Continual Learning by Maximizing Transfer and Minimizing Interference

Matthew Riemer[1,2], Ignacio Cases[3], Robert Ajemian[4,2], Miao Liu[1,2], Irina Rish[1,2], Yuhai Tu[1,2], and Gerald Tesauro[1,2]

[1]IBM Research, Yorktown Heights, NY
[2]MIT-IBM Watson AI Lab
[3]Linguistics and Computer Science Departments, Stanford NLP Group, Stanford University
[4]Department of Brain and Cognitive Sciences, MIT

Abstract

Lack of performance when it comes to continual learning over non-stationary distributions of data remains a major challenge in scaling neural network learning to more human realistic settings. In this work we propose a new conceptualization of the continual learning problem in terms of a temporally symmetric trade-off between transfer and interference that can be optimized by enforcing gradient alignment across examples. We then propose a new algorithm, Meta-Experience Replay (MER), that directly exploits this view by combining experience replay with optimization based meta-learning. This method learns parameters that make interference based on future gradients less likely and transfer based on future gradients more likely.[1] We conduct experiments across continual lifelong supervised learning benchmarks and non-stationary reinforcement learning environments demonstrating that our approach consistently outperforms recently proposed baselines for continual learning. Our experiments show that the gap between the performance of MER and baseline algorithms grows both as the environment gets more non-stationary and as the fraction of the total experiences stored gets smaller.[2]

1 Solving the Continual Learning Problem

Solutions to the continual learning problem [41] have been largely driven by prominent conceptualizations of the issues faced by neural networks. One popular view is *catastrophic forgetting* (interference) [28], in which the primary concern is the lack of stability in neural networks, and the main solution is to limit the extent of weight sharing across experiences by focusing on preserving past knowledge [18, 52, 22]. Another popular and more complex conceptualization is the *stability-plasticity dilemma* [6]. In this view, the primary concern is the balance between network stability (to preserve past knowledge) and plasticity (to rapidly learn the current experience). Recently proposed techniques focus on balancing limited weight sharing with some mechanism to ensure fast learning [23, 36, 25, 42, 21, 46]. We extend this view by noting that for continual learning over an unbounded number of distributions, we need to consider gradient alignment, weight sharing, and the stability-plasticity trade-off in both the forward and backward directions in time (Figure 1A).

The transfer-interference trade-off proposed in this paper (section 2) makes explicit the connection between controlling the extent of weight sharing and managing the continual learning problem. The key novelty is that we are not just concerned with current interference and transfer with respect to past examples, but also with the dynamics of transfer and interference moving forward as we learn.

[1]We consider task agnostic *future gradients*, referring to gradients of the model parameters with respect to unseen data points. These can be drawn from tasks that have already been partially learned or unseen tasks.

[2]See `https://arxiv.org/pdf/1810.11910.pdf` for an extended version of this paper.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

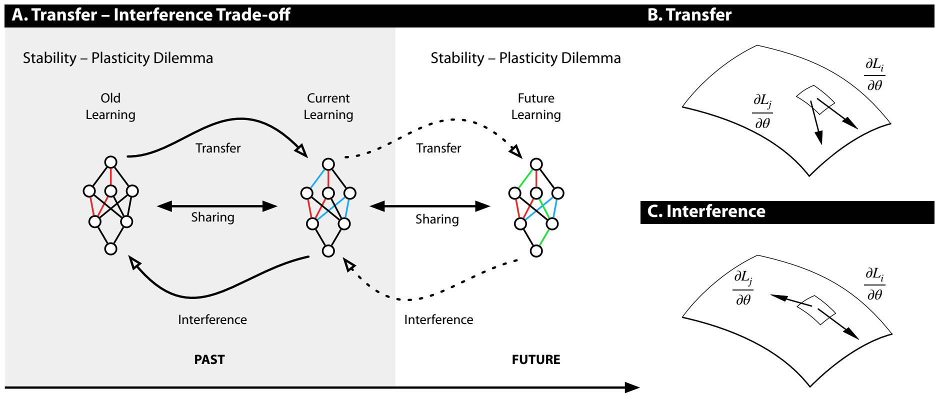
A. Transfer – Interference Trade-off
B. Transfer
C. Interference
Stability – Plasticity Dilemma
Stability – Plasticity Dilemma
Old Learning
Current Learning
Future Learning
Transfer
Transfer
Sharing
Sharing
Interference
Interference
PAST
FUTURE
$\frac{\partial L_i}{\partial \theta}$
$\frac{\partial L_j}{\partial \theta}$

Figure 1: A) The stability-plasticity dilemma considers plasticity with respect to the current learning and how it degrades old learning. The transfer-interference trade-off is the stability-plasticity dilemma considered in both forward and backward directions. B, C) Examples of transfer and interference in weight space.

This new view of the problem leads to a natural meta-learning perspective on continual learning: we would like to learn to change our parameters, so that we effect the dynamics of transfer and interference in a general sense, and not just in the past, but in the future as well. To the extent that our meta-learning into the future generalizes, this should effectively make it easier for our model to perform continual learning in non-stationary settings.

2 The Transfer-Interference Trade-off for Continual Learning

At an instant in time with parameters θ and loss L, we can define[3] operational measures of transfer and interference between two arbitrary distinct examples (x_i, y_i) and (x_j, y_j) while training with SGD. We can say that *transfer* occurs when:

$$\frac{\partial L(x_i, y_i)}{\partial \theta} \cdot \frac{\partial L(x_j, y_j)}{\partial \theta} > 0, \tag{1}$$

where $\cdot$ is the dot product operator. This implies that learning example i will without repetition improve performance on example j and vice versa (Figure 1B). On the other hand we can say that *interference* occurs when:

$$\frac{\partial L(x_i, y_i)}{\partial \theta} \cdot \frac{\partial L(x_j, y_j)}{\partial \theta} < 0. \tag{2}$$

Here, in contrast, learning example i will lead to unlearning (i.e. forgetting) of example j and vice versa (Figure 1C). The potential for transfer is maximized when weight sharing is maximized while potential for interference is minimized when weight sharing is minimized (Appendix C).

Past approaches to the stability-plasticity dilemma considered reducing weight sharing (see Appendix B and D). However, the learning process should not only reduce backward interference, but also it should not undermine effective learning on future data. This suggests extending the temporal horizon of the stability-plasticity problem forward, turning it into a continual learning problem that we call the *Transfer-Interference Trade-off* (Figure 1A). Enabling transfer to improve future performance must not disrupt performance on what has come previously. Likewise, if we mitigate interference by reducing weight-sharing, we are likely to worsen the networks capacity for transfer into the future, since transfer depends precisely on weight sharing as well. As such, our work adopts a meta-learning perspective on the continual learning problem. We would like to learn to learn each example in a way that generalizes to other examples from the overall distribution.

3 A System for Learning to Learn without Forgetting

In typical offline supervised learning, our optimization objective over the stationary distribution of x, y pairs within the dataset D and loss function L can be written as:

$$\theta = arg \min_{\theta} \mathbb{E}_{(x,y) \in D}[L(x, y)], \tag{3}$$

[3]Throughout our paper we will discuss ideas in terms of the supervised learning problem formulation. We extended the formulation to reinforcement learning as well and we provide the details in the appendix.

To maximize transfer and minimize interference, it is possible to add an auxiliary loss to the objective to bias the learning process in that direction. For example, let us consider the gradients with respect to the loss evaluated at random datapoints. Maximizing the dot products between gradients at these points would encourage sharing parameters where gradients align and separating parameters where they interfere. Ideally we want to optimize for the following objective:

$$\theta = arg\min_{\theta} \mathbb{E}_{(x_i,y_i)\&(x_j,y_j)\in D}[L(x_i,y_i) + L(x_j,y_j) - \alpha \frac{\partial L(x_i,y_i)}{\partial \theta} \cdot \frac{\partial L(x_j,y_j)}{\partial \theta}], \tag{4}$$

where (x_i,y_i) and (x_j,y_j) are randomly sampled data points. A continual learning system that optimizes for this objective needs to consider learning over a continuous non-stationary stream of data. We address this issue by implementing an experience replay module that augments online learning so that we can approximately optimize over the stationary distribution of all examples seen to the extent that our memory M captures the variation in D (Appendix H). Another problem is that the gradients of this loss depend on the second derivative, which is expensive to compute. We instead indirectly approximate this objective to a first order Taylor expansion using an online meta-learning algorithm with minimal overhead.

Building off the popular MAML [9] algorithm for optimization based meta-learning, [32] has proposed an algorithm, Reptile, that optimizes for this objective while also allowing for online updates (Appendix F). Reptile approximately optimizes for the following objective over a set of s batches:

$$\theta = arg\min_{\theta} \mathbb{E}_{B_1,...,B_s\in D}[2\sum_{i=1}^{s}[L(B_i) - \sum_{j=1}^{i-1} \alpha \frac{\partial L(B_i)}{\partial \theta} \cdot \frac{\partial L(B_j)}{\partial \theta}]], \tag{5}$$

where $B_1,...,B_s$ are batches within D. This is similar to our motivation in equation 4 to the extent that gradients produced on these batches approximate samples from the stationary distribution.

The MER learning objective: In this work, we modify the Reptile algorithm to properly integrate it with an experience replay module, facilitating continual learning while maximizing transfer and minimizing interference. As we describe in more detail in the appendix, achieving the Reptile objective in an online setting where examples are provided sequentially is non-trivial and is only achievable because of our sampling strategies for both the buffer and batch. This allows us to optimize for the following objective in a continual learning setting using our proposed MER algorithm:

$$\theta = arg\min_{\theta} \mathbb{E}_{(x_{11},y_{11}),...,(x_{sk},y_{sk})\in M}[2\sum_{i=1}^{s}\sum_{j=1}^{k}[L(x_{ij},y_{ij}) - \sum_{q=1}^{i-1}\sum_{r=1}^{j-1} \alpha \frac{\partial L(x_{ij},y_{ij})}{\partial \theta} \cdot \frac{\partial L(x_{qr},y_{qr})}{\partial \theta}]]. \tag{6}$$

The MER algorithm: MER maintains an experience replay style memory M with reservoir sampling and at each time step draws s batches including $k-1$ random samples from the buffer to be trained alongside the current example. Each of the k examples within each batch is treated as its own Reptile batch of size 1 with an inner loop Reptile meta-update after that batch is processed. We then apply the Reptile meta-update again in an outer loop across the s batches. We provide further details for MER in algorithm 4. This procedure approximates the objective of equation 6 when $\beta = 1$.

Controlling the degree of regularization: Following equation 4 we can see that using a SGD batch size of 1 has an advantage over larger batches because it allows for the second derivative information conveyed to the algorithm to be fine grained on the example level. Another reason to use sample level effective batches is that for a given number of samples drawn from the buffer we maximize s from equation 5. In equation 5, the typical offline learning loss has a weighting proportional to s and the regularizer term to maximize transfer and minimize interference has a weighting proportional to $\alpha s(s-1)/2$. So by maximizing the effective s we can put more weight on the regularization term.

Prioritizing current learning: To ensure strong regularization, we would like our number of batches processed in a Reptile update to be large. As such, we also need to make sure we provide enough priority to learning the current example, particularly because we may not store it in M. To achieve this in algorithm 4, we sample s separate batches from M that are processed sequentially and each interleaved with the current example. In the appendix we also outline two additional variants of MER with very similar properties in that they effectively approximate for the same objective.

4 Evaluation for Supervised Continual Lifelong Learning

To test the efficacy of MER we compare it to relevant baselines for continual learning of many supervised tasks from [25] and [18] (see the appendix for in-depth descriptions of these tasks).

Model	MNIST Rotations			MNIST Permutations		
	RA	LA	BTI	RA	LA	BTI
Online	46.40 ± 0.78	78.70 ± 0.52	-32.30 ± 0.97	55.42 ± 0.65	69.18 ± 0.99	-13.76 ± 1.19
Independent	60.74 ± 4.55	60.74 ± 4.55	-	55.80 ± 4.79	55.80 ± 4.79	-
Task Input	79.98 ± 0.66	81.04 ± 0.34	-1.06 ± 0.59	80.46 ± 0.80	81.20 ± 0.52	-0.74 ± 1.10
EwC	57.96 ± 1.33	78.38 ± 0.72	-20.42 ± 1.60	62.32 ± 1.34	75.64 ± 1.18	-13.32 ± 2.24
GEM	87.12 ± 0.44	86.34 ± 0.09	+0.80 ± 0.42	82.50 ± 0.42	80.30 ± 0.80	**+2.20** ± 0.57
MER	**89.56** ± 0.11	**87.62** ± 0.16	**+1.94** ± 0.18	**85.50** ± 0.16	**86.36** ± 0.21	-0.86 ± 0.21

Table 1: Performance on continual lifelong learning 20 tasks benchmarks from [25].

Model	Buffer	Many Permutations			Omniglot		
		RA	LA	BTI	RA	LA	BTI
Online	0	32.62 ± 0.43	51.68 ± 0.65	-19.06 ± 0.86	4.36 ± 0.37	5.38 ± 0.18	-1.02 ± 0.33
EWC	0	33.46 ± 0.46	51.30 ± 0.81	-17.84 ± 1.15	4.63 ± 0.14	9.43 ± 0.63	-4.80 ± 0.68
GEM	5120	56.76 ± 0.29	59.66 ± 0.46	-2.92 ± 0.52	18.03 ± 0.15	3.86 ± 0.09	**+14.19** ± 0.19
	500	32.14 ± 0.50	55.66 ± 0.53	-23.52 ± 0.87	-	-	-
MER	5120	**61.84** ± 0.25	**60.40** ± 0.36	**+1.44** ± 0.21	**75.23** ± 0.52	**69.12** ± 0.83	+6.11 ± 0.62
	500	**47.40** ± 0.35	**65.18** ± 0.20	**-17.78** ± 0.39	**32.05** ± 0.69	**28.78** ± 0.91	+3.27 ± 1.04

Table 2: Performance on many task non-stationary continual lifelong learning benchmarks.

We follow [25] and consider final retained accuracy (RA) across all tasks after training sequentially on all tasks as our main metric for comparing approaches. We also report *learning accuracy (LA)*, the average accuracy for each task after it is learned, and *backward transfer and interference (BTI)*, the average change in accuracy from when a task is learned to the end of training. A highly negative BTI reflects catastrophic forgetting. Forward transfer and interference [25] is only applicable for one task we explore, so we provide details in Appendix M.

Performance of MER on supervised continual learning benchmarks: we consider two MNIST-based, continual learning benchmarks from [25]: *MNIST Permutations* and *MNIST Rotations* (see the appendix). In Table 1 we report how clearly GEM outperforms other baselines and how our approach adds significantly over GEM in terms of retained accuracy. MER strikes a superior balance between transfer and interference, displaying the best adaption to incoming tasks while also providing very strong retention of knowledge when learning future tasks.

Effect of MER in increasingly non-stationary settings: continual learning with only relatively few examples per task is particularly difficult because we have less data to characterize each class to learn from and our distribution is increasingly non-stationary over a fixed amount of training. To explore this we consider two new benchmarks: *Many Permutations* a variant of *MNIST Permutations* that has 5 times more tasks (100 total) and 5 times less training examples per task (200 each); and *Omniglot* [20], treating each of the 50 alphabets to be a task (details in Appendix E).

In Table 2 we show how GEM and MER both achieve significant performance gains on top of EWC and online learning. We also see that increasingly non-stationary settings lead to a larger performance gain for MER over GEM. Gains are quite significant for *Many Permutations* and remarkable for *Omniglot*. *Omniglot* is even more non-stationary including slightly fewer examples per task and MER nearly quadruples the performance of baseline techniques. Both EWC and GEM have major deficits for learning on Omniglot that are resolved by MER which achieves far better performance when quickly learning the current task with high computational efficiency (see Appendix E).

Shifting in the distribution of gradient dot products: Confirming the motivation of our work, MER results in significant changes in the distribution of gradient dot products between new incoming examples and past examples over the course of learning when compared to experience replay (ER). We designed an experiment precisely measuring these distances (see appendix for details). Table 3 shows a significant and reproducible difference in the mean gradient encountered is seen for MER in comparison to ER alone, confirming the desired effect of the objective function (equation 6) proposed in this paper and the power of the proposed transfer-interference trade-off (Section 2).

Model	MNIST Permutations	MNIST Rotations	Many Permutations
ER	-0.569 ± 0.077	-1.652 ± 0.082	-1.280 ± 0.078
MER	+0.042 ± 0.017	+0.017 ± 0.007	+0.131 ± 0.027

Table 3: Analysis of the mean dot product across the period of learning between gradients on incoming examples and gradients on randomly sampled past examples across 5 runs on MNIST based benchmarks.

Acknowledgments

We would like to thank Pouya Bashivan, Christopher Potts, Dan Jurafsky, and Joshua Greene for their input and support of this work. This research was supported by the MIT-IBM Watson AI Lab, and is based in part upon work supported by the Stanford Data Science Initiative and by the NSF under Grant No. BCS-1456077 and the NSF Award IIS-1514268.

References

[1] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *ICLR*, 2018.

[2] Rahaf Aljundi, Jay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings CVPR 2017*, pages 3366–3375, 2017.

[3] Bernard Ans and Stéphane Rousset. Avoiding catastrophic forgetting by coupling two reverberating neural networks. *Comptes Rendus de l'Académie des Sciences-Series III-Sciences de la Vie*, 320(12):989–997, 1997.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. 2017.

[5] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[6] Gail A Carpenter and Stephen Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer vision, graphics, and image processing*, 37(1):54–115, 1987.

[7] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. *arXiv preprint arXiv:1801.10112*, 2018.

[8] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[10] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *arXiv preprint arXiv:1710.11622*, 2017.

[11] Robert M French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. 1991.

[12] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[13] Stefano Fusi, Patrick J Drew, and Larry F Abbott. Cascade models of synaptically stored memories. *Neuron*, 45(4):599–611, 2005.

[14] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012.

[15] Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. 1987.

[16] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835, 2017.

[19] Subhaneil Lahiri and Surya Ganguli. A memory frontier for complex synapses. In *Advances in neural information processing systems*, pages 1034–1042, 2013.

[20] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Cognitive Science Society*, volume 33, 2011.

[21] Jeongtae Lee, Jaehong Yun, Sungju Hwang, and Eunho Yang. Lifelong learning with dynamically expandable networks. *ICLR*, 2018.

[22] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, pages 4652–4662, 2017.

[23] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, pages 614–629. Springer, 2016.

[24] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[25] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continuum learning. *NIPS*, 2017.

[26] Daniel J Mankowitz, Timothy A Mann, Pierre-Luc Bacon, Doina Precup, and Shie Mannor. Learning robust options. *arXiv preprint arXiv:1802.03236*, 2018.

[27] James L McClelland, Bruce L McNaughton, and Randall C O'reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[28] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989.

[29] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3994–4003, 2016.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[31] Jacob MJ Murre. Learning and categorization in modular neural networks. 1992.

[32] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

[33] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[34] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H Lampert. icarl: Incremental classifier and representation learning. *CVPR*, 2017.

[35] Matthew Riemer, Michele Franceschini, Djallel Bouneffouf, and Tim Klinger. Generative knowledge distillation for general purpose function compression. *NIPS 2017 Workshop on Teaching Machines, Robots, and Humans*, 5:30, 2017.

[36] Matthew Riemer, Elham Khabiri, and Richard Goodwin. Representation stability as a regularizer for improved text analytics transfer learning. *arXiv preprint arXiv:1704.03617*, 2016.

[37] Matthew Riemer, Tim Klinger, Michele Franceschini, and Djallel Bouneffouf. Scalable recollections for continual lifelong learning. *arXiv preprint arXiv:1711.06761*, 2017.

[38] Matthew Riemer, Sophia Krasikov, and Harini Srinivasan. A deep learning and knowledge transfer based architecture for social media user characteristic determination. In *Proceedings of the third International Workshop on Natural Language Processing for Social Media*, pages 39–47, 2015.

[39] Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. *NIPS*, 2018.

[40] Matthew Riemer, Aditya Vempaty, Flavio Calmon, Fenno Heath, Richard Hull, and Elham Khabiri. Correcting forecasts with multifactor neural attention. In *International Conference on Machine Learning*, pages 3010–3019, 2016.

[41] Mark Bishop Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

[42] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *ICLR*, 2018.

[43] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

[44] Jürgen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004.

[45] Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

[46] Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

[47] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[48] Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[49] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

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

[51] Yongxin Yang and Timothy Hospedales. Deep multi-task representation learning: A tensor factorisation approach. *ICLR*, 2017.

[52] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017.

[53] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

A Continual Learning Problem Formulation

In the classical *offline supervised learning* setting, a learning agent is given a fixed training data set $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ of n samples, each containing an input feature vector $\boldsymbol{x}_i \in \mathscr{X}$ associated with the corresponding output (target, or label) $y_i \in \mathscr{Y}$; a common assumption is that the training samples are i.i.d. samples drawn from the same unknown joint probability distribution $P(\boldsymbol{x}, y)$. The learning task is often formulated as a *function approximation* problem, i.e. finding a function, or model, $f_{\boldsymbol{\theta}}(\boldsymbol{x}) : \mathscr{X} \to \mathscr{Y}$ from a given class of models (e.g., neural networks, decision trees, linear functions, etc.)

where $\boldsymbol{\theta}$ are the parameters estimated from data. Given a loss function $L(f_{\boldsymbol{\theta}}(\boldsymbol{x}),y)$, the parameter estimation is formulated as an empirical risk minimization problem: $\min_{\boldsymbol{\theta}} \frac{1}{|D|}\sum_{(\mathbf{x}_i,y_i)\in D} L(f_{\boldsymbol{\theta}}(\boldsymbol{x}),y)$.

On the contrary, the *online learning* setting does not assume a fixed training dataset but rather a stream of data samples, where unlabeled feature vectors arrive one at a time, or in small mini-batches, and the learner must assign labels to those inputs, receive the correct labels, and update the model accordingly, in iterative fashion. While classical online learning assumes i.i.d. samples, *continual* or *lifelong learning* does not make such an assumption, and requires a learning agent to handle non-stationary data streams.

In this work, we define *continual learning* as online learning from a non-stationary input data stream, with a specific type of non-stationarity as defined below. Namely, we follow a commonly used setting to define non-stationary conditions for continual learning, dubbed *locally i.i.d* by [25], where the agent learns over a sequence of separate stationary distributions one after another. We call the individual stationary distributions *tasks*, where each task t_k is an online supervised learning problem associated with its own data probability distribution $P_k(\boldsymbol{x},y)$. Namely, we are given a (potentially infinite) sequence

$$(\boldsymbol{x}_1,y_1,t_1),...,(\boldsymbol{x}_i,y_i,t_i),...,(\boldsymbol{x}_{i+j},y_{i+j},t_{i+j})$$

While many continual learning methods assume the task descriptors t_k are available to a learner, we are interested in developing approaches which do not have to rely on such information and can learn continuously without explicit announcement of the task change. Borrowing terminology from [7], we explore the *single-headed* setting in most of our experiments, which keeps learning a common function f_θ across changing tasks. In contrast, *multi-headed* learning, which we consider for our Omniglot experiments, involves a separate final classification layer for each task. This makes more sense in case of Omniglot dataset, where the number of classes for each task varies considerably from task to task. We should also note that for Omniglot we consider a setting that is locally i.i.d. at the class level rather than the task level.

B Relation to Past Work

With regard to the continual learning setting specifically, other recent work has explored similar operational measures of transfer and interference. For example, the notions of Forward Transfer and Backward Transfer were explored in [25]. However, the approach of that work, GEM, was primarily concerned with solving the classic stability-plasticity dilemma [6] at a specific instance of time. Adjustments to gradients on the current data are made in an adhoc manner solving a quadratic program separately for each example. In our work we try to learn a generalizeable theory about weight sharing that can learn to influence the distribution of gradients not just in the past and present, but in the future as well. Additionally, in [7] similar ideas were explored with operational measures of *intransigenc*e (the inability to learn new data) and *forgetting* (the loss of previous performance). These measures are also intimately related to the stability-plasticity dilemma as *intransigence* is high when plasticity is low and *forgetting* is high when stability is low. The major distinction in the transfer-interference tradeoff proposed in this work is that we aim to learn the optimal weight sharing scheme to optimize for the stability-plasticity dilemma with the hope that our learning about weight sharing will improve the stability and efficacy of learning on unseen data as well.

With regard to the problem of weight-sharing in neural networks more generally, a host of different strategies have been proposed in the past to deal with the problems of catastrophic forgetting and/or the stability-plasticity dilemma (for review, see [12]). For example, one strategy for alleviating catastrophic forgetting is to make distributed representations less distibuted – or semi-distributed [11] – for the case of past learning. Activation sharpening as introduced by [11] is a prominent example. A second strategy known as dual network models [27, 3] is based on the neurobiological finding that both hippocampal and cortical circuits contributed differentially to memory. The cortical circuits are highly distributed with overlapping representations suitable for task generalization, while the more sparse hippocampal representations tend to be non-overlapping. The existence of dual circuits provides an extra degree of freedom for balancing the dual constriants of stability and plasticity. In a similar spirit, models have been proposed that have two classes of weights operating on two different timescales [15]. A third strategy also motivated by neurobiological considerations is the use of latent synaptic dynamics [13, 19]. Here the basic idea is that synaptic strength is determined by a multiple of variables, including latent ones not easily observed, operating at different timsescales such that their net effect is to provide the system with additional degrees-of-freedom to store past experience without interfering with current learning. A fourth strategy is the use of feedback mechanisms to stabilize representations [6, 31]. In this class of models, a previously experienced memory will trigger top down feedback that prevents plasticity, while novel stimuli that experience no such feedback trigger plasticity. All of these approaches have their own strengths and weaknesses with respect to the stability-plasticity dilemma and, by extension, the transfer-interfererence tradeoff we propose.

Another relevant work is the POWERPLAY framework [44, 45] which is a method for asymptotically optimal curriculum learning that by definition cannot forget previously learned skills. POWERPLAY also uses environment-independent replay of behavioral traces to avoid forgetting previous skills. However, POWERPLAY is orthogonal to our work as we consider a different setting where the agent cannot directly control the new tasks that will be encountered in the environment and thus must instead learn to adapt and react to non-stationarity conditions.

In contrast to past work on meta-learning for few shot learning [43, 49, 33, 9] and reinforcement learning across successive tasks [1], we are not only trying to improve the speed of learning on new data, but also trying to do it in a way that preserves knowledge of past data and generalizes to future data. While past work has considered learning to influence gradient angles, so that there is more alignment and thus faster learning within a task, we focus on a setting where we would like to influence gradient angles from all tasks at all points in time.

As our model aims to influence the dynamics of weight sharing, it bears conceptual similarity to mixtures of experts [16] style models for lifelong and multi-task learning [29, 40, 2, 8, 47, 42]. MER implicitly effects the dynamics of weight sharing, but it is possible that combining it with mixtures of experts models could further amplify the ability for the model to control these dynamics. This is potentially an interesting avenue for future work.

The options framework has also been considered as a solution to a similar continual RL setting to the one we explore [26]. Options formalize the notion of temporally abstraction actions in RL. Interestingly, generic architectures designed for shallow [4] or deep [39] hierarchies of options in essence learn very complex patterns of weight sharing over time. The option hierarchies constitute an explicit mechanism of controlling the extent of weight sharing for continual learning, allowing for orthogonalization of weights relating to different skills. In contrast, our work explores a method of implicitly optimizing weight sharing for continual learning that improves the efficacy of experience replay. MER should be simple to implement in concert with options based methods and combining the two is an interesting direction for future work.

C The Connection Between Weight Sharing and the Transfer-Interference Trade-off

In this section we would like to generalize our interpretation of a large set of different weight sharing schemes including [38, 5, 42, 46] and how the concept of weight sharing impacts the dynamics of transfer (equation 1) and interference (equation 2). We will assume that we have a total parameter space θ that can be used by our network at any point in time. However, it is not a requirement that all parameters are actually used at all points in time. So, we can consider two specific instances in time. One where we receive data point (x_1, y_1) and leverage parameters θ_1. Then, at the other instance in time, we receive data point (x_2, y_2) and leverage parameters θ_2. θ_1 and θ_2 are both subsets of θ and critically the overlap between these subsets influences the possible extent of transfer and interference when training on either data point.

First let us consider two extremes. In the first extreme imagine θ_1 and θ_2 are entirely non-overlapping. As such $\frac{\partial L(x_1,y_1)}{\partial \theta} \cdot \frac{\partial L(x_2,y_2)}{\partial \theta} = 0$. On the positive side, this means that our solution has no potential for interference between the examples. On the other hand, there is no potential for transfer either. On the other extreme, we can imagine that $\theta_1 = \theta_2$. In this case, the potential for both transfer and interference is maximized as gradients with respect to every parameter have the possibility of a non-zero dot product with each other.

From this discussion it is clear that both the extreme of full weight sharing and the extreme of no weight sharing have value depending on the relationship between data points. What we would really like for continual learning is to have a system that learns when to share weights and when not to on its own. To the extent that our learning about weight sharing generalizes, this should allow us to find an optimal solution to the transfer-interference trade-off.

D Further Descriptions and Comparisons with Baseline Algorithms

Independent: originally reported in [25] is the performance of an independent predictor per task which has the same architecture but with less hidden units proportional to the number of tasks. The independent predictor can be initialized randomly or clone the last trained predictor depending on what leads to better performance.

EWC: Elastic Weight Consolidation (EWC) [18] is an algorithm that modifies online learning where the loss is regularized to avoid catastrophic forgetting by considering the importance of parameters in the model as measured by their fisher information. EWC follows the catastrophic forgetting view of the continual learning problem by promoting less sharing of parameters for *new learning* that were deemed to be important for performance on *old memories*. We utilize the code provided by [25] in our experiments. The only difference in our setting is that we provide the model one example at a time to test true continual learning rather than providing a batch of 10 examples at a time.

GEM: Gradient Episodic Memory (GEM) [25] is an algorithm meant to enhance the effectiveness of episodic storage based continual learning techniques by allowing the model to adapt to incoming examples using SGD as long as the gradients do not interfere with examples from each task stored in a memory buffer. If gradients interfere leading to a decrease in the performance of a past task, a quadratic program is used to solve for the closest gradient to the original that does not have negative gradient dot products with the aggregate memories from any previous tasks. GEM is known to achieve superior performance in comparison to other recently proposed techniques that use episodic storage like [34], making superior use of small memory buffer sizes. GEM follows similar motivation to our approach in that it also considers the intelligent use of gradient dot product information to improve the use case of supervised continual learning. As a result, it a very strong and interesting baseline to compare with our approach. We modify the original code and benchmarks provided by [25]. Once again the only difference in our setting is that we provide the model one example at a time to test true continual learning rather than providing a batch of 10 examples at a time.

We can consider the GEM algorithm as tailored to the stability-plasticity dilemma conceptualization of continual learning in that it looks to preserve performance on past tasks while allowing for maximal plasticity to the new task. To achieve this, GEM [25] solves a quadratic program to find an approximate gradient g_{new} that closely matches $\frac{\partial L(x_{new}, y_{new})}{\partial \theta}$ while ensuring that the following constraint holds:

$$g_{new} \cdot \frac{\partial L(x_{old}, y_{old})}{\partial \theta} > 0. \tag{7}$$

An independent adhoc analysis is performed to alter each incoming gradient. In contrast to MER, nothing generalizable is learned across examples about how to alter gradients.

E Continual Learning Experiments: Further Details

MNIST Permutations: is a variant of MNIST first proposed in [18] where each task is transformed by a fixed permutation of the MNIST pixels. As such, the input distribution of each task is unrelated.

MNIST Rotations: is another variant of MNIST proposed in [25] where each task contains digits rotated by a fixed angle between 0 and 180 degrees. We follow the standard benchmark setting from [25] using a modest memory buffer of size 5120 to learn 1000 sampled examples across each of 20 tasks. Following conventions, we use a two layer MLP architecture for all models with 100 hidden units in each layer. We also model our hyperparameter search after [25]. We provide detailed information about the parameters we searched for and picked in Appendix L.

Omniglot: is a variant of the Omniglot dataset [20] testing continual learning by treating each of the 50 alphabets to be a task. Following multi-task learning conventions, 90% of the data is used for training and 10% is used for testing [51]. Overall there are 1623 different character classes. We learn each class and task sequentially. Following [49] we scale the images to 28x28 and use

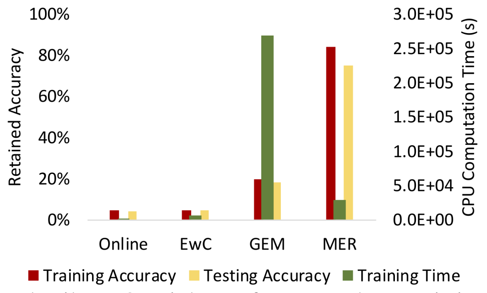

Figure 2: Further details on Omniglot performance characteristics for each model.

an architecture that consists of a stack of 4 modules before a fully connected softmax layer. Each module includes a 3x3 convolution with 64 filters, a ReLU non-linearity and 2x2 max-pooling. Considering the poor performance of online learning and EWC it is natural to question whether or not examples were learned in the first place. We experiment with using as many as 100 gradient descent steps per incoming example to ensure each example is learned when first seen. However, due to the extremely non-stationary setting no run of any variant we tried surpassed 5.5% retained accuracy. GEM also has major deficits for learning on Omniglot that are resolved by MER which achieves far better performance when it comes to quickly learning the current task. GEM maintains a buffer using a recent item based sampling strategy and thus can not deal with non-stationarity within the task nearly as well as reservoir sampling. Additionally, we found that the optimization based on the buffer was significantly less effective and less reliable as the quadratic program fails for many hyperparameter values that lead to non-positive definite matrices. Unfortunately, we could not get GEM to consistently converge on Omniglot for a memory size of 500 (significantly less than the number of classes), meanwhile MER handles it well. In fact, MER greatly outperforms GEM with an order of magnitude smaller buffer.

We provide additional details about our experiments on Omniglot in Figure 2. We plot retained training accuracy, retained testing accuracy, and computation time for the entire training period using one CPU. We find that MER strikes the best balance of computational efficiency and performance even when using algorithm 4 for MER which performs more computation than algorithm 6. The computation involved in the GEM update does not scale well to large CNN models like those that are popular for Omniglot. MER is far better able to fit the training data than our baseline models while maintaining a computational efficiency closer to online update methods like EWC than GEM.

Gradient dot product experiments: For these experiments, we maintain a history of all examples seen that is totally separate from our notion of memory buffers that only include a partial history of examples. Every time we receive a new example we use the current model to extract a gradient direction and we also randomly sample five examples from the previous history. We save the dot products of the incoming example gradient with these five past example gradients and consider the mean of the distribution of dot products seen over the course of learning for each model. We run this experiment on the best hyperparameter setting for both our ER model and our MER model with one batch per example for fair comparison. Each model is evaluated five times over the course of learning. We report mean and standard deviations of the mean gradient dot product across runs in Table 3. We can thus verify that a very significant and reproducible difference in the mean gradient encountered is seen for MER in comparison to ER alone. This difference alters the learning process making incoming examples on average result in slight transfer rather than significant interference. This analysis confirms the desired effect of the objective function in equation 6. For these tasks there are enough similarities that our meta-learning generalizes very well into the future. We should also expect it to perform well in the case of drastic domain shifts like other meta-learning algorithms driven by SGD alone [10].

F Reptile Algorithm

First order meta-learning: Model Agnostic Meta-Learning (MAML) [9] is a popular optimization based meta-learning algorithm that has the ability to both approximate any learning algorithm and generalize well to learning data out of distribution [10]. MAML's scalability is limited by the need to compute second derivatives, which is addressed with very similar performance by simply ignoring those terms. Recently, this was explained by [32] who noted through Taylor expansion that the algorithms were approximately optimizing for the same objective. We detail the standard Reptile algorithm from [32] in algorithm 1. The *sample* function randomly samples s batches of size k from dataset D. The *SGD* function applies min-batch stochastic gradient descent over a batch of data given a set of current parameters and learning rate.

Algorithm 1 Reptile for Stationary Data

```
procedure TRAIN($D, \theta, \alpha, \beta, s, k$)
    while not done do
        // Draw batches from data:
        $B_1, ..., B_s \leftarrow sample(D, s, k)$
        $\theta_0 \leftarrow \theta$
        for $i = 1, ..., s$ do
            $\theta_i \leftarrow SGD(B_i, \theta_{i-1}, \alpha)$
        end for
        // Reptile meta-update:
        $\theta \leftarrow \theta_0 + \beta(\theta_s - \theta_0)$
    end while
    return $\theta$
end procedure
```

G Details on Reservoir Sampling

Throughout this paper we refer to updates to our memory M as $M \leftarrow M \cup \{(x,y)\}$. We would like to now provide details on how we update our memory buffer using reservoir sampling as outlined in [50] (algorithm 2). Reservoir sampling solves the problem of keeping some limited number M of N total items seen before with equal probability $\frac{M}{N}$ when you don't know what number N will be in advance. The *randomInteger* function randomly draws an integer inclusively between the provided minimum and maximum values.

Algorithm 2 Reservoir Sampling with Algorithm R

```
procedure RESERVOIR($M, N, x, y$)
    if $M > N$ then
        $M[N] \leftarrow (x, y)$
    else
        $j = randomInteger(min = 0, max = N)$

        if $j < M$ then
            $M[j] \leftarrow (x, y)$
        end if
    end if
    return $M$
end procedure
```

H Experience Replay

Learning objective: The continual lifelong learning setting poses a challenge for the optimization of neural networks as examples come one by one in a non-stationary stream. Instead, we would like our network to optimize over the stationary distribution of all examples seen so far. Experience replay [24, 31] is an old technique that remains a central component of deep learning systems attempting to learn in non-stationary settings, and we will adopt here conventions from recent work [53, 37] leveraging this approach. The central feature of experience replays is keeping a memory of examples seen M that is interleaved with the training of the current example with the goal of making the neural network training more stable. As a result, experience replay approximates the objective in equation 3 to the extent that M approximates D:

$$\theta = arg \min_{\theta} \mathbb{E}_{(x,y) \in M}[L(x,y)], \tag{8}$$

M has a current size M_{size} and maximum size M_{max}. In our work, we update the buffer with reservoir sampling (Appendix G). This ensures that at every time-step the probability that any of the N examples seen has of being in the buffer is equal to M_{size}/N. The content of the buffer resembles a stationary distribution over all examples seen to the extent that the items stored captures the variation of past examples. Following the standard practice in offline learning, we train by randomly sampling a batch B from the distribution captured by M.

Prioritizing the current example: the variant of experience replay we explore differs from offline learning in that the current example has a special role ensuring that it is always interleaved with the examples sampled from the replay buffer. This is because before we proceed to the next example, we want to make sure our algorithm has the ability to optimize for the current example (particularly if it is not added to the memory). Over N examples seen, this still implies that we have trained with each example as the current example with probability per step of $1/N$. We provide an algorithm further detailing how experience replay is used in this work in the appendix (algorithm 3).

Concerns about storing examples: Obviously, it is not scalable to store every experience seen in memory. As such, in this work we focus on showing that we can achieve greater performance than baseline techniques when each approach is provided with only a small memory buffer.

We detail the our variant of the experience replay in algorithm 3. This procedure closely follows recent enhancements discussed in [53, 37, 35] The *sample* function randomly samples $k-1$ examples from the memory buffer M and interleaves them with the current example to form a single size k batch. The *SGD* function applies min-batch stochastic gradient descent over a batch of data given a set of current parameters and learning rate.

Algorithm 3 Experience Replay (ER) with Reservoir Sampling

```
procedure TRAIN($D, \theta, \alpha, k$)
    $M \leftarrow \{\}$
    for $t = 1, ..., T$ do
        for $(x, y)$ in $D_t$ do
            // Draw batch from buffer:
            $B \leftarrow sample(x, y, k, M)$
            // Update parameters with mini-batch SGD:
            $\theta \leftarrow SGD(B, \theta, \alpha)$
            // Reservoir sampling memory update:
            $M \leftarrow M \cup \{(x, y)\}$ (algorithm 2)
        end for
    end for
    return $\theta, M$
end procedure
```

I The Variants of MER

The primary version of MER, which we utilize this work is summarized in algorithm 4

Algorithm 4 Meta-Experience Replay (MER)

```
procedure TRAIN($D, \theta, \alpha, \beta, \gamma, s, k$)
    $M \leftarrow \{\}$
    for $t = 1, ..., T$ do
        for $(x, y)$ in $D_t$ do
            // Draw batches from buffer:
            $B_1, ..., B_s \leftarrow sample(x, y, s, k, M)$
            $\theta_0^A \leftarrow \theta$
            for $i = 1, ..., s$ do
                $\theta_{i,0}^W \leftarrow \theta$
                for $j = 1, ..., k$ do
                    $x_c, y_c \leftarrow B_i[j]$
                    $\theta_{i,j}^W \leftarrow SGD(x_c, y_c, \theta_{i,j-1}^W, \alpha)$
                end for
                // Within batch Reptile meta-update:
                $\theta \leftarrow \theta_{i,0}^W + \beta(\theta_{i,k}^W - \theta_{i,0}^W)$
                $\theta_i^A \leftarrow \theta$
            end for
            // Across batch Reptile meta-update:
            $\theta \leftarrow \theta_0^A + \gamma(\theta_s^A - \theta_0^A)$
            // Reservoir sampling memory update:
            $M \leftarrow M \cup \{(x, y)\}$ (algorithm 2)
        end for
    end for
    return $\theta, M$
end procedure
```

We detail two additional variants of MER (algorithm 4) in algorithms 5 and 6. The *sample* function takes on a slightly different meaning in each variant of the algorithm. In algorithm 4 *sample* is used to produce s batches consisting of $k-1$ random examples from the memory buffer and the current example. In algorithm 5 *sample* is used to produce one batch consisting of $sk - s$ examples from the memory buffer and s copies of the current example. In algorithm 6 *sample* is used to produce one batch consisting of $k-1$ examples from the memory buffer. In contrast, the *SGD* function carries a common meaning across algorithms, applying stochastic gradient descent over a particular input and output given a set of current parameters and learning rate.

Algorithm 5 Meta-Experience Replay (MER) - One Big Batch

```
procedure TRAIN($D, \theta, \alpha, \gamma, sk$)
    $M \leftarrow \{\}$
    for $t = 1, ..., T$ do
        for $(x, y)$ in $D_t$ do
            // Draw batch from buffer:
            $B \leftarrow sample(x, y, s, k, M)$
            $\theta_0 \leftarrow \theta$
            for $i = 1, ..., sk$ do
                $x_c, y_c \leftarrow B_i[j]$
                $\theta_i \leftarrow SGD(x_c, y_c, \theta_{i-1}, \alpha)$
            end for
            // Reptile meta-update:
            $\theta \leftarrow \theta_0 + \gamma(\theta_{sk} - \theta_0)$
            // Reservoir sampling memory update:
            $M \leftarrow M \cup \{(x, y)\}$ (algorithm 2)
        end for
    end for
    return $\theta, M$
end procedure
```

Algorithm 6 Meta-Experience Replay (MER) - Current Example Learning Rate

```
procedure TRAIN($D, \theta, \alpha, \gamma, s, k$)
    $M \leftarrow \{\}$
    for $t = 1, ..., T$ do
        for $(x, y)$ in $D_t$ do
            // Draw batch from buffer:
            $B \leftarrow sample(k-1, M)$
            $\theta_0 \leftarrow \theta$
            // SGD on individual samples from batch:
            for $i = 1, ..., k-1$ do
                $x_c, y_c \leftarrow B_i[j]$
                $\theta_i \leftarrow SGD(x_c, y_c, \theta_{i-1}, \alpha)$
            end for
            // High learning rate SGD on current example:
            $\theta_k \leftarrow SGD(x, y, \theta_{k-1}, s\alpha)$
            // Reptile meta-update:
            $\theta \leftarrow \theta_0 + \gamma(\theta_k - \theta_0)$
            // Reservoir sampling memory update:
            $M \leftarrow M \cup \{(x, y)\}$ (algorithm 2)
        end for
    end for
    return $\theta, M$
end procedure
```

J Deriving the Effective Objective of MER

We would like to derive what objective Meta-Experience Replay (algorithm 4) approximates and show that it is approximately the same objective from algorithms 5 and 6. We follow conventions from [32] and first demonstrate what happens to the effective gradients computed by the algorithm in the most trivial case. As in [32], this allows us to extrapolate an effective gradient that is a function of the number of steps taken. We can then consider the effective loss function that results in this gradient. Before we begin, let us define the following terms from [32]:

$$g_i = \frac{\partial L(\theta_i)}{\partial \theta_i} \text{ (gradient obtained during SGD)} \tag{9}$$

$$\theta_{i+1} = \theta_i - \alpha g_i \text{ (sequence of parameter vectors)} \tag{10}$$

$$\bar{g}_i = \frac{\partial L(\theta_i)}{\partial \theta_0} \text{ (gradient at initial point)} \tag{11}$$

$$g_i^j = \frac{\partial L(\theta_i)}{\partial \theta_j} \text{ (gradient evaluated at point i with respect to parameters j)} \tag{12}$$

$$\bar{H}_i = \frac{\partial^2 L(\theta_i)}{\partial \theta_0^2} \text{ (Hessian at initial point)} \tag{13}$$

$$H_i^j = \frac{\partial^2 L(\theta_i)}{\partial \theta_j^2} \text{ (Hessian evaluated at point i with respect to parameters j)} \tag{14}$$

In [32] they consider the effective gradient across one loop of reptile with size $k = 2$. As we have both an outer loop of Reptile applied across batches and an inner loop applied within the batch to consider, we start with a setting where the number of batches $s = 2$ and the number of examples per batch $k = 2$. Let's recall from the original paper that the gradients of Reptile with $k = 2$ was:

$$g_{Reptile,k=2,s=1} = g_0 + g_1 = \bar{g}_0 + \bar{g}_1 - \alpha \bar{H}_1 \bar{g}_0 + O(\alpha^2) \tag{15}$$

So, we can also consider the gradients of Reptile if we had 4 examples in one big batch (algorithm 5) as opposed to 2 batches of 2 examples:

$$\begin{aligned} g_{Reptile,k=4,s=1} &= g_0 + g_1 + g_2 + g_3 \\ = \bar{g}_0 + \bar{g}_1 + \bar{g}_2 + \bar{g}_3 - \alpha \bar{H}_1 \bar{g}_0 - \alpha \bar{H}_2 \bar{g}_0 - \alpha \bar{H}_2 \bar{g}_1 &- \alpha \bar{H}_3 \bar{g}_0 - \alpha \bar{H}_3 \bar{g}_1 - \alpha \bar{H}_3 \bar{g}_2 + O(\alpha^2) \end{aligned} \tag{16}$$

Now we can consider the case for MER where we define the parameter values as follows extending algorithm 4 where A stands for across batches and W stands for within batches:

$$\theta_0 = \theta_0^A = \theta_{00}^W \tag{17}$$

$$\theta_{01}^W = \theta_{00}^W - \alpha g_{00} \tag{18}$$

$$\theta_{02}^W = \theta_{01}^W - \alpha g_{01} \tag{19}$$

$$\theta_1^A = \theta_0^A + \beta \frac{(\theta_{02}^W - \theta_0^A)}{\alpha} = \theta_0 + \beta \frac{(\theta_{02}^W - \theta_0)}{\alpha} = \theta_{10}^W \tag{20}$$

$$\theta_{11}^W = \theta_{10}^W - \alpha g_{10} \tag{21}$$

$$\theta_{12}^W = \theta_{11}^W - \alpha g_{11} \tag{22}$$

$$\theta_2^A = \theta_1^A + \beta \frac{(\theta_{12}^W - \theta_1^A)}{\alpha} \tag{23}$$

$$\theta = \theta_0^A + \gamma \beta \frac{(\theta_2^A - \theta_0^A)}{\beta} = \theta_0^A + \gamma(\theta_2^A - \theta_0^A) \tag{24}$$

g_{MER} the gradient of Meta-Experience Replay can thus be defined analogously to the gradient of Reptile as:

$$g_{MER} = \frac{\theta_0^A - \theta_2^A}{\beta} = \frac{\theta_0 - \theta_2^A}{\beta} \tag{25}$$

By simply applying Reptile from equation 15 we can derive the value of the parameters θ_1^A after updating with Reptile within the first batch in terms of the original parameters θ_0:

$$\theta_1^A = \theta_0 - \beta \bar{g}_{00} - \beta \bar{g}_{01} + \beta \alpha \bar{H}_{01} \bar{g}_{00} + O(\beta \alpha^2) \tag{26}$$

By subbing equation 26 into equation 23 we can see that:

$$\theta_2^A = \theta_0 - \beta \bar{g}_{00} - \beta \bar{g}_{01} + \beta \alpha \bar{H}_{01} \bar{g}_{00} - \beta g_{10} - \beta g_{11} + O(\beta \alpha^2) \tag{27}$$

We can express g_{10} in terms of the initial point, by considering a Taylor expansion following the Reptile paper:

$$g_{10} = \bar{g}_{10} + \alpha \bar{H}_{10}(\theta_{10}^W - \theta_0) + O(\alpha^2) \tag{28}$$

Then substituting in for θ_{10}^W we express g_{10} in terms of θ_0:

$$g_{10} = \bar{g}_{10} - \alpha\beta\bar{H}_{10}\bar{g}_{00} - \alpha\beta\bar{H}_{10}\bar{g}_{01} + O(\alpha^2) \tag{29}$$

We can then rewrite g_{11} by taking a Taylor expansions with respect to θ_{10}^W:

$$g_{11} = g_{11}^{10} - \alpha H_{11}^{10} g_{10} + O(\alpha^2) \tag{30}$$

Taking another Taylor expansion we find that we can transform our expression for the Hessian:

$$H_{11}^{10} = \bar{H}_{11} + O(\alpha) \tag{31}$$

We can analogously also transform our expression our expression for g_{11}^{10}:

$$g_{11}^{10} = \bar{g}_{11} + \alpha\bar{H}_{11}(\theta_{10}^W - \theta_0) + O(\alpha^2) \tag{32}$$

Substituting for θ_{10}^W in terms of θ_0

$$g_{11}^{10} = \bar{g}_{11} - \alpha\beta\bar{H}_{11}\bar{g}_{00} - \alpha\beta\bar{H}_{11}\bar{g}_{01} + O(\alpha^2) \tag{33}$$

We then substitute equation 31, equation 33, and equation 29 into equation 34:

$$g_{11} = \bar{g}_{11} - \alpha\beta\bar{H}_{11}\bar{g}_{00} - \alpha\beta\bar{H}_{11}\bar{g}_{01} - \alpha\bar{H}_{11}\bar{g}_{10} + O(\alpha^2) \tag{34}$$

Finally, we have all of the terms we need to express θ_2^A and we can then derive an expression for the MER gradient g_{MER}:

$$\begin{aligned} g_{MER} &= \bar{g}_{00} + \bar{g}_{01} + \bar{g}_{10} + \bar{g}_{11} \\ -\alpha\bar{H}_{01}\bar{g}_{00} - \alpha\bar{H}_{11}\bar{g}_{10} - \alpha\beta\bar{H}_{10}\bar{g}_{00} - \alpha\beta\bar{H}_{10}\bar{g}_{01} &- \alpha\beta\bar{H}_{11}\bar{g}_{00} - \alpha\beta\bar{H}_{11}\bar{g}_{01} + O(\alpha^2) \end{aligned} \tag{35}$$

This equation is quite interesting and very similar to equation 16. As we would like to approximate the same objective, we can remove one hyperparameter from our model by setting $\beta = 1$. This yields:

$$\begin{aligned} g_{MER} &= \bar{g}_{00} + \bar{g}_{01} + \bar{g}_{10} + \bar{g}_{11} \\ -\alpha\bar{H}_{01}\bar{g}_{00} - \alpha\bar{H}_{11}\bar{g}_{10} - \alpha\bar{H}_{10}\bar{g}_{00} - \alpha\bar{H}_{10}\bar{g}_{01} &- \alpha\bar{H}_{11}\bar{g}_{00} - \alpha\bar{H}_{11}\bar{g}_{01} + O(\alpha^2) \end{aligned} \tag{36}$$

Indeed, with β set to equal 1, we have shown that the gradient of MER is the same as one loop of Reptile with a number of steps equal to the total number of examples in all batches of MER (algorithm 5) if the current example is mixed in with the same proportion. If we include in the current example for s of sk examples in our meta-replay batch, it gets the same overall priority in both cases which is s times larger than that of a random example drawn from the buffer. As such, we can also optimize an equivalent gradient using algorithm 6 because it uses a factor s to increase the priority of the gradient given to the current example.

While $\beta = 1$ is an interesting special case of MER in algorithm 4, in general we find it can be useful to set β to be a value smaller than 1. In fact, in our experiments we consider the case when β is smaller than 1 and $\gamma = 1$. The success of this approach makes sense because the higher order terms in the Taylor expansion that reflect the mismatch between parameters across replay batches creates variance in the learning process. By setting β to a value below 1 we can reduce our comparative weighting on promoting inter batch gradient similarities rather than intra batch gradient similarities.

It was noted in the Reptile paper that the following equality holds if the examples and order are random:

$$\mathbb{E}[\bar{H}_1\bar{g}_0] = \mathbb{E}[\bar{H}_0\bar{g}_1] = \frac{1}{2}\mathbb{E}[\frac{\partial}{\partial\theta_0}(\bar{g}_0 \cdot \bar{g}_1)] \tag{37}$$

Model	Buffer	MNIST Rotations			MNIST Permutations		
		RA	LA	BTI	RA	LA	BTI
GEM	5120	87.12 ± 0.44	86.34 ± 0.09	+0.80 ± 0.42	82.50 ± 0.42	80.30 ± 0.80	**+2.20** ± 0.57
	500	72.08 ± 1.29	82.92 ± 0.49	-10.82 ± 1.22	69.26 ± 0.66	80.28 ± 0.52	-11.02 ± 0.71
	200	66.88 ± 0.72	**85.46** ± 0.38	-18.58 ± 0.48	55.42 ± 1.10	79.84 ± 1.01	-24.42 ± 1.10
MER	5120	**89.56** ± 0.11	**87.62** ± 0.16	**+1.94** ± 0.18	**85.50** ± 0.16	**86.36** ± 0.21	-0.86 ± 0.21
	500	**81.82** ± 0.52	**87.28** ± 0.50	**-5.46** ± 0.13	**77.40** ± 0.38	**83.64** ± 0.18	**-6.24** ± 0.34
	200	**77.24** ± 0.47	84.82 ± 0.25	**-7.58** ± 0.26	**72.74** ± 0.46	**82.18** ± 0.23	**-9.44** ± 0.34

Table 4: Performance varying the buffer size on continual learning benchmarks [25].

In our work to make sure this equality holds in an online setting, we must take multiple precautions as noted in the main text. The issue is that examples are received in a non-stationary sequence so when applied in a continual learning setting the order is not totally random or arbitrary as in the original Reptile work. We address this by maintaining our buffer using reservoir sampling, which ensures that any example seen before has a probability $\frac{1}{N}$ of being a particular element in the buffer. We also randomly select over these elements to form a batch. As this makes the order largely arbitrary to the extent that our buffer includes all examples seen, we are approximating the random offline setting from the original Reptile paper. As such we can view the gradients in equation 16 and equation 36 as leading to approximately the following objective function:

$$\theta = arg\min_{\theta} \mathbb{E}_{(x_{11},y_{11}),\ldots,(x_{sk},y_{sk})\in M}[2\sum_{i=1}^{s}\sum_{j=1}^{k}[L(x_{ij},y_{ij}) - \sum_{q=1}^{i-1}\sum_{r=1}^{j-1}\alpha \frac{\partial L(x_{ij},y_{ij})}{\partial \theta}\cdot\frac{\partial L(x_{qr},y_{qr})}{\partial \theta}]]. \quad (38)$$

This is precisely equation 6 in the main text.

K Additional Experiments

Question *How do the performance gains from MER vary as a function of the buffer size?*

To make progress towards the greater goals of lifelong learning, we would like our algorithm to make the most use of even a modest buffer. This is because in extremely large scale settings it is unrealistic to assume a system can store a large percentage of previous examples in memory. As such, we would like to compare MER to GEM, which is known to perform well with an extremely small memory buffer [25]. We consider a buffer size of 500, that is over 10 times smaller than the standard setting on these benchmarks. Additionally, we also consider a buffer size of 200, matching the smallest setting explored in [25]. This setting corresponds to an average storage of 1 example for each combination of task and class. We report our results in Table 4. The benefits of MER seem to grow as the buffer becomes smaller. In the smallest setting, MER provides more than a 10% boost in retained accuracy on both benchmarks.

K.1 Further Analysis of the Approach

In this section we would like to dive deeper into how MER works. To do so we run additional detailed experiments across our three MNIST based continual learning benchmarks.

Question *What components of MER are most important?*

We would like to further analyze our MER model to understand what components add the most value and when. We want to understand how powerful our proposed variant of ER is on its own and how much is added by adding meta-learning to ER. In Table 5 we provide detailed results considering ablated baselines for our experiments on the MNIST lifelong learning benchmarks. Our version of ER consistently provides gains over GEM on its own, but the techniques perform very comparably when we also maintain GEM's buffer with reservoir sampling our use ER with a GEM style buffer. Additionally, we see that adding meta-learning to ER consistently results in performance gains. In fact, meta-learning appears to provide increasing value for smaller buffers.

Model	Buffer Size	Rotations	Permutations	Many Permutations
ER with SGD (algorithm 3)	5120	88.30 ± 0.57	83.90 ± 0.21	59.78 ± 0.22
	500	76.58 ± 0.89	74.02 ± 0.33	42.36 ± 0.42
	200	70.32 ± 0.86	67.62 ± 0.27	–
MER (algorithm 4)	5120	**89.56** ± 0.11	**85.50** ± 0.16	61.84 ± 0.25
	500	**81.82** ± 0.52	77.40 ± 0.38	**47.40** ± 0.35
	200	**77.24** ± 0.47	72.74 ± 0.46	-
MER (algorithm 5)	5120	88.94 ± 0.17	84.70 ± 0.17	60.78 ± 0.22
	500	80.00 ± 0.22	75.54 ± 0.87	44.28 ± 0.49
	200	73.08 ± 0.64	69.40 ± 0.85	-
MER (algorithm 6)	5120	89.38 ± 0.16	**85.64** ± 0.23	**63.04** ± 0.30
	500	**82.40** ± 0.64	**78.20** ± 0.57	46.68 ± 0.13
	200	**77.26** ± 1.19	**73.22** ± 0.37	-
ER with Adam [17]	5120	88.68 ± 0.29	83.78 ± 0.19	58.82 ± 0.31
	500	77.84 ± 0.97	72.14 ± 1.01	41.10 ± 0.24
	200	69.48 ± 1.21	63.52 ± 0.96	–
ER with RMSProp [14]	5120	88.28 ± 0.16	82.84 ± 0.50	59.00 ± 0.32
	500	76.32 ± 1.34	67.80 ± 0.80	36.68 ± 0.51
	200	66.66 ± 0.71	55.00 ± 0.79	–
ER with GEM style buffer	5120	86.78 ± 0.37	81.18 ± 0.28	54.56 ± 0.59
	500	74.26 ± 0.81	70.04 ± 0.48	38.12 ± 0.64
	200	66.02 ± 0.55	62.98 ± 0.69	-
GEM [25]	5120	87.12 ± 0.44	82.50 ± 0.42	56.76 ± 0.29
	500	72.08 ± 1.29	69.26 ± 0.66	32.14 ± 0.50
	200	66.88 ± 0.72	55.42 ± 1.10	-
GEM with Reservoir Sampling	5120	87.16 ± 0.41	83.68 ± 0.40	58.94 ± 0.53
	500	77.26 ± 2.09	74.82 ± 0.29	42.24 ± 0.48
	200	69.00 ± 0.84	68.90 ± 0.71	-

Table 5: Retained accuracy ablation experiments on MNIST based learning lifelong learning benchmarks.

We would also like to compare the variants of MER proposed in algorithms 4, 5, and 6. Conceptually algorithms 4 and 6 represent different mechanisms of increasing the importance of the current example in algorithm 5. We find that all variants of MER result in significant improvements on ER in all settings. Meanwhile, the variants that increase the importance of the current example see a further improvement in performance, performing quite comparably to each other. Overall, in our MNIST experiments algorithm 6 displays the best tradeoff of computational efficiency and performance. Finally, we conducted experiments demonstrating that adaptive optimizers like Adam and RMSProp can not account for the gap between ER and MER. Particularly for smaller buffer sizes, these approaches overfit more on the buffer and actually hurt generalization in comparison to SGD.

L Hyperparameter Search

Here we report the hyper-parameter grids that we searched over in our experiments. The best values for the MNIST Rotations (ROT) at each buffer size (ROT-5120, ROT-500, ROT-200), MNIST Permutations (PERM) at each buffer size (PERM-5120, PERM-500, PERM-200), Many Permutations (MANY) at each buffer size (MANY-5120, MANY-500), and Omniglot (OMNI) at each buffer size (OMNI-5120, OMNI-500) are noted accordingly in parenthesis.

- Online Learning
  - learning rate: [0.0001, 0.0003, 0.001 (PERM, ROT), 0.003 (MANY), 0.01, 0.03, 0.1 (OMNI)]
- EWC
  - learning rate: [0.001 (OMNI), 0.003 (MANY), 0.01, 0.03 (ROT, PERM), 0.1, 0.3, 1.0]
  - regularization: [1, 3 (MANY), 10 (OMNI), 30, 100, 300 (ROT, PERM), 1000, 3000, 10000, 30000]

- GEM
  - learning rate: [0.001, 0.003 (MANY-500), 0.01 (ROT, PERM, OMNI, MANY-5120), 0.03, 0.1, 0.3, 1.0]
  - memory strength (γ): [0.0 (PERM-500, MANY-500), 0.1 (PERM-200, MANY-200), 0.5 (OMNI), 1.0 (ROT-5120, ROT-500, ROT-200, PERM-5120)]
- Experience Replay
  - learning rate: [0.00003, 0.0001, 0.0003, 0.001, 0.003 (MANY), 0.01 (ROT, PERM), 0.03, 0.1]
  - batch size (k-1): [5, 10 (ROT-500, PERM-200), 25 (ROT-5120, PERM-5120, PERM-500), 50 (MANY-5120, ROT-200), 100, 250]
- Meta-Experience Replay (Algorithm 4)
  - learning rate (α): [0.01 (OMNI-5120), 0.03 (ROT, PERM, MANY), 0.1 (OMNI-500)]
  - across batch meta-learning rate (γ): 1.0
  - within batch meta-learning rate (β): [0.03 (ROT-5120, ROT-200, PERM-5120, PERM-200, MANY), 0.1 (ROT-500, PERM-500), 0.3, 1.0 (OMNI)]
  - batch size (k-1): [5 (MANY-500, OMNI-500), 10, 25 (PERM-500, PERM-200, OMNI-5120), 50 (PERM-5120, MANY-5120), 100 (ROT)]
  - number of batches per example: [1, 2 (PERM-500, OMNI-500), 5 (ROT, PERM-200, OMNI-5120, MANY-5120), 10 (PERM-5120, MANY-500)]
- Meta-Experience Replay (Algorithm 5)
  - learning rate (α): [0.01 (PERM-500), 0.03 (ROT, PERM-5120, PERM-200, MANY), 0.1]
  - meta-learning rate (γ): [0.03 (ROT-200), 0.1 (ROT-5120, ROT-500, PERM-5120, MANY), 0.3 (PERM-200), 0.6 (PERM-500), 1.0]
  - batch size (k-1): [5, 10, 25 (MANY), 50 (ROT-200, PERM-500), 100 (ROT-5120, PERM-200), 250 (ROT-500)]
  - number of batches per example: 1
- Meta-Experience Replay (Algorithm 6)
  - learning rate (α): [0.01 (PERM-5120, PERM-500), 0.03 (ROT, PERM-200, MANY), 0.1]
  - within batch meta-learning rate (γ): [0.03 (ROT, MANY-5120), 0.1 (PERM, MANY-500), 0.3, 1.0]
  - batch size (k-1): [5 (PERM-200, MANY-500), 10, 25 (PERM-500), 50 (ROT-200, ROT-500, MANY-5120), 100 (ROT-5120, PERM-5120)]
  - current example learning rate multiplier (s): [1, 2 (PERM-200, MANY-500), 5 (ROT-500, ROT-200), 10 (ROT-5120, PERM-5120, PERM-500, MANY-5120)]

M Forward Transfer and Interference

Forward transfer was a metric defined in [25] based on the average increased performance on a task relative to performance at random initialization before training on that task. Unfortunately, this metric does not make much sense for tasks like MNIST Permutations where inputs are totally uncorrelated across tasks or Omniglot where outputs are totally uncorrelated across tasks. As such, we only provide performance for this metric on MNIST Rotations in Table 6.

Model	FTI
Online	58.22 ± 2.03
Task Input	1.62 ± 0.87
EWC	58.26 ± 1.98
GEM	**65.96** ± 1.67
MER	**66.74** ± 1.41

Table 6: Forward transfer and interference (FTI) experiments on MNIST Rotations.

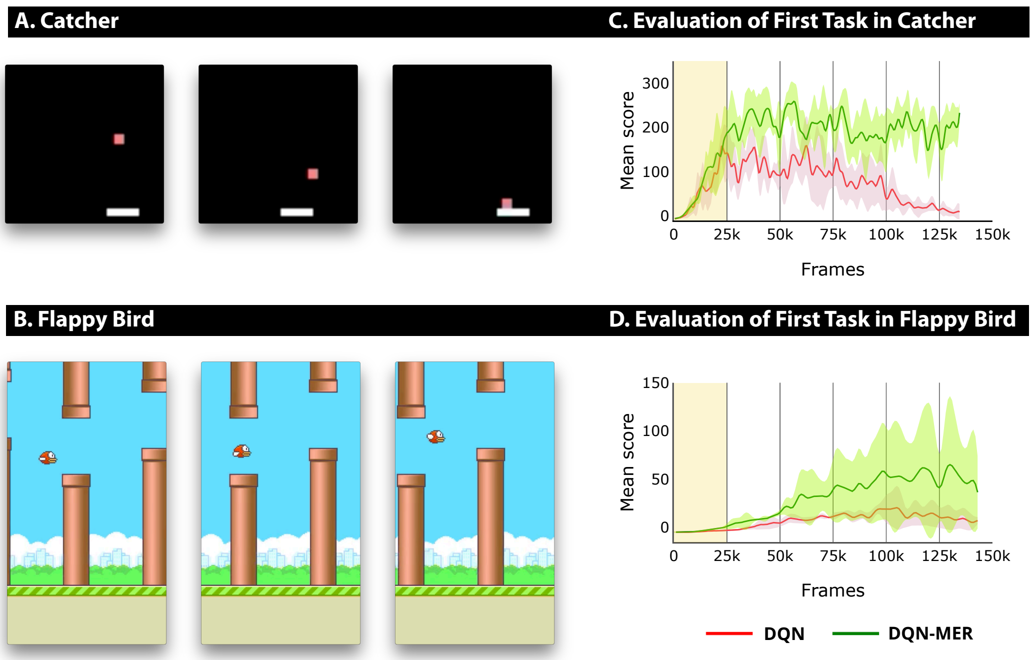

Figure 3: *Right, above and below*: sequence of frames for the game Catcher and Flappy Bird respectively. The goal in Catcher is to capture the falling pellet by horizontally moving the racket on the bottom of the screen. Non-stationarity is introduced in Catcher by changing the velocity with which the pellet falls from the top of the screen to the bottom. In Flappy Bird, the goal is to navigate the bird through as many pipes as possible by making it go up or leave it fall. We change the gap between upper and lower pipes, which is decreased when the task switches. *Left, above and below*: average score in Catcher and Flappy Bird respectively for the evaluation on the first task, corresponding to slower falling pellet in Catcher and a larger gap in Flappy Bird.

N Continual Reinforcement Learning

N.1 Evaluation for Continual Reinforcement Learning

Question *Can MER improve a DQN with ER in continual reinforcement learning settings?*

We considered the evaluation of MER in a continual reinforcement learning setting where the environment is highly non-stationary. In order to produce these non-stationary environments in a controlled way suitable for our experimental purposes, we utilized different arcade games provided by [48]. In our experiments we used Catcher and Flappy Bird, two simple but interesting enough environments.

We detail the application of MER to deep Q-learning in algorithm 7, using notation from [30].

Algorithm 7 Deep Q-learning with Meta-Experience Replay (MER)

```
procedure DQN-MER($env, frameLimit, \theta, \alpha, \beta, \gamma, steps, k, E_Q$)
    // Initialize action-value function $Q$ with parameters $\theta$:
    $Q \leftarrow Q(\theta)$
    // Initialize action-value function $\hat{Q}$ with the same parameters $\hat{\theta} = \theta$:
    $\hat{Q} \leftarrow \hat{Q}(\hat{\theta}) = \hat{Q}(\theta)$
    // Initialize experience replay buffer:
    $M \leftarrow \{\}$
    $M.age \leftarrow 0$
    while $M.age \leq frameLimit$ do
        // Begin new episode:
        $env.reset()$
        // Initialize the s state with the initial observation:
        while episode not done do
            // Select with probability p an action a from set of possible actions:
            $a = \begin{cases} \text{random selection of action } \hat{a} & p \leq \varepsilon \\ \arg\max_a Q(s_t, a'; \theta) & p > \varepsilon \end{cases}$
            // Perform the action a in the environment:
            $s', r_t \leftarrow env.step(s, a)$
            // Store current transition with reward r:
            $M \leftarrow M \cup \{(s, a, r, s')\}$ (algorithm 2)
            $B_1, ..., B_{steps} \leftarrow sample(s, a, r, s', steps, k, M)$
            // Store current weights:
            $\theta_0^A \leftarrow \theta$
            for $i = 1, ..., steps$ do
                $\theta_{i,0}^W \leftarrow \theta$
                for $j = 1, ..., k$ do
                    // Sample one set of processed sequences, actions, and rewards from M:
                    $s, a, r, s' = B_i[j]$
                    $y = \begin{cases} r & \text{if final frame in episode} \\ r + \Gamma \max_a \hat{Q}(s', a; \hat{\theta}) & \text{otherwise} \end{cases}$
                    // Optimize the Huber loss $H(y, Q(s, a; \theta_{i,j-1}^W))$:
                    $L \leftarrow H(y, Q(s, a; \theta_{i,j-1}^W))$
                    $\theta_{i,j}^W \leftarrow \theta_{i,j-1}^W - \alpha \frac{\partial L}{\partial \theta_{i,j-1}^W}$
                end for
                // Within batch Reptile meta-update:
                $\theta \leftarrow \theta_{i,0}^W + \beta(\theta_{i,k}^W - \theta_{i,0}^W)$
                $\theta_i^A \leftarrow \theta$
            end for
            // Across batch Reptile meta-update:
            $\theta \leftarrow \theta_0^A + \gamma(\theta_{steps}^A - \theta_0^A)$
            // Reset target action-value network $\hat{Q}$ to $Q$ every $E_Q$ number of episodes:
            $\hat{Q} = Q$
        end while
    end while
    return $\theta, M$
end procedure
```

N.2 Description of Catcher and Flappy Bird

In Catcher, the agent controls a segment that lies horizontally in the bottom of the screen, i.e. a basket, and can move right or left, or stay still. The goal is to move the basket to catch as many pellets as possible. Missing a pellet results in losing one of the three available lives. Pellets emerge one by one from the top of the screen, and have a descending velocity that is fixed for each task.

The reward and thus y axis in our Catcher experiments refers to the number of fruits caught during the full game span.

In the case of the very popular game Flappy Bird, the agent has to navigate a bird in an environment full of pipes by deciding whether to flap or not flap its wings. The pipes appear always in pairs, one from the bottom of the screen and one from the top of the screen, and have a gap that allows the bird to pass through them. Flapping the wings results in the bird ascending, otherwise the bird descends to ground naturally. Both ascending and descending velocities are presets by the physics engine of the game. The goal is to pass through many pairs of pipes as possible without hitting a pipe, as this results in losing the game. The scoring scheme in this game awards a point each time a pipe is crossed. Despite very simple mechanics, Flappy Bird has proven to be challenging for many humans. According to the original game scoring scheme, players with a score of 10 receive a Bronze medal; with 20 points, a Silver medal; 30 results in a Gold medal, and any score better than 40 is rewarded with a Platinum medal.

N.3 DQN with Meta-Experience Replay

The DQN used to train on both games follows the classic architecture from [30]: it has a CNN consisting of 3 layers, the first with 32 filters and an 8x8 kernel, the second layer with 64 filters and a 4x4 kernel, and a final layer with 64 filters and a 3x3 kernel. The CNN is followed by two fully connected layers. A ReLU non-linearity was applied after each layer. We limited the memory buffer size for our models to 50k transitions, which is roughly the proportion of the total memories used in the benchmark setting for our supervised learning tasks.

N.4 Parameters for Continual Reinforcement Learning Experiments

For the continual reinforcement learning setting we set the parameters using results from the experiments in the supervised setting as a guidance. Both Catcher and Flappy Bird used the same hyper parameters as detailed below with the obvious exception of the game-dependent parameter that defines each task. Models were trained with a maximum number of frames of 150k and 6 total tasks, switching every 25k frames. Runs used different random seeds for the initialization as stated in the figures.

- Game Parameters
  - Catcher: Δ: 0.03 (vertical velocity of pellet increased from default 0.608).
  - Flappy Bird: Δ: -5 (pipe gap decreased 5 from default 100).
- Experience Replay
  - learning rate: 0.0001
  - batch size (k-1): 16
- Meta-Experience Replay
  - learning rate (α): 0.0001
  - within batch meta-learning rate (β): 1
  - across batch meta-learning rate (γ): 0.3
  - batch size (k-1): 16
  - number of steps: 1
  - buffer size: 50000

N.5 Continual Reinforcement Learning Evaluation for Flappy Bird

Performance during training in continual learning for a non-stationary version of Flappy Bird is shown in Figure (4). Graphsshow averaged values over three validation episodes across three different seed initializations. Vertical grid lineson the frame axis indicate task switch

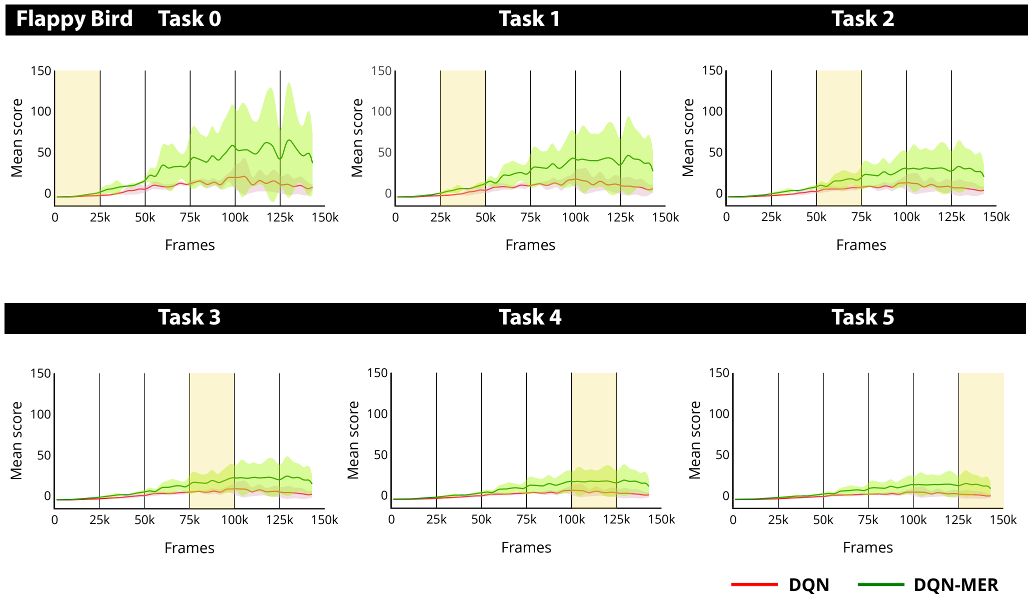

Figure 4: Continual learning for a non-stationary version of Flappy Bird.