Learn the Time to Learn: Replay Scheduling for Continual Learning

Marcus Klasson[1], Hedvig Kjellström[1], and Cheng Zhang[2]

[1]KTH Royal Institute of Technology, Stockholm, Sweden, [2]Microsoft Research, Cambridge, UK

Motivation & Summary

- **Companies in the real-world keep their historical data.**
- Current CL approaches ignore the time to learn old tasks again.
- Education methods for humans often focus on scheduling of rehearsal to enhance memory retention, e.g., spaced repetition [1].
- The time when replay is applied can influence the final performance significantly (see Fig. 1).

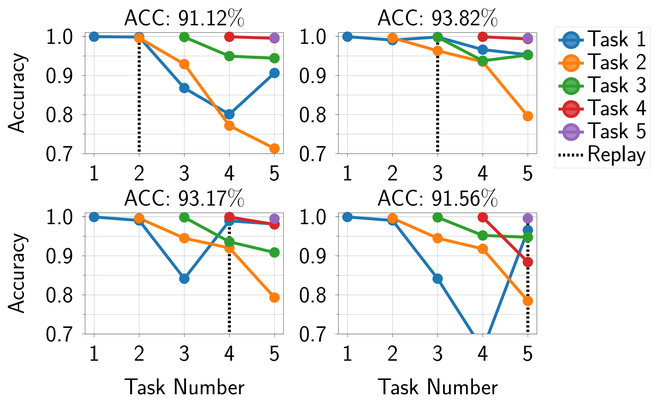

Figure 1: Task accs. on Split MNIST when replaying 10 examples of $0/1$ at one time step (black vertical line). ACC = avg. acc. over all tasks after learning Task 5.

- We introduce **Replay Scheduling** in CL as learning which tasks to replay at different time steps.
- We apply **Monte Carlo tree search** (MCTS) [2] for learning replay schedules by searching over a finite set of memory compositions.
- We demonstrate that learned **scheduling can improve the CL performance** significantly with fixed size memory on four datasets.

Problem Setting

We assume that **historical data is accessible** but only a limited replay memory can be used when the CL system adapts to novel data.

Goal: **Schedule which tasks to replay at different times** by learning task proportions over the number of memory examples per task to use.

Training Flow:

1: Initialize network $\boldsymbol{\theta}_0$
2: $\boldsymbol{\theta}_1 \leftarrow$ Network $\boldsymbol{\theta}_0$ learns dataset $\mathcal{D}_1$
3: Set historical data $\mathcal{H}_2 = \{\mathcal{D}_1\}$
4: **for** $t = 2 : T$ **do**
5: Sample $\mathcal{M}_t \overset{M}{\sim} \mathcal{H}_t$ using selected task proportions $(a_1, ..., a_{t-1})$
6: $\boldsymbol{\theta}_{t+1} \leftarrow$ Network $\boldsymbol{\theta}_t$ learns dataset $\mathcal{D}_t$ and replays $\mathcal{M}_t$
7: Update historical data $\mathcal{H}_{t+1} = [\mathcal{H}_t; \mathcal{D}_t]$

We **construct a discrete set of task proportions** $(a_1, ..., a_{t-1})$ to enable searches for replay schedules. See example in Fig. 2.

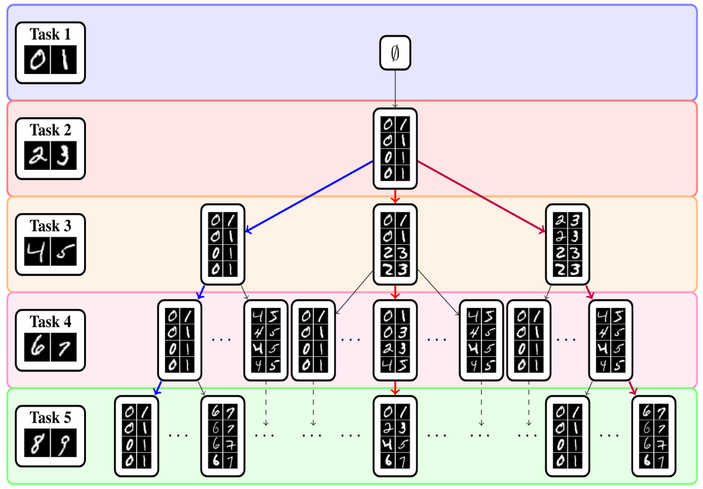

Figure 2: An exemplar tree of memory compositions $\mathcal{M}_t$ from our proposed discretization method for Split MNIST. The memory compositions are structured according to the task where they can be replayed.

MCTS for Replay Scheduling

Essential Steps for MCTS Simulation:

1. Select $\mathcal{M}_t$ using UCT function when fully-expanded task level t.
2. Select $\mathcal{M}_t$ randomly when task level t has unvisited memory compositions until reaching task T.
3. Train network with every selected $\mathcal{M}_t$ along path and backpropagate reward for simulation of replay schedule.

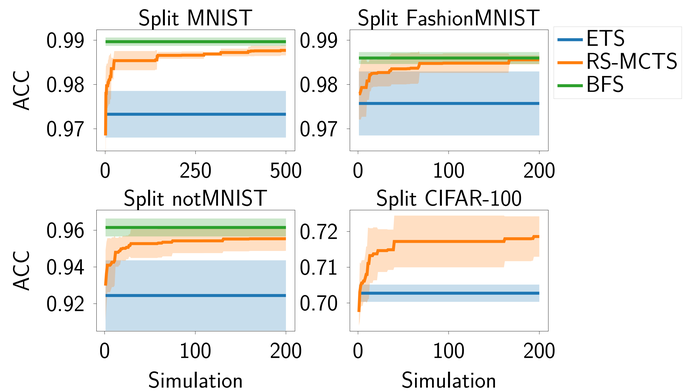

Figure 3: Schedules found with RS-MCTS improves significantly over ETS after 100 simulations. M=24 for first three datasets and M=285 for Split CIFAR-100.

Experimental Settings

- Replay Scheduling MCTS (RS-MCTS) is our method.
- Equal Task Schedule (ETS) involves using an equally distributed amount of memory examples per task for replay.
- $\mathcal{H}_t$ contains M examples from the previous tasks, such that $|\mathcal{H}_t| = M \cdot (t-1)$ before learning task t.

Experimental Results

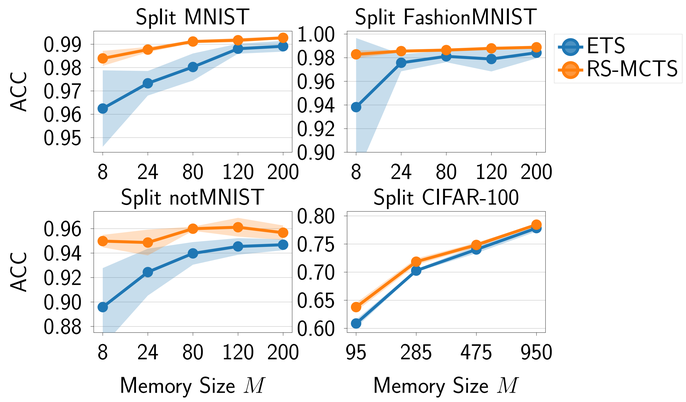

Figure 4: RS-MCTS performs significantly better than ETS for small M.

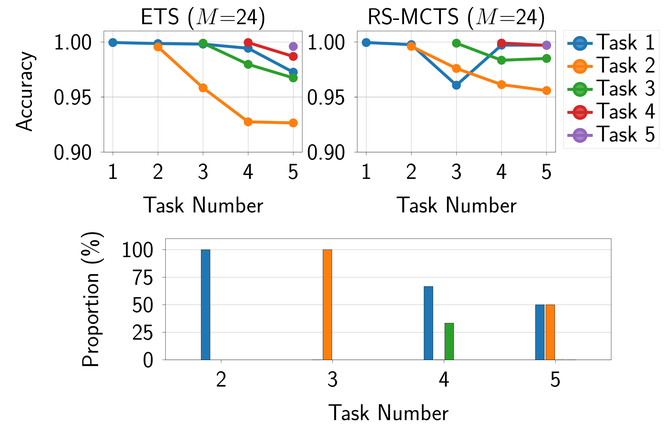

Figure 5: Comparison of task accuracies on Split MNIST for ETS and RS-MCTS (top) and task proportions at each task from RS-MCTS (bottom). The RS-MCTS schedule manages to retain its performance on Task 2 better than ETS.

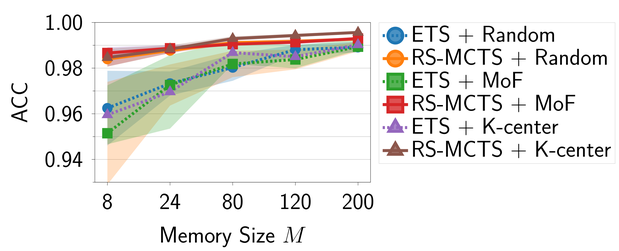

Figure 6: RS-MCTS improves over ETS on Split MNIST for different memory selection methods, such as Random Selection (Random), Mean-of-Features (MoF), and K-center Coreset (K-center).

References

[1] F. N. Dempster, "Spacing effects and their implications for theory and practice," *Educational Psychology Review*, vol. 1, no. 4, pp. 309–330, 1989.

[2] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*, pp. 72–83, Springer, 2006.