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

\[ y_{i+1} = w_i \cdot y_i \]

Working Mem* \[ \max (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

Parameters \[ \sum sz(w_i) \]

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

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

Convolution Layer \( i \)

DATE24 tutorial - M. Rusci
Recap: ODL costs

Backward the input gradients

\[ G_y = w_i \cdot G_{y_i+1} \]

\[ y_{i+1} = w_i \cdot y_i \]

\[ \text{Working Mem}\* = \max (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

\[ \sum sz(w_i) \]

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

\[ G_x = \frac{\partial L}{\partial x} \]

Convolution Layer \( i \)

\[ y_i = w_i \cdot y_{i-1} \]

\[ G_y = w_i \cdot G_{y_i+1} \]

Parameters

\[ \sum sz(w_i) \]

\[ \text{Working Mem}\* = \max (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

\[ \sum sz(w_i) \]

\[ \text{Working Mem}\* = \max (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

\[ \sum sz(w_i) \]

\[ \text{Working Mem}\* = \max (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

\[ \sum sz(w_i) \]

\[ \text{Working Mem}\* = \max (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

\[ \sum sz(w_i) \]
Recap: ODL costs

Backward the weight gradients

- Working Mem* \( \max (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \)
- Parameters \( \sum sz(w_i) \)
- Weight Gradients \( \sum sz(G_{wi}) (== \sum sz(w_i)) \)
- 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

Note: \( G_x = \frac{\delta L}{\delta x} \)
Recap: ODL costs

\[ T = E \cdot N_{\text{batch}} \cdot N_{\text{data b}} \cdot (T_{\text{FW}} + T_{BW_{wg}} + T_{BW_{wg}}) \]

- **#epochs**
- **#batches**
- **#data per batch**

**Online (Streaming) Learning:**
\[ N_{\text{data b}} = 1 \text{ and } E = 1 \]

**Working Mem**
\[ \text{max} (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

**Parameters**
\[ \sum sz(w_i) \]

**Weight Gradients**
\[ \sum sz(G_{w_i}) (== \sum sz(w_i)) \]

**Activation Storage for BW**
\[ \sum sz(y_i) \cdot N_{\text{data b}} \]

**Data/Replay Mem Buffer**
\[ N_{\text{batch}} \cdot N_{\text{data b}} \cdot sz(\text{data}) \]

**Note:**
\[ sz(\cdot) \text{ returns the number of elements} \]

* assume layer-wise execution, sample-by-sample
Recap: ODL costs

\[ T = E \cdot N_{batch} \cdot N_{data_b} \cdot (T_{FW} + T_{BW_{wg}} + T_{BW_{wg}}) \]

- **#epochs**
- **#batches**
- **#data per batch**

**Online (Streaming) Learning:**
\[ N_{data_b} = 1 \text{ and } E = 1 \]

### Working Mem*
\[ \text{max } (sz(y_i) + sz(y_{i+1}) + sz(w_i)) \]

### Parameters
\[ \sum sz(w_i) \]

### 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
\[ N_{batch} \cdot N_{data_b} \cdot sz(data) \]

**Problem:** High latency and (activation) memory costs

Notes:
- sz(·) returns the number of elements
- Assume layer-wise execution, sample-by-sample

**Diagram:**
- New data
- Replay data
- Loss
- Convolution Layer \( i \)
  \[ y_{i+1} = w_i \cdot y_i \]
  + storing \( y_i \)
  \[ G_{y_i} = w_i \cdot G_{y_{i+1}} \]
  \[ G_{w_i} = y_i \cdot G_{y_{i+1}} \]
- Forward (\( F_W \))
- Backward input gradient (\( B_{W_{ig}} \))
- Backward weight gradient (\( B_{W_{wg}} \))

DATE24 tutorial - M. Rusci
Efficient Trainable Models

Update only few layers ($w_1, w_3$)
- ✓ 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]

Retraining only the biases

\[ y_{i+1} = w_i \cdot y_i + b_i \rightarrow G_{b_i} = G_{y_{i+1}} \]

does not depend from \( y_i \)
Retraining only the biases

\[ y_{i+1} = w_i \cdot y_i + b_i \rightarrow G_{b_i} = G_{y_{i+1}} \]

doest not depend from \( y_i \)

**Lite Residual Modules (LRM)**

\[ y_{i+1} = w_i \cdot y_i + b_i + f_w^r(y'_i = pool(y_i)) \]

**TinyTL: Tiny Transfer Learning** [Cai2020]
(Structured) Sparse Updates

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

DATE24 tutorial - M. Rusci
Other Relevant Tricks

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

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

- 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 + occasionally training
  ▪ New opportunities for heterogeneity
Reference


Questions

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

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

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