Fault pruning:
Robust training of neural networks with memristive weights

Ceca Kraišniković
Institute of Theoretical Computer Science
Graz University of Technology, Austria
Lab of Dr. Robert Legenstein

Co-authors

Spyros Stathopoulos Themis Prodromakis Robert Legenstein

University of Edinburgh,
United Kingdom

Graz University of
Technology, Austria

Motivation

- **Artificial Intelligence (AI):** large amounts of data processed, demands on computing speed and efficiency

Motivation

- **Artificial Intelligence (AI):** large amounts of data processed, demands on computing speed and efficiency

- **Neuro-inspired chips:** emulate the structure and part of the working mechanisms of the biological brain.

Motivation

- **Artificial Intelligence (AI):** large amounts of data processed, demands on computing speed and efficiency

- **Neuro-inspired chips:** emulate the structure and part of the working mechanisms of the biological brain.
  - Information is stored in the form of *synaptic weights*.

Motivation

- **Artificial Intelligence (AI):** large amounts of data processed, demands on computing speed and efficiency

- **Neuro-inspired chips:** emulate the structure and part of the working mechanisms of the biological brain.
  - Information is stored in the form of *synaptic weights*.
  - *Synaptic plasticity*: ability to increase or decrease synaptic weights by means of changes in conductance.

Motivation

- **Artificial Intelligence (AI):** large amounts of data processed, demands on computing speed and efficiency

- **Neuro-inspired chips:** emulate the structure and part of the working mechanisms of the biological brain.
  - Information is stored in the form of *synaptic weights*.
  - *Synaptic plasticity*: ability to increase or decrease synaptic weights by means of changes in conductance.

  - Main features: neuron-synapse structure, in-memory computation, learning capabilities

Motivation

- **Non-volatile memory devices**:
  - For hardware implementation of biological synapses

Motivation

- **Non-volatile memory devices**:
  - For hardware implementation of biological synapses
  - Two-terminal:
    (a) Resistive Random Access Memory (RRAM),
    (b) Phase-Change Memory (PCM),
    (c) Magnetic Random Access Memory (MRAM)
  - Three-terminal (d)

Motivation

- **Non-volatile memory devices**:
  - For hardware implementation of biological synapses
  - Two-terminal:
    (a) Resistive Random Access Memory (RRAM),
    (b) Phase-Change Memory (PCM),
    (c) Magnetic Random Access Memory (MRAM)
  - Three-terminal (d)

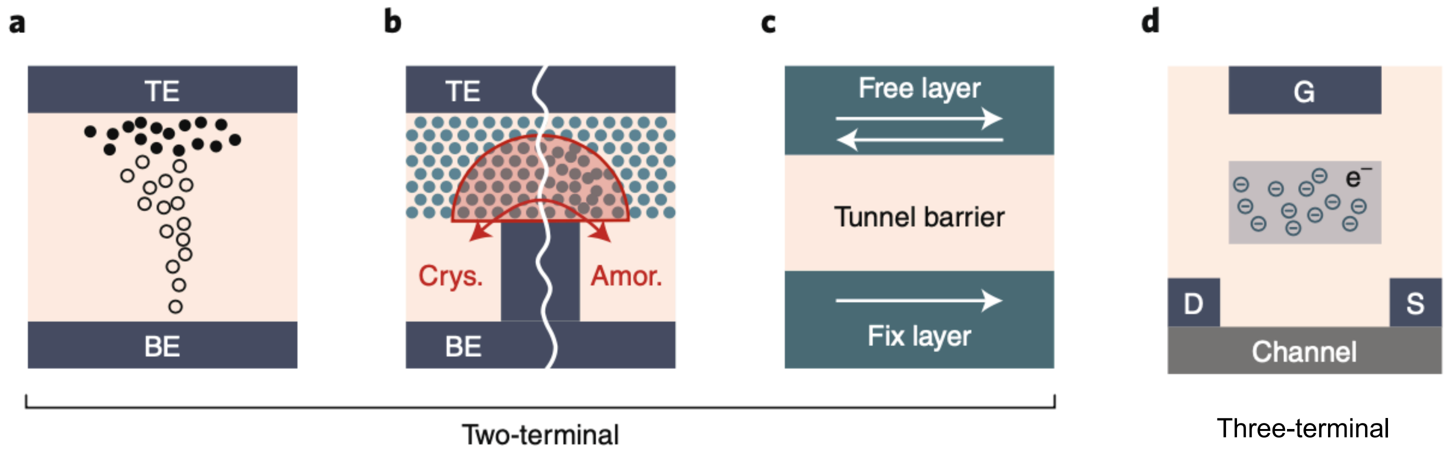


Image source:
Zhang, Wenqiang, et al. "Neuro-inspired computing chips." *Nature electronics* 3.7 (2020): 371-382.

Motivation

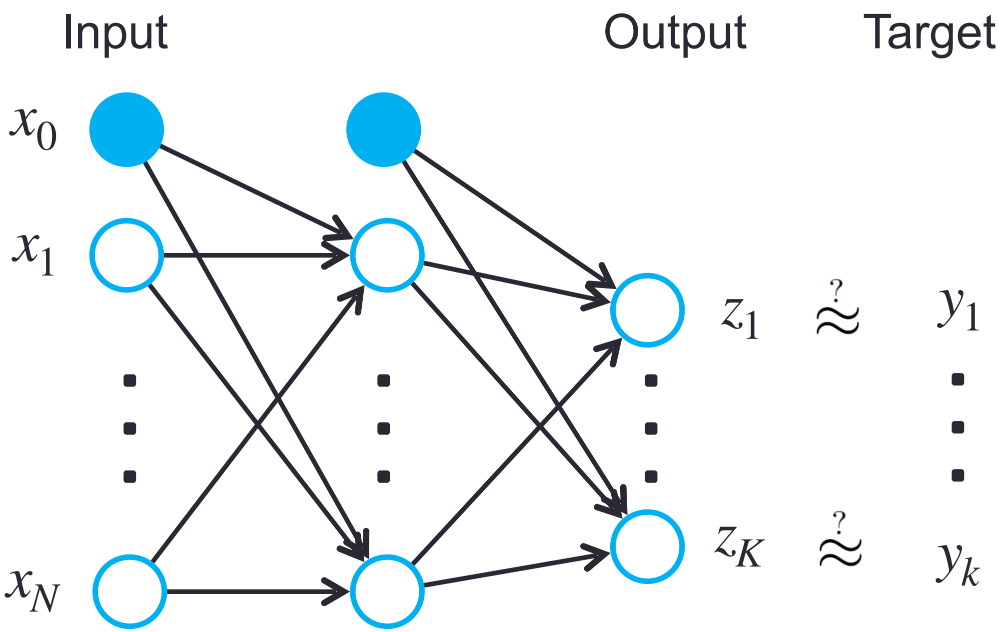
- **Key metrics** for performance evaluation:
  - Computing density
  - Energy-efficiency
  - Computing accuracy: influenced by non-idealities of devices
  - Learning capabilities: off-chip, on-chip, hybrid

Motivation

- **Key metrics** for performance evaluation:
  - Computing density
  - Energy-efficiency
  - Computing accuracy: influenced by non-idealities of devices
  - Learning capabilities: off-chip, on-chip, hybrid

- **Our focus:**
  - RRAM devices (“**memristors**”)
  - Improving energy-efficiency
  - Learning “in-the-loop”:
    - robust training of neural networks with memristive weights
    - detection of faulty memristors
    - improving computing accuracy

Introduction: Neural networks

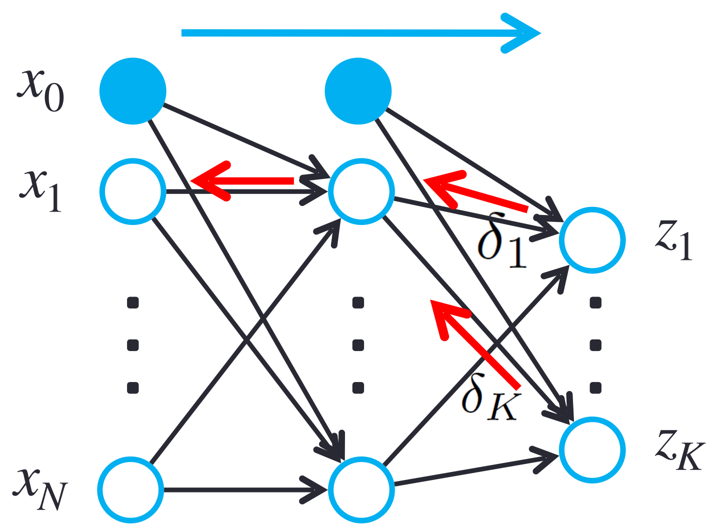


Learning:
the weights are optimized by minimizing the training error.

e.g., $E = \frac{1}{2}\sum_{k=1}^{K} e_k^2 = \frac{1}{2}\sum_{k=1}^{K} (z_k - y_k)^2$

Introduction: Neural networks

- For learning, the gradient of the error function is needed.
  - Forward: Calculate activations and outputs of all neurons.
  - Backward: Calculate errors and propagate them back



Introduction: Memristors

- Mimic biological synapses
- Analog non-volatile two-terminal memory cells
- Resistance R (conductance $G = \frac{1}{R}$) serves as a probed state variable

Introduction: Memristors

- Mimic biological synapses
- Analog non-volatile two-terminal memory cells
- Resistance R (conductance $G = \frac{1}{R}$) serves as a probed state variable

Advantages:

- Can be integrated with ultra-high density
- Can operate in an analog fashion
- Suited for implementation of matrix-vector multiplications
- Low-power consumption

Introduction: Memristors

- Mimic biological synapses
- Analog non-volatile two-terminal memory cells
- Resistance R (conductance $G = \frac{1}{R}$) serves as a probed state variable

Advantages:

- Can be integrated with ultra-high density
- Can operate in an analog fashion
- Suited for implementation of matrix-vector multiplications
- Low-power consumption

Challenges:

- Fabrication, operational constraints
- Limited endurance of the devices
- Yield and repeatability issues

Memristive neural network training

Faulty behavior of memristors

- Stuck memristors
- Faulty updates
  - Concordant switching faults
  - Discordant switching faults

This significantly reduces network performance.

Memristive neural network training

Faulty behavior of memristors

- Stuck memristors
- Faulty updates
  - Concordant switching faults
  - Discordant switching faults

This significantly reduces network performance.

Our approach:

- Analyze impact of faulty memristor behavior on neural network training
- Strategy: Use Fault pruning.
  Detection of faults during training and pruning of connections on the fly.

Memristive weights

- Mapping resistance $R_i \in [R_{\text{min}}, R_{\text{max}}]$ to weight $w_i \in [w_{\text{min}}, w_{\text{max}}]$:

$$w_i = \alpha\left(\frac{1}{R_i} - \frac{1}{R_C}\right)$$

Memristive weights

- Mapping resistance $R_i \in [R_{\text{min}}, R_{\text{max}}]$ to weight $w_i \in [w_{\text{min}}, w_{\text{max}}]$:

$$w_i = \alpha\left(\frac{1}{R_i} - \frac{1}{R_C}\right)$$

- Inverse mapping from weight to resistance:

$$R_i = \frac{1}{\frac{1}{R_C} + \frac{w_i}{\alpha}}$$

Memristive weights

- Mapping resistance $R_i \in [R_{\text{min}}, R_{\text{max}}]$ to weight $w_i \in [w_{\text{min}}, w_{\text{max}}]$:

$$w_i = \alpha\left(\frac{1}{R_i} - \frac{1}{R_C}\right)$$

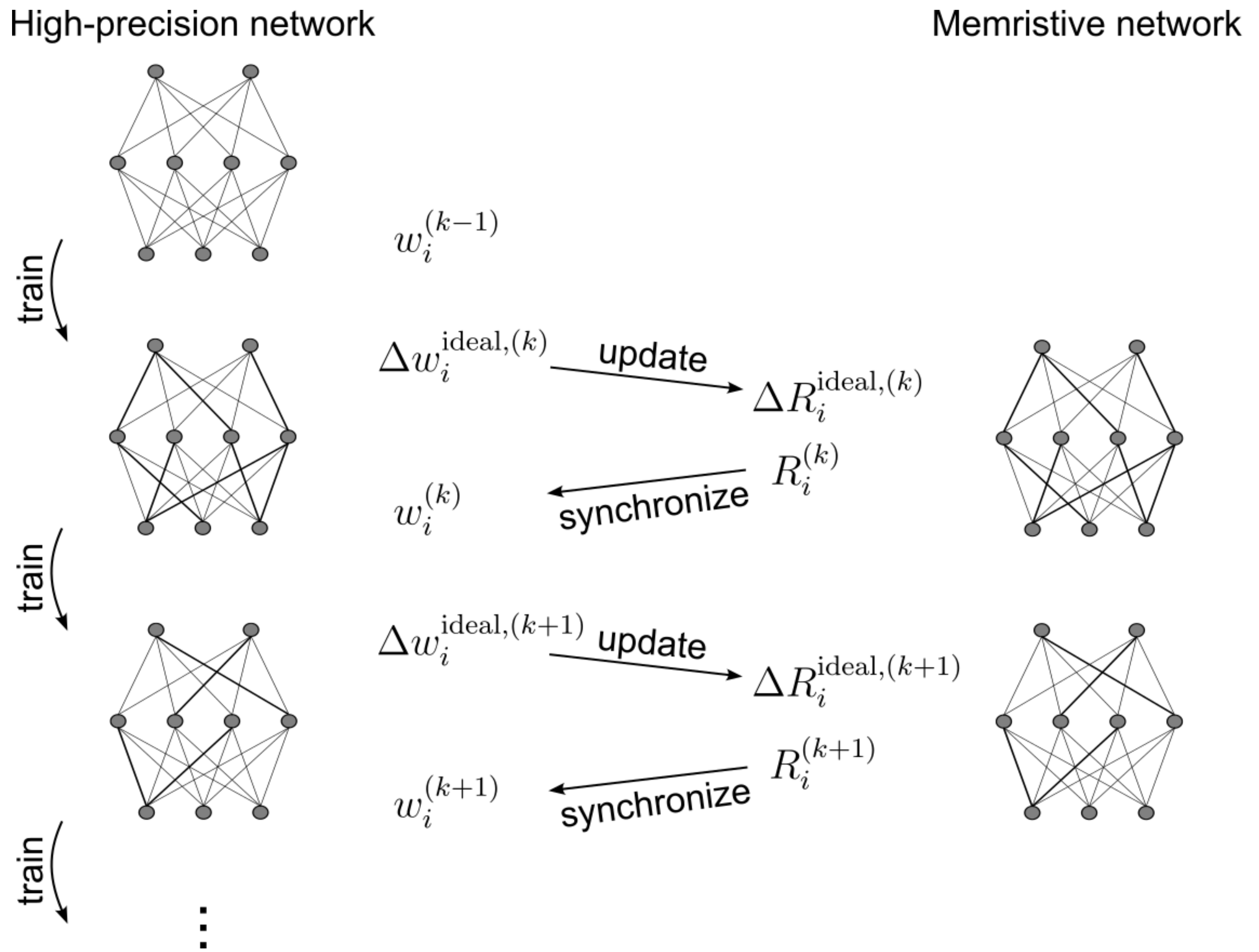
- Inverse mapping from weight to resistance:

$$R_i = \frac{1}{\frac{1}{R_C} + \frac{w_i}{\alpha}}$$

- Weight and resistance updates:

$$\Delta w_i = w_i^{(k)} - w_i^{(k-1)}$$
$$\Delta R_i = R_i^{(k)} - R_i^{(k-1)}$$

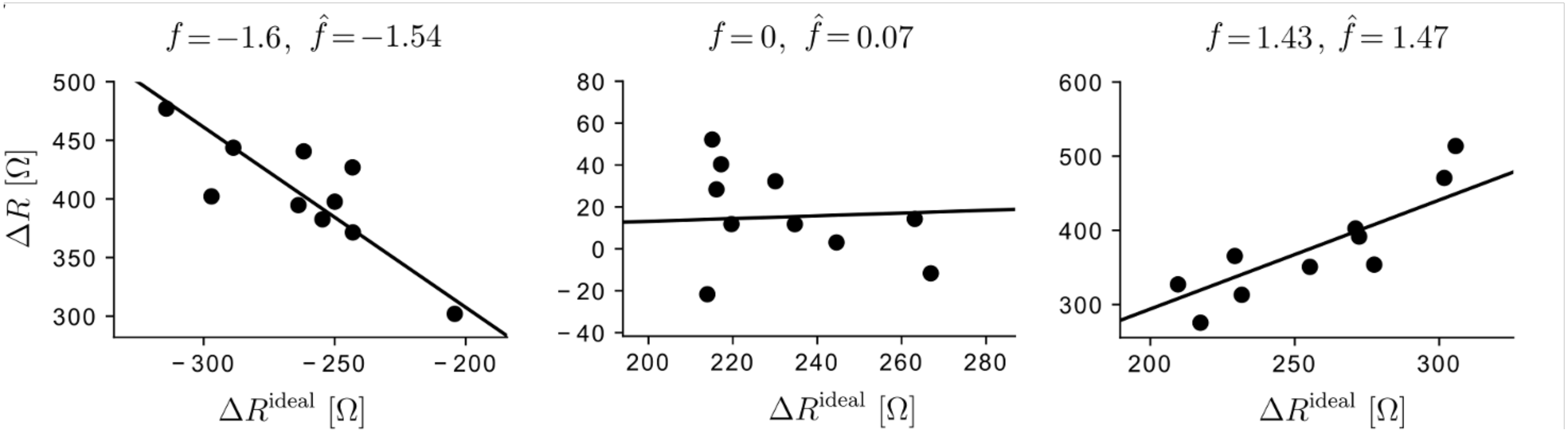
In-the-loop training



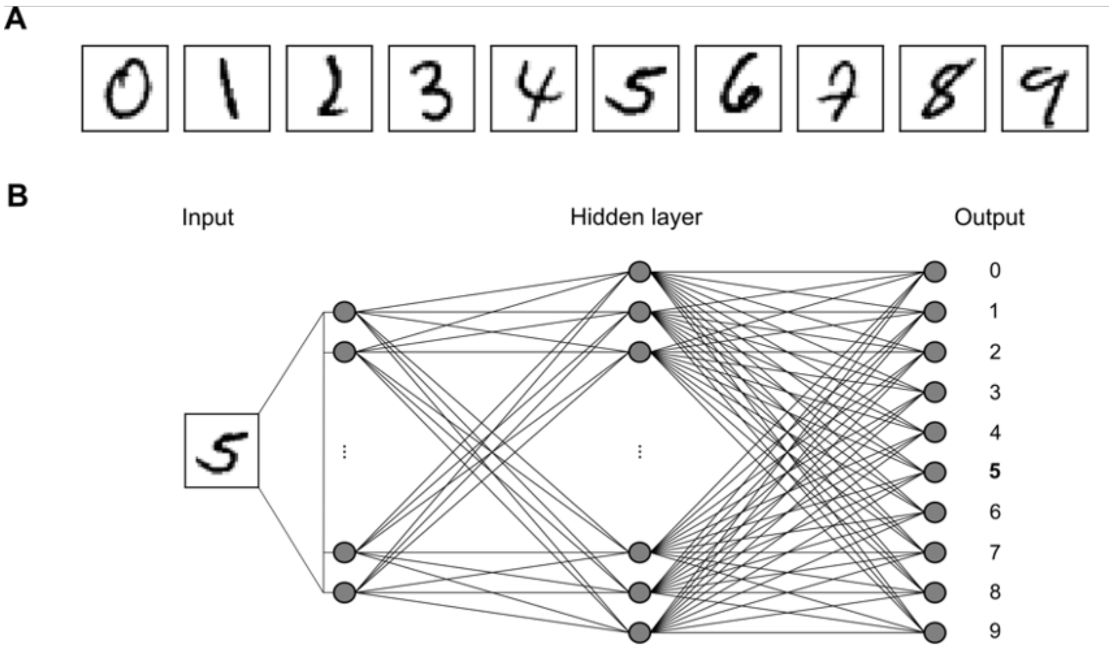
Ceca Kraišniković
Graz University of Technology, Austria

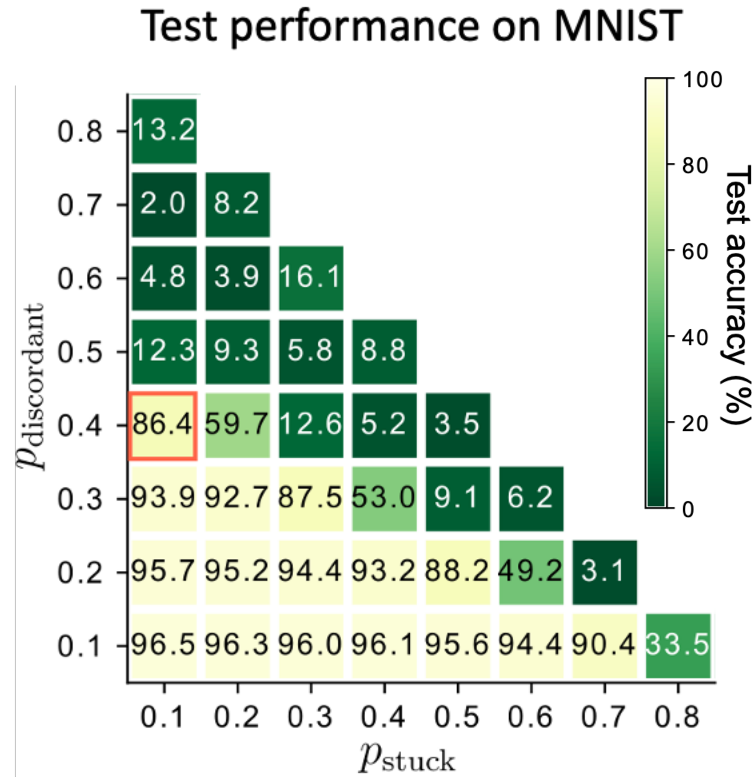
Model of imperfect memristor

- Memristor faults modeled by fault factor f_i
  - Modulates memristance change: $\Delta R_i^{(k)} = f_i \cdot \Delta R_i^{\text{ideal}, (k)} + \eta_i^{(k)}$
  - Stuck memristors: $f_i = 0$
  - Concordant changes: $f_i > 0$
  - Discordant changes: $f_i < 0$
  - Switching and readout noise $\eta_i^{(k)}$ added.



The MNIST task





Discordant memristive changes are detrimental.

Neural networks can be pruned significantly and achieve little loss in accuracy, hence we asked if one can prune faulty memristive connections.

Ceca Kraišniković
Graz University of Technology, Austria

Fault pruning algorithm

- Estimate fault factor over a window of previous updates:

$$\hat{f}_i = \frac{\sum_l \Delta R_i^{\text{ideal},(l)} \Delta R_i^{(l)}}{\sum_l \left(\Delta R_i^{\text{ideal},(l)}\right)^2}$$

Fault pruning algorithm

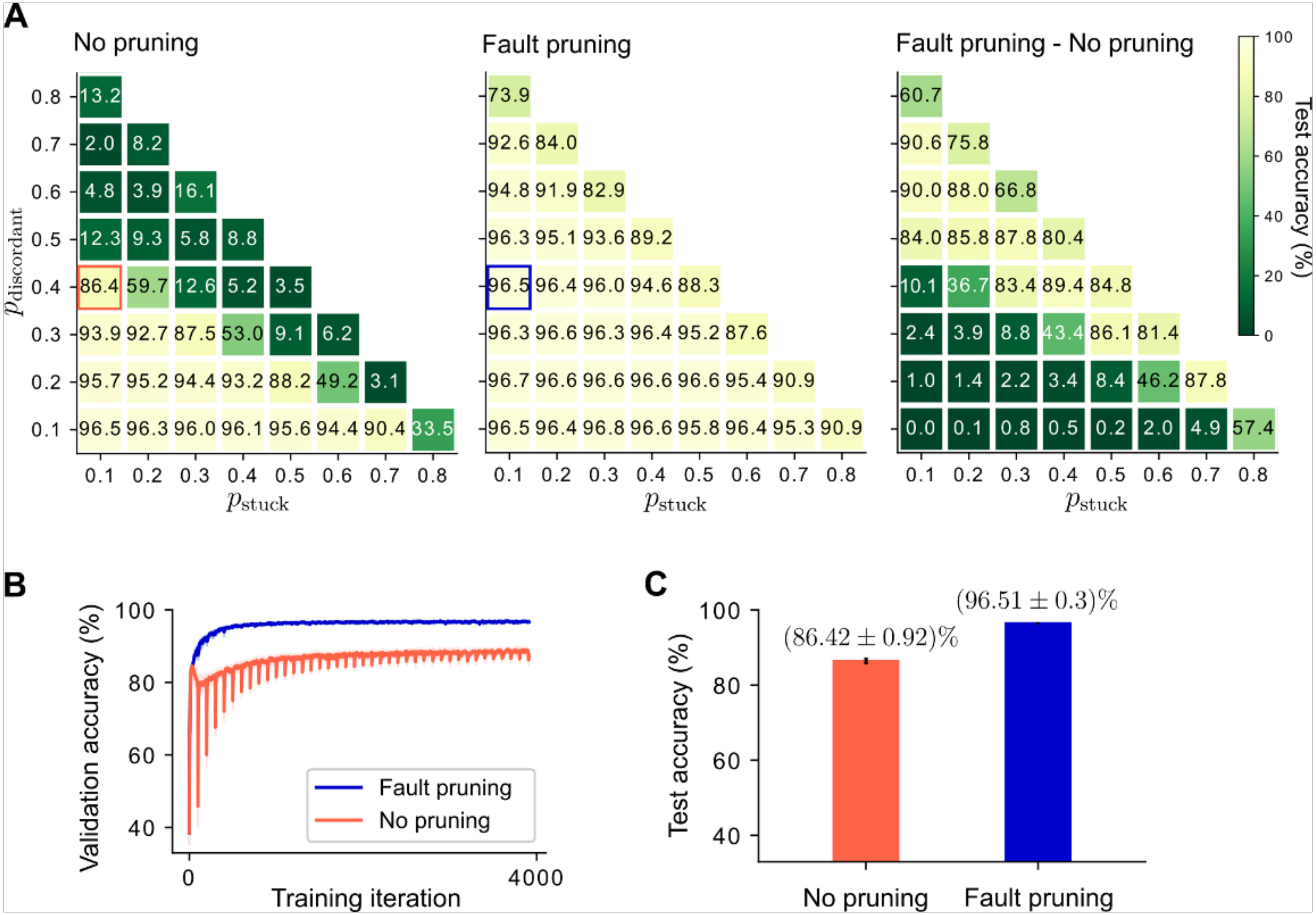
- Estimate fault factor over a window of previous updates:

$$\hat{f}_i = \frac{\sum_l \Delta R_i^{\mathrm{ideal},(l)} \Delta R_i^{(l)}}{\sum_l \left(\Delta R_i^{\mathrm{ideal},(l)}\right)^2}$$

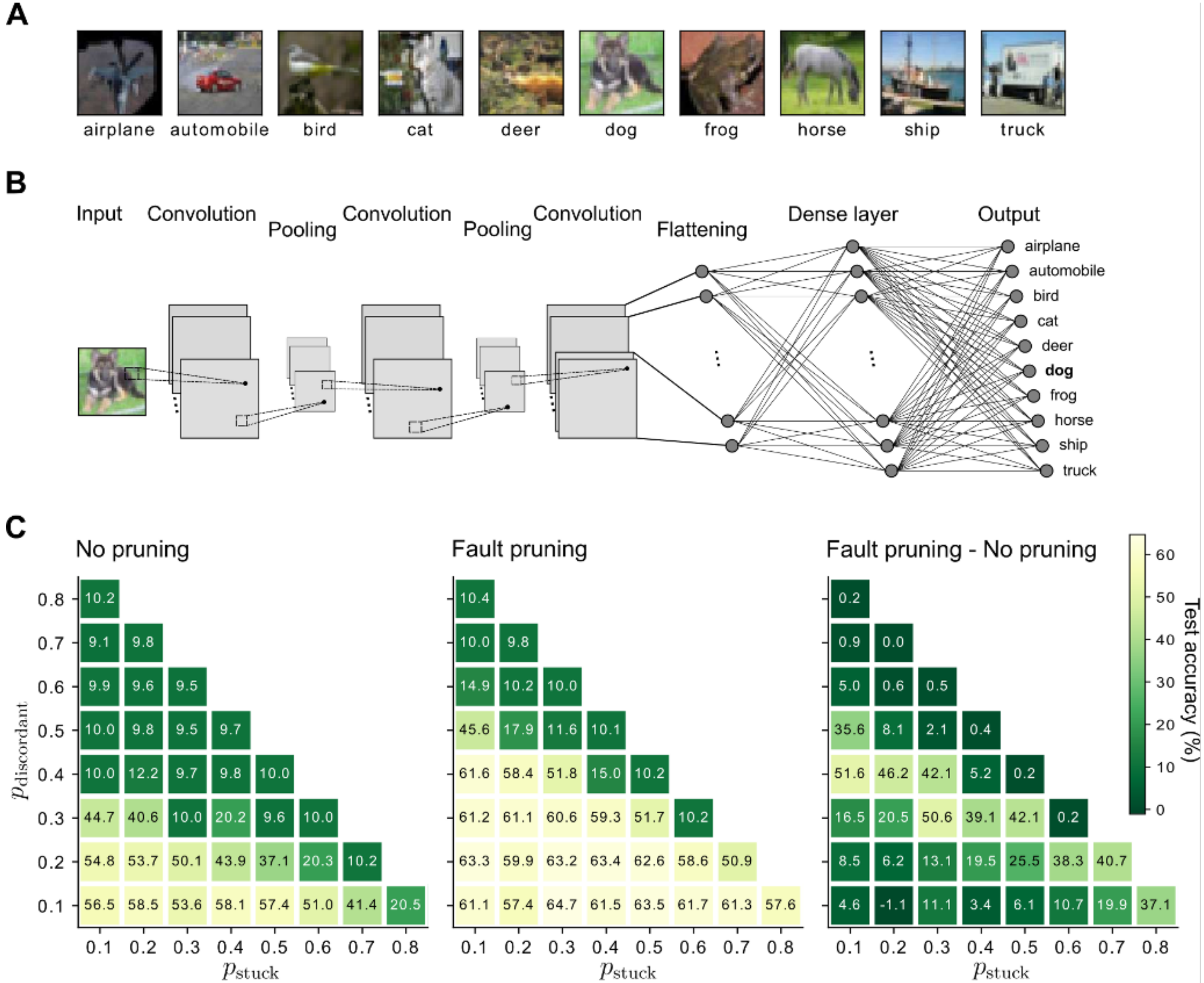
- Remove detected unreliable memristors from the network if $\hat{f}_i < \theta$, and we set $\theta = 0.1$

- Two variants of the algorithm
  - Variant 1: Prune faulty weights (set to zero)
  - Variant 2: Don't update faulty weights (keep last weight)

Results on MNIST

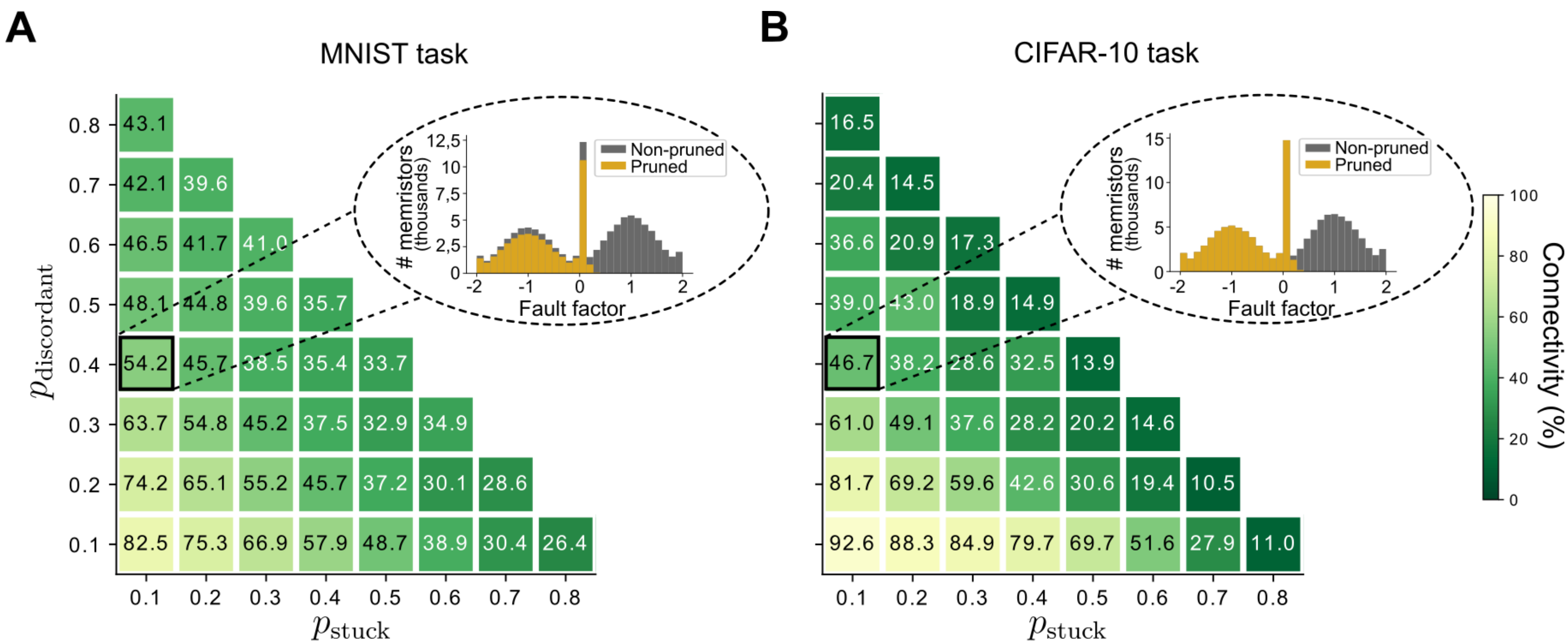


Results on CIFAR-10



Ceca Kraišniković
Graz University of Technology, Austria

Connectivity in the network after pruning



Summary and Conclusion

- Fault pruning managed to preserve very good performance
- Estimation of faults on the fly, and acting accordingly
- General approach, independent of the network structure and trained tasks
- A simple linear regression to estimate faults
  - Can be substituted by more advanced approaches
- **Future work:**
  - Test the algorithm in a real-world scenario
  - Handling memristors with discordant faults - by adapting the requested update

Questions?

Estimation of the fault factors $\hat{f}_i$

$$\Delta R_i = \hat{f}_i \cdot \Delta R_i^{\text{ideal}} + \epsilon$$

Estimated from $N = 10$ data points $(\Delta R_i^{\text{ideal},(l)}, \Delta R_i^{(l)})$, $l \in \{k-N+1, k-N+2, \ldots, k-1, k\}$

The least-squares estimator of $\hat{f}_i$ minimises the error

$$\mathscr{L}(\hat{f}_i) := \sum_l \left(\Delta R_i^{(l)} - \hat{f}_i \cdot \Delta R_i^{\text{ideal},(l)}\right)^2,$$

$$\frac{\partial \mathscr{L}}{\partial \hat{f}_i} = 2 \sum_l \left(\Delta R_i^{(l)} - \hat{f}_i \cdot \Delta R_i^{\text{ideal},(l)}\right)\left(-\Delta R_i^{\text{ideal},(l)}\right) \stackrel{!}{=} 0$$

$$\hat{f}_i \sum_l \left(\Delta R_i^{\text{ideal},(l)}\right)^2 = \sum_l \Delta R_i^{(l)} \Delta R_i^{\text{ideal},(l)}$$

$$\hat{f}_i = \frac{\sum_l \Delta R_i^{(l)} \Delta R_i^{\text{ideal},(l)}}{\sum_l \left(\Delta R_i^{\text{ideal},(l)}\right)^2}$$