Learn the Time to Learn: Replay Scheduling for Continual Learning

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

Abstract

Memory-based continual learning has shown to be successful in mitigating catastrophic forgetting. However, in many real-world applications, the memory is limited due to constraints on processing time rather than storage capacity. Inspired by human learning, we demonstrate that scheduling over which previous tasks to revisit is critical to the final performance with limited resources. To this end, we propose to learn the time to learn for a continual learning system, in which we learn schedules of what data to rehearse at different times using Monte Carlo tree search. We empirically show on multiple datasets that our method can learn replay schedules that significantly improve over standard replay strategies given the same memory budget.

1. Introduction

Machine learning systems in real-world applications are often required to handle streaming data; for example, an industrial sorting system may regularly receive examples of new object classes to be classified in addition to the old ones. In this setting, the system is required to continuously update and adapt to the new tasks while retaining the previously learned abilities. Continual learning methods (Delange et al., 2021; McCloskey & Cohen, 1989; Parisi et al., 2019) address this challenge. In particular, many memory-based continual learning methods have shown to be very effective and achieve a great prediction performance and storage balance.

Current continual learning methods assume that there is no access to historical data apart from a memory with finite size due to storage constraints. However, in the real-world, many companies provide secure solutions for cloud data storage making access to historical data feasible. Nevertheless, as data accumulates, it is impossible and not necessary to re-train the model from all historical data. Thus, a limited memory is still a constraint due to processing time consideration even when historical data are accessible.

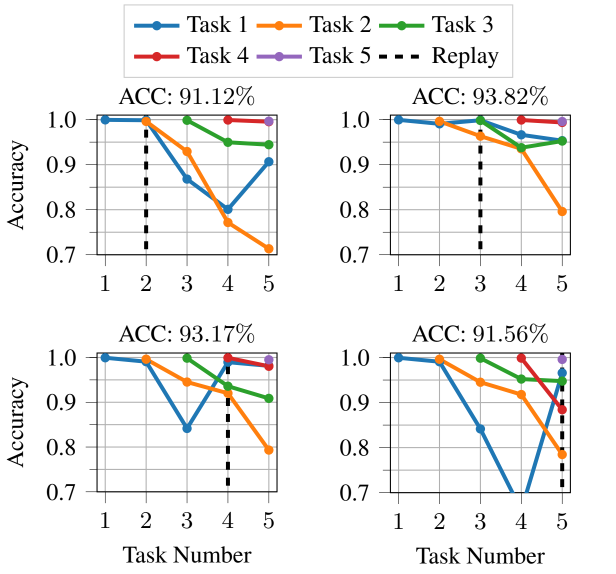

Figure 1. Task accuracies on Split MNIST (Zenke et al., 2017) when replaying only 10 examples of classes $0/1$ at a single time step. The black vertical line indicates when replay is used. ACC denotes the average accuracy over all tasks after learning Task 5. Results are averaged over 5 seeds. These results show that the time to replay the previous task is critical for the final performance.

Most research on memory-based continual learning commonly assumes a simple rule to sample the memory, such as an equal amount of samples from past tasks, and focus on the sample quality (Aljundi et al., 2019; Chaudhry et al., 2019; Nguyen et al., 2017; Rebuffi et al., 2017) or compression quality (Hayes et al., 2020; Pellegrini et al., 2019). However, these methods lack efficiency and ignore the time to learn past tasks again which is important in human learning. Humans are continual learning systems, and different methods have been developed to enhance memory retention, such as spaced repetition (Dempster, 1989; Ebbinghaus, 2013; Landauer & Bjork, 1977) which is often used in education. These education methods focus on the scheduling of learning and rehearsal of the already learned knowledge.

In this work, we argue that finding the proper schedule of what tasks to rehearse in the fixed memory situation

[1]Robotics, Perception, and Learning, KTH Royal Institute of Technology, Stockholm, Sweden [2]Microsoft Research, Cambridge, UK. Correspondence to: Marcus Klasson <mklas@kth.se>, Cheng Zhang <Cheng.Zhang@microsoft.com>.

Workshop on Theory and Foundation of Continual Learning at ICML 2021. Copyright 2021 by the author(s).

is critical for continual learning as well. To demonstrate our claim, we perform a simple experiment on the Split MNIST (Zenke et al., 2017) dataset where the memory only contains data from the first task. Furthermore, the memory can only be replayed, i.e. rehearsed, at one point in time and we show the performance on each task when the memory is replayed at different time steps. As shown in Figure 1, choosing different time points to replay the same memory leads to noticeably different results in the final performance, and in this example, the best final performance is achieved when the memory is used when learning Task 3.

To this end, inspired by human learning, we propose learning the time to learn, in which we learn schedules of which tasks to replay at different times. We illustrate the advantages with replay scheduling using Monte Carlo tree search (MCTS) (Coulom, 2006). More specifically, we train a neural network on the current task dataset mixed with the scheduled replay examples and measure the final performance of the network to evaluate the schedules selected by MCTS. In summary, our contributions are:

- We demonstrate the importance of replay scheduling in continual learning and propose to learn the time to learn which tasks to replay.
- We propose to use MCTS to learn replay schedules by establishing a finite set of memory compositions that can be replayed at every task.
- We demonstrate with four datasets that learned scheduling can improve the continual learning performance significantly with fixed size memory.

2. Method

2.1. Problem Setting

We describe our problem setting for learning replay schedules in scenarios where subsets of historical data can be used to overcome catastrophic forgetting. We assume that historical data are accessible to the system for replaying previous tasks at any time which is consistent with several real-world scenarios. However, the number of examples allowed for replay is limited due to processing time constraints. Our goal is therefore to select subsets of historical data for replay every time the system must adapt to novel data. We focus on composing the memories by selecting the replay samples based on tasks rather than instances in the historical data.

The notation of our problem setting resembles the traditional continual learning setting. We let a neural network $\boldsymbol{\theta}$ learn T classification tasks sequentially in some order. The dataset of the t-th task is denoted as $\mathcal{D}_t = \{(\boldsymbol{x}_t^{(i)}, y_t^{(i)})\}_{i=1}^{N_t}$ where $\boldsymbol{x}_t^{(i)}$ and $y_t^{(i)}$ are the i-th data point and class label respectively among N_t examples. The network after learning task t is denoted $\boldsymbol{\theta}_t$.

We assume that historical data $\mathcal{H}_t$ is accessible at any time t, where $\mathcal{H}_t = \bigcup_{i=1}^{t-1} \mathcal{D}_i$ and consists of H_t examples in total from the past datasets. The network has an external memory $\mathcal{M}_t$ that can be filled with historical data, such that $\mathcal{M}_t \subset \mathcal{H}_t$, and mixed with the current dataset to avoid catastrophic forgetting. At task t, we sample M historical examples to store in the memory for replay, i.e., $\mathcal{M}_t \overset{M}{\sim} \mathcal{H}_t$. The M examples come from a composition of old tasks and the size is extremely small comparing to the history, such that $M \ll H_t$. Before sampling from $\mathcal{H}_t$, we need to decide on the proportion $(a_1, \dots, a_{t-1})$ of examples to sample from each task, where $\sum_{i=1}^{t-1} a_i = 1$ and $a_i \geq 0$ is the proportion of M examples that comes from task i. The challenge is how to decide these proportions from each historical task to replay. To make this selection tractable, we construct discrete choices of possible proportions of examples that can be added to $\mathcal{M}_t$ at every task.

2.2. Replay Scheduling for Continual Learning

We define a replay schedule as a selection of $T-1$ memory compositions that have been used for replaying past data when learning T tasks. At task t, we sample a subset of M memory examples from the available historical data to store in a memory, i.e., $\mathcal{M}_t \overset{M}{\sim} \mathcal{H}_t$, which is used for replay. To this end, we construct a discrete number of choices for the proportions of data from old tasks to reduce the action space while keeping different types of memory choices available. Particularly, we include the option for an equal proportion from all previous tasks in the memory since this is a common choice in memory-based continual learning.

We construct the action space of proportions in the following way: At task t, we create $t-1$ bins $[b_1, b_2, ...b_{t-1}]$ and choose a historical task to sample for each bin $b_i \in 1, .., t-1$. We treat the bins as exchangeable and only keep the unique choices. For example, at Task 3, we have Task 1 and 2 in history; so the unique choices of bins are $[1,1], [1,2], [2,2]$, where $[1,1]$ indicates that all memory samples are from Task 1, $[1,2]$ indicates that half memory is from Task 1 and the other half is from Task 2 etc. If the memory size is not divisible by the number of bins, we round up and down accordingly while keeping the same memory size. From this specification, we can build a tree of different replay schedules to evaluate with the network.

Figure 2 shows an example of such replay schedule tree with Split MNIST (Zenke et al., 2017) where the memory size has been set to $M = 8$. Each row corresponds to the task of learning the two classes in the left box. On the right of the task boxes, we show examples of possible memory compositions that can be used for replay. At Task 1, the memory $\mathcal{M}_1 = \emptyset$ is empty, but is filled with M Task 1 examples at Task 2 by sampling from the historical data, as this is the only task in $\mathcal{H}$ at this moment. At Task 3, we

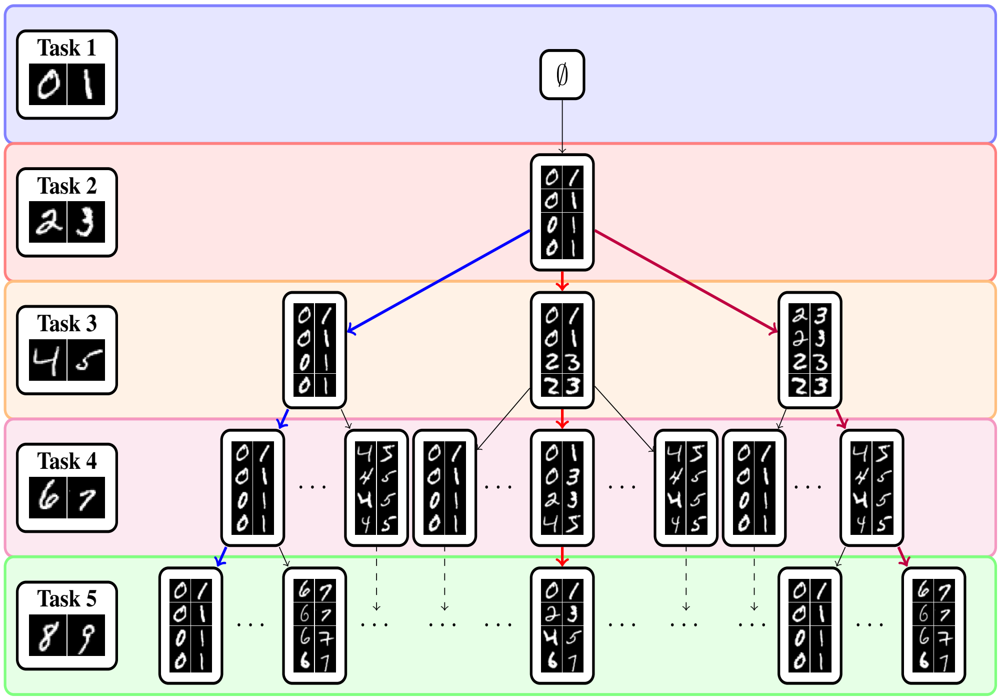

Figure 2. An exemplar tree of memory compositions from our proposed discretization method described in Section 2.2 for Split MNIST. A replay schedule is represented by a selection of $T-1$ memory compositions found by traversing from Task 1 to Task 5 through the tree on the right hand side. The memory composition choices have been structured according to the task where they can be used for replay. Note that the memory composition is the empty set, i.e., $\mathcal{M}_1 = \emptyset$, at Task 1. Example images for each task are shown on the left.

can choose three compositions; fill $\mathcal{M}_3$ with examples from either Task 1 or 2, or equally filling $\mathcal{M}_3$ with examples from both tasks. A replay schedule is represented as a path of memory compositions from Task 1 to Task 5. We have color-coded three examples of possible schedules in Figure 2 to use for illustration: the blue path represents a replay schedule where only Task 1 examples are replayed.The red path represents using an equally distributed amount of examples per task in the memory, and the purple path represents a schedule where the memory is filled with examples from the most recently visited task. Note that all other possible paths are valid replay schedules.

2.3. Monte Carlo Tree Search for Replay Schedules

Our proposed setup to make the replay scheduling problem tractable was to discretize the number of possible schedules per task. This turned replay scheduling into a tree search problem where each node represents a memory composition of examples from past tasks, as can be seen in Figure 2. Since the tree grows fast with the number of tasks, we need a scalable method that enables tree searches in large search spaces. We thus propose using Monte Carlo tree search (MCTS) (Browne et al., 2012; Coulom, 2006) which has been successful in searches in large state spaces (Chaudhry & Lee, 2018; Gelly et al., 2006; Silver et al., 2016). In our setting, MCTS can help to concentrate the search in paths of promising replay schedules that brings high classification performance for the network and thus learn the time to learn.

Following one path in the search tree consists of selecting a possible memory composition and training the network on the current task and replaying the selected memory. The tree levels are structured in the sequential task order $t = (1, \dots, T)$ where the corresponding task dataset $\mathcal{D}_t$ is available only at its corresponding level. Tree level t contains a set of nodes $\{v_t^i\}_{i=1}^{K_t}$ where K_t is the number of nodes. Every node v_t^i is used to retrieve the memory composition at the current node. The memory composition at node v_t^i is denoted as $\mathcal{M}(v_t^i)$ which gives memory composition i to replay at the task t. Referring to Figure 2, the memory composition on the root node v_1^1 is the empty set $\mathcal{M}(v_1^1) = \emptyset$ and, at Task 2, the memory composition $\mathcal{M}(v_2^1)$ corresponds to a memory with examples from Task 1 only, etc. In the rest of the paper, a node at task t is denoted as v_t by ignoring the superscript i to avoid cluttered notation.

Next, we briefly outline the MCTS procedure for finding replay schedules. The historical data $\mathcal{H}$ is initialized with at least M examples from every dataset $\mathcal{D}_{1:T}$ before starting MCTS. An MCTS simulation consists of moving to subsequent nodes and, at each node, using the corresponding memory composition for replay when the network learns the current task. The next node is selected by evaluating the Upper Confidence Tree (UCT) function (Kocsis & Szepesvári, 2006) if the tree level is fully-expanded, i.e., all nodes have been visited on the specific level. Otherwise, the tree is

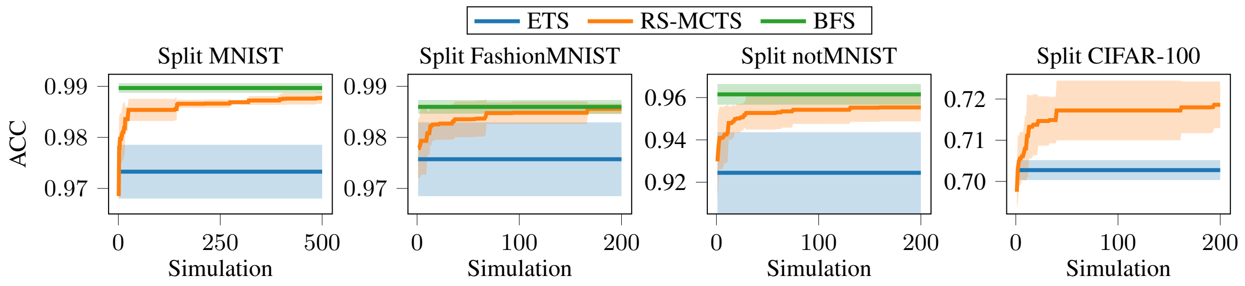

Figure 3. MCTS simulation performance on Split MNIST, Split FashionMNIST, Split notMNIST, and Split CIFAR-100, where the average test classification accuracies over tasks after training on the final task (ACC) is used as the reward. The ETS performance is shown in blue as a baseline. For the first three datasets, the green lines show the ACC from the optimal schedules found from a breadth-first search (BFS). We use $M = 24$ for the first three datasets and $M = 285$ for Split CIFAR-100. All results have been averaged over 5 seeds.

expanded by randomly selecting the next node among the unvisited ones. After an expansion step, the search proceeds by traversing randomly through the tree until reaching a terminal node v_T. When the network has been trained on the final task T, we backpropagate the reward for the simulation which is the average test classification accuracy of all tasks after learning task T. The full details of the algorithm are provided in Appendix B. We emphasize that MCTS is used to showcase in a simplistic setting the importance of learning the time to learn from memory examples.

3. Experiments

We evaluated our replay scheduling method on four common benchmark datasets for continual learning, namely Split MNIST (Zenke et al., 2017), Split Fashion-MNIST (Xiao et al., 2017), Split notMNIST (Bulatov, 2011), and Split CIFAR-100 (Lopez-Paz & Ranzato, 2017). Our results show the importance of learning the time to learn based on improved classification performance compared to using replay schedules with equally distributed memory examples across the tasks. We denote results from replay schedules found using MCTS as RS-MCTS. In all experiments, the historical data $\mathcal{H}_t$ contains M examples from all previous tasks such that the historical data size is $H_t = M \cdot (t-1)$ before learning task t. The memory composition $\mathcal{M}_t$ consists of M examples drawn from $\mathcal{H}_t$ where the task proportions are given by the tree node selected by MCTS. We compare the performance of RS-MCTS with training a network using an equally distributed amount of memory examples per task in the memory, which we name Equal Task Schedule (ETS). See the red path in Figure 2 for an example of ETS.

We use the average test classification accuracy of all tasks after training on the final task (ACC) as evaluation metric. The experiments are mainly in the task incremental learning setting where task labels are available at test time. The experimental settings can be found in Appendix C. Full experimental analysis is provided in Appendix D, where we show that RS-MCTS have greater advantage when the memory size is small in Appendix D.1, and that learned replay scheduling is consistent with human learning processes in Appendix D.2. Furthermore, we show that RS-MCTS can be used with various sample selection strategies other than random selection in Appendix D.3, and also be applied to the class incremental learning setting in Appendix D.4.

3.1. Results

In the first experiments, we show the progress in classification performance from using the learned scheduling policies from RS-MCTS over the number of MCTS simulations in Figure 3. We have set the memory sizes to $M = 24$ for the first three datasets and $M = 285$ for Split CIFAR-100. These options makes the ETS method store at least three examples of every class when training on the final task. For the first three datasets, we provide the optimal replay schedule found from a breadth-first search (BFS) over all possible replay schedules as an upper bound for RS-MCTS. As the search space grows super exponentially with the number of tasks, it will even with only 5 continual learning tasks (which associate to a tree with depth of 4) yield a tree with 1050 leaf nodes. Thus, BFS becomes impracticable for the 20 tasks in Split CIFAR-100. We observe that RS-MCTS quickly finds a significantly better ACC than ETS for all datasets with commonly less than 50 iterations. Furthermore, RS-MCTS can reach similar performance as from BFS with significantly less budget than BFS. These results confirm that scheduling which tasks to replay is important for achieving the best possible final performance.

4. Conclusion

We propose learning the time to learn, i.e., learning schedules of what previous tasks to rehearse at different times. To the best of our knowledge, we are the first to consider the time to learn in a continual learning setting, which corresponds well with real-world needs. We demonstrate with an example method that learned replay schedules produce significantly improved results on retaining previously learned abilities under the same memory budget comparing with a method without scheduling.

Acknowledgements

Marcus Klasson is funded by the Promobilia foundation and the Swedish e-Science Research Centre.

References

Adel, T., Zhao, H., and Turner, R. E. Continual learning with adaptive weights (claw). *arXiv preprint arXiv:1911.09514*, 2019.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. Gradient based sample selection for online continual learning. *arXiv preprint arXiv:1903.08671*, 2019.

Ball, P. J., Li, Y., Lamb, A., and Zhang, C. A study on efficiency in continual learning inspired by human learning. *arXiv preprint arXiv:2010.15187*, 2020.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Bulatov, Y. The notMNIST dataset. http://yaroslavvb.com/upload/notMNIST/, 2011.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 532–547, 2018a.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018b.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Chaudhry, M. U. and Lee, J.-H. Feature selection for high dimensional data using monte carlo tree search. *IEEE Access*, 6:76036–76048, 2018.

Chrysakis, A. and Moens, M.-F. Online continual learning from imbalanced data. In *ICML*, 2020.

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

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Dempster, F. N. Spacing effects and their implications for theory and practice. *Educational Psychology Review*, 1 (4):309–330, 1989.

Douillard, A., Cord, M., Ollion, C., Robert, T., and Valle, E. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *Computer Vision – ECCV 2020*, pp. 86–102. Springer International Publishing, 2020.

Dunlosky, J., Rawson, K. A., Marsh, E. J., Nathan, M. J., and Willingham, D. T. Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest*, 14(1):4–58, 2013. ISSN 15291006. URL http://www.jstor.org/stable/23484712.

Ebbinghaus, H. Memory: A contribution to experimental psychology. *Annals of neurosciences*, 20(4):155, 2013.

Ebrahimi, S., Meier, F., Calandra, R., Darrell, T., and Rohrbach, M. Adversarial continual learning. *arXiv preprint arXiv:2003.09553*, 2020.

Feng, K., Zhao, X., Liu, J., Cai, Y., Ye, Z., Chen, C., and Xue, G. Spaced learning enhances episodic memory by increasing neural pattern similarity across repetitions. *Journal of Neuroscience*, 39(27):5351–5360, 2019.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. ISSN 1364-6613. doi: https://doi.org/10.1016/S1364-6613(99)01294-2. URL https://www.sciencedirect.com/science/article/pii/S1364661399012942.

Gelly, S., Wang, Y., Munos, R., and Teytaud, O. Modification of uct with patterns in monte-carlo go. Technical Report RR-6062, INRIA, 2006. inria-00117266v3f.

Gonzalez, T. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

Hawley, K. S., Cherry, K. E., Boudreaux, E. O., and Jackson, E. M. A comparison of adjusted spaced retrieval versus a uniform expanded retrieval schedule for learning a name–face association in older adults with probable alzheimer's disease. *Journal of Clinical and Experimental Neuropsychology*, 30(6):639–649, 2008.

Hayes, T. L., Cahill, N. D., and Kanan, C. Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9769–9776. IEEE, 2019.

Hayes, T. L., Kafle, K., Shrestha, R., Acharya, M., and Kanan, C. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pp. 466–483. Springer, 2020.

Hou, S., Pan, X., Loy, C. C., Wang, Z., and Lin, D. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 831–839, 2019.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Iscen, A., Zhang, J., Lazebnik, S., and Schmid, C. Memory-efficient incremental learning through feature adaptation. In *European Conference on Computer Vision*, pp. 699–715. Springer, 2020.

Isele, D. and Cosgun, A. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Joseph, K. and Balasubramanian, V. N. Meta-consolidation for continual learning. *arXiv preprint arXiv:2010.00352*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Landauer, T. and Bjork, R. Optimum rehearsal patterns and name learning. *Practical aspects of memory*, 1, 11 1977.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. *arXiv preprint arXiv:1706.08840*, 2017.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Mirzadeh, S. I., Farajtabar, M., Gorur, D., Pascanu, R., and Ghasemzadeh, H. Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*, 2020.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

Pan, P., Swaroop, S., Immer, A., Eschenhagen, R., Turner, R. E., and Khan, M. E. Continual deep learning by functional regularisation of memorable past. *arXiv preprint arXiv:2004.14070*, 2020.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Pellegrini, L., Graffieti, G., Lomonaco, V., and Maltoni, D. Latent replay for real-time continual learning. *arXiv preprint arXiv:1912.01100*, 2019.

Rannen, A., Aljundi, R., Blaschko, M. B., and Tuytelaars, T. Encoder based lifelong learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1320–1328, 2017.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. Experience replay for continual learning. *arXiv preprint arXiv:1811.11682*, 2018.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pp. 4528–4537. PMLR, 2018.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Smolen, P., Zhang, Y., and Byrne, J. H. The right time to learn: mechanisms and optimization of spaced learning. *Nature Reviews Neuroscience*, 17(2):77, 2016.

van de Ven, G. M. and Tolias, A. S. Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*, 2018.

van de Ven, G. M., Siegelmann, H. T., and Tolias, A. S. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1):1–14, 2020.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080*, 2016.

Vitter, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1): 37–57, 1985.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

Willis, J. Review of research: Brain-based teaching strategies for improving students' memory, learning, and test-taking success. *Childhood Education*, 83(5):310–315, 2007.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

Yoon, J., Kim, S., Yang, E., and Hwang, S. J. Scalable and order-robust continual learning with additive parameter decomposition. *arXiv preprint arXiv:1902.09432*, 2019.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.

Appendix

This supplementary material provides details that were excluded from the paper submission due to space constraints. First, we provide a brief overview of the related works in Section A. We give more details on the steps for using Monte Carlo tree search (MCTS) to find replay schedules in Section B. In Section C, we describe the full details of the experimental settings and Section D gives a more thorough experimental analysis than in the main paper (Section 3.1). More specifically, we show that our method have greater advantage when the memory size is small (see Section D.1), and the learned memory scheduling is consistent with human learning process (see Section D.2). Furthermore, we provide experimental results on Split MNIST with alternative memory selection strategies such as Mean-of-Features (Rebuffi et al., 2017) and K-center Coreset (Nguyen et al., 2017) (see Section D.3). These experiments show that our method can work with any other memory-based continual learning method to further improve the performance (see Figure 11 and 12). Last, we show results from replay scheduling in the class incremental learning (Class-IL) scenario in Section D.4, which demonstrates that replay scheduling is necessary in the more challenging Class-IL scenario when memory budgets are limited (see Figure 13 and 14).

A. Related Work

We give an overview of continual learning methods, essentially memory-based methods, as well as spaced repetition techniques for human continual learning.

Continual Learning: Traditional continual learning can be divided into three main areas, namely regularization-based, architecture-based, and memory-based approaches. Regularization-based methods aim to mitigate catastrophic forgetting by protecting parameters influencing the predictive performance from wide changes and use the rest of the parameters for learning the new tasks (Adel et al., 2019; Chaudhry et al., 2018a; Kirkpatrick et al., 2017; Li & Hoiem, 2017; Nguyen et al., 2017; Rannen et al., 2017; Schwarz et al., 2018; Zenke et al., 2017). Architecture-based methods isolate task-specific parameters by either increasing network capacity (Rusu et al., 2016; Yoon et al., 2019; 2017) or freezing parts of the network (Mallya & Lazebnik, 2018; Serra et al., 2018) to maintain good performance on previous tasks. Memory-based methods store examples from previous tasks in an external memory (Chaudhry et al., 2019; Hayes et al., 2020; Isele & Cosgun, 2018; Lopez-Paz & Ranzato, 2017), or uses a generative model to generate pseudo-samples from a distribution over tasks (Shin et al., 2017; van de Ven & Tolias, 2018), that are mixed with the current task dataset to mitigate catastrophic forgetting. Regularization-based approaches and dynamic architectures have been combined with memory-based approaches to methods to overcome their limitations (Chaudhry et al., 2018a;b; Douillard et al., 2020; Ebrahimi et al., 2020; Joseph & Balasubramanian, 2020; Mirzadeh et al., 2020; Nguyen et al., 2017; Pan et al., 2020; Pellegrini et al., 2019; Rolnick et al., 2018; von Oswald et al., 2019). Our work relates most to memory-based methods with external memory which we spend more time on describing in the next paragraph.

Memory-based Continual Learning: The simplest selection strategy is random selection of examples to store in the memory for replay. Much research effort has focused on selecting higher quality samples to store in memory (Chaudhry et al., 2019; Chrysakis & Moens, 2020; Hayes et al., 2019; Isele & Cosgun, 2018; Lopez-Paz & Ranzato, 2017; Nguyen et al., 2017; Rebuffi et al., 2017). Chaudhry (Chaudhry et al., 2019) reviews several selection strategies, e.g., reservoir sampling (Vitter, 1985), first-in first-out buffer (Lopez-Paz & Ranzato, 2017), k-Means, and Mean-of-Features (Rebuffi et al., 2017), in scenarios with tiny memory capacity. However, more elaborate selection strategies have been shown to give little benefit over random selection for image classification problems (Chaudhry et al., 2018a; Hayes et al., 2020). More recently, there has been work on compressing raw images to feature representations to increase the number of memory examples for replay (Hayes et al., 2020; Iscen et al., 2020). Our approach differs from the above mentioned works since we focus on which memory examples to choose for training at the current task rather than which examples to store in the memory. Replay scheduling can however be combined with any selection strategy as well as storing feature representations.

Another important aspect of memory-based methods is how to adjust the memory size when storing old examples from the most recent task. There exist two popular approaches for setting the memory size that maintain an equal distribution of past examples in the memory, which is important when the memory size is small. In the first strategy, a constant number of samples per class M_{per} are stored, so that the total memory size grows with the number of classes (Douillard et al., 2020; Hou et al., 2019). The second strategy sets a fixed size capacity on the memory meaning that we are only allowed to store a total of M_{total} number of samples from all previously seen classes/tasks (Chaudhry et al., 2019; Lopez-Paz & Ranzato, 2017). For our task, we assume that we can easily access historical data and focus on which tasks to select for replay. In Section 2.2, we describe our memory setting for learning policies to select replay schedules.

Human Continual Learning: Humans are continual learning systems in the sense of learning tasks and concepts sequentially. Furthermore, humans have an impressive ability to memorize experiences but can forget learned knowledge gradually rather than catastrophically (French, 1999).

Different learning techniques have been suggested for humans to memorize better (Dunlosky et al., 2013; Willis, 2007). An example is spaced repetition which gradually increases time-intervals between rehearsals for retaining long-term memory (Dempster, 1989) and has been known to be superior to rehearsing many times in short time, i.e., massed learning or massed training, to retain memory since the studies by Ebbinghaus (Ebbinghaus, 2013). Landauer and Bjork (Landauer & Bjork, 1977) demonstrated that memory training schedules using adjusted spaced repetition were better at preserving memory than uniformly spaced training, while Hawley (Hawley et al., 2008) compares its efficacy on adults with probable Alzheimer's disease for learning face-name association. More recently, computational models and neural representational analysis have been used to bring insights on the underlying mechanisms enabling spaced repetition to improve long-term memory (Feng et al., 2019; Smolen et al., 2016). Several works in continual learning with neural networks are inspired by or have a connection to human learning techniques, including spaced repetition (Feng et al., 2019; Smolen et al., 2016), mechanisms of sleep (Ball et al., 2020; Schwarz et al., 2018), and reactivation of memories (Hayes et al., 2020; van de Ven et al., 2020). Our replay scheduling method is inspired by spaced repetition; we learn schedules of which memory examples to use for replay at different time steps.

B. Replay Scheduling Monte Carlo Tree Search Algorithm

In this section, we describe the steps in our Replay Scheduling MCTS method and pseudo-code in Algorithm 1.

B.1. Details on the MCTS Steps

We outline the steps for performing MCTS in the search for replay schedules:

Selection: An MCTS iteration begins by selecting a traversal path from the root node to an expanded or visited node. In our case, only the Task 1 dataset $\mathcal{D}_1$ is available in the root node v_1. Terminal nodes are denoted as v_T since then we are training at the final dataset $\mathcal{D}_T$ for the given continual learning experiment. For selecting the next node, we evaluate the Upper Confidence Tree (UCT) (Kocsis & Szepesvári, 2006) function for all previously visited child nodes v_{t+1} and select the node with the highest score. We use the UCT function proposed by Chaudhry & Lee (2018) which has the following form when evaluting the UCT score for moving from node v_t to its child v_{t+1}:

$$UCT(v_t, v_{t+1}) = \max(Q(v_{t+1})) + C\sqrt{\frac{2\log(N(v_t))}{N(v_{t+1})}}, \tag{1}$$

The reward function $Q(\cdot)$ includes the test classification accuracy that has been measured at the end of a whole replay schedule simulation when starting from the child v_{t+1}. The exploration constant $C \geq 0$ determines the degree of exploration of less visited replay schedules based on the number of visits $N(v_t)$ and $N(v_{t+1})$ to the corresponding nodes in the tree. The node with the highest UCT score is selected at each task level in the tree.

Expansion: Whenever the current node has unvisited child nodes, the next step is to expand the search tree with one of the unvisited child nodes v_{t+1}. The policy for selecting the child to visit is most commonly decided by uniform random sampling which is the policy we use as well. After expanding to child node v_{t+1}, we train the model on the memory composition $\mathcal{M}(v_{t+1})$ to obtain the model parameters $\boldsymbol{\theta}_{t+1}$ for this node. A node is considered to be fully-expanded when all its child nodes have been visited once.

Simulation: After expansion, the algorithm selects a sequence of moves starting from the current node v_t and ends in a terminal node v_T where the final classification performance of the network can be computed. The sequence of selected moves is chosen uniformly at random. At every selected node v_t, we train the network on the current dataset $\mathcal{D}_t$ and the current memory $\mathcal{M}(v_t)$ until the network has been trained at the terminal node v_T. We then calculate the reward the network has achieved by following the replay schedules in the current simulation.

Reward Calculation and Backpropagation: After training the network on a whole simulation of replay schedules, we calculate the reward achieved by the network for following these replay schedules. We use the average test classification accuracy of all tasks after learning the final task T as the reward from a simulation, which we refer to as ACC. The last step is to backpropagate the calculated reward from the node v_t where the simulation steps started up to the root node v_1. Every node visited in the backprop updates its statistics, i.e., the classification reward and number of visits, which will be used in the selection phase by the UCT function of the next MCTS iteration.

B.2. Algorithm

We provide pseudo-code in Algorithm 1 that outlines our method Replay Scheduling Monte Carlo tree search (RS-MCTS) described in the main paper (Section 2.2). The algorithm requires the task datasets $\mathcal{D}_{1:T}$, all nodes in the search tree $v_{1:T}$, and the memory size M. We assume that each node v contains the memory combinations of the path from the root node that is used for sampling the memory $\mathcal{M}_t$ from the historical data $\mathcal{H}_t$. Furthermore, each node v stores model parameters $\boldsymbol{\theta}$ that have been obtained from training the model using the replay memory combination at

Algorithm 1 Replay Scheduling Monte Carlo tree search

```
Require: Datasets $\mathcal{D}_{1:T}$, Tree Nodes $v_{1:T}$
Require: Memory Capacity $M$
 1: Initialize historical data $\mathcal{H}$ using $\mathcal{D}_{1:T}$
 2: Initialize model $\boldsymbol{\theta}_0$
 3: best_acc $\leftarrow 0$
 4: $\boldsymbol{\theta}_1 \leftarrow \text{TRAINMODEL}(\mathcal{D}_1, \boldsymbol{\theta}_0)$
 5: while within computational budget do
 6:     $v_t, \boldsymbol{\theta}_t \leftarrow \text{TREEPOLICY}(v_1, \boldsymbol{\theta}_1, \mathcal{D}_{1:T}, \mathcal{H}, M)$
 7:     $v_T, \boldsymbol{\theta}_T, \text{acc} \leftarrow \text{DEFAULTPOLICY}(v_t, \boldsymbol{\theta}_t, \mathcal{D}_{1:T}, \mathcal{H}, M)$
 8:     $\text{BACKPROPAGATE}(v_t, \text{acc})$
 9:     if acc > best_acc then
10:         $v_T^{best} \leftarrow v_T$
11:         $\boldsymbol{\theta}_T^{best} \leftarrow \boldsymbol{\theta}_T$
12:         best_acc $\leftarrow$ acc
13: return $v_T^{best}, \boldsymbol{\theta}_T^{best}$, best_acc

14: function TREEPOLICY($v_t, \boldsymbol{\theta}_t, \mathcal{D}_{1:T}, \mathcal{H}, M$)
15:     while $v_t$ is non-terminal do
16:         if $v_t$ not fully expanded then
17:             return $\text{EXPANSION}(v_t, \boldsymbol{\theta}_t, \mathcal{D}_{t+1}, \mathcal{H}, M)$
18:         else
19:             $v_t, \boldsymbol{\theta}_t \leftarrow \text{BESTCHILD}(v_t)$
20:     return $v_t, \boldsymbol{\theta}_t$

21: function EXPANSION($v_t, \boldsymbol{\theta}_t, \mathcal{D}_{t+1}, \mathcal{H}, M$)
22:     Sample $v_{t+1}$ uniformly among unvisited children of $v_t$
23:     $\mathcal{M} \overset{M}{\sim} \mathcal{H}$ using memory combination in $v_{t+1}$
24:     $\boldsymbol{\theta}_{t+1} \leftarrow \text{TRAINMODEL}(\mathcal{D}_{t+1} \cup \mathcal{M}, \boldsymbol{\theta}_t)$
25:     Add new child $v_{t+1}$ with model $\boldsymbol{\theta}_{t+1}$ to node $v_t$
26:     return $v_{t+1}, \boldsymbol{\theta}_{t+1}$

27: function BESTCHILD($v_t$)
28:     $v_{t+1} = \underset{v_{t+1} \in \text{ children of } v}{\arg\max} \max(Q(v_{t+1})) + C\sqrt{\frac{2\log(N(v_t))}{N(v_{t+1})}}$
29:     Get model $\boldsymbol{\theta}_{t+1}$ from node $v_{t+1}$
30:     return $v_{t+1}, \boldsymbol{\theta}_{t+1}$

31: function DEFAULTPOLICY($v_t, \boldsymbol{\theta}_t, \mathcal{D}_{1:T}, \mathcal{H}, M$)
32:     while $v_t$ is non-terminal do
33:         Sample $v_{t+1}$ uniformly among children of $v_t$
34:         $\mathcal{M} \overset{M}{\sim} \mathcal{H}$ using memory combination in $v_{t+1}$
35:         $\boldsymbol{\theta}_{t+1} \leftarrow \text{TRAINMODEL}(\mathcal{D}_{t+1} \cup \mathcal{M}, \boldsymbol{\theta}_t)$
36:         Update $v_t \leftarrow v_{t+1}$, $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t+1}$
37:     acc $\leftarrow \text{TESTMODEL}(\mathcal{D}_{1:T}, \boldsymbol{\theta}_t)$
38:     return $v_t, \boldsymbol{\theta}_t$, acc

39: function BACKPROPAGATE($v_t$, acc)
40:     while $v_t$ is not root do
41:         $N(v_t) \leftarrow N(v_t) + 1$
42:         $Q(v_t) \leftarrow$ acc
43:         $v_t \leftarrow$ parent of $v_t$
```

the current node.

We let all the historical data $\mathcal{H}_T = \bigcup_{i=1}^{T} \mathcal{D}_i$ to be initialized in the beginning of the algorithm to avoid doing this step in every simulation. The memory composition at task t will prevent the memory to sample from future tasks anyway. Before learning task $t > 1$, we randomly sample M memory examples in $\mathcal{M}_t \overset{M}{\sim} \mathcal{H}_T$ based on the proportion of memory examples across tasks $1, \dots, (t-1)$ given by the memory combination in the current node v_t. The memory $\mathcal{M}_t$ is then concatenated with the current dataset, i.e. $\mathcal{D}_t \cup \mathcal{M}_t$, and then used for training the model in $\text{TRAINMODEL}(\cdot)$ using mini-batch stochastic gradient descent. The command $\text{TESTMODEL}(\cdot)$ consists of calculating the average task accuracy after learning task T on the held-out test sets to retrieve the reward in ACC. The algorithm returns the node v_T^{best}, the final model $\boldsymbol{\theta}_T^{best}$, and the final accuracy best_acc that was found from the search within the computational budget.

C. Experimental Settings

In this section, we describe the full details of the experimental settings used in this paper.

Datasets: We conduct experiments on four datasets commonly used benchmarks in the continual learning literature. Split MNIST (Zenke et al., 2017) is a variant of the MNIST (LeCun et al., 1998) dataset where the classes have been divided into five tasks incoming in the following order: 0/1, 2/3, 4/5, 6/7, and 8/9. Fashion-MNIST (Xiao et al., 2017) is of similar size to MNIST and consists of grayscale images of different clothes, where the classes have been divided into five tasks: T-shirt/Trouser, Pullover/Dress, Coat/Sandals, Shirt/Sneaker, and Bag/Ankle boots. Similar to MNIST, Split notMNIST (Bulatov, 2011) consists of ten classes of the letters A-J with various fonts. The classes are divided into five tasks: A/B, C/D, E/F, G/H, and I/J. Furthermore, we use training/test split provided by Ebrahimi (Ebrahimi et al., 2020). For Split CIFAR-100 (Krizhevsky & Hinton, 2009), we divide the 100 classes into 20 tasks with five classes each (Lopez-Paz & Ranzato, 2017; Rebuffi et al., 2017).

Architectures and Hyperparameters: In all experiments, we assume that task information is available at test time. We use a fully-connected multihead network with two hidden layers consisting of 256 hidden units with ReLU activations for Split MNIST, Split FashionMNIST, and Split notMNIST. For Split CIFAR-100, we use a multihead convolutional neural network built according to the architecture used in (Adel et al., 2019; Schwarz et al., 2018; Vinyals et al., 2016), which consists of four 3x3 convolutional blocks, i.e. convolutional layer followed by batch normalization (Ioffe & Szegedy, 2015), with 64 filters, ReLU activations, and 2x2 Max-pooling. We train all networks with the Adam optimizer (Kingma & Ba, 2014) with starting learning rate $\eta = 0.001$ and hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Note that the learning rate for Adam is not reset before training on a new task. We train the networks for 20, 50, 100, and 25 epochs per task for Split MNIST, Split FashionMNIST, Split notMNIST, and Split CIFAR-100 respectively. The

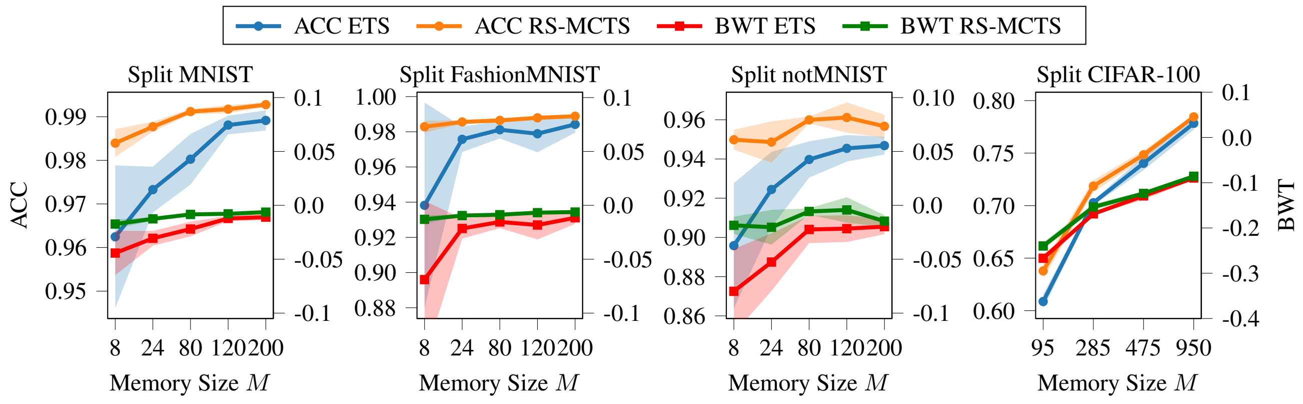

Figure 4. Average test classification accuracies over tasks after training on the final task (ACC) and backward transfer (BWT) over different memory sizes M for Split MNIST, Split FashionMNIST, Split notMNIST, and Split CIFAR-100. We show accuracies obtained by using an equal proportion of examples from previous tasks (ETS) as well as the result from the best replay schedules found with MCTS (RS-MCTS). All results have been averaged over 5 seeds.

mini-batch size is set to 128 for Split MNIST and 256 for the remaining datasets.

Monte Carlo Tree Search: The replay schedule protocol and the tree are built as described in Section 2.2. The exploration constant for UCT in Equation 1 is set to $C = 0.1$ in all experiments (Chaudhry & Lee, 2018). The number of MCTS simulations varies across the datasets based on training time per task and the number of tasks.

Historical Data Size: In Section 2.1, we assume that the historical data at time t to be given by $\mathcal{H}_t = \bigcup_{i=1}^{t-1} \mathcal{D}_i$ and consisting of H examples in total. In all experiments, we have reduced the historical data size by updating $\mathcal{H}_t$ with a subset of M examples randomly drawn from $\mathcal{D}_t$ after learning task t, such that $M << |\mathcal{D}_t|$. The historical data size is then given by $H = M \times (t-1)$ at task t. We chose to select M examples from every task since the option to replay memory $\mathcal{M}$ with M examples from one task only exists in the search tree. Furthermore, we balance the number of examples for each class within a task when we update $\mathcal{H}_t$ with M examples from task t.

Memory Size: We select the different memory sizes M to use in the experiments based on the number of tasks and classes per task in each dataset. At the final task, we want to ensure that we can replay at least one memory example from every class. We can then select the memory size using $M = C \times (T-1) \times S$ where C is the number of classes per task, T is the total number of tasks in the dataset, and S are the number of examples per class that we want when learning task T. For example, Split MNIST has $C = 2$ classes per task and $T = 5$ tasks, which yields $M = 8$ and $M = 24$ if we want $S = 1$ and $S = 3$ examples per class respectively. Split CIFAR-100 has $C = 5$ classes per task and $T = 20$ tasks, which gives $M = 95$ and $M = 285$ for $S = 1$ and $S = 3$ examples per class respectively.

Evaluation Metrics: We report results on commonly used metrics for evaluating the classification performance and amount of catastrophic forgetting in the network. We use the average test classification accuracy of all tasks after training on the final task (ACC). To assess catastrophic forgetting, we use backward transfer (BWT) (Lopez-Paz & Ranzato, 2017) which measures how much learning new tasks have influenced the performance on older tasks. Formally, both metrics are given by:

$$\text{ACC} = \frac{1}{T} \sum_{i=1}^{T} R_{T,i} \tag{2}$$

$$\text{BWT} = \frac{1}{T-1} \sum_{i=1}^{T-1} R_{T,i} - R_{i,i} \tag{3}$$

where $R_{t,i}$ is the test accuracy on task i after sequentially learning the t-th task.

Experiment with Single Task Replay Memory: We motivated the need for replay scheduling in continual learning with Figure 1 in Section 1. This simple experiment was performed on Split MNIST where the replay memory only contains examples from the first task, i.e., learning the classes $0/1$. Furthermore, the memory can only be replayed at one point in time and we show the performance on each task when the memory is replayed at different time steps. We set the memory size to $M = 10$ examples such that the memory holds 5 examples from both classes. We use the same network architecture and hyperparameters as described above for Split MNIST. The ACC metric above each subfigure corresponds to the ACC for training a network with the single task memory replay at different tasks. We observe that choosing different time points to replay the same memory leads to noticeably different results in the final performance, and in this example, the best final performance is achieved when the memory is used when learning Task 3. Therefore, we argue that finding the proper schedule of what tasks to replay at what time in the fixed memory situation can be critical for continual learning.

D. Experimental Analysis

In this section, we present experimental analysis on learning replay schedules with MCTS. First, we show that our method has greater advantage over the standard replay strategy without scheduling when the memory size is small in Section D.1. In Section D.2, we analyze the progress of task accuracies during learning by inspecting the replay schedules and deduce that the learned replay schedules are consistent with human learning processes, such as spaced repetition. Furthermore, we illustrate that RS-MCTS can be used with various sample selection strategies than random selection in Appendix D.3, and also be applied to the class incremental learning setting in Appendix D.4.

D.1. Results with Varying Memory Size M

In these experiments, we show that RS-MCTS improves on both ACC and BWT across different memory sizes M for all four datasets with Figure 4. We observe that RS-MCTS replay schedules yield better final classification performance compared to ETS, especially for small M. Furthermore, the replay schedules from RS-MCTS yields slightly better BWT than ETS. As the memory size increases, the results from using ETS approaches the ACC and BWT achieved by RS-MCTS. This indicates that replay scheduling is extremely needed when the memory budget is limited as in many real-world cases.

D.2. Task Accuracies and Replay Schedule Visualizations

We bring further insights about the performance boost from good replay schedules by inspecting how the task accuracies progress during the learning phase.

Split MNIST: In Figure 8, we show the progress in classification performance for each task when using ETS and RS-MCTS with memory size $M = 24$ on Split MNIST. For comparison, we also show the performance from a network that is fine-tuning on the current task without using replay. Both ETS and RS-MCTS overcome catastrophic forgetting to a large degree compared to the fine-tuning network. Our method RS-MCTS further improves the performance compared to ETS with the same memory, which indicates that learning the time to learn can be more efficient against catastrophic forgetting. Especially, Task 2 in Split MNIST seems to be the most difficult task to remember since it has the lowest final performance using the fine-tuning network. Our RS-MCTS schedule manages to retain its performance on Task 2 better than ETS. To bring more insights to this behavior, we look into the proportions of the memory examples per task in the corresponding replay schedule from RS-MCTS in Figure 5 (top). At Task 3, we see that the schedule fills the memory with data from Task 2 and discards replaying Task 1. This helps the network to retain knowledge about Task 2 better than ETS at the cost of forgetting Task 1 slightly. This shows that the learned policy has considered the difficulty level of different tasks. At the next task, the RS-MCTS schedule has decided to rehearse on Task 1 and discards rehearse on Task 2 until training on Task 5. This behavior is similar to spaced repetition, where increasing the time interval between rehearsal helps memory retention. We emphasize that even on datasets with few tasks, using learned replay schedules can overcome catastrophic forgetting better than standard ETS approaches.

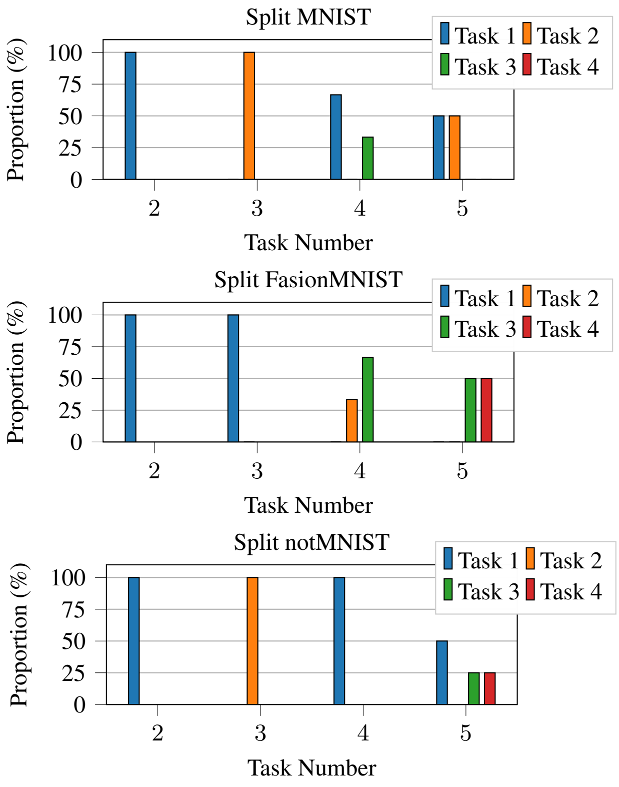

Figure 5. The proportion of memory examples per task from replay schedule found by RS-MCTS on Split MNIST (top), Split FashionMNIST (middle), and Split notMNIST (bottom) with memory size $M = 24$. The x-axis displays the current task and the y-axis shows the proportion of examples from the historical tasks. These replay schedules correspond to the learning progress in task accuracy performance for RS-MCTS in Figures 8, 9, and 10 for Split MNIST, Split FashionMNIST, and Split notMNIST respectively.

Split Fashion-MNIST: Figure 9 shows the progress in classification performance for a network using fine-tuning and replay schedules from ETS and RS-MCTS with memory size $M = 24$ when training on each task in Split FashionMNIST. Both ETS and RS-MCTS overcome catastrophic

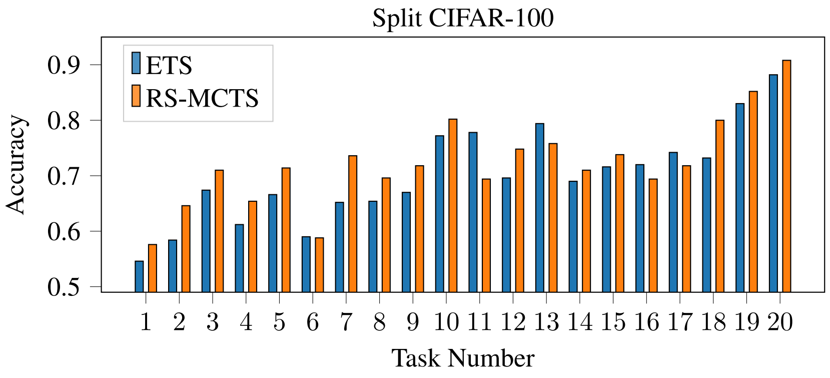

Figure 6. Comparison of test classification accuracies after learning the final task on Split CIFAR-100 from a network trained with ETS and RS-MCTS with memory size $M = 285$. RS-MCTS yields higher accuracies on all tasks except Task 6, 11, 13, 16, and 17. The corresponding replay schedule for RS-MCTS are found in Figure 7. Results are shown for a single seed.

forgetting significantly compared to the fine-tuning network. We observe that our RS-MCTS schedule manages to retain its performance on Task 1 on the final time step even if the Task 1 performance had dropped to almost 85% at the previous time step. Figure 5 (middle) shows however that this performance boost on Task 1 was the outcome of replaying Task 3 and 4 rather than Task 1 at the final time step. The performance increase of Task 1 is observed for the ETS schedule as well, which could mean that Task 1 benefits from learning Task 5. Still, RS-MCTS manages to receive better overall performance after learning Task 5 than ETS which shows the advantages of using a flexible replay schedule.

Split notMNIST: Figure 10 shows a similar visualization of the task classification performance for Split notMNIST. The fine-tuning network forgets Task 1 and 3, which both ETS and RS-MCTS manage to remember. However, RS-MCTS maintains its performance on Task 1 and 2 better than ETS which only yields good performance on the succeeding tasks. Figure 5 (bottom) shows the proportion of memory examples for RS-MCTS. In particular, the schedule fills the memory with data from Task 2 at Task 3 but still retains its performance on the future time steps even if the schedule replays other tasks than Task 2.

Split CIFAR-100: Due to the large number of tasks, instead of showing the whole learning process, we show the results at the final step for Split CIFAR-100 in Figure 6. We observe that our method performs better than ETS on 15 out of 20 tasks with the same memory size $M = 285$. We visualize the corresponding replay schedule to gain insights into the behavior of the RS-MCTS policy. Figure 7 shows a bubble plot of the proportion of memory examples that are used for replay at each task. Each color of the circles corresponds to a historical task and its size represents the proportion of examples that are replayed at the current task. Thus, each column of the circles corresponds to the memory composition at the current task time. As the memory is

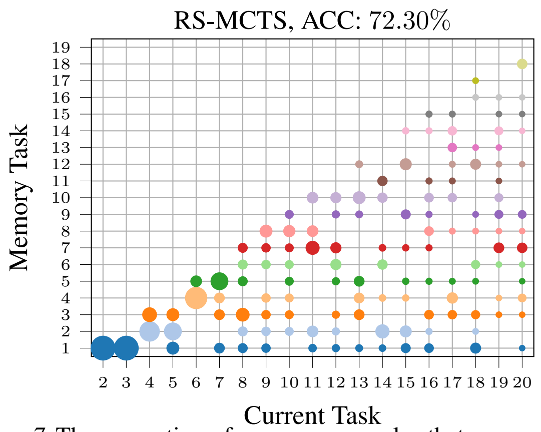

Figure 7. The proportion of memory examples that are used for replay at each task from replay schedules using RS-MCTS when training on Split CIFAR-100 with memory size $M = 285$. The color of the circles corresponds to a historical task and its size represents the proportion of examples that are replayed at the current task. The sum of areas of circles in each column is fixed across different time. We show the ACC yielded by the replay schedules above each figure.

limited, the sum of areas of circles in each column is fixed across different time steps. The memory examples per task vary dynamically over time in a sophisticated nonlinear way which would be difficult to replace by any heuristics. This motivates the importance of learning the time to learn in continual learning with fixed size memory. Furthermore, we observe that the schedule replays the early tasks with a similar proportion in the beginning of learning but gradually increases the time interval between rehearsal. This shows that our learned schedule establishment stems well with the idea of spaced repetition that early tasks require less repetition in the future for retaining knowledge. Moreover, our method is potentially more optimal than native spaced repetition as it considers the relationship among tasks. For example, Task 5-8 need less rehearsal which could potentially be that they are correlated with other tasks or they are simpler.

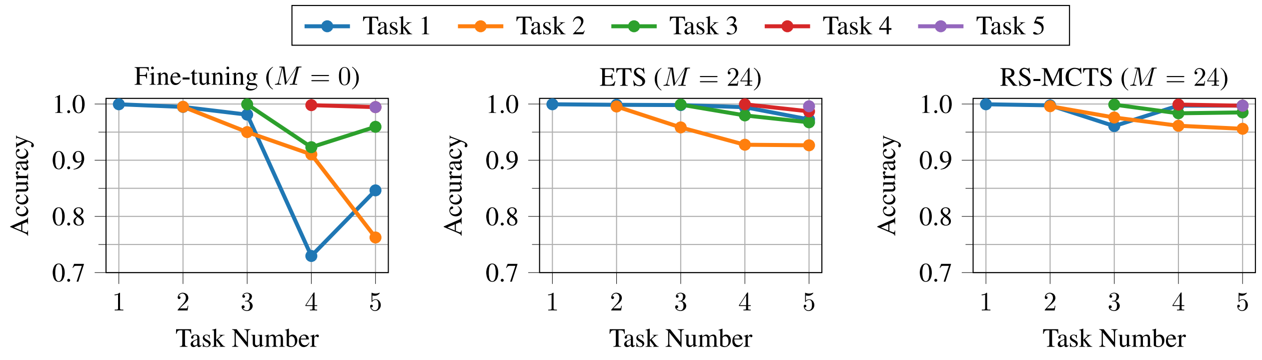

Figure 8. Comparison of test classification accuracies for Task 1-5 on Split MNIST from a network trained without replay (Fine-tuning), ETS, and RS-MCTS. The memory size is set to $M = 24$ for ETS and RS-MCTS. The replay schedule found by RS-MCTS is shown in Figure 5 (top). Results are shown for a single seed.

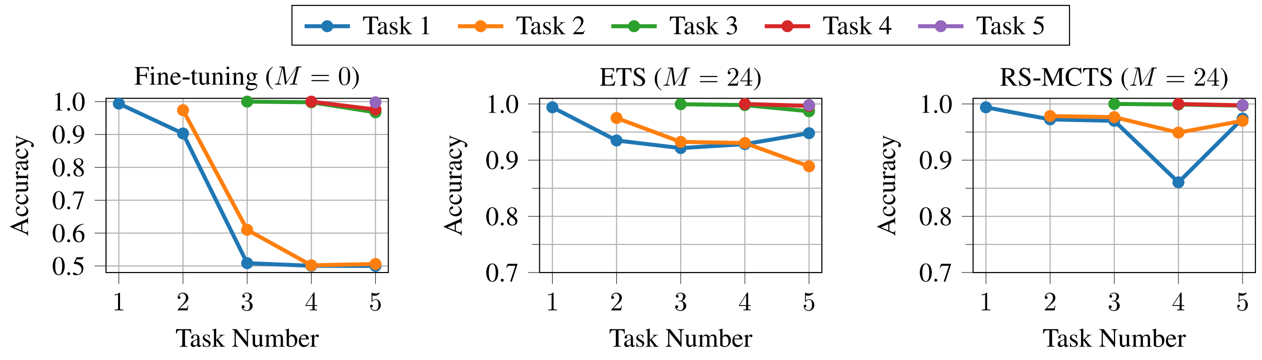

Figure 9. Comparison of test classification accuracies for Task 1-5 on Split FashionMNIST from a network trained without replay (Fine-tuning), ETS, and RS-MCTS. The memory size is set to $M = 24$ for ETS and RS-MCTS. The replay schedule found by RS-MCTS is shown in Figure 5 (middle). Results are shown for a single seed.

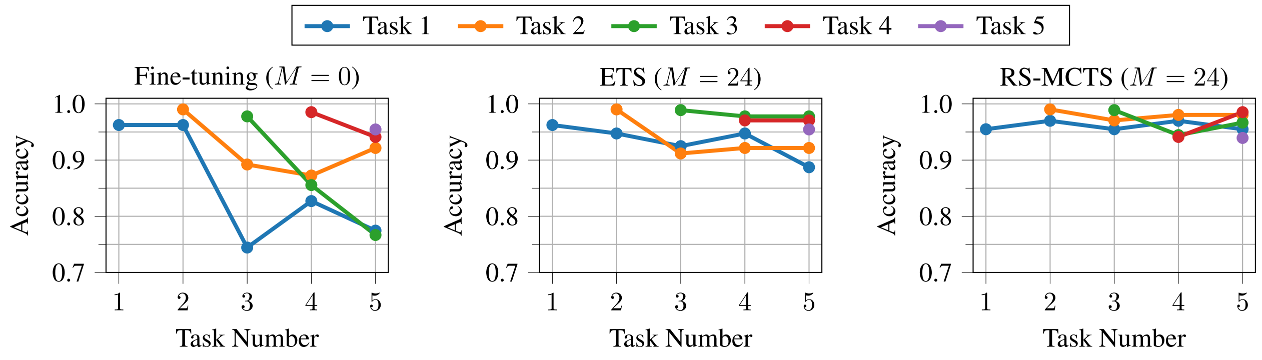

Figure 10. Comparison of test classification accuracies for Task 1-5 on Split notMNIST from a network trained without replay (Fine-tuning), ETS, and RS-MCTS. The memory size is set to $M = 24$ for ETS and RS-MCTS. The replay schedule found by RS-MCTS is shown in Figure 5 (bottom). Results are shown for a single seed.

D.3. Alternative Memory Selection Strategies

In this section, we evaluate RS-MCTS on two alternative selection strategies, namely Mean-of-Features (MoF) (Rebuffi et al., 2017) and K-center Coreset (Nguyen et al., 2017), and compare these to using random selection of examples to store as historical data. We begin by giving a brief description of both strategies:

Mean-of-Features: We extract feature representations before the classification layer of all images for every class. Then the examples to store in memory are selected based on the similarity between the feature representations and their moving mean value. This selection strategy was used in iCaRL (Rebuffi et al., 2017) where they grabbed such examples to create an exemplar set for performing classification in feature space. Similar to Chaudhry (Chaudhry et al., 2019), we use this strategy for selecting memory examples that should be close to the mode of the class in feature space.

K-center Coreset: We employ the greedy K-center algorithm (Gonzalez, 1985) that was used for memory replay in Variational Continual Learning (Nguyen et al., 2017). In contrast to MoF, this strategy operates in input space and returns examples that are spread throughout the input space.

We perform experiments on Split MNIST to compare MoF and K-center Coreset to random selection as selection strategies. Figure 11 shows the progress in ACC when searching for replay schedules with RS-MCTS using the three strategies and memory size $M = 24$. We also show the performance from using ETS with each strategy as baselines. All three strategies find better replay schedules than ETS using less than 50 simulations. Both MoF and K-center Coreset strategies perform on par with random selection over the 500 simulations with RS-MCTS.

In Figure 12, we compare the performance using the selection strategies for various memory sizes. The schedules from RS-MCTS for the selection strategies perform on par as the memory size M increases where K-center Coreset strategy gains slightly superior performance at $M = 120$ and $M = 200$. A potential reason for this slight performance increase could be that K-center Coreset selects memory examples to store that are spread out in the input space for each class rather than selecting examples that are near the mode for all examples as in MoF.

D.4. Class Incremental Learning

In this section, we apply replay scheduling to the class incremental learning (Class-IL) setting where task information is unavailable at test time. This scenario requires the model to both infer the task and class of the test input. This continual learning scenario is considered to be more challenging than the task incremental learning (Task-IL) setting where task labels are available at test time that we address in the main paper. We assess RS-MCTS for Class-IL on the Split MNIST dataset. The experimental settings are the same as in the main paper, except that the network architecture uses a single-head classification layer in this scenario.

We need to adjust the way to construct the action space of memory combinations to apply RS-MCTS to the Class-IL setting. As earlier, we create $t-1$ bins $[b_1, b_2, ...b_{t-1}]$ and choose a historical task to sample for each bin $b_i \in 1, .., t-1$ before learning task t. In the Class-IL setting, the model will catastrophically forget tasks that are excluded from the selected memory combination. Therefore, we ensure that the memory combinations includes some proportion of samples from all previous tasks by extending the memory with $t-1$ additional bins $[c_1, c_2, ..., c_{t-1}]$ where each c_i includes samples from its corresponding task i. For example, at Task 3, we have Task 1 and 2 in history; so the unique choices of bins are $[1,1], [1,2], [2,2]$. We then extend each of these unique choices with $[1,2]$, such that the unique choices become $[1,1,1,2], [1,2,1,2], [2,2,1,2]$, where $[1,1,1,2]$ indicates that 75% and 25% of the memory is from Task 1 and Task 2 respectively, and $[1,2,1,2]$ indicates that half memory is from Task 1 and the other half are from Task 2 etc.

In the first experiment, we inspect the performance progress measured in ACC over MCTS simulations when $M = 24$ for Split MNIST. Figure 13 shows that RS-MCTS quickly finds a significantly better ACC than ETS using less than 50 iterations. Furthermore, RS-MCTS can reach similar performance as from the breadth-first search (BFS) with significantly less computational budget. These results confirm that scheduling of which memory examples to replay is important in the Class-IL scenario.

In Figure 14, we show that the performance of RS-MCTS improves on both ACC and BWT across different memory sizes M. We observe that RS-MCTS replay schedules yield better final classification performance compared to ETS, especially for small M in the Class-IL scenario. The performance for both RS-MCTS and ETS schedules increases rapidly as the memory size becomes larger which confirms the importance of the memory size in Class-IL. The results from using the ETS schedules approach the ACC and BWT metrics achieved by RS-MCTS as the memory size grows. This indicates that replay scheduling is necessary for situations with limited memory budgets in the Class-IL scenario.

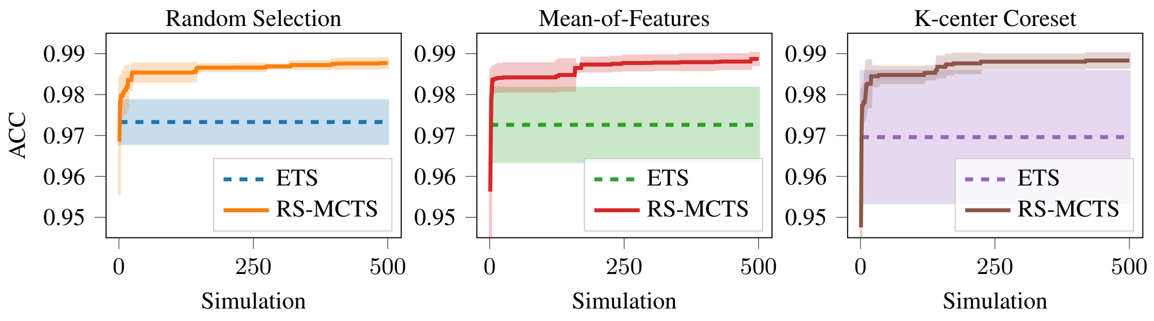

Figure 11. Best rewards measured in ACC for ETS and RS-MCTS using memory size $M = 24$ on Split MNIST for different memory selection strategies Random Selection, Mean-of-Features, and K-center Coreset. Results are averaged over 5 seeds.

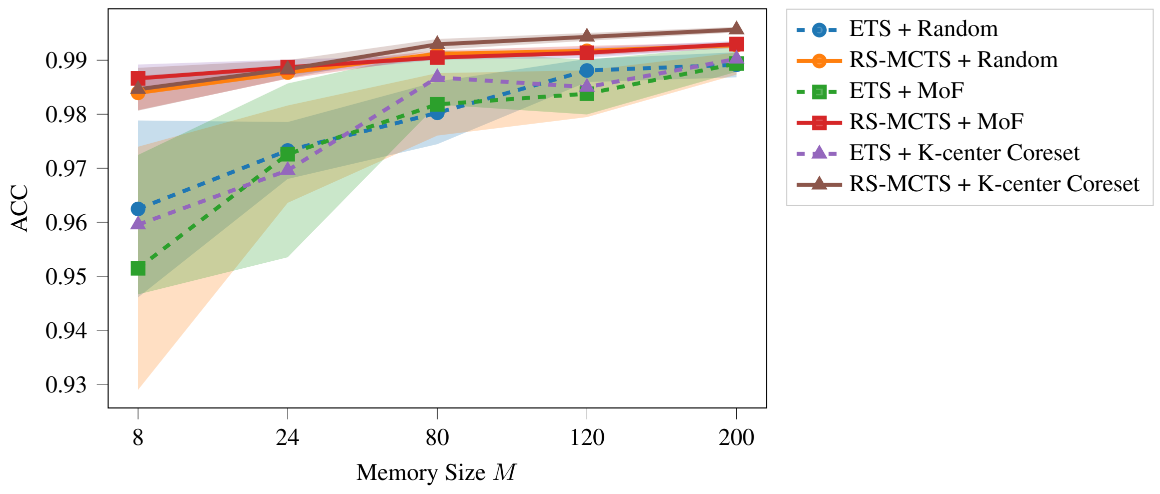

Figure 12. ACC over memory size M for ETS and RS-MCTS on Split MNIST using different memory selection methods, namely Random Selection (Random), Mean-of-Features (MoF), and K-center Coreset. All results have been averaged over 5 seeds.

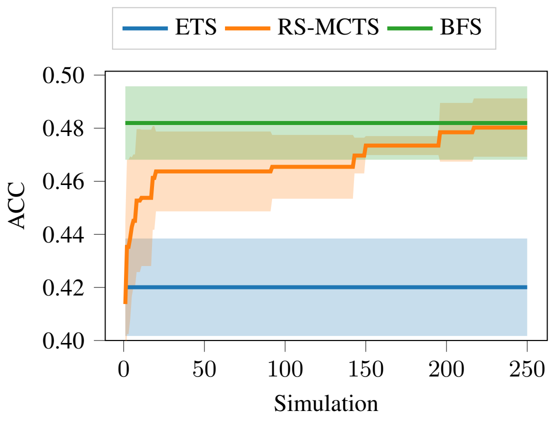

Figure 13. MCTS simulation performance on Split MNIST in the Class-IL setting where the average test classification accuracies over tasks after training on the final task (ACC) is used as the reward. The ETS performance is shown in blue as a baseline and the green line show the ACC from the optimal schedules found from a breadth-first search (BFS) as an upper bound. We use $M = 24$ and all results have been averaged over 5 seeds.

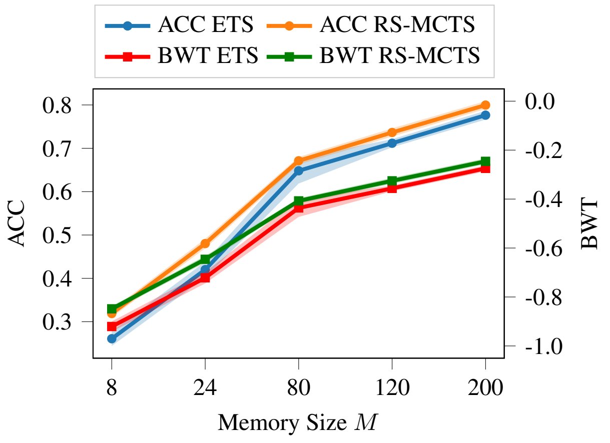

Figure 14. Average test classification accuracies over tasks after training on the final task (ACC) and backward transfer (BWT) over different memory sizes M for Split MNIST in the Class Incremental Learning setting. We show accuracies obtained by using an equal proportion of examples from previous tasks (ETS) as well as the result from the best replay schedules found with MCTS (RS-MCTS). All results have been averaged over 5 seeds.