Three continual learning scenarios

Gido M. van de Ven[1*] **& Andreas S. Tolias**[1,2]
[1] Center for Neuroscience and Artificial Intelligence, Baylor College of Medicine, Houston
[2] Department of Electrical and Computer Engineering, Rice University, Houston
{ven,astolias}@bcm.edu

Abstract

Standard artificial neural networks suffer from the well-known issue of catastrophic forgetting, making continual or lifelong learning problematic. Recently, numerous methods have been proposed for continual learning, but due to differences in evaluation protocols it is difficult to directly compare their performance. To enable more meaningful comparisons, we identified three distinct continual learning scenarios based on whether task identity is known and, if it is not, whether it needs to be inferred. Performing the split and permuted MNIST task protocols according to each of these scenarios, we found that regularization-based approaches (e.g., elastic weight consolidation) fail when task identity needs to be inferred. In contrast, generative replay combined with distillation (i.e., using class probabilities as "soft targets") achieves superior performance in all three scenarios.

1 Introduction

When trained on a new task, deep neural networks tend to forget most information related to previously learned tasks, a phenomenon referred to as "catastrophic forgetting". In recent years, numerous methods for alleviating catastrophic forgetting have been proposed. However, due to the wide variety of experimental protocols used to evaluate them, many of these methods claim "state-of-the-art" performance [1, 2, 3, 4, 5, 6]. To obscure things further, some methods shown to perform well in some experimental settings are reported to dramatically fail in others: compare the performance of elastic weight consolidation in [1] and [7] with that in [8] and [9].

To enable a fairer and more structured comparison of methods for reducing catastrophic forgetting, this paper identifies three distinct continual learning scenarios of increasing difficulty. Using these scenarios, we then perform a structured comparison of recently proposed methods for continual learning. We find that the here identified scenarios can explain seemingly contradictory results from the recent literature: even for experimental protocols involving the relatively simple classification of MNIST-digits, methods that perform well in one scenario can completely fail in another.

2 Continual learning scenarios

We consider the continual learning problem in which a single model needs to sequentially learn a series of tasks, whereby it is not allowed to store raw data. This continual learning framework has been actively studied in recent years: many methods for alleviating catastrophic forgetting are being proposed, with almost as many different experimental protocols being used for their evaluation. We found that an important difference between these experimental protocols is whether at test time information about the task identity is available and—if it is not—whether the model is required to identify the identity of the task it has to solve. Yet, this crucial experimental design consideration is not always clearly stated and differences in this regard are sometimes not appreciated. For example,

[*] Alternative email address: gidovandeven@gmail.com

Continual learning workshop (NIPS 2018). An extended version of this work has been submitted to ICLR 2019.

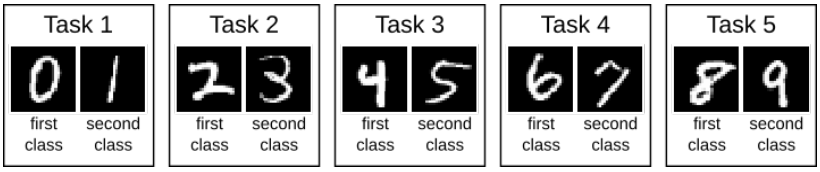

Figure 1: Schematic of the split MNIST task protocol.

Table 1: The split MNIST task protocol according to each continual learning scenario.

Incremental task learning	With task given, is it the first or second class? (*e.g.*, '0' or '1')
Incremental domain learning	With task unknown, is it a first or second class? (*e.g.*, in ['0','2','4','6','8'] or in ['1','3','5','7','9'])
Incremental class learning	With task unknown, which digit is it? (choice from '0' to '9')

in [4] a substantial improvement over state-of-the-art is reported, while their method assumes task identity is always available and the compared methods operate without this assumption. To enable more meaningful comparisons, we identify three distinct scenarios for continual learning.

In the first scenario, models are always informed about which task needs to be performed. This is the easiest continual learning scenario, and we refer to it as ***incremental task learning***. Since task identity is always provided, it is possible to train models with task-specific components. A typical neural network architecture used in this scenario has a "multihead" output-layer, meaning that each task has its own output units but the rest of the network is (potentially) shared between tasks.

In the second scenario, which we refer to as ***incremental domain learning***, task identity is not available at test time. Models however only need to solve the task at hand; they are not required to infer which task it is. Typical examples of this scenario are protocols whereby the structure of the tasks is always the same, but the input-distribution is changing.

Finally, in the third scenario, models need to be able both to solve each task seen so far and to infer which task they are presented with. We refer to this scenario as ***incremental class learning***, as it includes protocols in which new classes need to be learned incrementally.

Example task protocols To demonstrate the difference between these continual learning scenarios, as well as to compare the performance of several recently proposed methods for continual learning, we use two different task protocols and perform both of them according to all three scenarios. The first task protocol is 'split MNIST' [7], which is sequentially learning to classify MNIST-digits (Figure 1; Appendix A.1). This task protocol is most naturally performed under the incremental class learning scenario, but it has also been performed under the other two scenarios (Table 1). The second task protocol is 'permuted MNIST' [10], in which each task involves classifying all ten MNIST-digits but with a different permutation applied to the pixels for every new task (Figure 2; Appendix A.1). Again, although permuted MNIST is most naturally performed according to the incremental domain learning scenario, it can be performed according to the other scenarios too (Table 2).

3 Compared methods

Based on the insight that the reason for catastrophic forgetting is that after a neural network is trained on a new task its parameters are optimized for the new task and no longer for the previous one(s), methods for continual learning can be divided in two categories: (1) those that restrict the network's optimization when learning a new task, and (2) those that modify the training data to make it more representative for previous tasks.

3.1 Not optimizing entire network / regularized optimization

A straightforward way of not optimizing the full network on every task is to explicitly define a different sub-network for each task. Several recent papers use this strategy, with different approaches for selecting the parts of the network for each task. A simple approach is to randomly and *a priori* assign

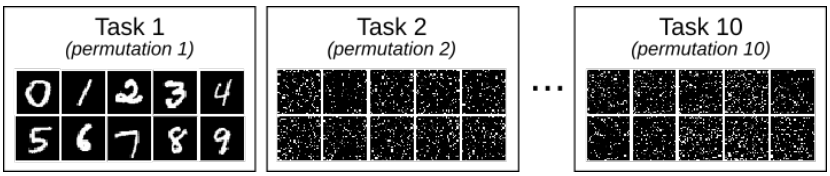

Figure 2: Schematic of the permuted MNIST task protocol.

Table 2: The permuted MNIST task protocol according to each continual learning scenario.

Incremental task learning	Given permutation *X* was applied, which digit is it?
Incremental domain learning	With permutation unknown, which digit is it?
Incremental class learning	Which digit is it *and* which permutation was applied?

which nodes will participate in each task (*Context-dependent Gating* [**XdG**; 4]). Other approaches use evolutionary algorithms [11] or gradient descent [12] to learn which sets of units to employ for each task. By design, however, these approaches are limited to the incremental task learning scenario, as they require knowledge of task identity to select the correct task-specific components.

A modification to make this strategy applicable in the other scenarios is to preferentially train a different part of the network for each task, but to always use the entire network for execution. One way to do this is by differently regularizing the network's parameters during training on each new task, which is the approach of *Elastic Weight Consolidation* [**EWC**, **Online EWC**; 1, 13] and *Synaptic Intelligence* [**SI**; 7]. Both methods estimate for all parameters of the network how important they are for the previously learned tasks and penalize future changes to them accordingly (i.e., learning is slowed down for parts of the network important for previous tasks).

3.2 Modifying training data

A second strategy is to complement the training data for each new task to be learned with "pseudo-data" representative of the previous tasks, which we refer to as replay. One option is to take the input data of the current task, label them based on the predictions of the model trained on the previous tasks, and use the resulting input-target pairs as pseudo-data. This is the approach of *Learning without Forgetting* [**LwF**; 14]. Another important aspect of this method is that instead of labeling the inputs to be replayed as the most likely category according to the previous tasks' model (i.e., "hard targets"), it pairs them with the by that model predicted probabilities for *all* target classes (i.e., "soft targets"). The approach of matching predicted probabilities of one network to those of another network had previously been used to compress (or "distill") information from one network to another [15].

Another option is to generate the input data to be replayed. For this, besides the main model for task performance (e.g., classification), a separate generative model is sequentially trained on all tasks to generate samples from their input data distributions. For the first application of this approach, which was called *Deep Generative Replay* [**DGR**], the generated input samples were paired with "hard targets" provided by the main model [16]. We note that it is possible to combine LwF and DGR by replaying input samples from a generative model and pairing them with soft targets [see also 6, 17]. We include this hybrid method in our comparison under the name **DGR+distill**.

4 Results

We comprehensively compared the above discussed methods, by performing both the split and the permuted MNIST task protocol according to each of the three continual learning scenarios (Tables 3 and 4; see Appendix A for implementation details). An important finding is that for the incremental class learning scenario (i.e., when task identity needs to be inferred as well), the regularization-based methods EWC, Online EWC and SI completely failed on both protocols. Only the replay-based methods DGR and DGR+distill produced good results in this scenario. Also striking is that for the split MNIST protocol, replaying images from the current task (LwF; e.g., replaying '2's and '3's in order not to forget how to recognize '0's and '1's), prevented the forgetting of previous tasks better

Table 3: Average test accuracy (over all tasks) on the split MNIST task protocol. Each experiment was performed 20 times with different random seeds, reported is the mean (± SEM) over these runs.

Method	**Incremental task learning**	**Incremental domain learning**	**Incremental class learning**
None – *lower bound*	85.15 (± 1.00)	57.33 (± 1.66)	19.90 (± 0.02)
XdG	98.74 (± 0.31)	-	-
EWC	85.48 (± 1.20)	57.80 (± 1.61)	19.90 (± 0.02)
Online EWC	85.22 (± 1.06)	57.60 (± 1.66)	19.90 (± 0.02)
SI	99.14 (± 0.11)	63.77 (± 1.18)	20.04 (± 0.08)
LwF	99.60 (± 0.03)	71.02 (± 1.26)	24.17 (± 0.51)
DGR	99.47 (± 0.03)	95.74 (± 0.23)	91.24 (± 0.33)
DGR+distill	99.59 (± 0.03)	96.94 (± 0.14)	91.84 (± 0.27)
Offline – *upper bound*	99.64 (± 0.03)	98.41 (± 0.06)	97.93 (± 0.04)

Table 4: Idem as Table 3, except on the permuted MNIST task protocol.

Method	**Incremental task learning**	**Incremental domain learning**	**Incremental class learning**
None – *lower bound*	81.79 (± 0.48)	78.51 (± 0.24)	17.26 (± 0.19)
XdG	91.40 (± 0.23)	-	-
EWC	94.79 (± 0.03)	94.43 (± 0.10)	27.65 (± 0.52)
Online EWC	96.09 (± 0.08)	94.25 (± 0.15)	34.41 (± 0.66)
SI	94.75 (± 0.14)	95.33 (± 0.11)	29.31 (± 0.62)
LwF	69.84 (± 0.46)	72.64 (± 0.52)	22.64 (± 0.23)
DGR	92.52 (± 0.08)	95.09 (± 0.04)	92.19 (± 0.09)
DGR+distill	97.51 (± 0.01)	97.35 (± 0.02)	96.38 (± 0.03)
Offline – *upper bound*	97.68 (± 0.01)	97.59 (± 0.01)	97.59 (± 0.02)

than EWC or SI. On the split MNIST protocol, EWC and online EWC actually did not prevent against catastrophic forgetting at all. We hypothesize that this is because the individual tasks of this protocol (distinguishing between 2 digits) are relatively easy, making that after finishing training on each task the gradients—and thus the Fisher Information—rely on very few errors.

5 Discussion

Catastrophic forgetting is a major obstacle to developing artificial intelligence applications capable of true lifelong learning [18, 19], and enabling neural networks to sequentially learn multiple tasks has become a topic of intense research. Despite its scope, this research field lacks common benchmarks (even when the same datasets are used), making direct comparisons between published methods difficult. The three scenarios identified here aim to make the continual learning field more structured. Although they were illustrated here only for supervised learning, these scenarios are also applicable for unsupervised or reinforcement learning.

The most challenging continual learning scenario is incremental class learning. Even for relatively simple task protocols involving the classification of MNIST-digits, regularization-based methods such as EWC and SI completely fail when task identity needs to be inferred as well. Only generative replay is capable of performing well in this scenario. We hypothesize that the difficulty is that a network needs to learn to distinguish classes that it never observes together, and that generative replay overcomes this by enabling a network to instead observe generated examples of all classes together.

A limitation of the current study is that MNIST-images are relatively easy to generate. It therefore remains an open question whether generative replay will still be so successful for task protocols with more complicated input distributions. An alternative / complement to generative replay could be to store examples from previous tasks and replay those. Due to privacy concerns or memory constraints

this is however not always possible, but when it is it can substantially boost performance [2, 3, 5; see Appendix C]. See also [20] for a discussion of the scalability of generative replay.

Acknowledgments

We thank Mengye Ren and Zhe Li for comments on earlier versions. This research project has been supported by an IBRO-ISN Research Fellowship, by the Lifelong Learning Machines (L2M) program of the Defence Advanced Research Projects Agency (DARPA) via contract number HR0011-18-2-0025 and by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, IARPA, DoI/IBC, or the U.S. Government.

References

[1] 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.

[2] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proc. CVPR*, 2017.

[3] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[4] Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *arXiv preprint arXiv:1802.01569*, 2018.

[5] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SJ1Xmf-Rb`.

[6] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, Zhengyou Zhang, and Yun Fu. Incremental classifier learning with generative adversarial networks. *arXiv preprint arXiv:1802.00853*, 2018.

[7] Friedemann Zenke, Ben Poole, and Surya Ganguli. Improved multitask learning through synaptic intelligence. *arXiv preprint arXiv:1703.04200*, 2017.

[8] Ronald Kemker, Angelina Abitino, Marc McClure, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. *arXiv preprint arXiv:1708.02072*, 2017.

[9] Nitin Kamra, Umang Gupta, and Yan Liu. Deep generative dual memory network for continual learning. *arXiv preprint arXiv:1710.10368*, 2017.

[10] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[11] 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.

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

[13] Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

[14] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[16] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2994–3003, 2017.

[17] Ragav Venkatesan, Hemanth Venkateswara, Sethuraman Panchanathan, and Baoxin Li. A strategy for an uncompromising incremental learner. *arXiv preprint arXiv:1705.00744*, 2017.

[18] Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

[19] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

[20] Anonymous. Generative replay with feedback connections as a general strategy for continual learning. In *Submitted to International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ByGVui0ctm`. under review.

[21] Ferenc Huszár. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences*, 115(11):E2496–E2497, 2018.

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

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[24] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[25] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[26] Matthew Riemer, Tim Klinger, Michele Franceschini, and Djallel Bouneffouf. Scalable recollections for continual lifelong learning. *arXiv preprint arXiv:1711.06761*, 2018.

A Additional experimental details

PyTorch-code that can be used to perform the experiments described in this paper is available online: `https://github.com/GMvandeVen/continual-learning`.

A.1 Task protocols

For the split MNIST task protocol [7] (Figure 1), the original MNIST-dataset was split into five tasks, where each task was a two digit classification. The original 28x28 pixel grey-scale images were used without pre-processing. The standard training/test-split was used resulting in 60,000 training images ($\sim$6000 per digit) and 10,000 test images ($\sim$1000 per digit).

For the permuted MNIST task protocol [10] (Figure 2), the tasks were classifying MNIST-digits (every task now had all ten digits), whereby in each task the pixels of the MNIST-images were permutated in a different way. We used a sequence of ten such tasks. To generate the permutated images, the original images were first zero-padded to 32x32 pixels. For each task, a random permutation was then generated and applied to these 1024 pixels. No other pre-processing was performed. Again the standard training/test-split was used.

A.2 Methods

For a fair comparison, the same neural network architecture was used for all methods. For the split MNIST experiments, this was a multi-layer perceptron with 2 hidden layers of 400 nodes each, followed by a softmax output layer. ReLU non-linearities were used in all hidden layers. For the permuted MNIST experiments each hidden layer consisted of 1000 nodes.

All methods used the standard cross entropy classification loss for the model's predictions on the current task ($\mathcal{L}_{\text{current}} = \mathcal{L}_{\text{classification}}$; see Appendix A.3.1). The regularization-based methods (*i.e.*, EWC, online EWC and SI) added a regularization term to this loss, with regularization strength controlled by a hyperparameter: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{current}} + \lambda\mathcal{L}_{\text{regularization}}$. The value of this hyperparameter was set by a grid search, even though it could be argued that this is problematic in the context of continual learning (see Appendix B). The replay-based methods (*i.e.*, LwF, DGR and DGR+distill) instead added a loss-term for the replayed data. In this case a hyperparameter could be avoided, as the loss for the current and replayed data could be weighted according to how many tasks the model has been trained on so far: $\mathcal{L}_{\text{total}} = \frac{1}{N_{\text{tasks so far}}}\mathcal{L}_{\text{current}} + (1 - \frac{1}{N_{\text{tasks so far}}})\mathcal{L}_{\text{replay}}$.

We compared the following approaches:

- **None:** The model was sequentially trained on all tasks in the standard way. This is also called *fine-tuning*, and can be seen as a lower bound.
- **XdG:** Following Masse et al. [4], for each task a random subset of $X\%$ of the units in each hidden layer was fully gated (*i.e.*, their activations set to zero), with X a hyperparameter whose value was set by a grid search (see Appendix B). As this method requires availability of task identity at test time, it could only be used in the incremental task learning scenario.
- **EWC:** The regularization term proposed in Kirkpatrick et al. [1] was added to the loss, see Appendix A.4.1 for implementation details.
- **Online EWC:** This is a modification of EWC proposed by Schwarz et al. [13], with inspiration from Huszár [21], that improves EWC's scalability by ensuring the computational cost of the regularization term does not grow with number of tasks (see Appendix A.4.2).
- **SI:** The regularization proposed in Zenke et al. [7] was added to the loss (see Appendix A.4.3).
- **LwF:** Images of the current task were replayed with soft targets provided by a copy of the model stored after finishing training on the previous task [14] (see Appendix A.3.2).
- **DGR:** A separate generative model was trained to generate the images to be replayed. Following Shin et al. [16], the replayed images were labeled with the most likely category predicted by a copy of the main model stored after training on the previous task (*i.e.*, hard targets).
- **DGR+distill:** A separate generative model was trained to generate the images to be replayed, but these were then paired with soft targets (as in LwF) instead of hard targets (as in DGR).

- Offline: The model was always trained using the data of all tasks so far. This is also called *joint training*, and was included as it can be seen as an upper bound.

For the split MNIST protocol, all models were trained for 2000 iterations per task using the ADAM-optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$; 22) with learning rate 0.001. The same optimizer was used for the permuted MNIST protocol, but with 5000 iterations and learning rate 0.0001. For each iteration, $\mathcal{L}_{\text{current}}$ (and $\mathcal{L}_{\text{regularization}}$) was calculated as average over 128 examples from the current task and—if replay was used—an additional 128 replayed examples (equally divided over all previous tasks) were used to calculate $\mathcal{L}_{\text{replay}}$. Importantly, since the total number of replayed examples does not depend on the number of previous tasks, for our implementation of the replay-based methods the training time per task does not need to increase with number of tasks so far.

For DGR and DGR+distill, a separate generative model was sequentially trained on all tasks. A symmetric variational autoencoder [VAE; 23] was used as generative model, with 2 fully connected hidden layers of 400 (split MNIST) or 1000 (permuted MNIST) units and a stochastic latent variable layer of size 100. A standard normal distribution was used as prior. See Appendix A.3.3 for more details. Training of the generative model was also done with generative replay (provided by its own copy stored after finishing training on the previous task) and with the same hyperparameters (i.e., learning rate, optimizer, iterations, batch sizes) as for the main model.

A.3 Loss functions

A.3.1 Classification

The standard per-sample cross entropy loss function for an input $\boldsymbol{x}$ labeled with a hard target y is given by:

$$\mathcal{L}_{\text{classification}}\left(\boldsymbol{x}, y; \boldsymbol{\theta}\right) = -\log p_{\boldsymbol{\theta}}\left(Y = y|\boldsymbol{x}\right) \tag{1}$$

where $p_{\boldsymbol{\theta}}$ is the conditional probability distribution defined by the neural network whose trainable bias and weight parameters are collected in $\boldsymbol{\theta}$. An important note is that in this paper this probability distribution is not always defined over all output nodes of the network, but only over the "active nodes". This means that the normalization performed by the final softmax layer only takes into account these active nodes, and that learning is thus restricted to those nodes. For experiments performed according to the incremental task learning scenario, for which we use a "multihead" softmax layer, always only the nodes of the task under consideration are active. Typically this is the current task, but for replayed data it is the task that is (intended to be) replayed. For the incremental domain learning scenario always all nodes are active. For the incremental class learning scenario, the nodes of all tasks seen so far are active, both when training on current and on replayed data.

For the method DGR, there are also some subtle differences between the continual learning scenarios when generating hard targets for the inputs to be replayed. With the incremental task learning scenario, only the classes of the task that is intended to be replayed can be predicted (in each iteration the available replays are equally divided over the previous tasks). With the incremental domain learning scenario always all classes can be predicted. With the incremental class learning scenario only classes from up to the previous task can be predicted.

A.3.2 Distillation

The methods LwF and DGR+distill use distillation loss for their replayed data. For this, each input $\boldsymbol{x}$ to be replayed is labeled with a "soft target", which is a vector containing a probability for each active class. This target probability vector is obtained using a copy of the main model stored after finishing training on the most recent task, and the training objective is to match the probabilities predicted by the model being trained to these target probabilities (by minimizing the cross entropy between them). Moreover, as is common for distillation, these two probability distributions that we want to match are made softer by temporary raising the temperature T of their models' softmax layers.[2] This means that before the softmax normalization is performed on the logits, these logits are first divided by T. For an input $\boldsymbol{x}$ to be replayed during training of task K, the soft targets are given by the vector $\tilde{\boldsymbol{y}}$

[2]The same temperature should be used for calculating the target probabilities and for calculating the probabilities to be matched during training; but during testing the temperature should be set back to 1. A typical value for this temperature is 2, which is the value used in this paper.

whose i^{th} element is given by:

$$\tilde{y}_i = p^T_{\hat{\boldsymbol{\theta}}^{(K-1)}}\left(Y = i|\boldsymbol{x}\right) \tag{2}$$

where $\hat{\boldsymbol{\theta}}^{(K-1)}$ is the vector with parameter values at the end of training of task $K-1$ and $p^T_{\boldsymbol{\theta}}$ is the conditional probability distribution defined by the neural network with parameters $\boldsymbol{\theta}$ and with the temperature of its softmax layer raised to T. The distillation loss function for an input $\boldsymbol{x}$ labeled with a soft target vector $\tilde{\boldsymbol{y}}$ is then given by:

$$\mathcal{L}_{\text{distillation}}\left(\boldsymbol{x}, \tilde{\boldsymbol{y}}; \boldsymbol{\theta}\right) = -T^2 \sum_{i=1}^{N_{\text{classes}}} \tilde{y}_i \log p^T_{\boldsymbol{\theta}}\left(Y = i|\boldsymbol{x}\right) \tag{3}$$

where the scaling by T^2 is included to ensure that the relative contribution of this objective matches that of a comparable objective with hard targets [15].

When generating soft targets for the inputs to be replayed, there are again subtle differences between the three continual learning scenarios. With the incremental task learning scenario, the soft target probability distribution is defined only over the classes of the task intended to be replayed. With the incremental domain learning scenario this distribution is always over all classes. With the incremental class learning scenario, the soft target probability distribution is first generated only over the classes from up to the previous task and then zero probabilities are added for all classes in the current task.

A.3.3 Symmetrical VAE

The separate generative model that is used for DGR and DGR+distill is a variational autoencoder [VAE; 23], of which both the encoder network $q_{\boldsymbol{\phi}}$ and the decoder network $p_{\boldsymbol{\psi}}$ are multi-layer perceptrons with 2 hidden layers containing 400 (split MNIST) or 1000 (permuted MNIST) units with ReLU non-linearity. The stochastic latent variable layer $\boldsymbol{z}$ has 100 units and the prior over them is the standard normal distribution. Following Kingma and Welling [23], the "latent variable regularization term" of this VAE is given by:

$$\mathcal{L}_{\text{latent}}(\boldsymbol{x}, \boldsymbol{\phi}) = \frac{1}{2}\sum_{j=1}^{100}\left(1 + \log\left(\left(\sigma_j^{(\boldsymbol{x})}\right)^2\right) - \left(\mu_j^{(\boldsymbol{x})}\right)^2 - \left(\sigma_j^{(\boldsymbol{x})}\right)^2\right) \tag{4}$$

whereby $\mu_j^{(\boldsymbol{x})}$ and $\sigma_j^{(\boldsymbol{x})}$ are the j^{th} elements of respectively $\boldsymbol{\mu}^{(\boldsymbol{x})}$ and $\boldsymbol{\sigma}^{(\boldsymbol{x})}$, which are the outputs of the encoder network $q_{\boldsymbol{\phi}}$ given input $\boldsymbol{x}$. Following Doersch [24], the output layer of the decoder network $p_{\boldsymbol{\psi}}$ has a sigmoid non-linearity and the "reconstruction term" is given by the binary cross entropy between the original and decoded pixel values:

$$\mathcal{L}_{\text{recon}}\left(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{\psi}\right) = \sum_{i=1}^{N_{\text{pixels}}} x_i \log\left(\tilde{x}_i\right) + \left(1 - x_i\right)\log\left(1 - \tilde{x}_i\right) \tag{5}$$

whereby x_i is the value of the i^{th} pixel of the original input image $\boldsymbol{x}$ and $\tilde{x}_i$ is the value of the i^{th} pixel of the decoded image $\tilde{\boldsymbol{x}} = p_{\boldsymbol{\psi}}\left(\boldsymbol{z}^{(\boldsymbol{x})}\right)$ with $\boldsymbol{z}^{(\boldsymbol{x})} = \boldsymbol{\mu}^{(\boldsymbol{x})} + \boldsymbol{\sigma}^{(\boldsymbol{x})} \cdot \boldsymbol{\epsilon}$, whereby $\boldsymbol{\epsilon}$ is sampled from $\mathcal{N}\left(0, \boldsymbol{I}_{100}\right)$. The per-sample VAE loss for an input $\boldsymbol{x}$ is then given by [23]:

$$\mathcal{L}_{\text{generative}}\left(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{\psi}\right) = \mathcal{L}_{\text{recon}}\left(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{\psi}\right) + \mathcal{L}_{\text{latent}}\left(\boldsymbol{x}; \boldsymbol{\phi}\right) \tag{6}$$

A.4 Regularization terms

A.4.1 EWC

The regularization term of elastic weight consolidation [EWC; 1] consists of a quadratic penalty term for each previously learned task, whereby each task's term penalizes the parameters for how different they are compared to their value directly after finishing training on that task. The strength of each parameter's penalty depends for every task on how important that parameter was estimated to be for that task, with higher penalties for more important parameters. For EWC, a parameter's importance is estimated for each task by the parameter's corresponding diagonal element of that task's Fisher Information matrix, evaluated at the optimal parameter values after finishing training on that task. The EWC regularization term for task $K > 1$ is given by:

$$\mathcal{L}^{(K)}_{\text{regularization}_{\text{EWC}}}\left(\boldsymbol{\theta}\right) = \sum_{k=1}^{K-1}\left(\frac{1}{2}\sum_{i=1}^{N_{\text{params}}} F_{ii}^{(k)}\left(\theta_i - \hat{\theta}_i^{(k)}\right)^2\right) \tag{7}$$

whereby $\hat{\theta}_i^{(k)}$ is the i^{th} element of $\hat{\boldsymbol{\theta}}^{(k)}$, which is the vector with parameter values at the end of training of task k, and $F_{ii}^{(k)}$ is the i^{th} diagonal element of $\boldsymbol{F}^{(k)}$, which is the Fisher Information matrix of task k evaluated at $\hat{\boldsymbol{\theta}}^{(k)}$. Following the definitions and notation in Martens [25], the i^{th} diagonal element of $\boldsymbol{F}^{(k)}$ is defined as:

$$F_{ii}^{(k)} = \mathbb{E}_{\boldsymbol{x}\sim Q_{\boldsymbol{x}}^{(k)}}\left[\mathbb{E}_{p_{\boldsymbol{\theta}}(y|\boldsymbol{x})}\left[\left(\frac{\delta \log p_{\boldsymbol{\theta}}\left(Y=y|\boldsymbol{x}\right)}{\delta\theta_i}\right)^2\right]\right]\Bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}^{(k)}} \tag{8}$$

whereby $Q_{\boldsymbol{x}}^{(k)}$ is the (theoretical) input distribution of task k and $p_{\boldsymbol{\theta}}$ is the conditional distribution defined by the neural network with parameters $\boldsymbol{\theta}$. Note that in Kirkpatrick et al. [1] it is not specified exactly how these $F_{ii}^{(k)}$ are calculated (except that it is said to be "easy"); but we have been made aware that they are calculated as the diagonal elements of the "true Fisher Information":

$$F_{ii}^{(k)} = \frac{1}{|S^{(k)}|}\sum_{\boldsymbol{x}\in S^{(k)}}\left(\frac{\delta \log p_{\boldsymbol{\theta}}\left(Y=\hat{y}_{\boldsymbol{x}}^{(k)}|\boldsymbol{x}\right)}{\delta\theta_i}\Bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}^{(k)}}\right)^2 \tag{9}$$

whereby $S^{(k)}$ is the training data of task k and $\hat{y}_{\boldsymbol{x}}^{(k)} = \arg\min_y \log p_{\hat{\boldsymbol{\theta}}^{(k)}}\left(Y=y|\boldsymbol{x}\right)$, the label predicted by the model with parameters $\hat{\boldsymbol{\theta}}^{(k)}$ given $\boldsymbol{x}$.[3] The calculation of the Fisher Information is time-consuming, especially if tasks have a lot of training data. In practice it might therefore sometimes be beneficial to trade accuracy for speed by using only a subset of a task's training data for this calculation (e.g., by introducing another hyperparameter N_{Fisher} that sets the maximum number of samples to be used in equation 9).

A.4.2 Online EWC

A disadvantage of the original formulation of EWC is that the number of quadratic terms in its regularization term grows linearly with the number of tasks. This is an important limitation, as for a method to be applicable in a true lifelong learning setting its computational cost should not increase with the number of tasks seen so far. It was pointed out by Huszár [21] that a slightly stricter adherence to the approximate Bayesian treatment of continual learning, which had been used as motivation for EWC, actually results in only a single quadratic penalty term on the parameters that is anchored at the optimal parameters after the most recent task and with the weight of the parameters' penalties determined by a running sum of the previous tasks' Fisher Information matrices. This insight was adopted by Schwarz et al. [13], who proposed a modification to EWC called *online EWC*. The regularization term of online EWC when training on task $K > 1$ is given by:

$$\mathcal{L}^{(K)}_{\text{regularization}_{\text{oEWC}}} = \sum_{i=1}^{N_{\text{params}}} \tilde{F}_{ii}^{(K-1)}\left(\theta_i - \hat{\theta}_i^{(K-1)}\right)^2 \tag{10}$$

whereby $\hat{\theta}_i^{(K-1)}$ is the value of parameter i after finishing training on task $K-1$ and $\tilde{F}_{ii}^{(K-1)}$ is a running sum of the i^{th} diagonal elements of the Fisher Information matrices of the first $K-1$ tasks, with a hyperparameter $\gamma \leq 1$ that governs a gradual decay of each previous task's contribution. That is: $\tilde{F}_{ii}^{(k)} = \gamma\tilde{F}_{ii}^{(k-1)} + F_{ii}^{(k)}$, with $\tilde{F}_{ii}^{(1)} = F_{ii}^{(1)}$ and $F_{ii}^{(k)}$ is the i^{th} diagonal element of the Fisher Information matrix of task k calculated according to equation 9.

A.4.3 SI

Similar as for online EWC, the regularization term of synaptic intelligence [SI; 7] consists of only one quadratic term that penalizes changes to parameters away from their values after finishing training on

[3] An alternative way to calculate $F_{ii}^{(k)}$ would be, instead of taking for each training input $\boldsymbol{x}$ only the most likely label predicted by model $p_{\hat{\boldsymbol{\theta}}^{(k)}}$, to sample for each $\boldsymbol{x}$ multiple labels from the entire conditional distribution defined by this model (i.e., to approximate the inner expectation of equation 8 for each training sample $\boldsymbol{x}$ with Monte Carlo sampling from $p_{\hat{\boldsymbol{\theta}}^{(k)}}\left(\cdot|\boldsymbol{x}\right)$). Another option is to use the "empirical Fisher Information", by replacing in equation 9 the predicted label $\hat{y}_{\boldsymbol{x}}^{(k)}$ by the observed label y. The results reported in Tables 3 and 4 do not depend much on the choice of how to calculate $F_{ii}^{(k)}$.

the previous task, with the strength of each parameter's penalty depending on how important that parameter is thought to be for the tasks learned so far. To estimate parameters' importance, for every new task k a per-parameter contribution to the change of the loss is first calculated for each parameter i as follows:

$$\omega_i^{(k)} = \sum_{t=1}^{N_{\text{iters}}} \left(\theta_i[t^{(k)}] - \theta_i[(t-1)^{(k)}]\right) \frac{-\delta \mathcal{L}_{\text{total}}[t^{(k)}]}{\delta \theta_i} \tag{11}$$

with N_{iters} the total number of iterations per task, $\theta_i[t^{(k)}]$ the value of the i^{th} parameter after the t^{th} training iteration on task k and $\frac{\delta \mathcal{L}_{\text{total}}[t^{(k)}]}{\delta \theta_i}$ the gradient of the loss with respect to the i^{th} parameter during the t^{th} training iteration on task k. For every task, these per-parameter contributions are normalized by the square of the total change of that parameter during training on that task plus a small dampening term ξ (set to 0.1, to bound the resulting normalized contributions when a parameter's total change goes to zero), after which they are summed over all tasks so far. The estimated importance of parameter i for the first $K-1$ tasks is thus given by:

$$\Omega_i^{(K-1)} = \sum_{k=1}^{K-1} \frac{\omega_i^{(k)}}{\left(\Delta_i^{(k)}\right)^2 + \xi} \tag{12}$$

with $\Delta_i^{(k)} = \theta_i[{N_{\text{iters}}}^{(k)}] - \theta_i[0^{(k)}]$, where $\theta_i[0^{(k)}]$ indicates the value of parameter i right before starting training on task k. (An alternative formulation is $\Delta_i^{(k)} = \hat{\theta}_i^{(k)} - \hat{\theta}_i^{(k-1)}$, with $\hat{\theta}_i^{(0)}$ the value of parameter i it was initialized with and $\hat{\theta}_i^{(k)}$ its value after finishing training on task k.) The regularization term of SI to be used during training on task K is then given by:

$$\mathcal{L}^{(K)}_{\text{regularization}_{\text{SI}}} = \sum_{i=1}^{N_{\text{params}}} \Omega_i^{(K-1)} \left(\theta_i - \hat{\theta}_i^{(K-1)}\right)^2 \tag{13}$$

B Hyperparameters

As discussed in Section A.2 and Appendix A.4, several of the in this paper compared continual learning methods have one or more hyperparameters. The typical way of setting the value of hyperparameters is by training models on the training set for a range of hyperparameter-values, and selecting those that result in the best performance on a separate validation set. This strategy has been adapted to the continual learning setting as training models on the full protocol with different hyperparameter-values using only every task's training data, and comparing their overall performances using separate validation sets (or sometimes the test sets) for each task [e.g., see 10, 1, 8, 13]. However, here we would like to stress that this means that these hyperparameters are set (or learned) based on an evaluation using data from all tasks, which violates the continual learning principle of only being allowed to visit each task once and in sequence. Although it is tempting to think that it is acceptable to relax this principle for tasks' validation data, we argue here that it is not. A clear example of how using each task's validation data continuously throughout an incremental training protocol can lead to an in our opinion unfair advantage is provided by Wu et al. [6], in which after finishing training on each task a "bias-removal parameter" is set that optimizes performance on the validation sets of all tasks seen so far (see their Section 3.3). Although the hyperparameters of the methods compared here are much less influential than those in the above paper, we believe that it is important to realize this issue associated with traditional grid searches in a continual learning setting and that at a minimum influential hyperparameters should be avoided in methods for continual learning.

Nevertheless, to give each method the best possible chance—and to explore how influential their hyperparameters are—we do perform grid searches to set the values of their hyperparameters (see Figures 3 and 4). Given the issue discussed above we do not see much value in using validation sets for this, and we evaluate the performances of all hyperparameter(-combination)s using the tasks' test sets. For this grid search each experiment is run once, after which 20 new runs are executed using the selected hyperparameter-values to obtain the results in Tables 3 and 4.

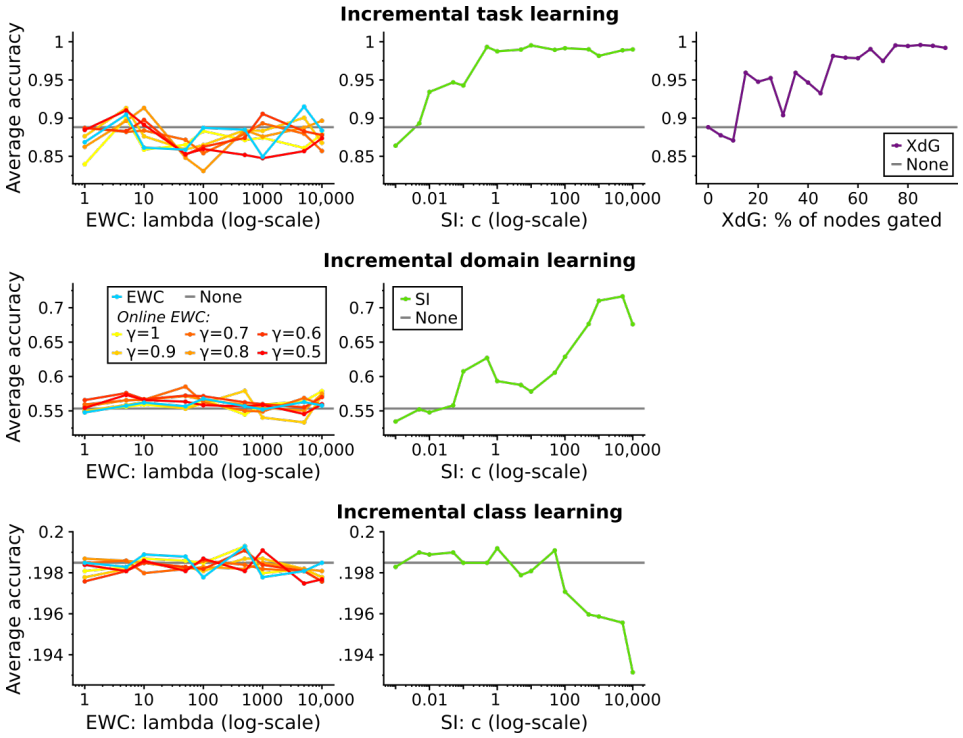

Figure 3: Grid searches for the split MNIST task protocol. Shown are the average test set accuracies (over all 5 tasks) for the (combination of) hyperparameter-values tested for each method.

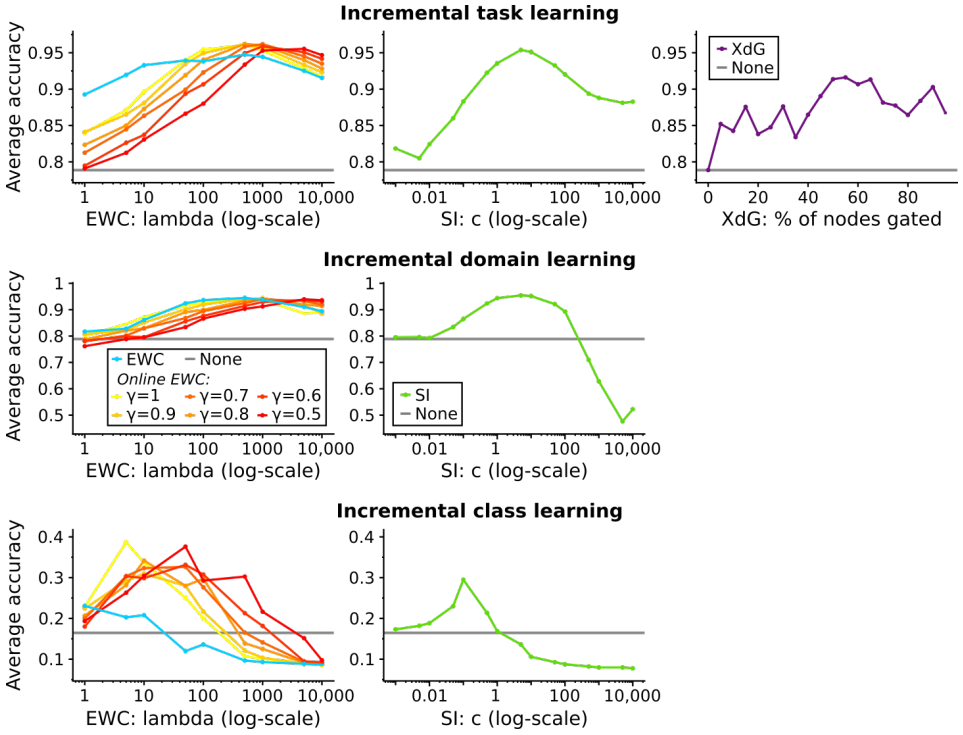

Figure 4: Grid searches for the permuted MNIST task protocol. Shown are the average test set accuracies (over all 10 tasks) for the (combination of) hyperparameter-values tested for each method.

C Additional discussion: storing data

In the current paper, based on the argument that storing data is not always possible due to privacy concerns or memory constraints, we only considered methods that do not store data. However, as indicated by methods such as iCaRL [2] and FearNet [5], when possible, storing data can substantially boost performance, especially in the incremental class learning scenario. Of particular note is that these two methods point out that it is not necessary that all of the original data is stored permanently. Indeed, iCaRL demonstrates that storing a relatively small number of well-chosen examples can be helpful, while FearNet's good performance seems to suggest that temporary storage of data can already be useful. Both these reductionist approaches to storing data of course reduce memory storage demands. Finally, an interesting aspect of FearNet is that it also stores hidden summary statistics. The idea of storing hidden representations, which besides reducing memory storage demands could also address the privacy issue, is further worked out by Riemer et al. [26]. We expect that the sparse and/or temporary storage of hidden representations could be a useful complement to generative replay, which might help it to scale up to real-world continual learning problems. However, we want to stress that when allowing the storage of data, it is important to take extra care to ensure fair comparisons between methods (i.e., that they all have the same rules regarding to how much, for how long and what data can be stored).