nature machine intelligence

Article

https://doi.org/10.1038/s42256-022-00568-3

Three types of incremental learning

Received: 1 October 2021

Accepted: 18 October 2022

Published online: 5 December 2022

Check for updates

Gido M. van de Ven [1,2,3] ✉, Tinne Tuytelaars[3] & Andreas S. Tolias [1,4]

Incrementally learning new information from a non-stationary stream of data, referred to as 'continual learning', is a key feature of natural intelligence, but a challenging problem for deep neural networks. In recent years, numerous deep learning methods for continual learning have been proposed, but comparing their performances is difficult due to the lack of a common framework. To help address this, we describe three fundamental types, or 'scenarios', of continual learning: task-incremental, domain-incremental and class-incremental learning. Each of these scenarios has its own set of challenges. To illustrate this, we provide a comprehensive empirical comparison of currently used continual learning strategies, by performing the Split MNIST and Split CIFAR-100 protocols according to each scenario. We demonstrate substantial differences between the three scenarios in terms of difficulty and in terms of the effectiveness of different strategies. The proposed categorization aims to structure the continual learning field, by forming a key foundation for clearly defining benchmark problems.

An important open problem in deep learning is enabling neural networks to incrementally learn from non-stationary streams of data[1,2]. For example, when deep neural networks are trained on samples from a new task or data distribution, they tend to rapidly lose previously acquired capabilities, a phenomenon referred to as catastrophic forgetting[3,4]. In stark contrast, humans and other animals are able to incrementally learn new skills without compromising those that were already learned[5]. The field of continual learning, also referred to as lifelong learning, is devoted to closing the gap in incremental learning ability between natural and artificial intelligence. In recent years, this area of machine learning research has been rapidly expanding, fuelled by the potential utility of deploying continual learning algorithms for applications such as medical diagnosis[6], autonomous driving[7] or predicting financial markets[8].

Despite its scope, continual learning research is relatively unstructured and the field lacks a shared framework. Because of an abundance of subtle, but often important, differences between evaluation protocols, systematic comparison between continual learning algorithms is challenging, even when papers use the same datasets[9]. It is therefore not surprising that numerous continual learning methods claim to be state-of-the-art. To help address this, here we describe a structured and intuitive framework for continual learning.

We put forward the view that, at the computational level[10], there are three fundamental types, or 'scenarios', of supervised continual learning. Informally, (a) in task-incremental learning, an algorithm must incrementally learn a set of clearly distinguishable tasks; (b) in domain-incremental learning, an algorithm must learn the same kind of problem but in different contexts; and (c) in class-incremental learning, an algorithm must incrementally learn to distinguish between a growing number of objects or classes. In this article, we formally define these three scenarios and point out different challenges associated with each one of them. We also review existing strategies for continual learning with deep neural networks and we provide a comprehensive, empirical comparison to test how suitable these different strategies are for each scenario.

Three continual learning scenarios

In classical machine learning, an algorithm has access to all training data at the same time. In continual learning, the data instead arrives in a sequence, or in a number of steps, and the underlying distribution

[1]Center for Neuroscience and Artificial Intelligence, Department of Neuroscience, Baylor College of Medicine, Houston, TX, USA. [2]Computational and Biological Learning Lab, Department of Engineering, University of Cambridge, Cambridge, UK. [3]Processing Speech and Images, Department of Electrical Engineering, KU Leuven, Leuven, Belgium. [4]Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA. ✉e-mail: gido.vandeven@kuleuven.be

of the data changes over time. In this article, we propose that, depending on how the aspect of the data that changes over time relates to the function or mapping that must be learned, there are three fundamental ways in which a supervised learning problem can be incremental (Table 1). Below, we start by describing the resulting three continual learning scenarios intuitively. After that we define them more formally: first in a restricted, 'academic' setting, before generalizing them to more flexible continual learning settings.

Table 1 | Overview of the three continual learning scenarios

Scenario	Intuitive description	Mapping to learn
Task-incremental learning	Sequentially learn to solve a number of distinct tasks	$f : \mathcal{X} \times \mathcal{C} \rightarrow \mathcal{Y}$
Domain-incremental learning	Learn to solve the same problem in different contexts	$f : \mathcal{X} \rightarrow \mathcal{Y}$
Class-incremental learning	Discriminate between incrementally observed classes	$f : \mathcal{X} \rightarrow \mathcal{C} \times \mathcal{Y}$

Notation: $\mathcal{X}$ is the input space, $\mathcal{Y}$ is the within-context output space and $\mathcal{C}$ is the context space. In this article, the term 'context' refers to an underlying distribution from which observations are sampled. The context changes over time. In the continual learning literature, the term 'task' is often used in a way analogous to how the term 'context' is used here.

Intuitive descriptions and each scenario's challenges

The first continual learning scenario we refer to as 'task-incremental learning' (or Task-IL). This scenario is best described as the case where an algorithm must incrementally learn a set of distinct tasks (see refs. [11–13] for examples from the literature). The defining characteristic of task-incremental learning is that it is always clear to the algorithm—also at test time—which task must be performed. In practice, this could mean that task identity is explicitly provided, or that the tasks are clearly distinguishable. In this scenario it is possible to train models with task-specific components (for example, a separate output layer per task), or even to have a completely separate network for each task to be learned. In this last case there is no forgetting at all. The challenge with task-incremental learning, therefore, is not—or should not be—to simply prevent catastrophic forgetting, but rather to find effective ways to share learned representations across tasks, to optimize the trade-off between performance and computational complexity and to use information learned in one task to improve performance on other tasks (that is, to achieve positive forward or even backward transfer between tasks)[14,15]. These are still open challenges. Real-world examples of task-incremental learning are learning to play different sports or different musical instruments, because typically it is always clear which sport or instrument should be played.

We call the second scenario 'domain-incremental learning' (or Domain-IL). In this scenario, the structure of the problem is always the same, but the context or input-distribution changes (for example, there are domain-shifts; see refs. [16,17]). Similarly to task-incremental learning, this scenario can be described as that an algorithm must incrementally learn a set of 'tasks' (although now it might be more intuitive to think of them as 'domains'), but with the crucial difference that—at least at test time—the algorithm does not know to which task a sample belongs. However, identifying the task is not necessary, because each task has the same possible outputs (for example, the same classes are used in each task). Using task-specific components in this scenario is, however, only possible if an algorithm first identifies the task[18–23], but that is not necessarily the most efficient strategy. Preventing forgetting 'by design' is therefore not possible with domain-incremental learning, and alleviating catastrophic forgetting is still an important unsolved challenge. Examples of this scenario are incrementally learning to recognize objects under variable lighting conditions[24] (for example, indoors versus outdoors) or learning to drive in different weather conditions[17].

The third continual learning scenario is 'class-incremental learning' (or Class-IL). This scenario is best described as the case where an algorithm must incrementally learn to discriminate between a growing number of objects or classes (for example, refs. [25,26]). An often used set-up for this scenario is that a sequence of classification-based tasks (although now it might be more intuitive to think of them as 'episodes') is encountered, whereby each task contains different classes and the algorithm must learn to distinguish between all classes[19,27,28]. In this case, task identification is necessary to solve the problem, as it determines which possible classes the current sample might belong to. In other words, the algorithm should be able both to solve each individual task (that is, distinguish between classes within an episode) and to identify which task a sample belongs to (that is, distinguish between classes from different episodes). For example, an agent might first learn about cats and dogs, and later about cows and horses; while with task-incremental learning the agent would not be expected to distinguish between animals encountered in different episodes (for example, between cats and cows), with class-incremental learning this is required. An important challenge in this scenario is learning to discriminate between classes that are not observed together, which has turned out to be very challenging for deep neural networks, especially when storing examples of previously seen classes is not allowed[29,30].

Formalization in a restricted, 'academic' setting

To more formally define these three scenarios, we start by considering the simple, but frequently studied, continual learning setting in which a classification problem is split up into multiple parts or episodes that must be learned sequentially, with no overlap between the different episodes. In the continual learning literature, these episodes are often called tasks, but in this article we will refer to them as 'contexts'. The term task is problematic because in the literature it is used with several different meanings or connotations. From here on, we will use the term task only to refer to a context when it is always clear to the learning algorithm when a sample belongs to that context (as is the case with task-incremental learning).

In the 'academic continual learning setting' sketched above (that is, classification-based, non-overlapping contexts encountered sequentially), a clear distinction can be drawn between the three scenarios. To formalize this, we express each sample as consisting of three components: an input $x \in \mathcal{X}$, a within-context label $y \in \mathcal{Y}$ and a context label $c \in \mathcal{C}$. The three scenarios can then be defined based on how the function or mapping that must be learned relates to the context space $\mathcal{C}$. With task-incremental learning, an algorithm is expected to learn a mapping of the form $f : \mathcal{X} \times \mathcal{C} \rightarrow \mathcal{Y}$, with domain-incremental learning a mapping of the form $f : \mathcal{X} \rightarrow \mathcal{Y}$ must be learned and with class-incremental learning the shape of the mapping to be learned is $f : \mathcal{X} \rightarrow \mathcal{C} \times \mathcal{Y}$. For class-incremental learning this mapping can also be written as $f : \mathcal{X} \rightarrow \mathcal{G}$, with $\mathcal{G}$ the 'global label space' obtained by combining $\mathcal{C}$ and $\mathcal{Y}$.

These definitions imply that the three scenarios can be distinguished based on whether at test time context identity information is known to the algorithm and, in case it is not, whether it must be inferred (Fig. 1). Each scenario thus specifies whether context labels are available during testing, but not necessarily whether they are available during training. With task- and class-incremental learning, it is often implicit that context labels are provided during training (for example, in the case of supervised learning), but with domain-incremental learning it is good practice to explicitly state whether context labels (or context boundaries) are provided during training.

To illustrate the continual learning scenarios with an example, Fig. 2 shows how Split MNIST, which is a popular toy problem for continual learning[27,28,31,32], can be performed according to each of the three scenarios. Further examples illustrating these scenarios with other context sequences are provided in Supplementary Note 1.

https://doi.org/10.1038/s42256-022-00568-3

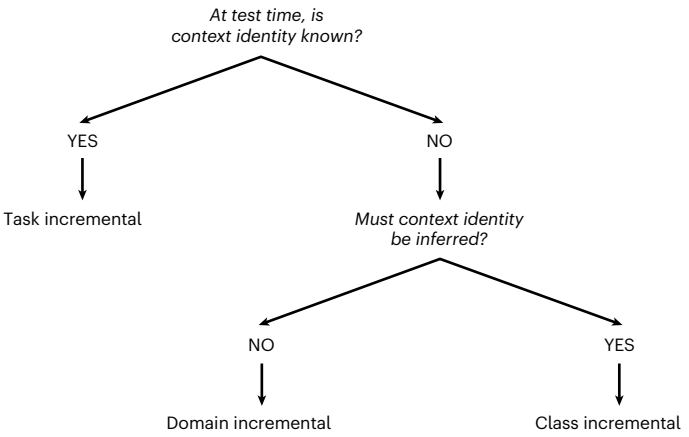


Fig. 1 | Decision tree for the three continual learning scenarios. The scenarios can be defined based on whether at test time context identity is known and if it is not, whether it must be inferred.

It might be unintuitive to distinguish domain- and class-incremental learning by whether context identity must be inferred, because with class-incremental learning context identification is often not explicitly performed, as typically a direct mapping is learned from the input space $\mathcal{X}$ to the set of global labels $\mathcal{Y}$. Another way to tell these two scenarios apart is by whether different contexts contain the same classes (domain-incremental learning) or different classes (class-incremental learning). However, it should then be realized that whether two samples belong to the same class can change depending on perspective: in the Split MNIST example (Fig. 2), with domain-incremental learning the digits '0' and '2' belong to the same class (as they are both even digits), but with class-incremental learning they are considered different classes.

Generalization to more flexible settings

The clear separation between the three scenarios makes the academic continual learning setting convenient for studying these scenarios and their different challenges in isolation. However, this setting does not reflect well the arbitrary non-stationarity that can be observed in the real world[33–40]. To generalize the three scenarios to more flexible continual learning settings, we first introduce a distinction between the concepts 'context set' and 'data stream':

The 'context set' is defined as a collection of underlying distributions, denoted by $\{\mathcal{D}_c\}_{c\in\mathcal{C}}$, from which the observations presented to the algorithm are sampled. For a supervised continual learning problem, for each context $c \in \mathcal{C}$, samples from $\mathcal{D}_c$ consist of an input $x \in \mathcal{X}$ and a within-context label $y \in \mathcal{Y}$. (With class-incremental learning each context could also contain a single class, in which case the within-context label y is not used.)

The 'data stream' is defined as a (possibly unbounded) stream of experiences that are sequentially presented to the algorithm: $e_1, e_2, \ldots$. Each experience consists of a set of observations sampled from one or more of the underlying distributions of the context set. These experiences are the incremental steps of a continual learning problem, in the sense that at each step, the algorithm has free access to the data of the current experience, but not to the data from past or future experiences (see also ref. [41]).

In the academic continual-learning setting, there is no distinction between the context set and the data stream, because each experience consists of all the training data of a particular context. In general, however, such a direct relation is not needed. Every observation within each experience can in principle be sampled from any combination of underlying datasets from the context set. This can be formalized as:

$$e_t[i] \sim \sum_{c\in\mathcal{C}} p_c^{t,i}\mathcal{D}_c \tag{1}$$

whereby $e_t[i]$ is observation i of experience t and $p_c^{t,i}$ is the probability that this observation is sampled from $\mathcal{D}_c$. Importantly, in this framework, from a probabilistic perspective, two observations at different points in time can only differ from each other with respect to the (combination of) contexts from which they are sampled. With this formulation, the context set describes the aspects of the data that 'can' change over time and the probabilities $p_c^{t,i}$ describe 'how' they change over time.

An advantage of distinguishing between the context set and the data stream is that it makes it possible to describe continual learning problems with gradual transitions between contexts[34,37,40,42] or whereby contexts are revisited[43–45]. In this framework, which is suitable for so-called 'task-free continual learning'[33,46–48], generalized versions of the three scenarios can be defined based on how the mapping that must be learned relates to the context space $\mathcal{C}$, which describes the non-stationary aspect of the data. Supplementary Note 2 illustrates how a 'task-free' data stream can be performed according to each of the generalized versions of the three scenarios.

We note that for complex real-world incremental learning problems, it might not be straight-forward to express the mapping that must be learned in terms of the context space $\mathcal{C}$, for example, because there are different aspects of the data that change over time. To accommodate this, a multidimensional context space $\mathcal{C}$ can be used, whereby each dimension could adhere to a different scenario. This allows for continual-learning problems that are mixtures of scenarios (Supplementary Note 3). Another generalization is that contexts do not need to be discrete, but can be continuous (in that case the summation in equation (1) becomes an integral); an example of a continuous context set is Rotated MNIST with arbitrary rotation (Supplementary Note 3).

Empirical comparison

To further explore the differences between the three continual learning scenarios, here we provide an empirical comparison of the performance of different deep learning strategies. To do this comparison in a structured manner, in Supplementary Note 4 we discuss and distinguish five computational strategies for continual learning (Fig. 3). For each strategy, we included a few representative methods in our comparison.

Compared methods

The use of context-specific components was represented by context-dependent gating (XdG[12]), which masks for each context a randomly selected subset of hidden units; and the separate networks approach, where the available parameter budget is divided over all contexts and a separate network is learned for each context.

Included parameter regularization methods were elastic weight consolidation (EWC[49]), which estimates parameter importance using a diagonal approximation to the Fisher information; and synaptic intelligence (SI[31]), which estimates parameter importance online based on the training trajectory.

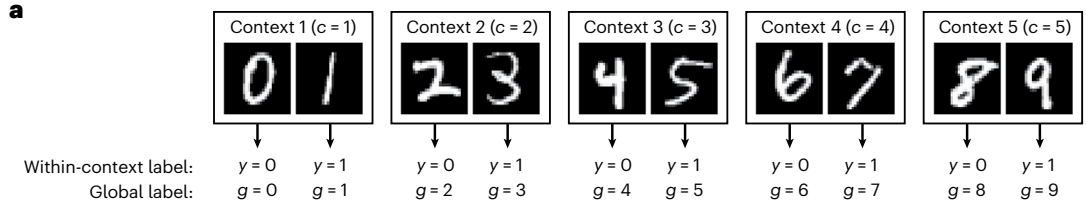
For functional regularization, compared were learning without forgetting (LwF[50]), which uses the inputs from the current context as anchor points; and functional regularization of the memorable past (FROMP[51]), which uses stored examples from past contexts as anchor points.

The included replay methods were deep generative replay (DGR[27]), which replays generated representations at the input level; brain-inspired replay (BI-R[28]), which replays generated representations at the latent feature level; experience replay (ER[52,53]), which replays stored samples in the 'standard way' (that is, loss on replayed data added to loss on current data); and averaged gradient episodic memory (A-GEM[54]), which replays the same stored samples but using the loss on replayed data as inequality constraint (that is, loss on current data optimized under constraint that loss on replayed data cannot increase).

The compared template-based methods were iCaRL[25], with mean latent feature representations of stored examples as templates; and the generative classifier from ref. [55], which uses class-specific generative models as templates.

https://doi.org/10.1038/s42256-022-00568-3

a



b

	Input (at test time)	Expected output	Intuitive description
Task-incremental learning	Image + context label	Within-context label[a]	Choice between two digits of same context (e.g. 0 or 1)
Domain-incremental learning	Image	Within-context label	Is the digit odd or even?
Class-incremental learning	Image	Global label	Choice between all ten digits

Fig. 2 | Split MNIST according to the three scenarios. a, The Split MNIST protocol is obtained by splitting the original MNIST dataset into five contexts, with each context consisting of two digits. **b**, Overview of what is expected of the algorithm at test time when the Split MNIST protocol is performed according to each continual learning scenario. [a]With task-incremental learning, at the computational level, there is no difference between whether the algorithm must return the within-context label or the global label, because the within-context label can be combined with the context label (which is provided as input) to get the global label.

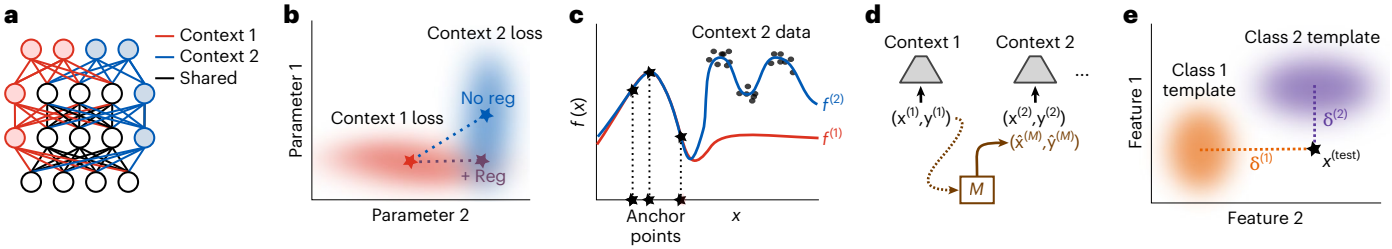


Fig. 3 | Schematic illustrations of different continual learning strategies. **a**, Context-specific components. Certain parts of the network are only used for specific contexts. **b**, Parameter regularization. Parameters important for past contexts are encouraged not to change too much when learning new contexts. **c**, Functional regularization. The input–output mapping learned previously is encouraged not to change too much at a particular set of inputs (the 'anchor points') when training on new contexts. **d**, Replay. The training data of a new context is complemented with data representative of past context. The replayed data is sampled from *M*, which can be a memory buffer or a generative model. **e**, Template-based classification. A 'template' is learned for each class (for example, a prototype, an energy value or a generative model), and classification is performed based on which template is most suitable for the sample to be classified. See Supplementary Note 4 for a detailed discussion of these strategies.

Finally, two baselines were included. As lower target, referred to as 'none', the model was incrementally trained on all contexts in the standard way. As upper target, referred to as 'joint', the model was trained on the data of all contexts at the same time.

Set-up

We performed both the Split MNIST and the Split CIFAR-100 protocol according to each of the three scenarios. All experiments used the academic continual learning setting and context identity information was available during training. To make the comparisons as informative as possible, we used similar network architectures and similar training protocols for all compared methods. Depending on the continual learning scenario, the output layer of the network was treated differently. With task-incremental learning, a multi-headed output layer was used whereby each context had its own output units and only the units of the context under consideration were used. For the other two scenarios, single-headed output layers were used, with the number of output units equal to the number of classes per context (domain-incremental learning) or to the total number of classes (class-incremental learning). See Methods for more detail.

Results

For both Split MNIST (Table 2) and Split CIFAR-100 (Table 3), we found clear differences between the three continual learning scenarios. With task-incremental learning, almost all tested methods performed well compared to the 'none' and 'joint' baselines, with domain-incremental learning the relative performances of many methods dropped considerably and with class-incremental learning they decreased even further.

The decline in performance across the three scenarios was most pronounced for the parameter regularization methods. On both protocols, EWC and SI performed close to the upper target when context identity was known during testing (that is, task-incremental learning); with domain-incremental learning the performance of both methods was substantially lower, but remained above the lower target of sequentially training a network in the standard way; and with class-incremental learning the performance of EWC and SI was similar to the lower target, indicating that in this scenario these methods failed completely. There was a similar trend across the three scenarios for the functional regularization methods, albeit less pronounced for FROMP than for LwF.

Replay-based methods performed relatively well in all three scenarios. Although on both protocols their performance still decreased from task- to domain- to class-incremental learning, the decline was less sharp than for the regularization-based methods, and replay-based methods were among the top performers in each scenario. Template-based classification also performed well with class-incremental learning, with iCaRL and the Generative Classifier among the best performing methods on both protocols.

For class-incremental learning, the methods that performed best either used a generative model or they stored previously seen data in a memory buffer. Directly comparing methods using these two

Table 2 | Results on Split MNIST

Strategy	Method	Budget	GM	Task-IL	Domain-IL	Class-IL
Baselines	None – lower target			84.32 (±0.99)	60.13 (±1.66)	19.89 (±0.02)
	Joint – upper target			99.67 (±0.03)	98.59 (±0.05)	98.17 (±0.04)
Context-specific components	Separate Networks	-	-	99.57 (±0.03)	-	-
	XdG	-	-	99.10 (±0.10)	-	-
Parameter regularization	EWC	-	-	99.06 (±0.15)	63.03 (±1.58)	20.64 (±0.52)
	SI	-	-	99.20 (±0.11)	66.94 (±1.13)	21.20 (±0.57)
Functional regularization	LwF	-	-	99.60 (±0.03)	71.18 (±1.42)	21.89 (±0.32)
	FROMP	100	-	99.12 (±0.13)	84.86 (±1.02)	77.38 (±0.64)
Replay	DGR	-	Yes	99.50 (±0.03)	95.57 (±0.30)	90.35 (±0.24)
	BI-R	-	Yes	99.61 (±0.03)	97.26 (±0.15)	94.41 (±0.15)
	ER	100	-	98.98 (±0.07)	93.75 (±0.24)	88.79 (±0.20)
	A-GEM	100	-	98.54 (±0.10)	87.67 (±1.33)	65.10 (±3.64)
Template-based classification	Generative Classifier	-	Yes	-	-	93.82 (±0.06)
	iCaRL	100	-	-	-	92.49 (±0.12)

Reported is the final test accuracy (as percentage, averaged over all contexts) of all compared methods on the Split MNIST protocol, which is performed according to all three scenarios. The experiments followed the academic continual learning setting and context identity information was available during training. The column 'Budget' indicates the number of examples per class that was allowed to be stored in a memory buffer. The column 'GM' indicates whether a generative model was learned, for which additional network capacity was used. Each experiment was performed 20 times with different random seeds, reported is the mean (±s.e.m.) over these runs.

Table 3 | Results on Split CIFAR-100

Strategy	Method	Budget	GM	Task-IL	Domain-IL	Class-IL
Baselines	None – lower target			61.43 (±0.36)	18.42 (±0.33)	7.71 (±0.18)
	Joint – upper target			78.78 (±0.25)	46.85 (±0.51)	49.78 (±0.21)
Context-specific components	Separate Networks	-	-	76.83 (±0.25)	-	-
	XdG	-	-	69.86 (±0.34)	-	-
Parameter regularization	EWC	-	-	76.34 (±0.29)	21.65 (±0.55)	8.24 (±0.25)
	SI	-	-	74.84 (±0.39)	22.58 (±0.42)	8.10 (±0.24)
Functional regularization	LwF	-	-	78.59 (±0.24)	29.45 (±0.39)	25.57 (±0.27)
Replay	DGR	-	Yes	71.40 (±0.32)	20.52 (±0.43)	9.67 (±0.22)
	BI-R	-	Yes	79.14 (±0.21)	30.26 (±0.44)	25.81 (±0.41)
	ER	100	-	76.43 (±0.24)	39.00 (±0.34)	37.57 (±0.21)
	A-GEM	100	-	73.30 (±0.39)	20.51 (±0.59)	20.38 (±1.45)
Template-based classification	Generative Classifier	-	Yes	-	-	46.83 (±0.18)
	iCaRL	100	-	-	-	37.83 (±0.21)

Reported is the final test accuracy (as percentage, averaged over all contexts) of all compared methods on the Split CIFAR-100 protocol, which is performed according to all three scenarios. The experiments followed the academic continual learning setting and context identity information was available during training. The column 'Budget' indicates the number of examples per class that was allowed to be stored in a memory buffer. The column 'GM' indicates whether a generative model was learned, for which additional network capacity was used. Note that we were not able to run the method FROMP on this protocol due to its high computational costs. Each experiment was performed 10 times with different random seeds, reported is the mean (±s.e.m.) over these runs. All compared methods used convolutional layers that were pre-trained on CIFAR-10, see Methods for full details.

types of memories can be arbitrary, as their performance can heavily depend on the number of stored examples or the kind of generative model. We instead focus on comparing methods using the same type of memory.

For methods using generative models, the largest differences were observed with Split CIFAR-100. In particular, DGR did not perform well on this protocol, indicating that standard generative replay (that is, at the input level) is not a good approach when the input data are complex (see also refs. [28,56,57]). There was also a substantial gap in performance between BI-R and the Generative Classifier. As both methods had a generative model on the latent features, this suggests that the way in which a generative model is used (that is, for generating replay or as templates) is important as well.

For methods using stored data, we found that replaying stored data in the standard way (as is done by ER) was not often outperformed by more complex ways of using stored data. In fact, perhaps surprisingly, on all experiments ER comfortably outperformed A-GEM, and ER performed significantly better than FROMP on two of the three scenarios of Split MNIST. These results held for different sizes of the memory buffer (Extended Data Fig. 1). A clear improvement over ER was only observed with iCaRL, and only when the size of the memory buffer was relatively small.

Discussion

Continual learning is a key feature of natural intelligence, but an open challenge for deep learning. Standard deep neural networks tend

to catastrophically forget previous tasks or data distributions when trained on a new one. Enabling these networks to incrementally learn, and retain, information from different contexts has become a topic of intense research. Yet, despite its scope, the continual learning field lacks structure and direct comparisons between published papers can be misleading. Here, we pointed out that an important difference between continual learning set-ups is whether context identity is known to the algorithm and—if it is not—whether it must be inferred. Based on these two distinctions, we identified three scenarios for continual learning: task-incremental learning, domain-incremental learning and class-incremental learning.

These three scenarios and their different challenges can be conveniently studied in an academic continual learning setting, where a classification-based problem is split up in discrete, non-overlapping contexts (which are often called 'tasks') that are encountered in sequence. We showed that in this setting there is a clear separation between the three scenarios. At least in part because of two preprints of this article[58,59], the terms 'task-incremental learning', 'domain-incremental learning' and 'class-incremental learning' are sometimes being used in the recent literature in a way that restricts them to this academic setting. Here, by interpreting these three scenarios as specifying how the non-stationary aspect of the data relates to the mapping that must be learned, we propose that they generalize to more flexible continual-learning settings. To demonstrate the value of such generalized versions of these three scenarios, Supplementary Note 2 shows how a 'task-free' data stream without sharp context boundaries can also be performed in three different ways.

A key insight of this article is that a useful way to categorize continual-learning problems is based on how the non-stationary aspect of the data relates to the mapping to be learned. For supervised classification this leads to the three continual learning scenarios discussed here, but the same perspective might also be useful for unsupervised or reinforcement learning (Supplementary Note 5). Continual learning can be categorized in other ways as well, some of which we discuss in Supplementary Note 6.

Using the academic continual-learning setting, for each scenario we performed an empirical comparison of a representative selection of continual learning algorithms. This comparison revealed marked differences between the three scenarios in overall difficulty level and in the relative effectiveness of different continual learning strategies. The only strategy among the top performers in all three scenarios is replay, with the replayed data sampled either from a memory buffer or a generative model. Surprisingly, within the class of methods using stored data, the strongest performance is often obtained by the method ER, which replays stored data 'in the standard way'. In our experiments, popular methods such as A-GEM and FROMP, which use stored data in more complex ways, are almost always outperformed by ER, even though the computational costs of A-GEM and FROMP are strictly higher than those of ER.

In the class-incremental learning scenario, we found that parameter regularization methods such as EWC and SI fail almost completely, even on Split MNIST. Functional regularization typically works better, especially when using stored data as anchor points, but this strategy also does not work optimally. We hypothesize that regularization-based strategies—at least by themselves—are not well suited for class-incremental learning because they do not provide a way to compare between classes that are not observed together. Regularization-based methods aim to learn new contexts while preserving the function or parameters learned in previous contexts. However, with class-incremental learning, learning a new context (for example, distinguishing '2' and '3') while preserving what was learned before (for example, distinguishing '0' and '1') is not enough; it is also needed to combine information from different contexts (for example, for distinguishing '1' and '2'). For learning to distinguish between classes not observed together, it might be unavoidable to use either replay, which allows for comparing between classes from different contexts during training, or template-based classification, which allows for comparing between classes from different contexts during inference (that is, during the classification decision).

Task-incremental learning is sometimes considered 'easy', and it has been argued that the continual-learning community should move away from the assumption that context identities (or task labels, as they are often called) are provided at test time[60]. A reason for this notion might be that with task-incremental learning, the bar is often set too low. In our experiments, while all methods indeed perform substantially better than the usual 'lower target' in which a single shared neural network is sequentially trained on all contexts, most methods perform worse than the more appropriate lower target in which a smaller, separate network is trained for each context. To do better than this 'Separate Networks' approach, positive forward or backward transfer between contexts is necessary, but achieving such positive transfer is not trivial[14,15].

Domain-incremental learning might be the least studied continual learning scenario. A few years ago this scenario was regularly studied with Permuted MNIST[31,49,61], but this protocol is not often used anymore as it is considered too artificial. The continual learning field is currently dominated by context sets created by splitting up existing image classification datasets based on class labels (for example, Split MNIST, Split CIFAR-100). Although in theory these context sets can be performed according to all three scenarios, they are typically less intuitive and/or realistic under the assumptions of domain-incremental learning. However, in recent years, several resources have been created that provide—or enable the generation of—more realistic context sets well suited for domain-incremental learning[24,41,62–64]. These resources might help to renew the community's interest in this scenario.

The three continual learning scenarios described in this article provide a useful basis for defining clear and unambiguous benchmark problems for continual learning. We hope this will accelerate progress to bridge the gap between natural and artificial intelligence. Moreover, we believe that it is an important conceptual insight that, at the computational level, a supervised learning problem can be incremental in these three different ways. Perhaps especially in the real world, where continual learning problems are often complex and 'mixtures' of scenarios, it might be fruitful to approach problems as consisting of a combination of these three fundamental types of incremental learning.

Methods

All experiments were run using custom-written code for the Python machine learning framework PyTorch[65].

Context sets

For the Split MNIST protocol, the MNIST dataset[66] was split into five contexts, such that each context contained two digits. The digits were randomly divided over the five contexts, so the order of the digits was different for each random seed. The original 28×28 pixel greyscale images were used without pre-processing. The standard training/test-split was used, which resulted in 60,000 training images (approximately 6,000 per digit) and 10,000 test images (approximately 1,000 per digit).

For the Split CIFAR-100 protocol, the CIFAR-100 dataset[67] was split up into ten contexts, such that each context contained ten image classes. The classes were randomly divided over the contexts, with a different class order for each random seed. The original 32×32 pixel RGB-colour images were normalized (that is, each pixel-value was subtracted by the relevant channel-wise mean and divided by the channel-wise standard deviation, with means and standard deviations calculated over all training images). No other pre-processing or augmentation was applied. The standard training/test-split was used,

which resulted in 50,000 training images (500 per class) and 10,000 test images (100 per class).

Base neural network architecture

To make the comparisons as informative as possible, we used the same base neural network architecture for all methods as much as possible. For Split MNIST, the base network had two fully connected hidden layers of 400 ReLU each and a softmax output layer. For Split CIFAR-100, the base network had five pre-trained convolutional layers followed by two fully connected layers with 2,000 ReLU each and a softmax output layer. The convolutional layers contained 16, 32, 64, 128 and 256 channels. Each convolutional layer used a 3×3 kernel, a padding of 1 and there was a stride of 1 in the first layer (that is, no downsampling) and a stride of 2 in the other layers (that is, image-size was halved in each of those layers). Batch norm[68] was used in all convolutional layers, followed by a ReLU non-linearity. No pooling was used. The convolutional layers were pre-trained on CIFAR-10, which is a dataset containing similar but non-overlapping images and image classes compared with CIFAR-100[67]. To pre-train the convolutional layers, the base neural network was trained to classify the 10 classes of CIFAR-10 for 100 epochs, using the ADAM-optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$) with learning rate of 0.0001 and mini-batch size of 256. For the pre-training on CIFAR-10, images were normalized and augmented by random cropping and horizontal flipping. A similar pre-training protocol was used in ref. [28]. During the incremental training on CIFAR-100, the parameters of the pre-trained convolutional layers were frozen. For all compared methods, freezing these parameters resulted in similar or better performance compared with not freezing them.

Output layer

The softmax output layer of the network was treated differently depending on the continual learning scenario that was performed. With task-incremental learning, a multi-headed output layer was used, meaning that each context had its own output units and only the output units of the context under consideration—that is, either the current context or the replayed context—were set to 'active' (see next paragraph). With domain- and class-incremental learning, a single-headed output layer was used. For domain-incremental learning, this meant that all contexts used the same output units (that is, there were 2 output units for Split MNIST and 10 for Split CIFAR-100); for class-incremental learning, this meant that each class had its own output unit (that is, there were 10 output units for Split MNIST and 100 for Split CIFAR-100). With both domain- and class-incremental learning, always all output units were set to 'active'. Note that with class-incremental learning another possibility is to use an 'expanding head' and only set the output units of classes seen so far to active (for example, see refs. [28,69]). We found that for our experiments there was not much difference in performance between these two options. Because all output units should always be active for the Bayesian interpretation of the parameter regularization methods[21], we decided to use that approach in this study.

Whether an output unit was set to 'active' controlled whether a network could assign a positive probability to its corresponding class. The probability predicted by a neural network with parameters θ that an input x belongs to output class o was calculated as:

$$p_\theta(o|x) = \begin{cases} \frac{e^{z_o^{(x,\theta)}}}{\sum_j e^{z_j^{(x,\theta)}}} & \text{if output unit } o \text{ is active} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

whereby $z_o^{(x,\theta)}$ was the logit of output class o obtained by putting input x through the neural network with parameters θ. The summation in the denominator was over all active classes in the output layer. Importantly, with task- and class-incremental learning, output class o refers to the 'global class' that is obtained by combining the within-context label y and the context label c (that is, the set of global classes is given by $\mathcal{G} = \mathcal{Y} \times \mathcal{C}$). With domain-incremental learning, output class o refers to the within-context label y.

Data stream

All experiments in this article used the academic continual learning setting, meaning that the different contexts were presented to the algorithm one after the other. Within each context, the training data was fed to the algorithm in a stream of independent and identically distributed experiences (or iterations). For Split MNIST, each context was trained for 2,000 iterations with mini-batch size of 128. For Split CIFAR-100, there were 5,000 iterations per context with mini-batch size of 256. Some of the compared methods (EWC, FROMP and iCaRL) performed an additional pass over each context's training data upon finishing training on that context.

Loss function and optimization

For all compared methods, the parameters of the neural network were sequentially trained on each context by optimizing a loss function (denoted by $\mathcal{L}_{\text{total}}$) using stochastic gradient descent. In each iteration, the loss was calculated as the average over all samples in the mini-batch and a single gradient step was taken with the ADAM-optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$; ref. [70]) and a learning rate of either 0.001 (Split MNIST) or 0.0001 (Split CIFAR-100).

For most compared methods, a central component of the loss function was the multi-class cross-entropy classification loss on the data of the current context. For an input x labeled with a hard target o, this classification loss was given by:

$$\mathcal{L}^{\text{C}}(x, o; \theta) = -\log p_\theta(o|x) \quad (3)$$

with p_θ the conditional probability distribution defined by the neural network with parameters θ, as given in equation (2).

Memory buffer and generative models

Several of the compared methods (FROMP, ER, A-GEM and iCaRL) maintained a memory buffer in which examples of previously seen classes were stored. Except for the experiments in Extended Data Fig. 1, 100 examples per class were allowed to be stored in the memory buffer (that is, the per-class memory budget B was set to 100). Some other methods (DGR, BI-R and the Generative Classifier) learned generative models, these methods used up to three times as many parameters compared with the other methods.

Baselines

For the baseline 'None', which was included as a lower target, the base neural network was sequentially trained on each context in the standard way, meaning that the loss function to be optimized was always just the classification loss on the current data (that is, $\mathcal{L}_{\text{total}} = \mathcal{L}^{\text{C}}$).

For the baseline 'Joint', which was included as an upper target, the base neural network was trained on the data from all contexts at the same time. For this baseline, the same total number of iterations was used as with the sequential training protocol (that is, 5×2,000 iterations for Split MNIST and 10×5,000 iterations for Split CIFAR), but each mini-batch was always sampled jointly from the data of all contexts.

Approaches using context-specific components

For XdG and the 'Separate Networks' approach, not all parts of the network were used for each context. These approaches require knowledge of which context a sample belongs to (to select the correct context-specific components), which meant that they could only be used in the task-incremental learning scenario. For both approaches, training was performed using just the classification loss on the current data (that is, $\mathcal{L}_{\text{total}} = \mathcal{L}^{\text{C}}$).

In the task-incremental learning scenario, the other methods (that is, all methods except XdG and Separate Networks) used the available context identity information only in the form of a separate output layer for each context. This is a common and often sensible way to use context identity information, although in Supplementary Note 7 we show that sometimes it is more efficient to use context identity information in other ways.

Separate Networks. For the Separate Networks approach, the available parameter budget was equally divided over all contexts to be learned, and a separate sub-network was trained for each context. For Split MNIST, each context-specific sub-network had two fully connected hidden layers of 100 ReLU each and a softmax output layer. For Split CIFAR-100, the pre-trained and frozen convolutional layers were shared between all contexts, and only the fully connected part of the network was split up into context-specific sub-networks. Each context-specific sub-network had two fully connected layers with 400 ReLU each and a softmax output layer.

XdG. With XdG[12], the base neural network was used and for each context a different, randomly selected subset of X% of the units in each hidden layer was fully gated (that is, their activations were set to zero), with X a hyperparameter whose value was set by a grid search (Supplementary Note 8).

Parameter regularization methods

For the parameter regularization methods EWC and SI, a regularization term was added to the classification loss: $\mathcal{L}_{\text{total}} = \mathcal{L}^{\text{C}} + \mathcal{L}_{\text{param-reg}}$. This regularization term penalized changes to parameters thought to be important for previously learned contexts.

EWC. The regularization term of EWC[49] consisted of a quadratic penalty term for each previously learned context, whereby the term of each context penalized parameters for how different they were compared to their value directly after finishing training on that context. When training on context $K > 1$, the EWC regularization term was given by:

$$\mathcal{L}^{(K)}_{\text{param-reg}}(\theta) = \lambda \sum_{k=1}^{K-1} \left(\frac{1}{2} \sum_{i=1}^{N_{\text{params}}} F^{(k)}_{ii} \left(\theta_i - \hat{\theta}^{(k)}_i \right)^2 \right) \tag{4}$$

with λ a hyperparameter controlling the regularization strength (which was set based on a grid search, Supplementary Note 8), $\hat{\theta}^{(k)}_i$ the value of the i^{th} parameter at the end of training on context k, and $F^{(k)}_{ii}$ the estimated importance of parameter i for context k. This importance estimate was calculated as the i^{th} diagonal element of the Fisher information matrix of context k:

$$F^{(k)}_{ii} = \frac{1}{|S^{(k)}|} \sum_{x \in S^{(k)}} \left(\sum_o \hat{o}^{(x)}_k \left(\frac{\delta \log p_\theta(o|x)}{\delta \theta_i} \Big|_{\theta = \hat{\theta}^{(k)}} \right)^2 \right) \tag{5}$$

whereby $S^{(k)}$ was the training data of context k and $\hat{o}^{(x)}_k$ was the probability that x belongs to output class o, as predicted by the network after finishing training on context k—that is, $\hat{o}^{(x)}_k = p_{\hat{\theta}^{(k)}}(o|x)$. The inner summation in equation (5) was over all output classes that were active during training on context k.

SI. The regularization term of SI (ref. [31]) consisted of a single quadratic term that penalized changes to the parameters away from the value they had after finishing training on the previous context. When training on context $K > 1$, the SI regularization term was given by:

$$\mathcal{L}^{(K)}_{\text{param-reg}}(\theta) = \gamma \sum_{i=1}^{N_{\text{params}}} \Omega^{(K-1)}_i \left(\theta_i - \hat{\theta}^*_i \right)^2 \tag{6}$$

with γ a hyperparameter controlling the regularization strength (which was set based on a grid search, see Supplementary Note 8), $\hat{\theta}^*_i$ the value of the i^{th} parameter at the end of training on context $K-1$, and $\Omega^{(K-1)}_i$ the estimated importance of parameter i after the first $K-1$ contexts have been learned. To compute these parameter importance estimates, after each context k, a per-parameter contribution to the change of the loss was calculated for each parameter i as follows:

$$\omega^{(k)}_i = \sum_{t=1}^{N_{\text{iters}}} \left(\theta_i[t^{(k)}] - \theta_i[(t-1)^{(k)}] \right) \frac{-\delta \mathcal{L}_{\text{total}}[t^{(k)}]}{\delta \theta_i} \tag{7}$$

with N_{iters} the number of iterations per context, $\theta_i[t^{(k)}]$ the value of parameter i after the t^{th} training iteration on context k and $\frac{\delta \mathcal{L}_{\text{total}}[t^{(k)}]}{\delta \theta_i}$ the gradient of the loss with respect to parameter i during the t^{th} training iteration on context k. For every context, these per-parameter contributions were normalized by the square of the total change of that parameter during training on that context 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 were summed over all contexts so far:

$$\Omega^{(K-1)}_i = \sum_{k=1}^{K-1} \frac{\omega^{(k)}_i}{\left(\Delta^{(k)}_i \right)^2 + \xi} \tag{8}$$

with $\Delta^{(k)}_i = \theta_i[N_{\text{iters}}{}^{(k)}] - \theta_i[0^{(k)}]$ where $\theta_i[0^{(k)}]$ was the value of parameter i right before starting training on context k.

Functional regularization methods

Similar as with parameter regularization, the functional regularization methods LwF and FROMP had a regularization term added to the classification loss: $\mathcal{L}_{\text{total}} = \mathcal{L}^{\text{C}} + \mathcal{L}_{\text{func-reg}}$. This regularization term encouraged the input–output mapping of the network not to change too much at a set of anchor points.

LwF. The method LwF (ref. [50]) used the inputs from the current context as anchor points in combination with knowledge distillation[71]. During training on context $K > 1$, the LwF regularization term was given by:

$$\mathcal{L}^{(K)}_{\text{func-reg}}(x, \theta) = -\sum_{k=1}^{K-1} \sum_{o \in \mathcal{O}_k} p^T_{\hat{\theta}^*}(o|x) \log \left[p^T_\theta(o|x) \right] \tag{9}$$

whereby $\mathcal{O}_k$ was the set of output classes in context k, $\hat{\theta}^*$ was the parameter vector with values as they were at the end of training on context $K-1$ and $p^T_\theta(o|x)$ was the 'temperature-raised' probability that input $\boldsymbol{x}$ belongs to output class o, as predicted by the network with parameters θ. These temperature-raised probabilities were defined as:

$$p^T_\theta(o|x) = \frac{\exp\left[z^{(x,\theta)}_o / T \right]}{\sum_j \exp\left[z^{(x,\theta)}_j / T \right]} \tag{10}$$

with T the temperature, which was set to 2, and $z^{(x,\theta)}_o$ the logit of output class o obtained by putting input $\boldsymbol{x}$ through the neural network with parameters θ. The summation in the denominator was over all active classes in the output layer. With task-incremental learning, for each context's term in the outer summation of equation (9), only the output classes contained in that context were active. With domain- and class-incremental learning, always all output classes were active. In each iteration, the LwF regularization term was computed as average over the same inputs that were used to compute $\mathcal{L}^{\text{C}}$.

We note that this implementation of LwF differs slightly from the implementation of LwF used in ref. [28]. Compared with that implementation, the regularization term here was weighted less strongly, which substantially improved the performance of LwF on Split CIFAR-100. Initial experiments indicated that by reducing the weight of the replay term in equation (16) it is also be possible to improve the performance

of several of the replay methods on Split CIFAR-100, but at the cost of impaired performance on Split MNIST.

FROMP. The method FROMP (ref. [51]) performed functional regularization in a Bayesian framework and used stored data from previous contexts, referred to as memorable inputs, as anchor points. During training on context K, the regularization term of FROMP was given by:

$$\mathcal{L}^{(K)}_{\text{func-reg}}(\theta) = \frac{1}{2}\tau\sum_{k=1}^{K-1}\sum_{o\in\mathcal{O}_k}\left(m^{(\theta)}_{k,o} - m^{(\hat{\theta}^*)}_{k,o}\right)^T \mathbf{K}^{(K-1)^{-1}}_{k,o}\left(m^{(\theta)}_{k,o} - m^{(\hat{\theta}^*)}_{k,o}\right) \quad (11)$$

with τ a hyperparameter controlling the regularization strength (which was set based on a grid search, see Supplementary Note 8) and $\hat{\theta}^*$ the parameter vector with values as they were at the end of training on context $K-1$. Further, $m^{(\theta)}_{k,o}$ was a vector containing for each memorable input from context k the probability that this input belongs to output class o as predicted by the network with parameters θ. That is, the i^{th} element of $m^{(\theta)}_{k,o}$ was given by $m^{(\theta)}_{k,o}[i] = p_\theta(o|x^{(i,k)})$, with $x^{(i,k)}$ the i^{th} memorable input of context k. Finally, $\mathbf{K}^{(K)}_{k,o}$ was a matrix whose elements were given by:

$$\mathbf{K}^{(K)}_{k,o}[i,j] = g_{k,o}[i]\mathbf{V}^{(K)}g_{k,o}[j]^T \quad (12)$$

with:

$$g_{k,o}[i] = \left.\frac{\delta p_\theta(o|x^{(i,k)})}{\delta\theta}\right|_{\theta=\hat{\theta}^*} \quad (13)$$

and $\mathbf{V}^{(K)}$ was a diagonal matrix with diagonal $v^{(K)}$ given by:

$$\frac{1}{v^{(K)}} = \sum_{k=1}^{K}\sum_{x\in\mathcal{D}_k}\text{diag}\left(\mathbf{J}^{(k)}(x)^T\Lambda^{(k)}(x)\mathbf{J}^{(k)}(x)\right) \quad (14)$$

whereby $\mathbf{J}^{(k)}(x) = \left.\frac{\delta f_\theta(x)}{\delta\theta}\right|_{\theta=\hat{\theta}^{(k)}}$, with $f_\theta(x)$ the logits obtained by putting input x through the neural network with parameters θ, and $\Lambda^{(k)}(x)[i,j] = p_{\theta^{(k)}}(i|x)(1-p_{\theta^{(k)}}(j|x))$.

The selection of memorable inputs, which are FROMP's anchor points, took place after finishing training on each context. After finishing on context k, for each input x in that context's training set, a relevance score was calculated as:

$$r(x) = \sum_{o\in\mathcal{O}_k} p_\theta(o|x)(1-p_\theta(o|x)) \quad (15)$$

whereby $\mathcal{O}_k$ was the set of output classes in context k and θ were the parameters after training on context k. Then, for each output class in context k, the B inputs with the highest relevance scores were selected as the memorable inputs for that class and stored in the memory buffer.

Replay-based methods

The replay-based methods had two separate loss terms: one for the data of the current context, denoted as $\mathcal{L}_{\text{current}}$, and one for the replayed data, denoted as $\mathcal{L}_{\text{replay}}$. Except with A-GEM, during training the objective was to optimize an overall loss function that was a weighted sum of these two terms, with the weights depending on how many contexts had been seen so far:

$$\mathcal{L}_{\text{total}} = \frac{1}{N_{\text{contexts so far}}}\mathcal{L}_{\text{current}} + \left(1-\frac{1}{N_{\text{contexts so far}}}\right)\mathcal{L}_{\text{replay}} \quad (16)$$

In each iteration, the number of replayed samples was always equal to the number of samples from the current context (that is, 128 for Split MNIST and 256 for Split CIFAR-100).

ER. With ER, the term $\mathcal{L}_{\text{current}}$ was the standard classification loss on the data of the current context (that is, $\mathcal{L}_{\text{current}} = \mathcal{L}^{\text{C}}$). The term $\mathcal{L}_{\text{replay}}$ was also the standard classification loss, but on the replayed data. In each iteration, the samples to be replayed were randomly sampled from the memory buffer. The memory buffer was updated after each context, when for each new class B samples were randomly selected from the training data and added to the buffer.

A-GEM. For the method A-GEM (ref. [54]), the loss terms $\mathcal{L}_{\text{current}}$ and $\mathcal{L}_{\text{replay}}$ were defined similarly as for ER. The population of the memory buffer and sampling of the the data to be replayed from the memory buffer were also the same. The only difference compared to ER was that with A-GEM, the objective was not to minimize the combined loss (that is, $\mathcal{L}_{\text{total}}$), but instead the objective was to minimize the loss on the current data (that is, $\mathcal{L}_{\text{current}}$) under the constraint that the loss on the replayed data (that is, $\mathcal{L}_{\text{replay}}$) did not increase. To achieve this, in every iteration, the gradient vector that was used to update the parameters (that is, the gradient vector that was put into the ADAM-optimizer) was required to have a positive angle with the gradient of $\mathcal{L}_{\text{replay}}$. Therefore, whenever the angle between the gradient of $\mathcal{L}_{\text{current}}$ and the gradient of $\mathcal{L}_{\text{replay}}$ was negative, the gradient of $\mathcal{L}_{\text{current}}$ was projected onto the orthogonal complement of the gradient of $\mathcal{L}_{\text{replay}}$. Let $\mathcal{B}_{\text{current}}$ be the mini-batch of data from the current context and $\mathcal{B}_{\text{replay}}$ the mini-batch of replayed data from the memory buffer. The gradient of $\mathcal{L}_{\text{current}}$ was then:

$$g_{\text{current}} = \frac{1}{|\mathcal{B}_{\text{current}}|}\sum_{(x,o)\in\mathcal{B}_{\text{current}}}\frac{\delta\mathcal{L}^{\text{C}}(x,o;\theta)}{\delta\theta} \quad (17)$$

and the gradient of $\mathcal{L}_{\text{replay}}$ was given by:

$$g_{\text{replay}} = \frac{1}{|\mathcal{B}_{\text{replay}}|}\sum_{(x,o)\in\mathcal{B}_{\text{replay}}}\frac{\delta\mathcal{L}^{\text{C}}(x,o;\theta)}{\delta\theta} \quad (18)$$

The gradient g^* used to update the parameters was then given by:

$$g^* = \begin{cases} g_{\text{current}} & \text{if } g_{\text{current}}{}^T g_{\text{replay}} \geq 0 \\ g_{\text{current}} - \frac{g_{\text{current}}{}^T g_{\text{replay}}}{(g_{\text{replay}}{}^T g_{\text{replay}}+\gamma)} g_{\text{replay}} & \text{otherwise,} \end{cases} \quad (19)$$

with γ a small constant to ensure numerical stability. In A-GEM's original formulation[54] there was no γ-term, but we found that without it, performance was unstable. We used $\gamma = 1\times10^{-7}$.

DGR. With DGR[27], two neural networks were sequentially trained on all contexts: a classifier, for which we used the base neural network, and a separate generative model.

For training of the classifier, as with ER and A-GEM, $\mathcal{L}_{\text{current}}$ and $\mathcal{L}_{\text{replay}}$ were the standard classification loss on the data of the current context and the replayed data, respectively. With DGR, the replayed data was obtained by sampling inputs from a copy of the generative model and labelling them as the most likely class predicted for those inputs by a copy of the classifier. The samples replayed during context K were generated by copies of the generator and classifier stored directly after finishing training on context $K-1$. With task-incremental learning, each replayed sample was labelled and evaluated separately for all previous contexts and $\mathcal{L}_{\text{replay}}$ was the average over those contexts.

As generative model a variational autoencoder (VAE; ref. [72]) was used, which consisted of an encoder network q_ϕ that mapped an input-vector x to a vector of latent variables z, and a decoder network p_ψ that mapped those latent variables back to a reconstructed or decoded input-vector $\hat{x}$. The architecture of these two networks was kept similar to that of the base neural network: for Split MNIST, the encoder and the decoder were both fully connected networks with two hidden layers of 400 ReLU each; for Split CIFAR-100, the encoder consisted of the same five pre-trained convolutional layers as the base neural network followed by two fully connected layers with 2,000 ReLU units, and the decoder consisted of two fully connected layers with

2,000 ReLU followed by five deconvolutional (or transposed convolutional) layers[73] that mirrored the convolutional layers and contained 128, 64, 32, 16 and 3 channels. The first four deconvolutional layers used a 4×4 kernel, a padding of 1 and a stride of 2 (that is, image size was doubled in each of those layers), while the final layer used a 3×3 kernel, a padding of 1 and a stride of 1 (that is, no upsampling). Batch-norm and ReLU non-linearities were used in all deconvolutional layers except for the last one. For both context sets, the VAE's latent variable layer z had 100 Gaussian units. The prior over the latent variables was the standard normal distribution.

For a given input x, the loss function for training the parameters of the VAE was:

$$\mathcal{L}^{\mathrm{G}}(x;\phi,\psi) = \mathcal{L}^{\mathrm{latent}}(x;\phi) + \mathcal{L}^{\mathrm{recon}}(x;\phi,\psi) \tag{20}$$

The first term in equation (20), the 'latent variable regularization term', was given by:

$$\mathcal{L}^{\mathrm{latent}}(x;\phi) = \frac{1}{2}\sum_{j=1}^{N_{\mathrm{latent}}}\left(1+\log(\sigma_j^{(x)^2}) - \mu_j^{(x)^2} - \sigma_j^{(x)^2}\right) \tag{21}$$

with N_{latent} the number of latent variables, and $\mu_j^{(x)}$ and $\sigma_j^{(x)}$ the j^{th} elements of $\mu^{(x)}$ and $\sigma^{(x)}$, which were the outputs of the encoder network q_ϕ for input x. The second term in equation (20), the 'reconstruction term', was given by the squared error between the original and decoded pixel values:

$$\mathcal{L}^{\mathrm{recon}}(x;\phi,\psi) = \sum_{p=1}^{N_{\mathrm{pixels}}}\left(x_p - \tilde{x}_p\right)^2 \tag{22}$$

whereby x_p was the value of the p^{th} pixel of the original input image x and $\tilde{x}_p$ was the value of the p^{th} pixel of the decoded image $\tilde{x} = p_\psi(z^{(x)})$, with $z^{(x)} = \mu^{(x)} + \sigma^{(x)} \times \epsilon$ and ϵ sampled from $\mathcal{N}(0, I)$.

Training of the generative model was also done with generative replay, which was provided by its own copy stored after finishing training on the previous context. The loss terms of the current and replayed data were weighted similarly to the classifier:

$$\mathcal{L}^{\mathrm{G}}_{\mathrm{total}} = \frac{1}{N_{\mathrm{contexts\,so\,far}}}\mathcal{L}^{\mathrm{G}}_{\mathrm{current}} + \left(1-\frac{1}{N_{\mathrm{contexts\,so\,far}}}\right)\mathcal{L}^{\mathrm{G}}_{\mathrm{replay}} \tag{23}$$

BI-R. For the method BI-R, we followed the protocol as described in the original paper[28]. For Split CIFAR-100, all five of the proposed modifications relative to DGR were used: distillation, replay-through-feedback, conditional replay, gating based on internal context and internal replay. For Split MNIST, internal replay was not used, but the other four components were used. We did not combine BI-R with SI. The hyperparameter X, which controlled the proportion of hidden units in the decoder that was gated per class, was set based on a grid search (Supplementary Note 8).

Compared with ref. [28] there were two slight differences: (1) here we used a different set of pre-trained convolutional layers for each random seed, while ref. [28] always used the same pre-trained convolutional layers; and (2) in the class-incremental learning scenario, here we used a softmax layer with the output units of all classes always set to active, while ref. [28] used an 'expanding head' (that is, only the output units of classes seen so far were set to active).

Template-based classification methods

Although for the context sets considered in this article, the template-based classification methods could, in theory, be used for all three continual learning scenarios, we considered them only for class-incremental learning. This was because, from an incremental-learning perspective, the specific benefit of template-based classification (that is, rephrasing a class-incremental learning problem as a task-incremental learning problem, see Supplementary Note 4) is only relevant in that scenario.

Generative classifier. For the generative classifier[55], a separate VAE model was trained for each class to be learned. Training of these models was done as described above for DGR, except that no replay was used and each VAE was only trained on the examples from its own class. Each class-specific VAE was trained for either 1,000 iterations (Split MNIST) or 500 iterations (Split CIFAR-100), which meant that the total number of training iterations was the same as for the other methods. The mini-batch size was also the same: 128 for Split MNIST and 256 for Split CIFAR-100.

The architecture of the VAE models was chosen so that the total number of parameters of the generative classifier was similar to the number of parameters used by generative replay. For Split MNIST, the encoder and the decoder were both fully connected networks with two hidden layers of 85 ReLU units each and the latent variable layer had five units. For Split CIFAR-100, the pre-trained convolutional layers were used as a feature extractor, and the VAE models were trained on the extracted features rather than on the raw inputs (that is, the reconstruction loss was in the feature space instead of at the pixel level). The encoder and decoder both had one fully connected hidden layer with 85 ReLU and a latent variable layer with 20 units.

Classification was performed based on Bayes' rule: a test sample was classified as the class under whose generative model it was estimated to be the most likely. That is, the output class label o^* predicted for an input x was given by:

$$o^* = \arg\max_o\, p(x|o) \tag{24}$$

whereby $p(\boldsymbol{x}|o)$ was the likelihood of input x under the generative model of class o. These likelihoods were estimated using importance sampling[74]:

$$p(x|o) = \frac{1}{S}\sum_{s=1}^{S}\frac{f\left(x\middle|\mu_{\psi_o}^{(z^{(s)})}, I\right)f\left(z^{(s)}\middle|\,0, I\right)}{f\left(z^{(s)}\middle|\,\mu_{\phi_o}^{(x)}, \sigma_{\phi_o}^{(x)^2} I\right)} \tag{25}$$

with $\mu_{\phi_o}^{(x)}$ and $\sigma_{\phi_o}^{(x)}$ the outputs of the encoder network for input x, $\mu_{\psi_o}^{(z)}$ the output of the decoder network for input z, S the number of importance samples and $z^{(s)}$ the s^{th} importance sample drawn from $\mathcal{N}\left(\mu_{\phi_o}^{(x)}, \sigma_{\phi_o}^{(x)^2} I\right)$. In this notation, $f(x\,|\,\mu,\Sigma)$ indicates the probability density of x under the multivariate normal distribution with mean μ and covariance matrix Σ. Similar to ref. [55], we used $S = 10{,}000$ importance samples per likelihood estimation.

iCaRL. The method iCaRL (ref. [25]) used a neural network for feature extraction and then performed classification based on a nearest-class-mean rule in that feature space, whereby the class means were calculated from stored data. To protect the feature extractor network from becoming unsuitable for previously learned contexts, iCaRL also replayed the stored data—as well as the inputs from the current context with a special form of distillation—during training of the feature extractor.

For the feature extractor we used the base neural network, except with the softmax output layer removed. We denote this feature extractor by $\psi_\phi(.)$, with trainable parameters ϕ. These parameters were trained based on a binary classification/distillation loss. For this, during training only, a sigmoid output layer was appended to ψ_ϕ. The resulting extended network outputs for any output class $o \in \{1, \ldots, N_{\mathrm{classes\,so\,far}}\}$ a binary probability whether input x belongs to it:

$$p_\theta^o(x) = \frac{1}{1+\mathrm{e}^{-w_o^T\psi_\phi(x)}} \tag{26}$$

with $\theta = (\phi, w_1, \ldots, w_{N_{\text{classes so far}}})$ a vector containing all the trainable parameters of iCaRL. Whenever a new output class o was encountered, new parameters w_o were added to θ.

In each context, the parameters in θ were trained on an extended dataset containing the current context's training data as well as all stored data in the memory buffer. When training on context K, each input x with hard target o in this extended dataset was paired with a new target-vector $\bar{o}$ whose j^{th} element was given by:

$$\bar{o}_j = \begin{cases} p^j_{\hat{\theta}^*}(x) & \text{if output class } j \text{ in context } 1, \ldots, K-1 \\ \mathbb{1}_{\{o=j\}} & \text{if output class } j \text{ in context } K \end{cases} \quad (27)$$

whereby $\hat{\theta}^*$ is the vector with parameter values at the end of training on context $K-1$. The binary classification/distillation loss function for an input x labelled with such an 'old-context-soft-target/new-context-hard-target' vector $\bar{o}$ was then given by:

$$\mathcal{L}_{\text{iCaRL}}(x, \bar{o}; \theta) = -\sum_{j=1}^{N_{\text{classes so far}}} \left[\bar{o}_j \log p^j_\theta(x) + (1-\bar{o}_j) \log\left(1 - p^j_\theta(x)\right)\right] \quad (28)$$

After finishing training on a context, data to be added to the memory buffer were selected as follows. For each new output class o, iteratively B samples (or 'exemplars') were selected based on their extracted feature vectors according to a procedure referred to as 'herding'. In each iteration, a new sample from output class o was selected such that the average feature vector over all selected examples was as close as possible to the average feature vector over all available examples of class o. Let $\mathcal{X}^o = \{x_1, \ldots, x_{N_o}\}$ be the set of all available examples of class o and let $\mu^o = \frac{1}{N_o}\sum_{x \in \mathcal{X}^o} \psi_\phi(x)$ be the average feature vector over set $\mathcal{X}^o$. The n^{th} exemplar (for $n = 1, \ldots, m$) to be selected for output class o was then given by:

$$p^o_n = \underset{x \in \mathcal{X}^o}{\arg\min} \left\| \mu^o - \frac{1}{n}\left(\psi_\phi(x) + \sum_{i=1}^{n-1} \psi_\phi(p^o_i)\right)\right\| \quad (29)$$

This resulted in ordered exemplar-sets $\mathcal{P}^o = \{p^o_1, \ldots, p^o_m\}$ for each new output class o that were stored in the memory buffer.

Finally, classification was performed based on a nearest-class-mean rule in feature space, whereby the class means were calculated from the stored exemplars. For this, let $\mu_o = \frac{1}{|\mathcal{P}^o|}\sum_{p \in \mathcal{P}^o} \psi_\phi(p)$ for $o = 1, \ldots, N_{\text{classes so far}}$. The output class label o^* predicted for a new input x was then given by:

$$o^* = \underset{o=1,\ldots,N_{\text{classes so far}}}{\text{argmin}} \left\| \psi_\phi(x) - \mu_o \right\| \quad (30)$$

Data availability

All datasets used in this study are freely available online resources: http://yann.lecun.com/exdb/mnist/ (MNIST[66]) and https://www.cs.toronto.edu/~kriz/cifar.html (CIFAR-10 and CIFAR-100[67]).

Code availability

Documented code that can be used to reproduce or build upon the reported experiments is available online under an MIT licence: https://github.com/GMvandeVen/continual-learning[75].

References

1. Chen, Z. & Liu, B. Lifelong machine learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **12**, 1–207 (2018).
2. Hadsell, R., Rao, D., Rusu, A. A. & Pascanu, R. Embracing change: continual learning in deep neural networks. *Trends Cognit. Sci.* **24**, 1028–1040 (2020).
3. McCloskey, M. & Cohen, N. J. In *Psychology of Learning and Motivation* Vol. 24, 109–165 (Elsevier, 1989).
4. French, R. M. Catastrophic forgetting in connectionist networks. *Trends Cognit. Sci.* **3**, 128–135 (1999).
5. Kudithipudi, D. et al. Biological underpinnings for lifelong learning machines. *Nat. Mach. Intell.* **4**, 196–210 (2022).
6. Lee, C. S. & Lee, A. Y. Clinical applications of continual learning machine learning. *Lancet Digital Health* **2**, e279–e281 (2020).
7. Shaheen, K., Hanif, M. A., Hasan, O. & Shafique, M. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. *J. Intell. Robot. Syst.* **105**, 9 (2022).
8. Philps, D., Weyde, T., Garcez, A. d. & Batchelor, R. Continual learning augmented investment decisions. Preprint at https://arxiv.org/abs/1812.02340 (2018).
9. Mundt, M., Lang, S., Delfosse, Q. & Kersting, K. CLEVA-compass: A continual learning evaluation assessment compass to promote research transparency and comparability. In *International Conference on Learning Representations* (2022).
10. Marr, D. Vision: A computational investigation into the human representation and processing of visual information (WH Freeman, 1982).
11. Ruvolo, P. & Eaton, E. ELLA: An efficient lifelong learning algorithm. In *International Conference on Machine Learning* 507–515 (PMLR, 2013).
12. Masse, N. Y., Grant, G. D. & Freedman, D. J. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proc. Natl Acad. Sci. USA* **115**, E10467–E10475 (2018).
13. Ramesh, R. & Chaudhari, P. Model Zoo: A growing brain that learns continually. In *International Conference on Learning Representations* (2022).
14. Lopez-Paz, D. & Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems* Vol. 30, 6470–6479 (2017).
15. Vogelstein, J. T. et al. Representation ensembling for synergistic lifelong learning with quasilinear complexity. Preprint at https://arxiv.org/abs/2004.12908 (2020).
16. Ke, Z., Liu, B., Xu, H. & Shu, L. CLASSIC: Continual and contrastive learning of aspect sentiment classification tasks. In *Proc. 2021 Conference on Empirical Methods in Natural Language Processing* 6871–6883 (Association for Computational Linguistics, 2021).
17. Mirza, M. J., Masana, M., Possegger, H. & Bischof, H. An efficient domain-incremental learning approach to drive in all weather conditions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* 3001–3011 (2022).
18. Aljundi, R., Chakravarty, P. & Tuytelaars, T. Expert gate: Lifelong learning with a network of experts. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 3366–3375 (2017).
19. von Oswald, J., Henning, C., Sacramento, J. & Grewe, B. F. Continual learning with hypernetworks. In *International Conference on Learning Representations* (2020).
20. Wortsman, M. et al. Supermasks in superposition. In *Advances in Neural Information Processing Systems* Vol. 33, 15173–15184 (2020).
21. Henning, C. et al. Posterior meta-replay for continual learning. In *Advances in Neural Information Processing Systems* Vol. 34, 14135–14149 (2021).
22. Verma, V. K., Liang, K. J., Mehta, N., Rai, P. & Carin, L. Efficient feature transformations for discriminative and generative continual learning. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 13865–13875 (2021).
23. Heald, J. B., Lengyel, M. & Wolpert, D. M. Contextual inference underlies the learning of sensorimotor repertoires. *Nature* **600**, 489–493 (2021).
24. Lomonaco, V. & Maltoni, D. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning* 17–26 (PMLR, 2017).

25. Rebuffi, S.-A., Kolesnikov, A., Sperl, G. & Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 2001–2010 (2017).
26. Tao, X. et al. Few-shot class-incremental learning. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 12183–12192 (2020).
27. Shin, H., Lee, J. K., Kim, J. & Kim, J. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems* Vol. 30, 2994–3003 (2017).
28. van de Ven, G. M., Siegelmann, H. T. & Tolias, A. S. Brain-inspired replay for continual learning with artificial neural networks. *Nat. Commun.* **11**, 4069 (2020).
29. Belouadah, E., Popescu, A. & Kanellos, I. A comprehensive study of class incremental learning algorithms for visual tasks. *Neural Networks* **135**, 38–54 (2021).
30. Masana, M. et al. Class-incremental learning: survey and performance evaluation on image classification. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (IEEE, 2022). https://doi.org/10.1109/TPAMI.2022.3213473
31. Zenke, F., Poole, B. & Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning* 3987–3995 (PMLR, 2017).
32. Zeng, G., Chen, Y., Cui, B. & Yu, S. Continual learning of context-dependent processing in neural networks. *Nat. Mach. Intell.* **1**, 364–372 (2019).
33. Aljundi, R., Kelchtermans, K. & Tuytelaars, T. Task-free continual learning. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 11254–11263 (2019).
34. Zeno, C., Golan, I., Hoffer, E. & Soudry, D. Task agnostic continual learning using online variational bayes. Preprint at https://arxiv.org/abs/1803.10123v3 (2019).
35. Rao, D. et al. Continual unsupervised representation learning. In *Advances in Neural Information Processing Systems* Vol. 32, 7647–7657 (2019).
36. De Lange, M. & Tuytelaars, T. Continual prototype evolution: Learning online from non-stationary data streams. In *Proc. IEEE/CVF International Conference on Computer Vision* 8250–8259 (2021).
37. Li, S., Du, Y., van de Ven, G. M. & Mordatch, I. Energy-based models for continual learning. Preprint at https://arxiv.org/abs/2011.12216 (2020).
38. Hayes, T. L. & Kanan, C. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* 220–221 (2020).
39. Mai, Z. et al. Online continual learning in image classification: An empirical survey. *Neurocomputing* **469**, 28–51 (2022).
40. Lesort, T., Caccia, M. & Rish, I. Understanding continual learning settings with data distribution drift analysis. Preprint at https://arxiv.org/abs/2104.01678 (2021).
41. Lomonaco, V. et al. Avalanche: an end-to-end library for continual learning. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* 3600–3610 (2021).
42. Gepperth, A. & Hammer, B. Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)* (2016).
43. Stojanov, S. et al. Incremental object learning from contiguous views. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 8777–8786 (2019).
44. Caccia, L., Belilovsky, E., Caccia, M. & Pineau, J. Online learned continual compression with adaptive quantization modules. In *International Conference on Machine Learning* 1240–1250 (PMLR, 2020).
45. Cossu, A. et al. Is class-incremental enough for continual learning? *Front. Artif. Intell.* **5**, 829842 (2022).
46. Lee, S., Ha, J., Zhang, D. & Kim, G.A neural dirichlet process mixture model for task-free continual learning. In *International Conference on Learning Representations* (2020).
47. Jin, X., Sadhu, A., Du, J. & Ren, X. Gradient-based editing of memory examples for online task-free continual learning. In *Advances in Neural Information Processing Systems* Vol. 34, 29193–29205 (2021).
48. Shanahan, M., Kaplanis, C. & Mitrović, J. Encoders and ensembles for task-free continual learning. Preprint at https://arxiv.org/abs/2105.13327 (2021).
49. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl Acad. Sci. USA* **114**, 3521–3526 (2017).
50. Li, Z. & Hoiem, D. Learning without forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.* **40**, 2935–2947 (2017).
51. Pan, P. et al. Continual deep learning by functional regularisation of memorable past. In *Advances in Neural Information Processing Systems* Vol. 33, 4453–4464 (2020).
52. Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. & Wayne, G. Experience replay for continual learning. In *Advances in Neural Information Processing Systems* Vol. 32 (2019).
53. Chaudhry, A. et al. On tiny episodic memories in continual learning. Preprint at https://arxiv.org/abs/1902.10486 (2019).
54. Chaudhry, A., Ranzato, M., Rohrbach, M. & Elhoseiny, M. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations* (2019).
55. van de Ven, G. M., Li, Z. & Tolias, A. S. Class-incremental learning with generative classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* 3611–3620 (2021).
56. Lesort, T., Caselles-Dupré, H., Garcia-Ortiz, M., Stoian, A. & Filliat, D. Generative models from the perspective of continual learning. In *International Joint Conference on Neural Networks* (IEEE, 2019).
57. Aljundi, R. et al. Online continual learning with maximally interfered retrieval. In *Advances in Neural Information Processing Systems* Vol. 32 (2019).
58. van de Ven, G. M. & Tolias, A. S. Generative replay with feedback connections as a general strategy for continual learning. Preprint at https://arxiv.org/abs/1809.10635 (2018).
59. van de Ven, G. M. & Tolias, A. S. Three scenarios for continual learning. Preprint at https://arxiv.org/abs/1904.07734 (2019).
60. Farquhar, S. & Gal, Y. Towards robust evaluations of continual learning. Preprint at https://arxiv.org/abs/1805.09733 (2018).
61. Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A. & Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. Preprint at https://arxiv.org/abs/1312.6211 (2013).
62. Douillard, A. & Lesort, T. Continuum: Simple management of complex continual learning scenarios. Preprint at https://arxiv.org/abs/2102.06253 (2021).
63. Normandin, F. et al. Sequoia: A software framework to unify continual learning research. Preprint at https://arxiv.org/abs/2108.01005 (2021).
64. Hess, T., Mundt, M., Pliushch, I. & Ramesh, V. A procedural world generation framework for systematic evaluation of continual learning. In *Thirty-fifth Conference on Neural Information Processing Systems, Datasets and Benchmarks Track* (2021).
65. Paszke, A. et al. Automatic differentiation in pytorch. In *NeurIPS Autodiff Workshop* (2017).
66. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. et al. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
67. Krizhevsky, A., Hinton, G. et al. *Learning Multiple Layers of Features from Tiny Images* (University of Toronto, 2009).

68. Ioffe, S. & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning* 448–456 (PMLR, 2015).
69. Maltoni, D. & Lomonaco, V. Continuous learning in single-incremental-task scenarios. *Neural Networks* **116**, 56–73 (2019).
70. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. Preprint at https://arxiv.org/abs/1412.6980 (2014).
71. Hinton, G., Vinyals, O. & Dean, J. Distilling the knowledge in a neural network. Preprint at https://arxiv.org/abs/1503.02531 (2015).
72. Kingma, D. P. & Welling, M. Auto-encoding variational bayes. Preprint at https://arxiv.org/abs/1312.6114 (2013).
73. Zeiler, M. D., Taylor, G. W., Fergus, R. et al. Adaptive deconvolutional networks for mid and high level feature learning. In *International Conference on Computer Vision* 2018–2025 (IEEE, 2011).
74. Rezende, D. & Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning* 1530–1538 (PMLR, 2015).
75. van de Ven, G. M. GMvandeVen/continual-learning: v1.0.0 (2022). https://doi.org/10.5281/zenodo.7189378

Acknowledgements

We thank K. Jensen, M. Mundt, W. Barfuss, T. Hess and M. De Lange for insightful comments. This research project was supported by an IBRO-ISN Research Fellowship (to G.M.v.d.V.), by the ERC-funded project KeepOnLearning (reference number 101021347; to T.T.), by the National Institutes of Health (NIH) under awards R01MH109556 (NIH/NIMH; to A.S.T.) and P30EY002520 (NIH/NEI; to A.S.T.), by the Lifelong Learning Machines (L2M) programme of the Defence Advanced Research Projects Agency (DARPA) via contract number HR0011-18-2-0025 (to A.S.T.) and by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003 (to A.S.T.). 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 NIH, DARPA, IARPA, DoI/IBC or the US government.

Author contributions

Conceptualization, G.M.v.d.V, T.T. and A.S.T.; formal analysis, G.M.v.d.V.; funding acquisition, A.S.T., T.T. and G.M.v.d.V.; investigation, G.M.v.d.V.; methodology, G.M.v.d.V.; resources, A.S.T.; software, G.M.v.d.V.; supervision, A.S.T. and T.T.; visualization, G.M.v.d.V.; writing – original draft, G.M.v.d.V.; writing – review and editing, G.M.v.d.V., T.T. and A.S.T.

Competing interests

The authors declare no competing interests.

Additional information

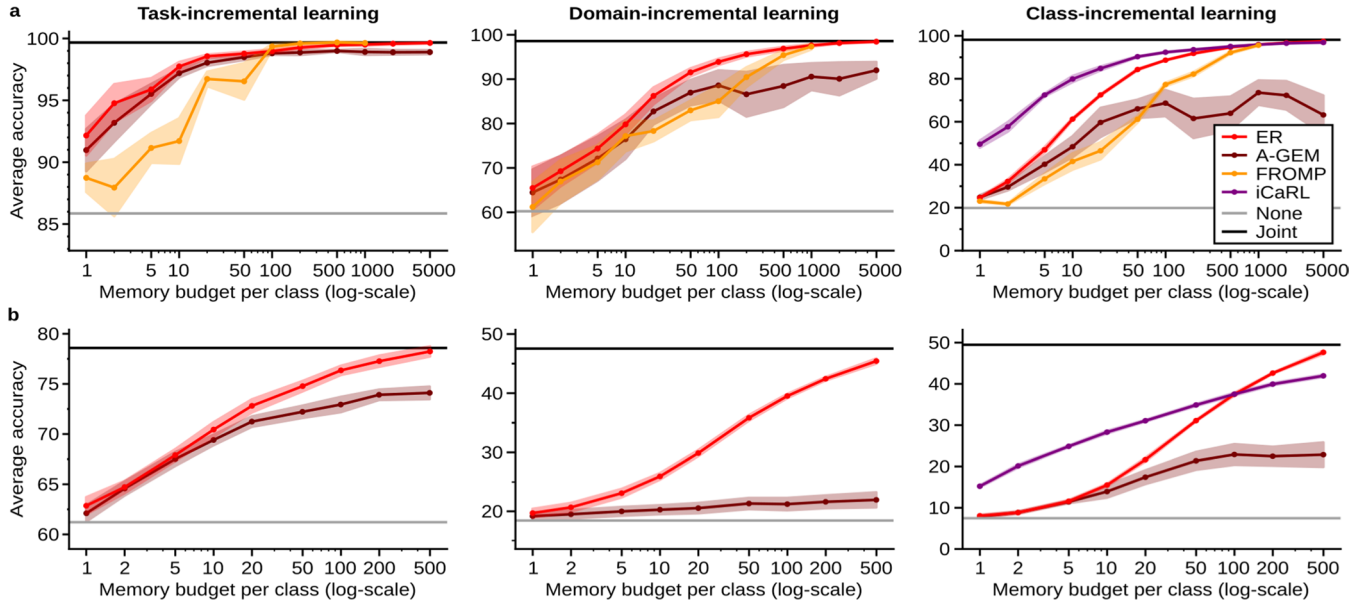
Extended data is available for this paper at https://doi.org/10.1038/s42256-022-00568-3.

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-022-00568-3.

Correspondence and requests for materials should be addressed to Gido M. van de Ven.

Peer review information *Nature Machine Intelligence* thanks Thomas Miconi and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

https://doi.org/10.1038/s42256-022-00568-3



Extended Data Fig. 1 | Comparison of methods using stored data with different buffer sizes. Shown is the average test accuracy (as %, over all contexts) of different methods that use stored data on Split MNIST (**a**) and on Split CIFAR-100 (**b**) as a function of the number of examples per class that is allowed to be stored in memory. Due to its high computational costs, we were not able to run FROMP on Split CIFAR-100, or on Split MNIST with a memory budget above 1,000 samples per class. Displayed are the means over 5 repetitions, shaded areas are ± 1 SEM. ER: exact replay, A-GEM: averaged gradient episodic memory, FROMP: functional regularization of the memorable past, iCaRL: incremental classifier and representation learning, None: sequential training in the standard way, Joint: training on all data at the same time.