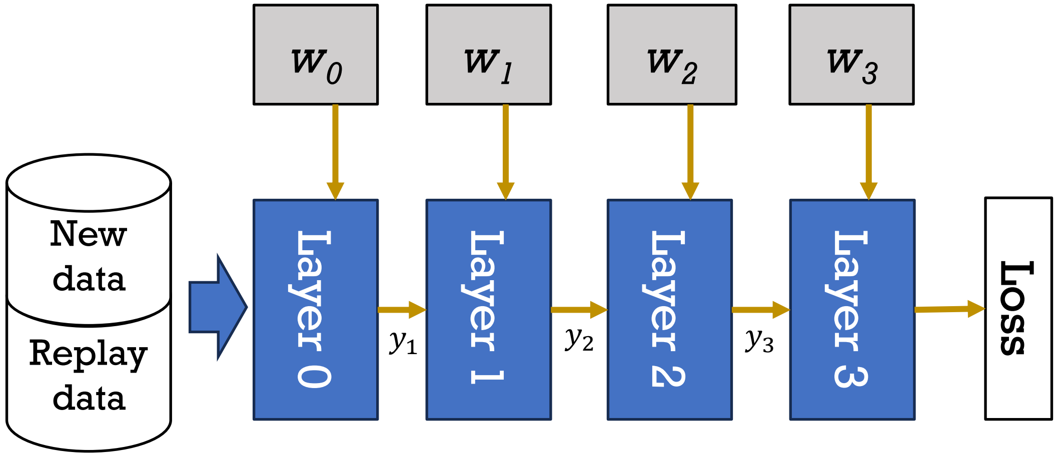
Memory and Latency Efficient On-Device Continual Learning: Trends & Tricks

Manuele Rusci, KU Leuven
manuele.rusci@kuleuven.be

Recap: ODL costs

Forward, a.k.a. inference



inference		
	Working Mem*	$\max\ (sz(y_i) + sz(y_{i+1}) + sz(w_i))$
	Parameters	$\sum sz(w_i)$

$sz(\cdot)$ returns the number of elements

* assume layer-wise execution, sample-by-sample

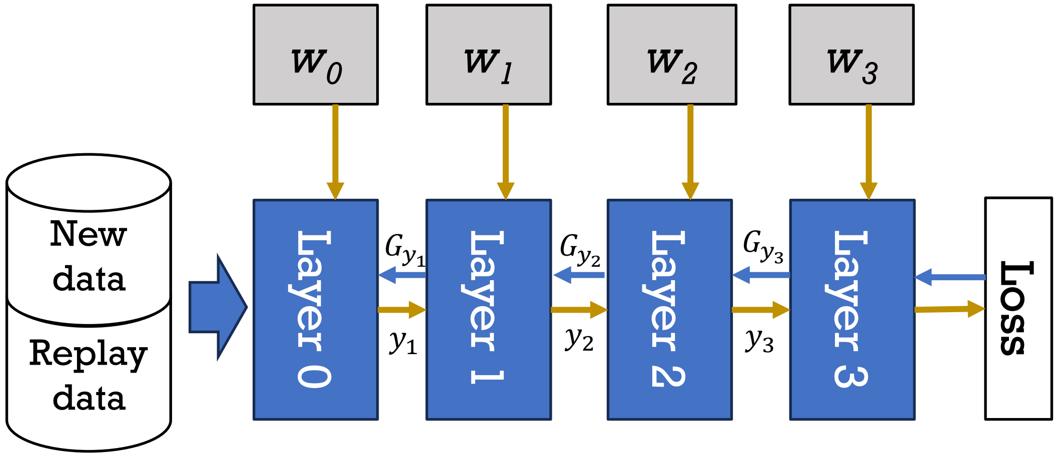
(batch)
Forward (FW)

<u>Convolution Layer ***i***</u>

$$y_{i+1} = w_i \cdot y_i$$

Recap: ODL costs

Backward the input gradients



inference		
	Working Mem*	$\max\ (sz(y_i) + sz(y_{i+1}) + sz(w_i))$
	Parameters	$\sum sz(w_i)$

$sz(\cdot)$ returns the number of elements
* assume layer-wise execution, sample-by-sample

Convolution Layer i

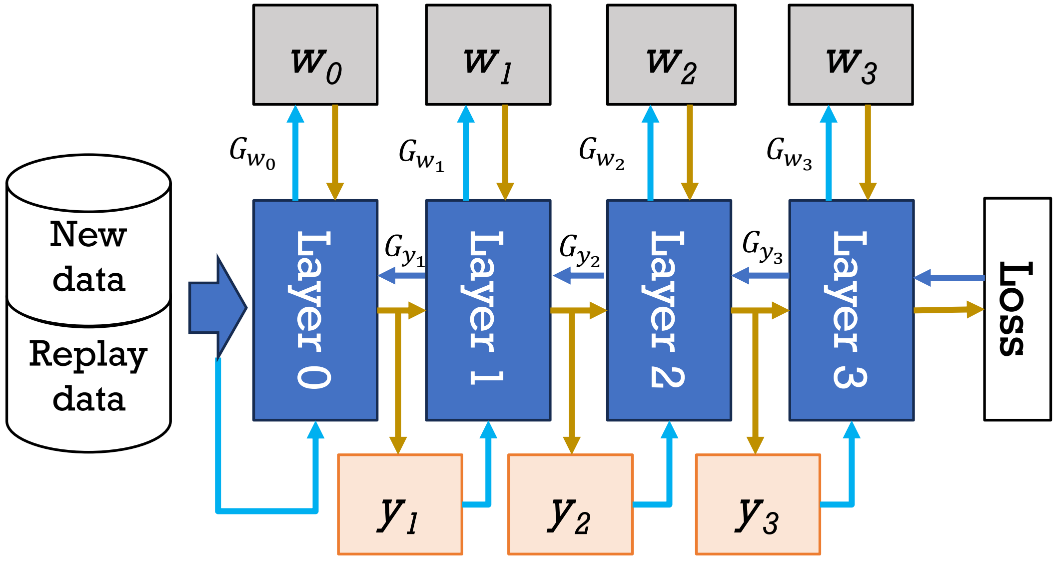
(batch) **Forward** (FW) → $y_{i+1} = w_i \cdot y_i$

Backward input gradient (BW_{ig}) ← $G_{y_i} = w_i \cdot G_{y_{i+1}}$

Note:
$G_x = \frac{\partial L}{\partial x}$

Recap: ODL costs

Backward the weight gradients



<u>Convolution Layer *i*</u>

(batch) **Forward** (FW)

$$y_{i+1} = w_i \cdot y_i$$

+ storing y_i

Backward input gradient (BW_{ig})

$$G_{y_i} = w_i \cdot G_{y_{i+1}}$$

Backward weight gradient (BW_{wg})

$$G_{w_i} = y_i \cdot G_{y_{i+1}}$$

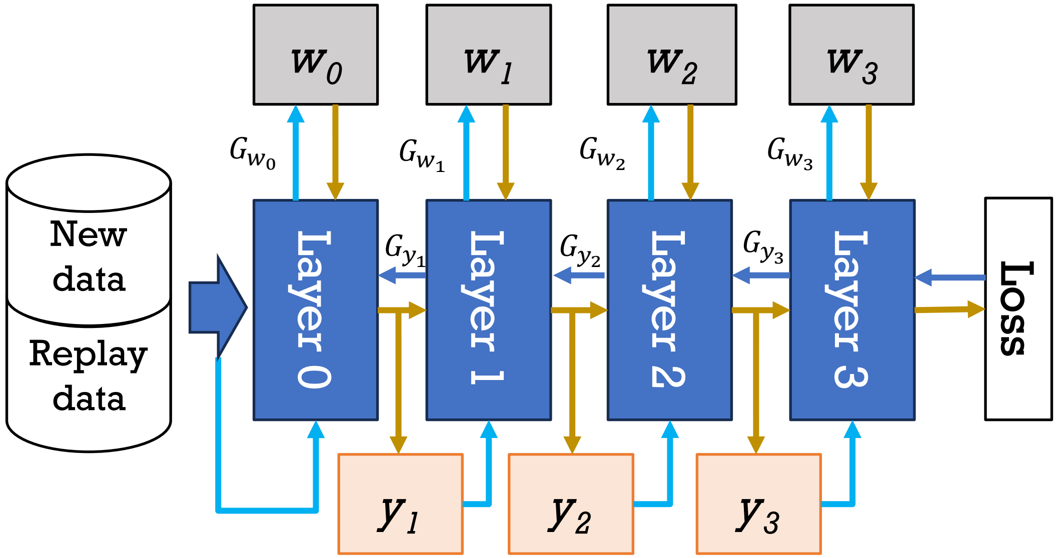
then, update
$w_i \leftarrow w_i + \eta\, G_{w_i}$

Note:
$G_x = \frac{\partial L}{\partial x}$

inference	**Working Mem***	$\max\ (sz(y_i) + sz(y_{i+1}) + sz(w_i))$
	Parameters	$\sum sz(w_i)$
+ training	**Weight Gradients**	$\sum sz(G_{w_i})\ (== \sum sz(w_i))$
	Activation Storage for BW	$\sum sz(y_i) \cdot N_{data_b}$
	Data/Replay Mem Buffer	$N_{batch} \cdot N_{data_b} \cdot sz(data)$

$sz(\cdot)$ returns the number of elements
* assume layer-wise execution, sample-by-sample

Recap: ODL costs



$$T = \underbrace{E}_{\#\text{epochs}} \cdot \overbrace{N_{batch}}^{\#\text{batches}} \cdot \underbrace{N_{data_b}}_{\#\text{data per batch}} \cdot \overbrace{(T_{FW} + T_{BW_{wg}} + T_{BW_{wg}})}^{\text{execution time per sample}}$$

Online (Streaming) Learning:
$N_{data_b} = 1$ and $E = 1$

inference	**Working Mem***	$\max\ (sz(y_i) + sz(y_{i+1}) + sz(w_i))$
inference	**Parameters**	$\sum sz(w_i)$
+ training	**Weight Gradients**	$\sum sz(G_{w_i})\ (== \sum sz(w_i))$
+ training	**Activation Storage for BW**	$\sum sz(y_i) \cdot N_{data_b}$
+ training	**Data/Replay Mem Buffer**	$N_{batch} \cdot N_{data_b} \cdot sz(data)$

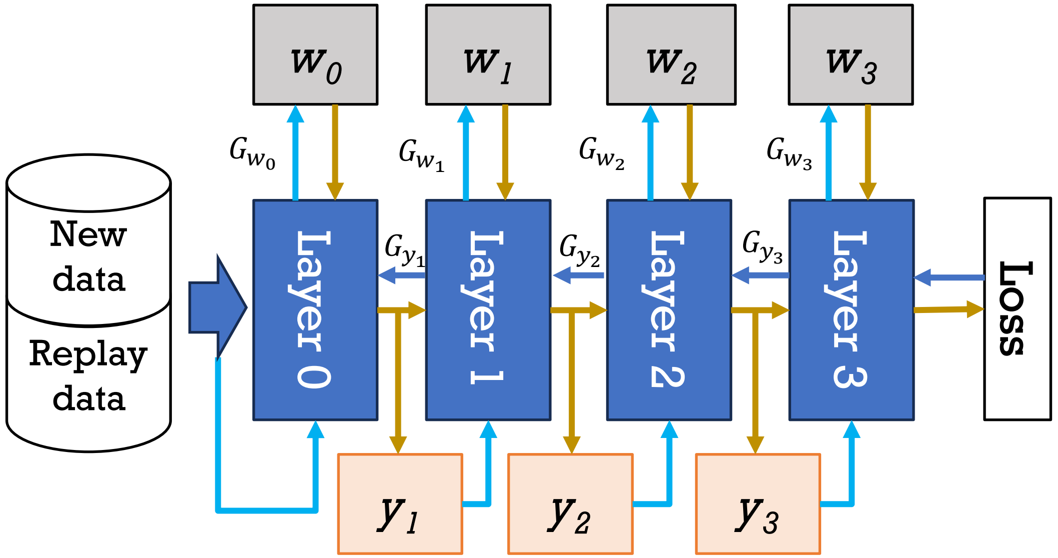
$sz(\cdot)$ returns the number of elements
* assume layer-wise execution, sample-by-sample

<u>Convolution Layer ***i***</u>

(batch) **Forward** (FW)	$y_{i+1} = w_i \cdot y_i$	+ storing y_i
Backward input gradient (BW_{ig})	$G_{y_i} = w_i \cdot G_{y_{i+1}}$	
Backward weight gradient (BW_{wg})	$G_{w_i} = y_i \cdot G_{y_{i+1}}$	then, update $w_i \leftarrow w_i + \eta\, G_{w_i}$

Note:
$G_x = \frac{\partial L}{\partial x}$

Recap: ODL costs



$$T = \underbrace{E}_{\text{\#epochs}} \cdot \overbrace{N_{batch}}^{\text{\#batches}} \cdot \underbrace{N_{data_b}}_{\text{\#data per batch}} \cdot \overbrace{(T_{FW} + T_{BW_{wg}} + T_{BW_{wg}})}^{\text{execution time per sample}}$$

Online (Streaming) Learning:
$N_{data_b} = 1$ and $E = 1$

inference	**Working Mem***	$\max\ (sz(y_i) + sz(y_{i+1}) + sz(w_i))$
	Parameters	$\sum sz(w_i)$
+ training	**Weight Gradients**	$\sum sz(G_{w_i})\ (== \sum sz(w_i))$
	Activation Storage for BW	$\sum sz(y_i) \cdot N_{data_b}$
	Data/Replay Mem Buffer	$N_{batch} \cdot N_{data_b} \cdot sz(data)$

$sz(\cdot)$ returns the number of elements
* assume layer-wise execution, sample-by-sample

<u>Convolution Layer ***i***</u>

(batch) **Forward** (FW) $$y_{i+1} = w_i \cdot y_i$$ + storing y_i

Backward input gradient (BW_{ig}) $$G_{y_i} = w_i \cdot G_{y_{i+1}}$$

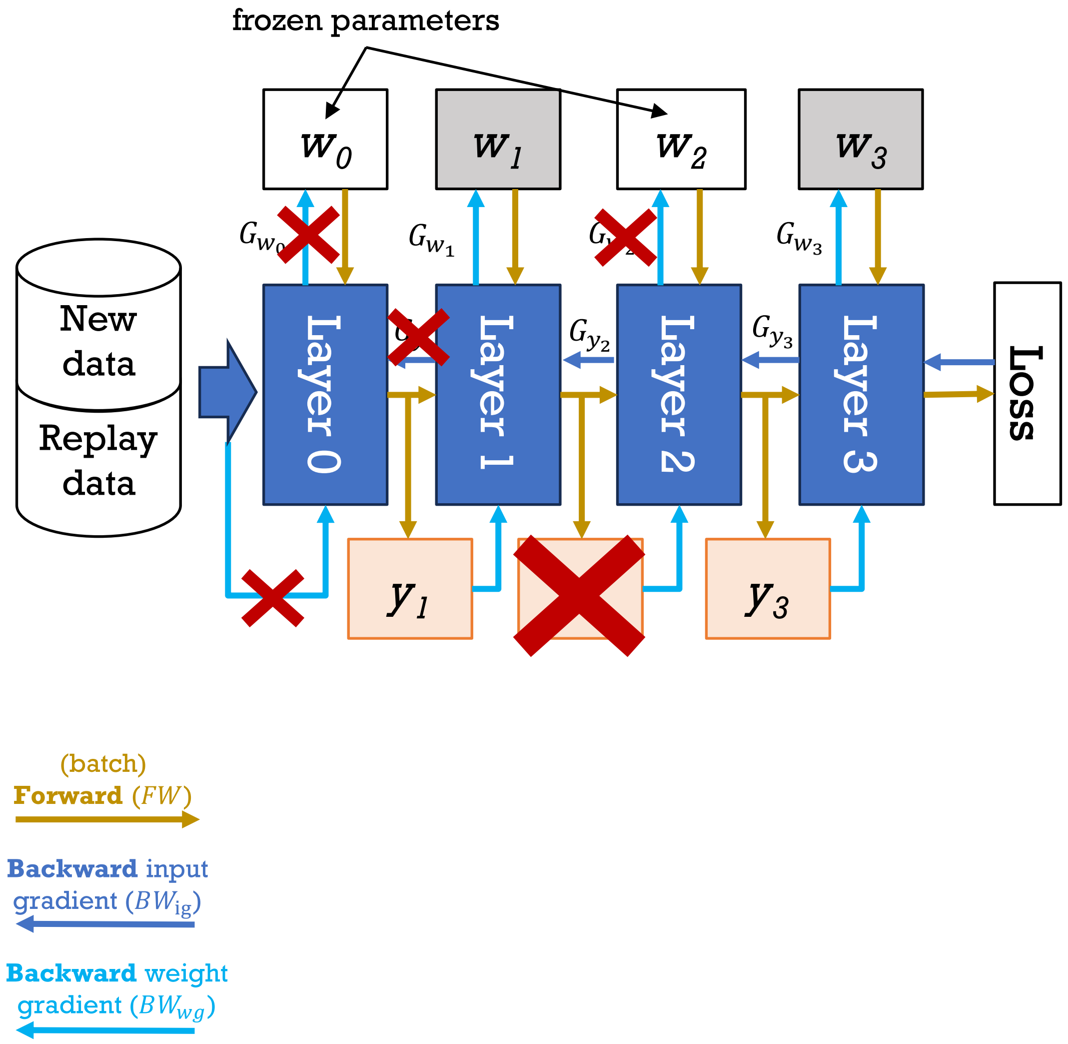
Backward weight gradient (BW_{wg}) $$G_{w_i} = y_i \cdot G_{y_{i+1}}$$

then, update
$w_i \leftarrow w_i + \eta\, G_{w_i}$

Note:
$G_x = \frac{\partial L}{\partial x}$

Problem: High latency and (activation) memory costs

Efficient Trainable Models

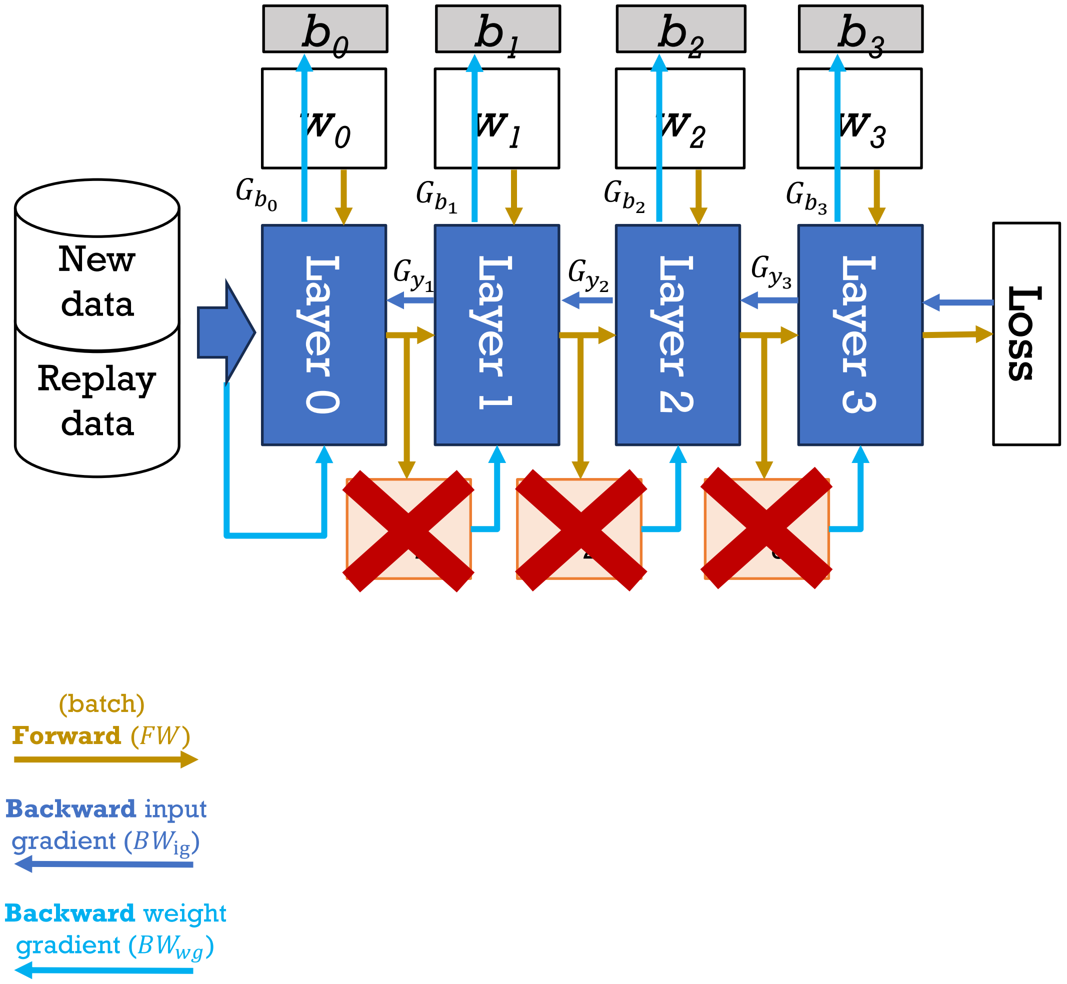


<u>Update only few layers</u> (w_1, w_3)

- ✓ Lower backward time ($T_{BW_{ig}}$, $T_{BW_{wg}}$) vs. full-backprop
- ✓ Lower gradient & activations vs. full-backprop
- × Lower accuracy than full-backprop

- Retraining only the last layer on MCUs, e.g. *TinyOL* [Ren2021]
- First layers have larger activation sizes (and more generic features): keep them frozen!

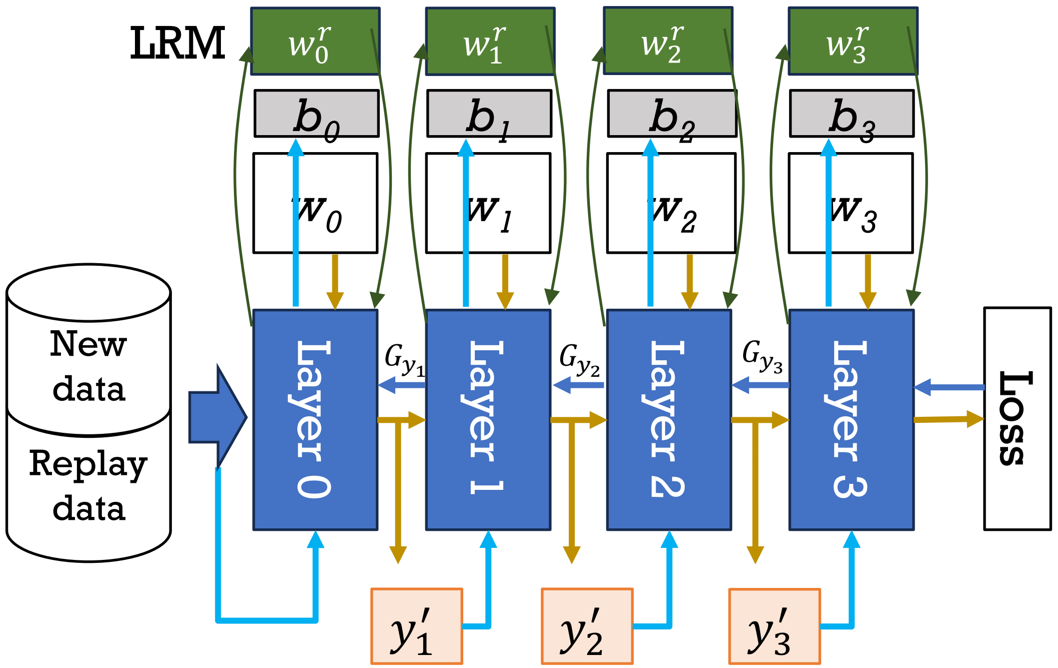
TinyTL: Tiny Transfer Learning [Cai2020]



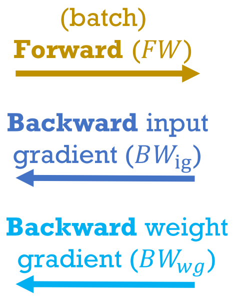
<u>Retraining only the biases</u>

$y_{i+1} = w_i \cdot y_i + b_i \rightarrow \boxed{G_{b_i} = G_{y_{i+1}}}$ does *not* depend from y_i

TinyTL: Tiny Transfer Learning [Cai2020]



smaller than y_1, y_2, y_3



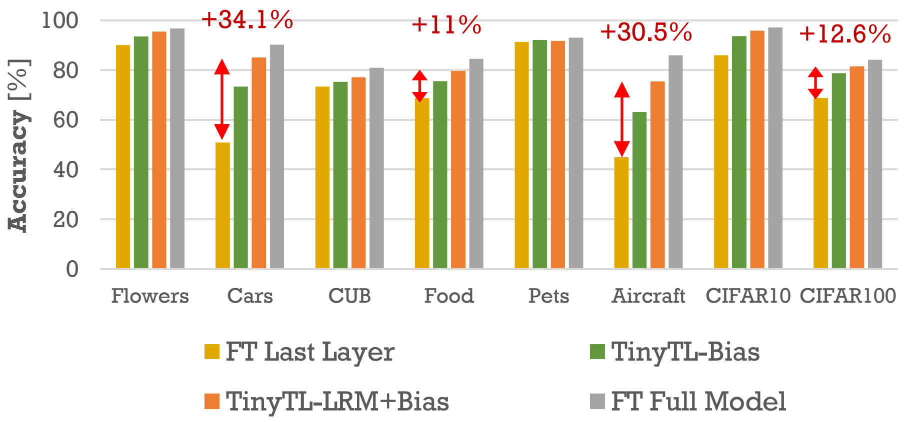
Retraining only the biases

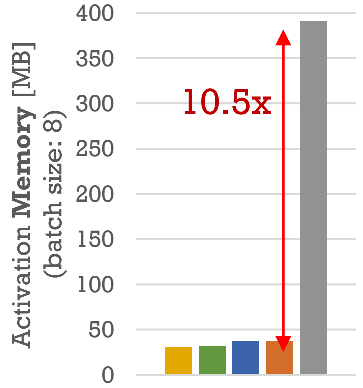
$$y_{i+1} = w_i \cdot y_i + b_i \rightarrow \boxed{G_{b_i} = G_{y_{i+1}}}$$

does *not* depend from y_i

Lite Residual Modules (LRM)

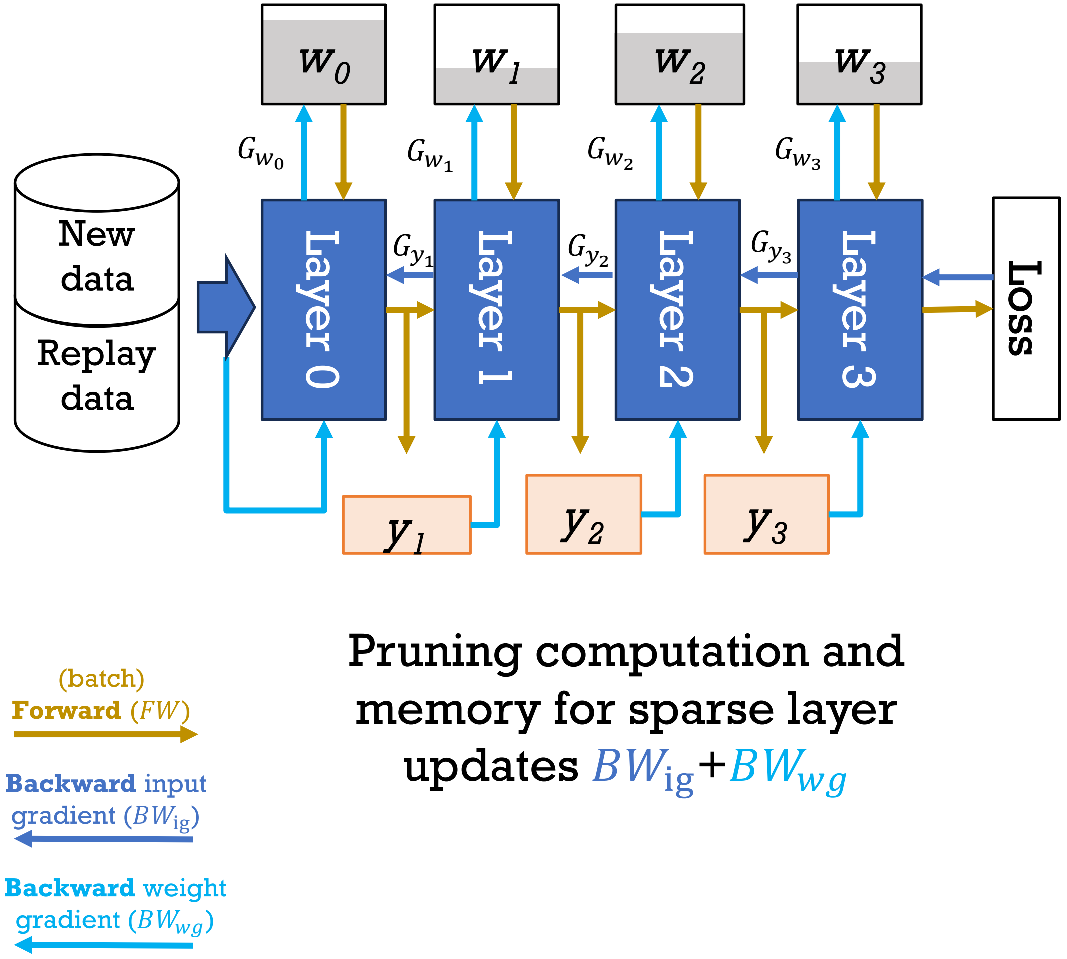
$$y_{i+1} = w_i \cdot y_i + b_i + \boxed{f_{w_i^r}(y'_i = pool(y_i))}$$





LRM+B achieves the best memory accuracy trade-off

(Structured) Sparse Updates



Pruning computation and memory for sparse layer updates BW_{ig}+BW_{wg}

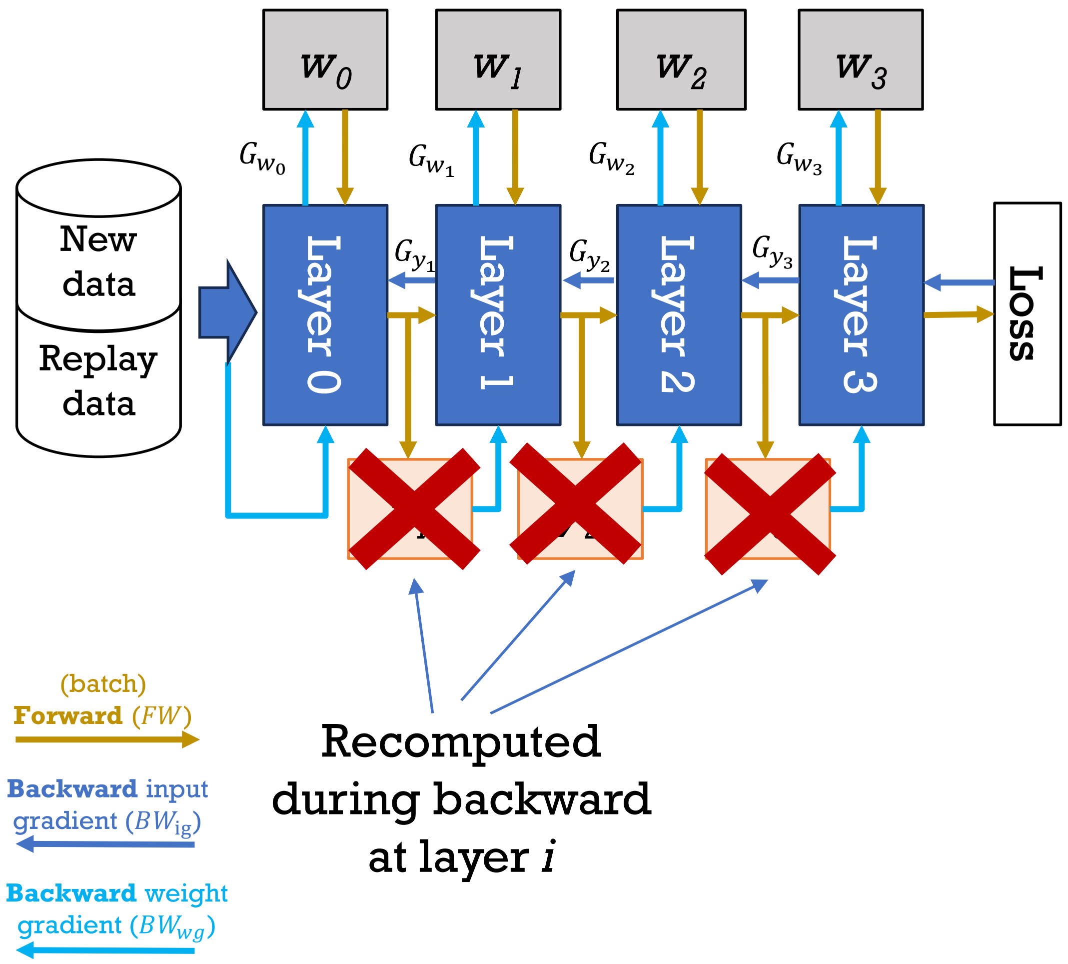
Pruning weight gradient computation BW_{wg} (and BW_{ig}) of less important weight or sub-tensors [Lin2022][Kwon2023]

- High transfer learning capacity (less overfitting vs. full-retraining) but **4.5-7x** memory saving [Lin2022]

Which weights to update?

- Evolutionary search (offline) with a per-layer contribution as the cost function [Lin2022]
- Multi-objective cost w/ Fisher information (online) [Kwon2023]

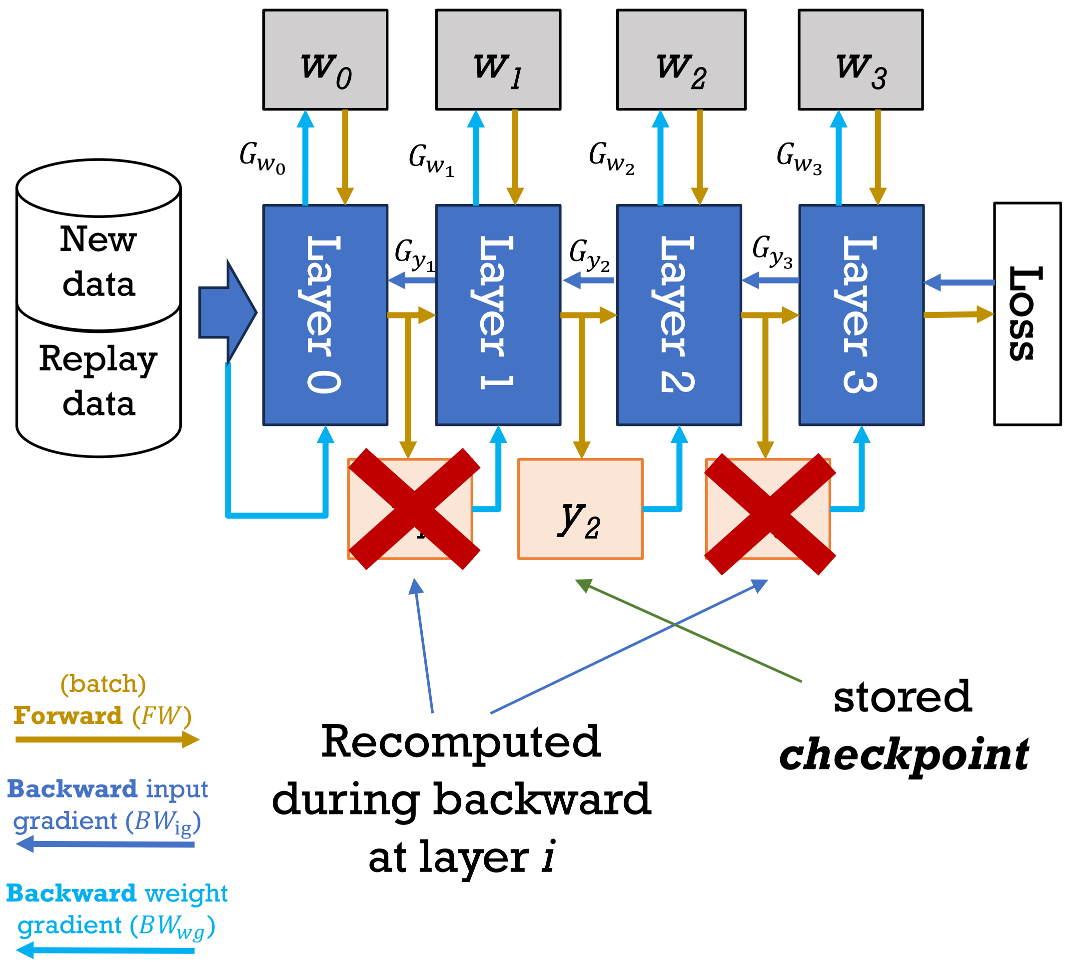
“Removing” the Memory Constraints



Update **all** parameters **without** storing the activations

- Activation tensors are recomputed layer-wise backward

"Removing" the Memory Constraints



Update **all** parameters **without** storing the activations

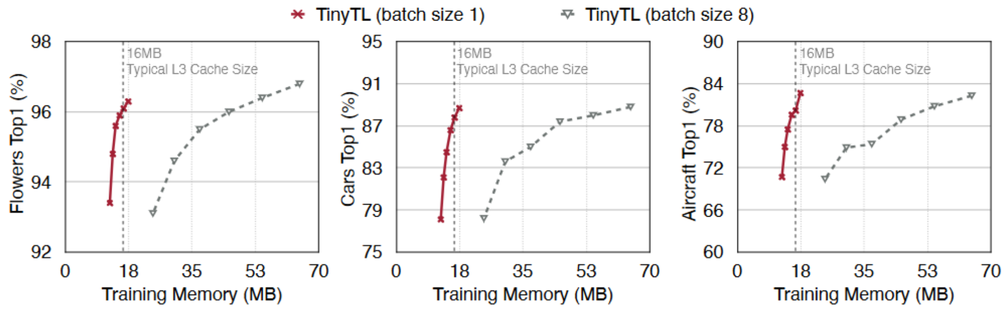
- Activation tensors are recomputed layer-wise backward

Only stores some **checkpoints** for faster training

- Convenient to recompute cheap-to-compute yet memory-intensive tensors, e.g., ReLU layers.
- *DaCapo* [Khan2023]
  - Exhaustive search to select checkpoints (w/ mem and latency constraints)
- *POET* [Patil2022]
  - Mixed Integer Linear Programming
  - Combined with paging: activations copied to off-chip memories
  - Trains ResNet-18 and BERT on tiny ARM Cortex M class devices

Other Relevant Tricks

- Batch Norm requires large batch sizes for accurate stats
  - **Group Norm** for small batch size [Cai2020]



Group Norm with batch size=1 same acc. vs. bs=8

- Lossless Low-precision training
  - Mixed-precision Training (FP32+FP16) [Narang2017]
  - INT8 [Lin2022] with Quantization-Aware Scaling: $G_W = G_W \cdot s_W^{-2}$

Scaling factor from quantization

- Replay Storage (& activation)
  - Low-bitwidth quantization (≤8-bit) [Ravaglia2021]
  - Product Quantization (PQ) compression [Hayes2020]

What is (or can be) next!

- HW-SW co-design for ODL for real-world continual learning benchmarks
  - Training algorithms under memory and latency constraints
  - Absence of or few labels available for continual learning
    - Few-shot & auxiliary tasks
  - Convergence time of the training algorithms (#epochs, #data)
    - Under-explored domain.

- Applications of On-Device Continual Learning
  - Detaching from “classic” benchmark datasets (Cifar10, Mnist,….)

- Novel MCU HW architectures
  - Always-on inference + <u>occasionally training</u>
  - New opportunities for heterogeneity

Reference

- [Lin2023] Lin, Ji, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han. "Tiny Machine Learning: Progress and Futures." IEEE Circuits and Systems Magazine, 2023.
- [Ren2021] Ren, Haoyu, Darko Anicic, and Thomas A. Runkler. "Tinyol: Tinyml with online-learning on microcontrollers." *IJCNN*, 2021.
- [Cai2020] Cai, Han, Chuang Gan, Ligeng Zhu, and Song Han. "Tinytl: Reduce memory, not parameters for efficient on-device learning.“, Neurips 2020.
- [Lin2022] Lin, Ji, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. "On-device training under 256kb memory." Neurips, 2022.
- [Kwon2023] Kwon, Young D., Rui Li, Stylianos I. Venieris, Jagmohan Chauhan, Nicholas D. Lane, and Cecilia Mascolo. "Tinytrain: Deep neural network training at the extreme edge." *arXiv,* 2023.
- [Patil2022] Patil, Shishir G., Paras Jain, Prabal Dutta, Ion Stoica, and Joseph Gonzalez. "Poet: Training neural networks on tiny devices with integrated rematerialization and paging." ICML, 2022.
- [Khan2023] Khan, Osama, Gwanjong Park, and Euiseong Seo. "DaCapo: An On-Device Learning Scheme for Memory-Constrained Embedded Systems." *TECS 2023*.
- [Narang2017] Narang, Sharan, Gregory Diamos, Erich Elsen, Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg et al. "Mixed precision training." ICLR. 2017.
- [Ravaglia2021] Ravaglia, Leonardo, Manuele Rusci, Davide Nadalini, Alessandro Capotondi, Francesco Conti, and Luca Benini. "A tinyml platform for on-device continual learning with quantized latent replays." IEEE JETCAS 2021.
- [Hayes2020] Hayes, Tyler L., Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. "Remind your neural network to prevent catastrophic forgetting.“ ECCV 2020.

Questions

Manuele Rusci, KU Leuven
manuele.rusci@kuleuven.be

Davide Nadalini, Università di Bologna
d.nadalini@unibo.it

Cristian Cioflan, ETH Zürich
cioflanc@iis.ee.ethz.ch