text{val}}$
Training
$\mathcal{D}_{\text{train}}$

Figure 3. Overview for the on-device learning when dynamic neuron selection is applied.

1. Select the sub-network to be updated for the first epoch: this phase can leverage knowledge from an evolutionary search algorithm as in SU, or in a random selection for the subgraph to update, in compliance with the maximum budget $\mathcal{B}^w_{\text{max}}$.

2. Train for one epoch.

3. Evaluate which part of the whole model should be updated for the next epoch. The sub-graph of neu-

Table 1. Comparison of pretrained MobileNetV2 final top1 test accuracies across different neuron selection methods for three different memory budgets expressed in percentage of network updated and in number of parameters. For the first epoch the neurons to update are given by the associated SU scheme. We highlight with different colors the best method for each budget (blue for Velocity, red for Random and green for SU).

% of network updated	$\mathcal{B}^w_{\max}$	Method	Cifar 10	Cifar 100	VWW	Flowers	Food	Pets	CUB
8.8	192 311	Sparse Update	95.13±0.21	78.60±0.22	90.66±0.29	93.77±0.38	77.81±0.26	85.82±0.22	67.82±0.29
		Velocity	95.25±0.29	79.46±0.12	91.40±0.16	93.03±0.47	79.16±0.16	85.50±0.17	67.52±0.05
		Random	94.41±0.13	78.15±0.26	90.29±0.05	92.19±0.17	77.78±0.00	85.50±0.28	65.56±0.45
21.2	464 639	Sparse Update	95.30±0.10	78.84±0.20	91.29±0.18	94.28±0.36	78.26±0.07	84.63±0.15	68.04±0.28
		Velocity	95.36±0.07	79.67±0.28	91.48±0.39	93.34±0.08	79.63±0.17	84.91±0.82	68.23±0.61
		Random	94.61±0.16	78.28±0.31	90.51±0.25	92.43±0.10	78.26±0.07	84.73±0.29	66.30±0.13
30.8	675 540	Sparse Update	95.16±0.29	78.62±0.18	91.46±0.21	94.22±0.14	78.03±0.08	84.38±0.28	67.59±0.22
		Velocity	95.49±0.16	79.43±0.19	91.57±0.20	93.77±0.26	79.56±0.24	84.44±0.50	68.26±0.36
		Random	94.57±0.20	78.58±0.11	90.58±0.10	92.83±0.03	79.00±0.11	84.39±0.42	66.17±0.42

rons to be updated will not change when using SU, it can change when employing either a velocity-based selection or a random selection. The training is iterated back to step 2 until some target training termination conditions are met (in our case, a wall number of epochs).

This very general framework allows for plugging any dynamic neuron selection strategy, and we will use it to compare static neuron selection (SU), velocity-based selection, and random selection. In the next section, we will present some results obtained on popular benchmarks.

4. Experiments

In this section, we will describe the experiments conducted to validate our hypotheses. We selected 7 target datasets to fine-tune our models upon, mostly to compare our results with SU performances as these are the datasets they selected in their study. The datasets are Cifar10, Cifar100, CUB [36], Flowers [24], Food [2], Pets [41] and Visual Wake Words [6]. We selected three models to test on (MobileNetV2, Resnet18, and Resnet50) and we averaged the results over 3 different seeds. Our fine-tuning policy is the same for all the experiments: we use SGD with no momentum and no weight decay and a cosine annealing scheduler over 200 epochs, ranging from 0.125 to 0 with 5 warm-up epochs. We report the final top1 accuracy obtained on the test set $\mathcal{D}_{\text{test}}$. All the experiments were conducted on an Nvidia RTX3090Ti with 24GB, algorithms are implemented in Python, using PyTorch 2.0.0.[1]

[1]The code used to run experiments is available at https://github.com/aelQuelennec/-WACV-2024-Ways-to-Select-Neurons-under-a-Budget-Constraint.

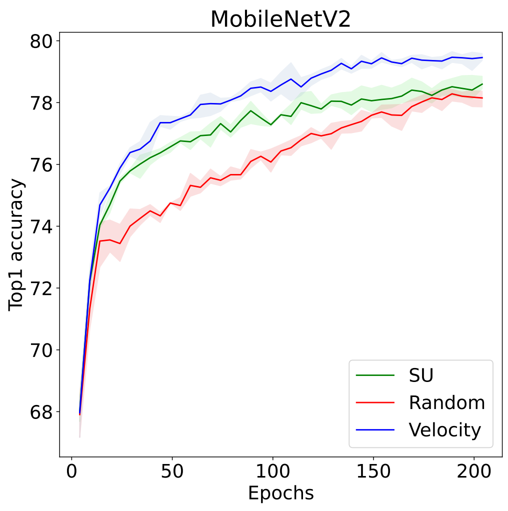


Figure 4. Comparison of MobileNetV2 testing accuracy through training on C100. We update 8.8% of the network's parameters.

4.1. Dynamic neuron selection VS SU

Our first set of experiments is the comparison of performance between SU and dynamic neuron selection methods. We isolated the SU code from the other technologies described in the original article and compared the results of fine-tuning with Velocity and Random (Table 1). We tested on MobileNetV2 with three different levels of budget constraint provided by the exact SU configurations Lin *et al.* designed for the network [21]. We calculate the number of parameters updated by a given SU scheme and we use that number as a budget for Velocity and Random. It is important to note that for the first epoch, we can not compute the neurons' velocities: we thus use the SU scheme to compare

Table 2. Comparison of final top1 test accuracies between Baseline, SU, Random and Velocity neuron selection over various pretrained models, datasets, and budgets. For the first epoch the neurons to update are randomly selected. For each budget, highlighted results correspond to the best accuracy between neuron selection methods (green if SU is better, blue for Velocity, and red for Random, in bold the overall best performance regardless of the budget).

Model	$\mathcal{B}^w_{\text{max}}$	Method	Cifar 10	Cifar 100	VWW	Flowers	Food	Pets	CUB
MbV2	192 311	SU	94.88±0.12	78.15±0.13	90.75±0.17	92.70±0.06	75.02±0.23	**86.93±0.22**	66.48±0.41
		Velocity	95.35±0.35	79.41±0.21	90.95±0.16	92.98±0.29	79.18±0.07	85.56±0.47	67.92±0.27
		Random	94.46±0.16	78.03±0.21	90.20±0.13	92.11±0.15	77.57±0.16	85.16±0.07	65.96±0.06
	464 639	SU	95.00±0.08	78.69±0.19	90.80±0.24	92.86±0.33	76.50±0.23	86.64±0.27	67.81±0.23
		Velocity	95.49±0.02	79.52±0.11	91.41±0.19	93.32±0.15	79.67±0.19	84.36±0.76	68.19±0.24
		Random	94.51±0.05	78.49±0.19	90.55±0.24	92.64±0.32	78.48±0.17	84.92±0.21	66.18±0.71
	675 540	SU	95.18±0.17	79.03±0.30	91.03±0.16	93.08±0.09	77.19±0.07	86.42±0.45	67.72±0.32
		Velocity	95.57±0.11	79.21±0.37	91.72±0.15	93.33±0.31	79.68±0.16	84.37±0.28	68.19±0.24
		Random	94.57±0.09	78.41±0.29	90.35±0.01	92.88±0.21	78.90±0.06	84.39±0.41	66.36±0.20
	2 189 760	Baseline	**95.93±0.14**	**79.83±0.29**	**91.80±0.03**	**94.02±0.03**	**80.63±0.10**	82.82±0.18	**69.24±0.34**
Resnet18	980 715	Velocity	95.51±0.10	78.77±0.41	88.78±0.51	90.78±0.24	75.09±0.13	**82.82±0.30**	63.64±0.35
		Random	95.20±0.20	77.98±0.38	88.33±0.45	89.39±0.47	74.57±0.15	79.49±0.51	60.93±0.54
	2 369 480	Velocity	95.36±0.15	**79.12±0.12**	89.16±0.31	**91.02±0.17**	75.72±0.23	82.01±0.60	**63.84±0.39**
		Random	95.68±0.10	78.28±0.22	89.21±0.23	89.53±0.17	75.17±0.12	79.40±0.34	61.42±0.32
	3 444 987	Velocity	95.58±0.21	78.95±0.13	89.17±0.33	90.76±0.15	75.83±0.12	81.45±0.63	63.59±0.03
		Random	95.80±0.09	78.52±0.18	88.92±0.19	89.66±0.30	75.28±0.12	79.28±0.34	61.45±0.32
	11 166 912	Baseline	**96.2±0.13**	78.86±0.08	**89.78±0.24**	90.14±0.26	**76.32±0.08**	79.76±0.63	60.97±0.52
Resnet50	2 059 888	Velocity	97.10±0.07	**82.94±0.35**	93.04±0.15	93.65±0.08	81.10±0.05	**90.11±0.25**	**73.73±0.52**
		Random	96.80±0.06	81.46±0.11	92.13±0.33	94.04±0.18	80.68±0.18	88.92±0.18	72.79±0.15
	4 976 842	Velocity	97.12±0.09	82.79±0.40	**93.37±0.14**	94.84±0.08	81.62±0.17	89.42±0.25	73.55±0.18
		Random	96.97±0.08	82.04±0.11	92.59±0.20	94.23±0.22	81.52±0.11	88.46±0.07	72.40±0.60
	7 235 830	Velocity	97.07±0.08	82.45±0.18	93.21±0.14	**95.03±0.18**	81.76±0.06	88.97±0.60	73.54±0.42
		Random	97.01±0.01	82.27±0.29	92.59±0.21	94.54±0.15	81.88±0.29	88.18±0.63	72.40±0.35
	23 454 912	Baseline	**97.30±0.04**	82.63±0.25	92.91±0.22	94.87±0.20	**82.49±0.15**	87.29±0.34	72.93±0.41

final results given an identical starting point. We observe that in the large majority of the cases, the update based on Velocity is competitive with SU and outperforms the random update baseline. This confirms that a proper selection of neurons to update guides the model to better generalizing solutions.

In Fig. 4, we display the test accuracy evolution for the different neuron selections. From this, we clearly observe that Velocity approaches faster than other approaches with high generalization. As intuitive, SU plateaus at intermediate epochs - seemingly, the updated parameters land at a local minimum. The Random update scheme is in general the worst, but progressively is able to catch up with the other approaches.

4.2. Velocity VS Random

In the attempt to assess a comparison between Velocity and Random neuron selection methods, we ran multiple experiments on the various models and datasets selected. Here the budgets we test are deduced from the percentage of total parameters updated in the SU schemes (on MobilenetV2) and applied to the Resnets architectures. For the

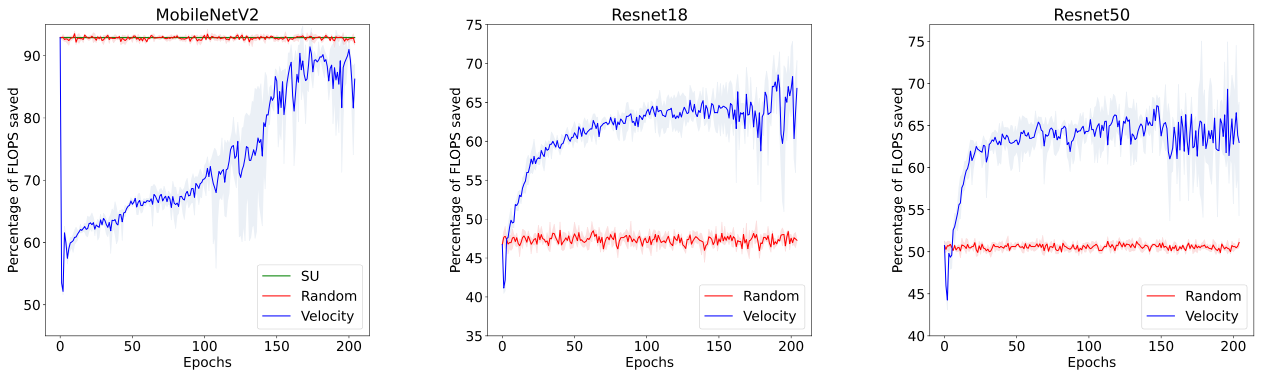


Figure 5. Percentage of FLOPS saved with SU, Velocity and Random neuron selection for each network during Cifar100 fine-tuning. We update 30.2% of the network's parameters.

first epoch, since we do not dispose of the neurons' velocities, we initialize the neurons randomly. The results obtained are displayed in Table 2. Comparing the experiments on Velocity with the SU initialization proposed in Table 1, we observe little to no impact on final accuracy for either of the two initialization methods. This suggests that we can dispense from finding the SU schemes and instead rely on dynamic neuron selection throughout the whole training. More broadly, we observe that MobileNetV2 clearly favors the Velocity selection (except in the case of the Pets dataset), whereas in Resnet networks the Random selection can result in marginally better performance, in some rare cases. It is important to note that as the budgets are computed as identical percentages of the network, they are tighter in the absolute value of parameters updated for the MobileNetV2 than for the Resnets.

Gap with the baseline models. Besides the comparisons between the Random and the Velocity approaches, in Table 2 we also report the baseline performance, when updating the full network. This gives us a reference accuracy (virtually achievable). We do not observe a big gap in performance between the low-budget approaches and the full model's update, and in some cases, the reported performance is even higher than the baseline, like in Pets for MobileNetV2 and ResNet50, where we observe an improvement of approximately 3 to 4%. We can explain this effect by the reduced size of the datasets, and the limited budget of parameters employed to fine-tune: we implicitly avoid overfitting.

FLOPs saved at training time. To qualitatively evaluate the impact on the training complexity for either of the two approaches, we introduce here as an efficiency metric the percentage of FLOPS saved (in comparison with the full model training). We observe that Random is a more economical strategy (more FLOPS are saved) for MobileNetV2, whereas it is the opposite for Resnets, as displayed in Fig. 5). While controlling FLOPs savings for Random is relatively straightforward, it becomes less manageable for a dynamic strategy like Velocity. This poses a critical consideration when implementing a learning strategy on the edge. This highlights the need for the development of new selection methods that also account for energy consumption, which remains a topic for future exploration.

5. Conclusions and Future works

In this work, two resource-saving neural update philosophies were presented: a static strategy, proposed in [21], where a sub-network in a pre-trained model is identified prior to training on the device (and statically determined for any downstream tasks), and a dynamic strategy. The results obtained suggest that the dynamic selection of neurons can be better than the static pre-selection of a sub-network to train. Specifically, the proposed dynamic strategy, which is an evolution of a resources-unconstrained training strategy [3], proved effective in almost all the scenarios.

Other aspects to be considered for progress in this field are the inclusion of activation cost and (expected) training cost for the selected neurons, and considering strategies potentially saving computation at forward-propagation time too, such as dropout [31].

Acknowledgements

This research was partially funded by Hi!PARIS Center on Data Analytics and Artificial Intelligence and by the project titled "PC2-NF-NAI" in the framework of the program "PEPR 5G et Réseaux du futur". This paper is supported by the European Union's HORIZON Research and Innovation Programme under grant agreement No 101120657, project ENFIELD (European Lighthouse to Manifest Trustworthy and Green AI).

References

[1] Toru Baji. Evolution of the gpu device widely used in ai and massive parallel processing. In *2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM)*, pages 7–9, 2018. 1

[2] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, 2014. 6

[3] Andrea Bragagnolo, Enzo Tartaglione, and Marco Grangetto. To update or not to update? neurons at equilibrium in deep models. *Advances in Neural Information Processing Systems*, 35, 2022. 2, 4, 8

[4] Maxime Burchi and Radu Timofte. Audio-visual efficient conformer for robust speech recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023. 1

[5] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce activations, not trainable parameters for efficient on-device learning. *arXiv preprint arXiv:2007.11622*, 2020. 2

[6] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. Visual wake words dataset. *arXiv preprint arXiv:1906.05721*, 2019. 3, 6

[7] David Elliott, Carlos E Otero, Steven Wyatt, and Evan Martino. Tiny transformers for environmental sound classification at the edge. *arXiv preprint arXiv:2103.12157*, 2021. 2

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 2

[9] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. The early phase of neural network training. In *International Conference on Learning Representations*, 2020. 4

[10] Robert M. French. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? In *Proceedings of the 6th International Conference on Neural Information Processing Systems*. Morgan Kaufmann Publishers Inc., 1993. 2

[11] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. *Advances in neural information processing systems*, 2016. 2

[12] Tyler L Hayes and Christopher Kanan. Online continual learning for embedded devices. *arXiv preprint arXiv:2203.10681*, 2022. 2

[13] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022. 2

[14] Ozlem Durmaz Incel and Sevda Ozge Bursa. On-device deep learning for mobile and wearable sensing applications: A review. *IEEE Sensors Journal*, 2023. 1

[15] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019. 2

[16] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114, 2017. 2

[17] Young D Kwon, Rui Li, Stylianos I Venieris, Jagmohan Chauhan, Nicholas D Lane, and Cecilia Mascolo. Tinytrain: Deep neural network training at the extreme edge. *arXiv preprint arXiv:2307.09988*, 2023. 4

[18] Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts. *arXiv preprint arXiv:2210.11466*, 2022. 3, 4

[19] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, 2020. 2

[20] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33, 2020. 2

[21] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35, 2022. 2, 3, 6, 8

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 2

[23] Shentong Mo, Zhun Sun, and Chao Li. Multi-level contrastive learning for self-supervised vision transformers. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023. 1

[24] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, 2008. 6

[25] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Netw.*, 113, 2019. 2

[26] Danilo Pietro Pau and Fabrizio Maria Aymone. Suitability of forward-forward and pepita learning to mlcommons-tiny benchmarks. In *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, 2023. 2

[27] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, 2018. 2

[28] Oleksandra Poquet and Maarten De Laat. Developing capabilities: Lifelong learning in the age of ai. *British Journal of Educational Technology*, 52, 2021. 2

[29] 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*, 2018. 2

[30] Gabriele Spadaro, Riccardo Renzulli, Andrea Bragagnolo, Jhony H Giraldo, Attilio Fiandrotti, Marco Grangetto, and Enzo Tartaglione. Shannon strikes again! entropy-based pruning in deep neural networks for transfer learning under

extreme memory and computation budgets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. 2

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15, 2014. 8

[32] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, 2019. Association for Computational Linguistics. 1

[33] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, 2019. 2

[34] Enzo Tartaglione. The rise of the lottery heroes: why zero-shot pruning is hard. In *2022 IEEE International Conference on Image Processing (ICIP)*, 2022. 2

[35] Yao-Hong Tsai, Dong-Meau Chang, and Tse-Chuan Hsu. Edge computing based on federated learning for machine monitoring. *Applied Sciences*, 12, 2022. 1

[36] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. 6

[37] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, Bo Long, et al. Graph neural networks for natural language processing: A survey. *Foundations and Trends® in Machine Learning*, 16, 2023. 1

[38] Junhuan Yang, Yi Sheng, Yuzhou Zhang, Weiwen Jiang, and Lei Yang. On-device unsupervised image segmentation. *arXiv preprint arXiv:2303.12753*, 2023. 2

[39] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020. 2

[40] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the International Conference on Machine Learning*, 2017. 2

[41] Hui Zhang, Shenglong Zhou, Geoffrey Ye Li, and Naihua Xiu. 0/1 deep neural networks via block coordinate descent. *arXiv preprint arXiv:2206.09379*, 2022. 6