Deep Partial Updating: Towards Communication Efficient Updating for On-device Inference

Zhongnan Qu[1], Cong Liu[2], and Lothar Thiele[1]

[1] ETH Zurich, Zurich, Switzerland
{quz,thiele}@ethz.ch
[2] UT Dallas, Texas, US
cong@utdallas.edu

Abstract. Emerging edge intelligence applications require the server to retrain and update deep neural networks deployed on remote edge nodes to leverage newly collected data samples. Unfortunately, it may be impossible in practice to continuously send fully updated weights to these edge nodes due to the highly constrained communication resource. In this paper, we propose the weight-wise deep partial updating paradigm, which smartly selects a small subset of weights to update in each server-to-edge communication round, while achieving a similar performance compared to full updating. Our method is established through analytically upper-bounding the loss difference between partial updating and full updating, and only updates the weights which make the largest contributions to the upper bound. Extensive experimental results demonstrate the efficacy of our partial updating methodology which achieves a high inference accuracy while updating a rather small number of weights.

Keywords: Partial updating, communication constraints, parameter reuse, server-to-edge, deep neural networks

1 Introduction

On-device inference is a new disruptive technology that enables new intelligent applications, *e.g.*, mobile assistants, Internet of Things and augmented reality. Compared to traditional cloud inference, on-device inference is subject to severe limitations in terms of storage, energy, computing power and communication. On the other hand, it has many advantages, *e.g.*, it enables fast and stable inference even with low communication bandwidth or interrupted communication, and can save energy by avoiding the transfer of data to the cloud, which often costs significant amounts of energy than sensing and computation [35,8,17]. To deploy deep neural networks (DNNs) on resource-constrained edge devices, extensive research has been done to compress a well pre-trained model via pruning [9,7,29] and quantization [5,28,26]. During on-device inference, compressed DNNs may achieve a good balance between model performance and resource demand.

However, due to a possible lack of relevant training data at the time of initial deployment or due to an unknown sensing environment, pre-trained DNNs may

either fail to perform satisfactorily or be significantly improved after the initial deployment. In other words, re-training the models by using newly collected data (from *edge devices* or *other sources*) is typically required to achieve the desired performance during the lifetime of devices.

Because of the resource-constrained nature of edge devices in terms of memory and computing power, on-device re-training (or federated learning) is typically restricted to tiny batch size, small inference (sub-)networks or limited optimization steps, all resulting in a performance degradation. Instead, retraining often occurs on a remote server with sufficient resources. One possible strategy to allow for a continuous improvement of the model performance is a two-stage iterative process: (*i*) at each round, edge devices collect new data samples and send them to the server, and (*ii*) the server retrains the model using all collected data, and sends updates to each edge device [4]. The first stage may be even not necessary if new data are collected in other ways and made directly available to the server.

Example scenarios. Example application scenarios of relevance include vision robotic sensing in an unknown environment (*e.g.*, Mars) [20], local translators of low-resource languages on mobile phones [3,36], and sensor networks mounted in alpine areas [21], automatic wildlife monitoring [34]. We detail two specific scenarios. *Hazard alarming on mountains:* Researchers in [21] mounted tens of sensor nodes at different scarps in high alpine areas with cameras, geophones and high-precision GPS. The purpose is to achieve fast, stable, and energy-efficient hazard monitoring for early warning to protect people and infrastructure. To this end, a DNN is deployed on each node to on-device detect rockfalls and debris flows. The nodes regularly collect and send data to the server for labeling and retraining, and the server sends the updated model back through a low-power wireless network. Retraining during deployment is essential for a highly reliable hazard warning. *Endangered species monitoring:* To detect endangered species, researchers often deploy some audio or image sensor nodes in virgin rainforests [34]. Edge nodes are supposed to classify the potential signal from endangered species and send these relevant data to the server. Due to the limited prior information from environments and species, retraining the initially classifier with received data or data from other sources (*e.g.*, other areas) is necessary.

Challenges. An essential challenge herein is that the transmissions in the server-to-edge stage are highly constrained by the limited communication resource (*e.g.*, bandwidth, energy [2]) in comparison to the edge-to-server stage, if necessary at all. Typically, state-of-the-art DNNs often require tens or even hundreds of mega-Bytes (MB) to store parameters, whereas a single batch of data samples (a number of samples that lead to reasonable updates in batch training) needs a relatively smaller amount of data. For example, for CIFAR10 dataset [15], the weights of a popular VGGNet require 56.09MB storage, while one batch of 128 samples only uses around 0.40MB [33,28]. As an alternative approach, the server sends a full update of the inference model once or rarely. But in this case, every node will suffer from a low performance until such an update occurs. Besides, edge devices could decide on and send only critical samples by using active learning schemes [1]. The server may also receive data from other sources,

e.g., through data augmentation based on the data collected in previous rounds or new data collection campaigns. These considerations indicate that the updated weights that are sent to edge devices by the server become a major bottleneck.

Facing the above challenges, we ask the following question: *Is it possible to update only a small subset of weights while reaching a similar performance as updating all weights?* Doing such a *partial updating* can significantly reduce the server-to-edge communication overhead. Furthermore, fewer parameter updates also lead to less memory access on edge devices, which in turn results in smaller energy consumption than full updating [11].

Why partial updating works. Since the model deployed on edge devices is trained with the data collected beforehand, some learned knowledge can be reused. In other words, we only need to distinguish and update the weights which are critical to the newly collected data.

How to select weights. Our key concept for partial updating is based on the hypothesis, that *a weight shall be updated only if it has a large contribution to the loss reduction* during the retraining given newly collected data samples. Specially, we define a binary mask $\boldsymbol{m}$ to describe which weights are subject to update and which weights are fixed (also reused). For any $\boldsymbol{m}$, we establish the analytical upper bound on the difference between the loss value under partial updating and that under full updating. We determine an optimized mask $\boldsymbol{m}$ by combining two different view points: (*i*) measuring each weight's "global contribution" to the upper bound through computing the Euclidean distance, and (*ii*) measuring each weight's "local contribution" to the upper bound using gradient-related information. The weights to be updated according to $\boldsymbol{m}$ will be further sparsely fine-tuned while the remaining weights are rewound to their initial values.

Our contributions can be summarized as follows.

- We formalize the deep partial updating paradigm, *i.e.*, how to iteratively perform partial updating of inference models on remote edge devices, if newly collected training samples are available at the server. This reduces the computation and communication demand on edge devices substantially.
- We propose a novel approach that determines the optimized subset of weights that shall be selected for partial updating, through measuring each weight's contribution to the analytical upper bound on the loss reduction. This simple yet effective metric can be applied to any models that are trained with gradient-based optimizers.
- Experimental results on public vision datasets show that, under the similar accuracy level along the rounds, our approach can reduce the size of the transmitted data by 95.3% on average (up to 99.3%), namely can update the model averagely 21 times more frequently than full updating.

2 Related Work

Partial updating. Although partial updating has been adopted in some prior works, it is conducted in a fairly coarse-grained manner, *e.g.*, layer-wise or neuron-wise, and targets at completely different objectives. Especially, under continual

learning settings, [37,13] propose to freeze all weights related to the neurons which are more critical in performing prior tasks than new ones, to preserve existing knowledge. Under adversarial attack settings, [32] updates the weights in the first several layers only, which yield a dominating impact on the extracted features, for better attack efficacy. Under meta learning settings, [27,31] reuse learned representations by only updating a subset of layers for efficiently learning new tasks. Unfortunately, such techniques do not focus on reducing the number of updated weights, and thus cannot be applied in our problem settings.

Federated learning. Communication-efficient federated learning [19,14,18] studies how to compress multiple gradients calculated on different sets of non-*i.i.d.* local data, such that the aggregation of these compressed gradients could result in a similar convergence performance as centralized training on all data. Such compressed updates are fundamentally different from our setting, where (*i*) updates are not transmitted in each optimization step; (*ii*) training data are incrementally collected; (*iii*) centralized training is conducted. Our typical scenarios focus on outdoor areas, which generally do not involve data privacy issues, since these collected data are not personal data. In comparison to federated learning, our pipeline has the following advantages: (*i*) we do not conduct resource-intensive gradient backward propagation on edge devices; (*ii*) the collected data are not continuously accumulated and stored on memory-constrained edge nodes; (*iii*) we also avoid the difficult but necessary labeling process on each edge node in supervised learning tasks; (*iv*) if few events occur on some nodes, the centralized training may avoid degraded updates in local training, *e.g.*, batch normalization.

Compression. The communication cost could also be reduced through some compression techniques, *e.g.*, quantizing/encoding the updated weights and the transmission signal [9]. But note that these techniques are orthogonal to our approach and could be applied in addition, see Appendix H.

Unstructured pruning. Deep partial updating is inspired by recent unstructured pruning methods, *e.g.*, [9,7,38,29,6,24]. Traditional pruning methods aim at reducing the number of operations and storage consumption by setting some weights to zero. Sending a pruned DNN with only non-zero weights may also reduce the communication cost, but to a much lesser extent as shown in the experimental results, see Sec. 5.2. Since our objective namely reducing the server-to-edge communication cost when updating the deployed DNN is fundamentally different from pruning, we can leverage some learned knowledge by retaining weights (partial updating) instead of zero-outing weights (pruning).

Domain adaptation. Domain adaptation targets reducing domain shift to transfer knowledge into new learning tasks [39]. This paper mainly considers the scenario where the inference task is not explicitly changed along the rounds, *i.e.*, the overall data distribution maintains the same along the data collection rounds. Thus, selecting critical weights (features) by measuring their impact on domain distribution discrepancy is invalid herein. Applying deep partial updating on streaming tasks where the data distribution varies along the rounds would be also worth studying, and we leave it for future works.

3 Notation and Setting

In this section, we define the notations used throughout this paper, and provide a formalized problem setting. We consider a set of remote edge devices that implement on-device inference. They are connected to a host server that is able to perform DNN training and retraining. We consider the necessary amount of information that needs to be communicated to each edge device to update its inference model.

Assume there are R rounds of model updates. The model deployed in the r-th round is represented with its weight vector $\boldsymbol{w}^r$. The training data used to update the model for the r-th round is represented as $\mathcal{D}^r = \delta\mathcal{D}^r \cup \mathcal{D}^{r-1}$. Also, newly collected data samples $\delta\mathcal{D}^r$ are made available to the server in round $r-1$.

To reduce the amount of information that needs to be sent to edge devices, only partial weights of $\boldsymbol{w}^{r-1}$ shall be updated when determining $\boldsymbol{w}^r$. The overall optimization problem for weight-wise partial updating in round $r-1$ is thus,

$$\min_{\delta\boldsymbol{w}^r} \ \ell\left(\boldsymbol{w}^{r-1} + \delta\boldsymbol{w}^r; \mathcal{D}^r\right) \tag{1}$$

$$\text{s.t.} \ \ \|\delta\boldsymbol{w}^r\|_0 \leq k \cdot I \tag{2}$$

where ℓ denotes the loss function, $\|.\|_0$ denotes the L0-norm, k denotes the updating ratio that is determined by the communication constraints in practical scenarios, and $\delta\boldsymbol{w}^r$ denotes the increment of $\boldsymbol{w}^{r-1}$. Note that both $\boldsymbol{w}^{r-1}$ and $\delta\boldsymbol{w}^r$ are drawn from $\mathbb{R}^I$, where I is the total number of weights.

In this case, only a fraction of $k \cdot I$ weights and the corresponding index information need to be communicated to each edge device for updating the model, namely the partial updates $\delta\boldsymbol{w}^r$. It is worth noting that the index information is relatively small in size compared to the partially updated weights (see Sec. 5). On each edge device, the weight vector is updated as $\boldsymbol{w}^r = \boldsymbol{w}^{r-1} + \delta\boldsymbol{w}^r$. To simplify the notation, we will only consider a single update, *i.e.*, from weight vector $\boldsymbol{w}$ (corresponding to $\boldsymbol{w}^{r-1}$) to weight vector $\widetilde{\boldsymbol{w}}$ (corresponding to $\boldsymbol{w}^r$) with $\widetilde{\boldsymbol{w}} = \boldsymbol{w} + \widetilde{\delta\boldsymbol{w}}$.

4 Deep Partial Updating

We developed a two-step approach for resolving the partial updating optimization problem in Eq.(1)-Eq.(2). The overall approach is depicted in Fig. 1.

The first step. The first step not only determines the subset of weights that are allowed to change their values, but also computes the initial values for the second step. In particular, we first optimize the loss function ℓ by updating all weights from the initialization $\boldsymbol{w}$ with a standard optimizer, *e.g.*, SGD or its variants. We thus obtain the minimized loss $\ell\left(\boldsymbol{w}^{\mathrm{f}}\right)$ with $\boldsymbol{w}^{\mathrm{f}} = \boldsymbol{w} + \delta\boldsymbol{w}^{\mathrm{f}}$, where the superscript f denotes "full updating". To consider the constraint of Eq.(2), the information gathered during this optimization is used to determine the subset of weights that will be changed, also that are communicated to the edge devices.

In the explanation of the method in Sec. 4.1, we use the binary mask $\boldsymbol{m}$ with $\boldsymbol{m} \in \{0,1\}^I$ to describe which weights are subject to change and which ones are not. The weights with $m_i = 1$ are trainable, whereas the weights with $m_i = 0$ will be rewound from the values in $\boldsymbol{w}^{\mathrm{f}}$ to their initial values in $\boldsymbol{w}$, *i.e.*, unchanged. Obviously, we find $\|\boldsymbol{m}\|_0 = \sum_i m_i = k \cdot I$.

The second step. In the second step we start a sparse fine-tuning from a DNN with $k \cdot I$ weights from the optimized model $\boldsymbol{w}^{\mathrm{f}}$ and $(1-k) \cdot I$ weights from the previous, still deployed model $\boldsymbol{w}$. In other words, the initial weights for the second step are $\boldsymbol{w} + \delta\boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m}$, where $\odot$ denotes an element-wise multiplication. To determine the final solution $\widetilde{\boldsymbol{w}} = \boldsymbol{w} + \widetilde{\delta\boldsymbol{w}}$, we conduct a sparse fine-tuning (still with a standard optimizer), *i.e.*, we keep all weights with $m_i = 0$ constant during the optimization. Therefore, $\widetilde{\delta\boldsymbol{w}}$ is zero wherever $m_i = 0$, and only weights where $m_i = 1$ are updated.

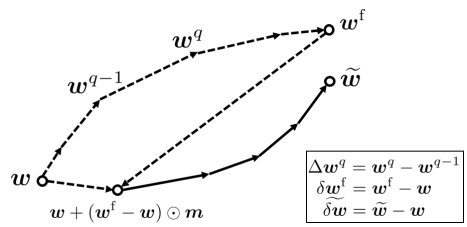

Fig. 1: The figure depicts the overall approach that consists of two steps. The first step is depicted with dotted arrows and starts from the deployed model $\boldsymbol{w}$. In Q optimization steps, all weights are trained to the optimum $\boldsymbol{w}^{\mathrm{f}}$. Based on the collected information, a binary mask $\boldsymbol{m}$ is determined that characterizes the set of weights that are rewound to the ones of $\boldsymbol{w}$. Therefore, the second step (solid arrows) starts from $\boldsymbol{w}+\delta\boldsymbol{w}^{\mathrm{f}}\odot\boldsymbol{m}$. According to the mask, this solution is sparsely fine-tuned to the final weights $\widetilde{\boldsymbol{w}}$, *i.e.*, $\widetilde{\delta\boldsymbol{w}}$ has only non-zero values where the mask value is 1

4.1 Metrics for Rewinding

We will now describe a new metric that determines the weights that should be kept constant, *i.e.*, with $m_i = 0$. Like most learning methods, we focus on minimizing a loss function. The two-step approach relies on the following assumption: the better the loss $\ell(\boldsymbol{w} + \delta\boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m})$ of the initial solution for the second step, the better the final performance. Therefore, the first step should select a mask $\boldsymbol{m}$ such that the loss difference $\ell(\boldsymbol{w} + \delta\boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m}) - \ell(\boldsymbol{w}^{\mathrm{f}})$ is as small as possible.

To determine an optimized mask $\boldsymbol{m}$, we propose to upper-bound the above loss difference in two view points, and measure each weight's contribution to the bounds. The "global contribution" uses the norm information of incremental weights $\delta\boldsymbol{w}^{\mathrm{f}} = \boldsymbol{w}^{\mathrm{f}} - \boldsymbol{w}$. The "local contribution" takes into account the gradient-based information that is gathered during the optimization in the first step, *i.e.*, in the path from $\boldsymbol{w}$ to $\boldsymbol{w}^{\mathrm{f}}$. Both contributions will be combined to determine the mask $\boldsymbol{m}$.

The two view points are based on the concept of smooth differentiable functions. A function $f(x)$ with $f : \mathbb{R}^d \to \mathbb{R}$ is called L-smooth if it has a Lipschitz continuous gradient $g(x)$: $\|g(x) - g(y)\|_2 \leq L\|x - y\|_2$ for all x, y. Note that

Lipschitz continuity of gradients is essential to ensuring convergence of many gradient-based algorithms. Under such a condition, one can derive the following bounds, see also [22]:

$$|f(y) - f(x) - g(x)^{\mathrm{T}} \cdot (y - x)| \leq L/2 \cdot \|y - x\|_2^2 \quad \forall x, y \tag{3}$$

Global contribution. One would argue that a large absolute value in $\delta \boldsymbol{w}^{\mathrm{f}} = \boldsymbol{w}^{\mathrm{f}} - \boldsymbol{w}$ indicates that this weight has moved far from its initial value in $\boldsymbol{w}$, and thus should not be rewound. This motivates us to adopt the widely used unstructured magnitude pruning to determine the mask $\boldsymbol{m}$. Magnitude pruning prunes the weights with the lowest magnitudes, which often achieves a good trade-off between the model accuracy and the number of zero's weights [29].

Using $a-b \leq |a-b|$, Eq.(3) can be reformulated as $f(y)-f(x)-g(x)^T(y-x) \leq |f(y) - f(x) - g(x)^T(y - x)| \leq L/2 \cdot \|y - x\|_2^2$. Thus, we can bound the relevant loss difference $\ell(\boldsymbol{w} + \delta \boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m}) - \ell(\boldsymbol{w}^{\mathrm{f}}) \geq 0$ as

$$\ell(\boldsymbol{w}+\delta \boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m}) - \ell(\boldsymbol{w}^{\mathrm{f}}) \leq \boldsymbol{g}(\boldsymbol{w}^{\mathrm{f}})^{\mathrm{T}} \cdot \big(\delta \boldsymbol{w}^{\mathrm{f}} \odot (\boldsymbol{m} - \boldsymbol{1})\big) + L/2 \cdot \|\delta \boldsymbol{w}^{\mathrm{f}} \odot (\boldsymbol{m} - \boldsymbol{1})\|_2^2 \tag{4}$$

where $\boldsymbol{g}(\boldsymbol{w}^{\mathrm{f}})$ denotes the loss gradient at $\boldsymbol{w}^{\mathrm{f}}$. As the loss is optimized at $\boldsymbol{w}^{\mathrm{f}}$, *i.e.*, $\boldsymbol{g}(\boldsymbol{w}^{\mathrm{f}}) \approx \boldsymbol{0}$, we can assume that the gradient term is much smaller than the norm of the weight differences in Eq.(4). Therefore, we have

$$\ell(\boldsymbol{w} + \delta \boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m}) - \ell(\boldsymbol{w}^{\mathrm{f}}) \lesssim L/2 \cdot \|\delta \boldsymbol{w}^{\mathrm{f}} \odot (\boldsymbol{1} - \boldsymbol{m})\|_2^2 \tag{5}$$

The right hand side is clearly minimized if $m_i = 1$ for the largest absolute values of $\delta \boldsymbol{w}^{\mathrm{f}}$. As $\boldsymbol{1}^{\mathrm{T}} \cdot \big(\boldsymbol{c}^{\mathrm{global}} \odot (\boldsymbol{1} - \boldsymbol{m})\big) = \|\delta \boldsymbol{w}^{\mathrm{f}} \odot (\boldsymbol{1} - \boldsymbol{m})\|_2^2$, this information is captured in the contribution vector

$$\boldsymbol{c}^{\mathrm{global}} = \delta \boldsymbol{w}^{\mathrm{f}} \odot \delta \boldsymbol{w}^{\mathrm{f}} \tag{6}$$

The $k \cdot I$ weights with the largest values in $\boldsymbol{c}^{\mathrm{global}}$ are assigned to mask values 1 and are further fine-tuned in the second step, whereas all others are rewound to their initial values in $\boldsymbol{w}$. Alg. 2 in Appendix A.1 shows this first approach.

Local contribution. As experiments show, one can do better when leveraging in addition some gradient-based information gathered during the first step, *i.e.*, optimizing the initial weights $\boldsymbol{w}$ in Q traditional optimization steps, $\boldsymbol{w} = \boldsymbol{w}^0 \rightarrow \cdots \rightarrow \boldsymbol{w}^{q-1} \rightarrow \boldsymbol{w}^q \rightarrow \cdots \rightarrow \boldsymbol{w}^Q = \boldsymbol{w}^{\mathrm{f}}$.

Using $-a+b \leq |a-b|$, Eq.(3) can be reformulated as $f(x) - f(y) + g(x)^T(y - x) \leq |f(y) - f(x) - g(x)^T(y - x)| \leq L/2 \cdot \|y - x\|_2^2$. Thus, each optimization step is bounded as

$$\ell(\boldsymbol{w}^{q-1}) - \ell(\boldsymbol{w}^q) \leq -\boldsymbol{g}(\boldsymbol{w}^{q-1})^{\mathrm{T}} \cdot \Delta \boldsymbol{w}^q + L/2 \cdot \|\Delta \boldsymbol{w}^q\|_2^2 \tag{7}$$

where $\Delta \boldsymbol{w}^q = \boldsymbol{w}^q - \boldsymbol{w}^{q-1}$. For a conventional gradient descent optimizer with a small learning rate we can use the approximation $|\boldsymbol{g}(\boldsymbol{w}^{q-1})^{\mathrm{T}} \cdot \Delta \boldsymbol{w}^q| \gg \|\Delta \boldsymbol{w}^q\|_2^2$ and obtain $\ell(\boldsymbol{w}^{q-1}) - \ell(\boldsymbol{w}^q) \lesssim -\boldsymbol{g}(\boldsymbol{w}^{q-1})^{\mathrm{T}} \cdot \Delta \boldsymbol{w}^q$. Summing up over all optimization iterations yields approximately

$$\ell(\boldsymbol{w}^{\mathrm{f}} - \delta \boldsymbol{w}^{\mathrm{f}}) - \ell(\boldsymbol{w}^{\mathrm{f}}) \lesssim -\sum_{q=1}^{Q} \boldsymbol{g}(\boldsymbol{w}^{q-1})^{\mathrm{T}} \cdot \Delta \boldsymbol{w}^q \tag{8}$$

Note that we have $\boldsymbol{w} = \boldsymbol{w}^{\mathrm{f}} - \delta\boldsymbol{w}^{\mathrm{f}}$ and $\delta\boldsymbol{w}^{\mathrm{f}} = \sum_{q=1}^{Q} \Delta\boldsymbol{w}^{q}$. Therefore, with a small updating ratio k, *i.e.*, $\boldsymbol{m} \sim \mathbf{0}$, we can reformulate Eq.(8) as $\ell\left(\boldsymbol{w} + \delta\boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m}\right) - \ell(\boldsymbol{w}^{\mathrm{f}}) \lesssim \mathrm{U}(\boldsymbol{m})$ with the upper bound $\mathrm{U}(\boldsymbol{m}) = -\sum_{q=1}^{Q} \boldsymbol{g}(\boldsymbol{w}^{q-1})^{\mathrm{T}} \cdot (\Delta\boldsymbol{w}^{q} \odot (\mathbf{1} - \boldsymbol{m}))$ where we suppose that the gradients are approximately constant for $\boldsymbol{m} \sim \mathbf{0}$ (*i.e.*, $\boldsymbol{m}$ has zero entries almost everywhere). Therefore, an approximate incremental contribution of each weight dimension to the upper bound on the loss difference $\ell\left(\boldsymbol{w} + \delta\boldsymbol{w}^{\mathrm{f}} \odot \boldsymbol{m}\right) - \ell(\boldsymbol{w}^{\mathrm{f}})$ can be determined by the negative gradient vector at $\boldsymbol{m} = \mathbf{0}$, denoted as

$$\boldsymbol{c}^{\mathrm{local}} = -\frac{\partial \mathrm{U}(\boldsymbol{m})}{\partial \boldsymbol{m}} = -\sum_{q=1}^{Q} \boldsymbol{g}(\boldsymbol{w}^{q-1}) \odot \Delta\boldsymbol{w}^{q} \tag{9}$$

which models the accumulated contribution to the overall loss reduction.

Combining global and local contribution. So far, we independently calculate the global and local contributions. To avoid the scale impact, we first normalize each contribution by its significance in its own set (either global or local contribution set). We investigated the impacts and the different combinations of both normalized contributions, see the results in Appendix G. Interestingly, the most straightforward combination (*i.e.*, the sum of both normalized metrics) often yields a satisfied and stable performance. Intuitively, local contribution can better identify critical weights w.r.t. the loss during training, while global contribution may be more robust for a highly non-convex loss landscape. Both metrics may be necessary when selecting weights to rewind. Therefore, the combined contribution is computed as

$$\boldsymbol{c} = \frac{1}{\mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{c}^{\mathrm{global}}} \boldsymbol{c}^{\mathrm{global}} + \frac{1}{\mathbf{1}^{\mathrm{T}} \cdot \boldsymbol{c}^{\mathrm{local}}} \boldsymbol{c}^{\mathrm{local}} \tag{10}$$

and $m_i = 1$ for the $k \cdot I$ largest values of $\boldsymbol{c}$ and $m_i = 0$ otherwise. The pseudocode of Deep Partial Updating (DPU), *i.e.*, rewinding according to the combined contribution to the loss reduction, is shown in Alg. 1. The complexity analysis of the corresponding algorithm are shown in Appendix B.

4.2 (Re-)Initialization of Weights

In this section, we discuss the initialization of DPU. $\mathcal{D}^1$ denotes the initial dataset used to train the model $\boldsymbol{w}^1$ from a randomly initialized model $\boldsymbol{w}^0$. $\mathcal{D}^1$ corresponds to the available dataset before deployment, or collected in the 0-th round if there are no data available before deployment. $\{\delta\mathcal{D}^r\}_{r=2}^{R}$ denotes newly collected samples in each subsequent round.

Experimental results show (see Appendix D) that training from a randomly initialized model can yield a higher accuracy *after a large number of rounds*, compared to always training from the last round $\boldsymbol{w}^{r-1}$. As a possible explanation, the optimizer could end in a hard to escape region of the search space if always trained from the last round for a long sequence of rounds. Thus, we propose to re-initialize the weights after a certain number of rounds. In such a case, Alg. 1

Algorithm 1: Deep Partial Updating

Input: weights $\boldsymbol{w}$, updating ratio k, learning rate $\{\alpha^q\}_{q=1}^{Q}$ in Q iterations
Output: weights $\widetilde{\boldsymbol{w}}$
`/* The first step: full updating and rewinding */`
Initiate $\boldsymbol{w}^0 = \boldsymbol{w}$ and $\boldsymbol{c}^{\text{local}} = \boldsymbol{0}$;
for $q \leftarrow 1$ **to** Q **do**
  Compute the loss gradient $\boldsymbol{g}(\boldsymbol{w}^{q-1}) = \partial \ell(\boldsymbol{w}^{q-1}) / \partial \boldsymbol{w}^{q-1}$;
  Compute the optimization step with learning rate α^q as $\Delta \boldsymbol{w}^q$;
  Update $\boldsymbol{w}^q = \boldsymbol{w}^{q-1} + \Delta \boldsymbol{w}^q$;
  Update $\boldsymbol{c}^{\text{local}} = \boldsymbol{c}^{\text{local}} - \boldsymbol{g}(\boldsymbol{w}^{q-1}) \odot \Delta \boldsymbol{w}^q$;
Set $\boldsymbol{w}^{\text{f}} = \boldsymbol{w}^Q$ and get $\delta \boldsymbol{w}^{\text{f}} = \boldsymbol{w}^{\text{f}} - \boldsymbol{w}$;
Compute $\boldsymbol{c}^{\text{global}} = \delta \boldsymbol{w}^{\text{f}} \odot \delta \boldsymbol{w}^{\text{f}}$;
Compute $\boldsymbol{c}$ as Eq.(10) and sort in descending order;
Create a binary mask $\boldsymbol{m}$ with 1 for the Top-$(k \cdot I)$ indices, 0 for others;
`/* The second step: sparse fine-tuning */`
Initiate $\widetilde{\delta \boldsymbol{w}} = \delta \boldsymbol{w}^{\text{f}} \odot \boldsymbol{m}$ and $\widetilde{\boldsymbol{w}} = \boldsymbol{w} + \widetilde{\delta \boldsymbol{w}}$;
for $q \leftarrow 1$ **to** Q **do**
  Compute the optimization step on $\widetilde{\boldsymbol{w}}$ with learning rate α^q as $\Delta \widetilde{\boldsymbol{w}}^q$;
  Update $\widetilde{\delta \boldsymbol{w}} = \widetilde{\delta \boldsymbol{w}} + \Delta \widetilde{\boldsymbol{w}}^q \odot \boldsymbol{m}$ and $\widetilde{\boldsymbol{w}} = \boldsymbol{w} + \widetilde{\delta \boldsymbol{w}}$;

does not start from the weights $\boldsymbol{w}^{r-1}$ but from the randomly initialized weights. The randomly re-initialized model (weights) can be efficiently sent to the edge devices via a single random seed. The device can determine the weights by means of a random generator. This process realizes a random shift in the search space, which is a communication-efficient way in comparison to other alternatives, such as learning to increase the loss or using the (averaged) weights in the previous rounds, as these fully changed weights still need to be sent to each node. Each time the model is randomly re-initialized, the new partially updated model might suffer from an accuracy drop in a few rounds. However, we can simply avoid such an accuracy drop by not updating the model if the validation accuracy does not increase compared to the last round, see Appendix E. Note that the learned knowledge thrown away by re-initialization can be re-learned afterwards, since all collected samples are continuously stored and accumulated in the server. This also makes our setting different from continual learning, that aims at avoiding catastrophic forgetting without accessing old data.

To determine after how many rounds the model should be re-initialized, we conduct extensive experiments on different partial updating settings, see Appendix E. In conclusion, the model is randomly re-initialized as long as the number of total newly collected data samples exceeds the number of samples when the model was re-initialized last time. For example, assume that at round r the model is randomly (re-)initialized and partially updated from this random model on dataset $\mathcal{D}^r$. Then, the model will be re-initialized again at round $r+n$, if $|\mathcal{D}^{r+n}| > 2 \cdot |\mathcal{D}^r|$, where $|.|$ denotes the number of samples in the dataset.

5 Evaluation

In this section, we experimentally show that through updating a small subset of weights, DPU can reach a similar accuracy as full updating while requiring a significantly lower communication cost. We implement DPU with Pytorch [23], and evaluate on public vision datasets, including MNIST [16], CIFAR10 [15], CIFAR100 [15], ImageNet [30], using multilayer perceptron (MLP), VGGNet [5,28,26], ResNet56 [10], MobileNetV1 [12], respectively. Particularly, we partition the experiments into multi-round updating and single-round updating.

Multi-round updating. We consider there are limited (or even zero) samples before the initial deployment, and data samples are continuously collected and sent from edge devices over a long period (the event rate is often low in real cases [21]). The server retrains the model and sends the updates to each device in multiple rounds. Regarding the highly-constrained communication resources, we choose low resolution image datasets (MNIST [16] and CIFAR10/100 [15]) to evaluate multi-round updating. We conduct one-shot rewinding in multi-round DPU, *i.e.*, rewinding is executed only once to achieve the desired updating ratio at each round as in Alg. 1, which avoids hand-tuning hyperparameters (*e.g.*, updating ratio schedule) frequently over a large number of rounds.

Single-round updating. The deployed model is updated once via server-to-edge communication when new data from other sources become available on the server after some time, *e.g.*, releasing a new version of mobile applications based on newly retrieved internet data. Although DPU is elaborated and designed under multi-round updating settings, it can be applied directly in single-round updating. Since transmission from edge devices may be even not necessary, we evaluate single-round DPU on the large scale ImageNet dataset. Iterative rewinding is adopted here due to its better performance. Particularly, we alternatively perform rewinding 20% of the remaining trainable weights according to Eq.(10) and sparse fine-tuning until reaching the desired updating ratio.

Settings for all experiments. We randomly select 30% of the original test dataset (original validation dataset for ImageNet) as the validation dataset, and the remainder as the test dataset. Let $\{|\mathcal{D}^1|, |\delta\mathcal{D}^r|\}$ represent the available data samples along rounds, where $|\delta\mathcal{D}^r|$ is supposed to be constant along rounds. Both $\mathcal{D}^1$ and $\delta\mathcal{D}^r$ are randomly sampled (without replacement) from the original training dataset to simulate the data collection. In each round, the test accuracy is reported, when the validation dataset achieves the highest Top-1 accuracy during retraining. When the validation accuracy does not increase compared to the previous round, the models are not updated to reduce the communication overhead. This strategy is also applied to other baselines to enable a fair comparison. We use the average cross-entropy as the loss function, and use Adam variant of SGD for MLP and VGGNet, Nesterov SGD for ResNet56 and MobileNetV1. More implementation details are provided in Appendix C.

Indexing. DPU generates a sparse tensor. In addition to the updated weights, the indices of these weights also need to be sent to each edge device. A simple implementation is to send the mask $\boldsymbol{m}$, *i.e.*, a binary vector of I elements.

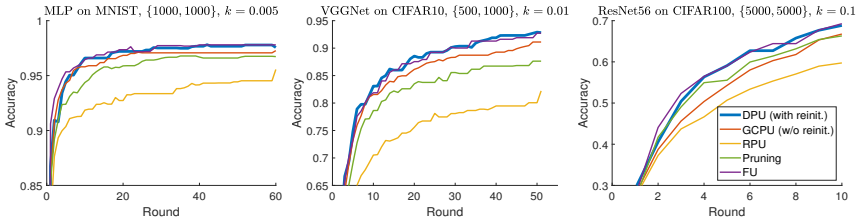

Fig. 2: DPU is compared with other baselines on different benchmarks in terms of the test accuracy during multi-round updating

Let S_w denote the bitwidth of each single weight, and S_x denote the bitwidth of each index. Directly sending $\boldsymbol{m}$ yields an overall communication cost of $I \cdot k \cdot S_w + I \cdot S_x$ with $S_x = 1$. To save the communication cost on indexing, we further encode $\boldsymbol{m}$. Suppose that $\boldsymbol{m}$ is a random binary vector with a probability of k to contain 1. The optimal encoding scheme according to Shannon yields $S_x(k) = k \cdot \log(1/k) + (1-k) \cdot \log(1/(1-k))$. Coding schemes such as Huffman block coding can come close to this bound. We use $S_w \cdot k \cdot I + S_x(k) \cdot I$ to report the size of data transmitted from server to each node at each round, contributed by the partially updated weights plus the encoded indices of these weights.

5.1 Benchmarking Multi-Round Updating

Settings. To the best of our knowledge, this is the first work on studying weight-wise partial updating a model using newly collected data in iterative rounds. Therefore, we developed three baselines for comparison, including (i) full updating (FU), where at each round the model is fully updated from a random initialization ($i.e.$, training from scratch, which yields a better performance see Sec. 4.2 and Appendix D); (ii) random partial updating (RPU), where the model is trained from $\boldsymbol{w}^{r-1}$, while we randomly fix each layer's weights with a ratio of $(1-k)$ and sparsely fine-tune the rest; (iii) global contribution partial updating (GCPU), where the model is trained with Alg. 2 without re-initialization described in Sec. 4.2; (iv) a state-of-the-art unstructured pruning method [29], where the model is first trained from a random initialization at each round, then conducts one-shot magnitude pruning, and finally is sparsely fine-tuned with learning rate rewinding. The ratio of nonzero weights in pruning is set to the same as the updating ratio k to ensure the same communication cost. The experiments are conducted on different benchmarks as mentioned earlier.

Results. We report the test accuracy of the model $\boldsymbol{w}^r$ along rounds in Fig. 2. All methods start from the same $\boldsymbol{w}^0$, an entirely randomly initialized model. As seen in this figure, DPU clearly yields the highest accuracy in comparison to the other partial updating schemes. For example, DPU can yield a final Top-1 accuracy of 92.85% on VGGNet, even exceeds the accuracy (92.73%) of full updating. In addition, we compare three partial updating schemes in terms of the accuracy difference related to full updating averaged over all rounds, and

Table 1: The average accuracy difference over all rounds and the ratio of communication cost over all rounds related to full updating

Method	Average accuracy difference			Ratio of communication cost		
	MLP	VGGNet	ResNet56	MLP	VGGNet	ResNet56
DPU	**−0.17%**	**+0.33%**	**−0.42%**	0.0071	0.0183	0.1147
GCPU	−0.72%	−1.51%	−3.87%	0.0058	0.0198	0.1274
RPU	−4.04%	−11.35%	−7.78%	0.0096	0.0167	0.1274
Pruning [29]	−1.45%	−4.35%	−2.35%	0.0106	0.0141	0.1274

the ratio of the communication cost related to full updating over all rounds in Tab. 1. As seen in the table, DPU reaches a similar accuracy as full updating, while incurring significantly fewer transmitted data sent from the server to each edge node. Specially, DPU saves around 99.3%, 98.2% and 88.5% of transmitted data on MLP, VGGNet, and ResNet56, respectively (95.3% in average). The communication cost ratios shown in Tab. 1 differ a little even for the same updating ratio. This is because if the validation accuracy does not increase, the model will not be updated to reduce the communication cost, as discussed earlier.

We further investigate the benefit due to DPU in terms of *the total communication cost reduction*, as DPU has no impact on the edge-to-server communication involving sending newly collected data samples. This experimental setup assumes that all samples in $\delta\mathcal{D}^r$ are collected by N edge nodes during all rounds and sent to the server on a per-round basis. For clarity, let S_d denote the data size of each training sample. During round r, we define per-node communication cost under DPU as $S_d \cdot |\delta\mathcal{D}^r|/N + (S_w \cdot k \cdot I + S_x(k) \cdot I)$. The results are shown in Appendix F.1. We observe that DPU can still achieve a significant reduction on the total communication cost, *e.g.*, reducing up to 88.2% even for the worst case (a single node). Moreover, DPU tends to be more beneficial when the size of data transmitted by each node to the server becomes smaller. This is intuitive because in this case server-to-edge communication cost (thus the reduction due to DPU) dominates the total communication cost.

5.2 Different Number of Data Samples and Updating Ratios

Settings. In this section, we show that DPU outperforms other baselines under varying number of training samples and updating ratios in multi-round updating. We also conduct ablations concerning the re-initialization of weights discussed in Sec. 4.2. We implement DPU with and without re-initialization, GCPU with and without re-initialization, RPU, and Pruning [29] (see more details in Sec. 5.1) on VGGNet using CIFAR10 dataset. We compare these methods with different amounts of samples $\{|\mathcal{D}^1|, |\delta\mathcal{D}^r|\}$ and different updating ratios k. Without further notations, each experiment runs three times using random data samples.

Results. We compare the difference between the accuracy under each method and that under full updating. The mean accuracy difference over three runs is

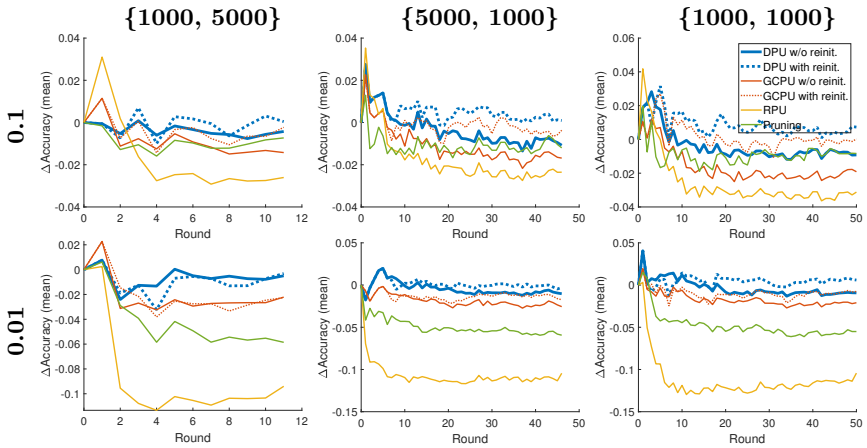

Fig. 3: Comparison w.r.t. the mean accuracy difference (full updating as the reference) under different $\{|\mathcal{D}^1|, |\delta\mathcal{D}^r|\}$ (representing the available data samples along rounds, see in Sec. 5) and updating ratio ($k = 0.1, 0.01$)

plotted in Fig. 3. A comprehensive set of results including the absolute accuracy and the standard deviations is provided in Appendix F.2. As seen in Fig. 3, DPU (with re-initialization) always achieves the highest accuracy. DPU also significantly outperforms the pruning method, especially under a small updating ratio. Note that we preferred a smaller updating ratio in our context because it explores the limits of the approach and it indicates that we can improve the deployed model more frequently with the same accumulated server-to-edge communication cost. The dashed curves and the solid curves with the same color can be viewed as the ablation study of our re-initialization scheme. Particularly given a large number of rounds, it is critical to re-initialize the start point $\boldsymbol{w}^{r-1}$ after several rounds (as discussed in Sec. 4.2).

In the first few rounds, partial updating methods almost always yield a higher test accuracy than full updating, *i.e.*, the curves are above zero. This is due to the fact that the amount of available samples is rather small, and partial updating may avoid some co-adaptation in full updating. The partial updating methods perform almost randomly in the first round compared to each other, because the limited data are not sufficient to distinguish critical weights from the random initialization $\boldsymbol{w}^0$. This also motivates us to (partially) update the deployed model when new samples are available.

Pruning weights vs. pruning incremental weights. One of our chosen baselines, global contribution partial updating (GCPU, see Alg. 2), could be viewed as a counterpart of the pruning method [29], *i.e.*, pruning the incremental weights with the least magnitudes. By comparing GCPU (with or without re-initialization) with "Pruning", we conclude that retaining previous weights yields better performance than zero-outing the weights.

5.3 Benchmarking Single-Round Updating

Settings. To show the versatility of our methods, we test single-round updating for MobileNetV1 [12] on ImageNet [30] with iterative rewinding. Single-round DPU is conducted on different initially deployed models, including a floating-point (FP32) dense model and two compressed models, *i.e.*, a 50%-sparse model and an INT8 quantized model. The sparse model is trained with a state-of-the-art dynamic pruning method [24]; the quantized model is trained with straight-through-estimator with a output-channel-wise floating point scaling factors similar as [28]. To maintain the same on-device inference cost, partial updating is only applied on nonzero values of sparse models; for quantized models, the updated weights are still in INT8 format.

Results. We compare DPU with the vanilla-updates, *i.e.*, the models are trained from a random initialization with the corresponding methods on all available samples. The test accuracy and the ratio of (server-to-edge) communication cost related to full updating on FP32 dense model are reported in Tab. 2. Results show that DPU often yields a higher accuracy than vanilla updating while requiring substantially lower communication cost.

Table 2: The test accuracy of single-round updating on different initial models (MobileNetV1 on ImageNet). The updating ratio $k = 0.2$. The ratio of communication cost related to full updating is reported in brackets

#Samples	$\{8 \times 10^5, 4.8 \times 10^5\}$		
	Initial	Vanilla-update	DPU
FP32 Dense	68.5%	70.7% (1)	71.1% (0.22)
50%-Sparse	68.1%	70.5% (0.53)	70.8% (0.22)
INT8	68.4%	70.6% (0.25)	70.6% (0.07)

6 Conclusion

In this paper, we present the weight-wise deep partial updating paradigm, motivated by the fact that continuous full weight updating may be impossible in many edge intelligence scenarios. We present DPU through analytically upper-bounding the loss difference between partial updating and full updating. DPU only updates the weights that make the largest contributions to the upper bound, while reuses the other weights that have less impact on the loss reduction. Extensive experimental results demonstrate the efficacy of DPU which achieves a high inference accuracy while updating a rather small number of weights.

Acknowledgement

Part of Zhongnan Qu and Lothar Thiele's work was supported by the Swiss National Science Foundation in the context of the NCCR Automation. Part of Cong Liu's work was supported by NSF CNS 2135625, CPS 2038727, CNS Career 1750263, and a Darpa Shell grant.

References

1. Ash, J.T., Zhang, C., Krishnamurthy, A., Langford, J., Agarwal, A.: Deep batch active learning by diverse, uncertain gradient lower bounds. In: International Conference on Learning Representations (ICLR) (2020), https://openreview.net/forum?id=ryghZJBKPS
2. Augustin, A., Yi, J., Clausen, T., Townsley, W.M.: A study of lora: Long range & low power networks for the internet of things. Sensors **16**(9) (2016). https://doi.org/10.3390/s16091466, https://www.mdpi.com/1424-8220/16/9/1466
3. Bhandare, A., Sripathi, V., Karkada, D., Menon, V., Choi, S., Datta, K., Saletore, V.: Efficient 8-bit quantization of transformer neural machine language translation model. In: International Conference on Machine Learning (ICML), Joint Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations (2019)
4. Brown, S., Sreenan, C.: Updating software in wireless sensor networks: A survey. Dept. of Computer Science, National Univ. of Ireland, Maynooth, Tech. Rep pp. 1–14 (2006)
5. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2015)
6. Evci, U., Gale, T., Menick, J., Castro, P.S., Elsen, E.: Rigging the lottery: Making all tickets winners. In: International Conference on Machine Learning (ICML) (2021)
7. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (ICLR) (2019), https://openreview.net/forum?id=rJl-b3RcF7
8. Guo, T.: Cloud-based or on-device: An empirical study of mobile deep inference. In: 2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018. pp. 184–190. IEEE Computer Society (2018). https://doi.org/10.1109/IC2E.2018.00042, https://doi.org/10.1109/IC2E.2018.00042
9. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
11. Horowitz, M.: 1.1 computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC) (2014)
12. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR (2017)
13. Jung, S., Ahn, H., Cha, S., Moon, T.: Adaptive group sparse regularization for continual learning. CoRR (2020)
14. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R.G.L., Rouayheb, S.E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P.B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi,

M., Javidi, T., Joshi, G., Khodak, M., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S.U., Sun, Z., Suresh, A.T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F.X., Yu, H., Zhao, S.: Advances and open problems in federated learning. CoRR (2019)

15. Krizhevsky, A., Nair, V., Hinton, G.: Cifar10/100 (canadian institute for advanced research) (2009), `http://www.cs.toronto.edu/~kriz/cifar.html`
16. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010), `http://yann.lecun.com/exdb/mnist/`
17. Lee, J., Chirkov, N., Ignasheva, E., Pisarchyk, Y., Shieh, M., Riccardi, F., Sarokin, R., Kulik, A., Grundmann, M.: On-device neural net inference with mobile gpus. CoRR (2019)
18. Li, A., Sun, J., Wang, B., Duan, L., Li, S., Chen, Y., Li, H.: Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. CoRR (2020)
19. Lin, Y., Han, S., Mao, H., Wang, Y., Dally, W.J.: Deep gradient compression: Reducing the communication bandwidth for distributed training. In: International Conference on Learning Representations (ICLR) (2018), `https://openreview.net/forum?id=SkhQHMW0W`
20. Meng, Z., Qin, H., Chen, Z., Chen, X., Sun, H., Lin, F., Ang, M.H.: A two-stage optimized next-view planning framework for 3-d unknown environment exploration, and structural reconstruction. IEEE Robotics and Automation Letters **2**(3), 1680–1687 (2017)
21. Meyer, M., Farei-Campagna, T., Pasztor, A., Forno, R.D., Gsell, T., Faillettaz, J., Vieli, A., Weber, S., Beutel, J., Thiele, L.: Event-triggered natural hazard monitoring with convolutional neural networks on the edge. In: International Conference on Information Processing in Sensor Networks (IPSN). IPSN '19, Association for Computing Machinery (2019)
22. Nesterov, Y.: Introductory lectures on convex programming volume i: Basic course. Lecture notes **1**, 25 (1998)
23. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques (2017)
24. Peste, A., Iofinova, E., Vladu, A., Alistarh, D.: Ac/dc: Alternating compressed/decompressed training of deep neural networks. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2021)
25. Pytorch: Pytorch example on resnet (2019), `https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py`, accessed: 2019-10-15
26. Qu, Z., Zhou, Z., Cheng, Y., Thiele, L.: Adaptive loss-aware quantization for multi-bit networks. In: The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
27. Raghu, A., Raghu, M., Bengio, S., Vinyals, O.: Rapid learning or feature reuse? towards understanding the effectiveness of MAML. In: International Conference on Learning Representations (ICLR) (2020), `https://openreview.net/forum?id=rkgMkCEtPB`
28. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision (ECCV) (2016)

29. Renda, A., Frankle, J., Carbin, M.: Comparing fine-tuning and rewinding in neural network pruning. In: International Conference on Learning Representations (ICLR) (2020), https://openreview.net/forum?id=S1gSj0NKvB
30. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)
31. Shen, Z., Liu, Z., Qin, J., Savvides, M., Cheng, K.: Partial is better than all: Revisiting fine-tuning strategy for few-shot learning. In: The AAAI Conference on Artificial Intelligence (AAAI) (2021)
32. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. CCS '15 (2015). https://doi.org/10.1145/2810103.2813687, https://doi.org/10.1145/2810103.2813687
33. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (ICLR) (2015)
34. Stowell, D., Wood, M., Pamuła, H., Stylianou, Y., Glotin, H.: Automatic acoustic detection of birds through deep learning: The first bird audio detection challenge. Methods in Ecology and Evolution **10** (10 2018). https://doi.org/10.1111/2041-210X.13103
35. Varshney, A.: Enabling Sustainable Networked Embedded Systems. Ph.D. thesis, Uppsala UniversityUppsala University, Division of Computer Systems, Computer Architecture and Computer Communication (2018)
36. Wang, Z., K, K., Mayhew, S., Roth, D.: Extending multilingual bert to low-resource languages. CoRR (2020)
37. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. In: International Conference on Learning Representations (ICLR) (2018), https://openreview.net/forum?id=Sk7KsfW0-
38. Zhou, H., Lan, J., Liu, R., Yosinski, J.: Deconstructing lottery tickets: Zeros, signs, and the supermask. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2019)
39. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q.: A comprehensive survey on transfer learning. CoRR (2019)