Continual Learning in Recurrent Neural Networks with Hypernetworks

Benjamin Ehret*, Christian Henning*, Maria R. Cervera*, Alexander Meulemans, Johannes von Oswald, Benjamin F. Grewe

*Equal contribution

Institute of Neuroinformatics
ETH Zürich and University of Zürich
Zürich, Switzerland
{behret,henningc,mariacer,ameulema,voswaldj,bgrewe}@ethz.ch

Abstract

The last decade has seen a surge of interest in continual learning (CL), and a variety of methods have been developed to alleviate catastrophic forgetting. However, most prior work has focused on tasks with static data, while CL on sequential data has remained largely unexplored. Here we address this gap in two ways. First, we evaluate the performance of established CL methods when applied to recurrent neural networks (RNNs). We primarily focus on elastic weight consolidation, which is limited by a stability-plasticity trade-off, and explore the particularities of this trade-off when using sequential data. We show that high working memory requirements, but not necessarily sequence length, lead to an increased need for stability at the cost of decreased performance on subsequent tasks. Second, to overcome this limitation we employ a recent method based on hypernetworks and apply it to RNNs to address catastrophic forgetting on sequential data. By generating the weights of a main RNN in a task-dependent manner, our approach disentangles stability and plasticity, and outperforms alternative methods in a range of experiments. Overall, our work provides several key insights on the differences between CL in feedforward networks and in RNNs, while offering a novel solution to effectively tackle CL on sequential data.

1 Introduction

The ability to continually learn from a non-stationary data distribution while transferring and protecting past knowledge is known as continual learning (CL). This ability requires neural networks to be stable to prevent forgetting, but also plastic to learn novel information, which is referred to as the stability-plasticity dilemma [1,2]. To address this dilemma, a variety of methods which tackle CL for static data with feedforward networks have been proposed (for reviews refer to [3,4]). However, CL for sequential data has only received little attention, despite recent work confirming that recurrent neural networks (RNNs) also suffer from catastrophic forgetting [5].

A prominent approach to address this problem are regularization methods, which do not require rehearsal of past data nor an increase in model capacity, but can benefit from either of the two [e.g., 6,7]. The most well-known regularization methods are weight-importance methods, such as elastic weight consolidation (EWC, [8]) and synaptic intelligence (SI, [9]), which are based on assigning importance values to weights. Some of these have a direct probabilistic interpretation as prior-focused CL methods [10], for which solutions of upcoming tasks must lie in the posterior parameter...
distribution of the current task (cf. Fig. 1b), highlighting the stability-plasticity dilemma. Whether this dilemma differently affects feedforward networks and RNNs, and whether weight-importance based methods can be used off the shelf for sequential data remains, however, unclear.

Here, we contribute to the development of CL approaches for sequential data in two ways. First, we evaluate the performance of established CL methods for feedforward networks when applied to RNNs. We particularly focus on a version of EWC [11] called Online EWC, [12, 13] as an exemplar of weight-importance methods. We show through empirical evidence that high requirements for working memory, i.e. the need to store and manipulate information when processing individual samples, lead to a saturation of weight importance values. This makes the RNN rigid and hinders its potential to learn new tasks. We complement these observations with a theoretical analysis of linear RNNs, which confirms that established weight-importance CL methods are prone to failure when applied to RNNs if the input-output mappings to be learned markedly differ across tasks. These results indicate that working memory plays a crucial role in the stability-plasticity trade-off in RNNs, and stresses the need to develop tailored CL methods. Second, we propose a CL regularization approach based on hypernetworks [14, 15] that mitigates the limitations of weight-importance methods in RNNs. We show this on variants of three well-known sequential datasets: the Copy Task [16], Sequential Stroke MNIST [17] and AudioSet [18]. Taken together, our experimental and theoretical results advance the development of state-of-the-art CL methods that are suited for sequential data.

2 Related work

Continual learning with sequential data. As in Parisi et al. [3], we categorize existing CL methods for RNNs into regularization approaches, dynamic architectures and complementary memory systems.

Regularization approaches set optimization constraints on the update of certain network parameters without requiring a model of past input data. EWC, for example, uses weight importance values to limit further updates of weights that are considered essential for solving previous tasks [11]. Throughout this work, we utilize a more mathematically sound and less memory intensive version of this algorithm, called Online EWC [12, 13]. Although a highly popular approach in feedforward networks, EWC has only been applied to RNNs as a comparison baseline for other methods [19, 20] and lacks a thorough investigation in the context of sequential data. A related CL approach that also relies on weight importance values is SI [9]. Variants of SI have been used for different sequential datasets, but have not been systematically compared against other established methods [21, 22]. Fixed expansion layers [23] are another method to limit the plasticity of weights and prevent forgetting, and in RNNs take the form of a sparsely activated layer between consecutive hidden states [24]. Lastly, some regularization approaches rely on the use of non-overlapping and orthogonal representations to overcome catastrophic forgetting [25, 26, 27]. Masse et al. [7], for example, proposed the use of context-dependent random subnetworks, which eliminates forgetting for disjoint networks but leads to a reduction of available capacity per task. Despite the use of different sets of neurons per task, context-dependent gating uses a fixed-size network while limiting the plasticity to task-specific subnetworks, thereby regularizing weight changes of masked weights.

Dynamic architecture approaches, which rely on the addition of neural resources in response to new information, have also been applied to RNNs. Cossu et al. [20] presented a combination of progressive networks [28] and gating autoencoders [29], where an RNN module is added for each new task and the reconstruction error of task-specific autoencoders is used to infer the RNN module to be used. Arguably, the main limitation of this type of approach is the increase in the number of parameters with the number of tasks, although methods have been presented that add resources for each new task only if needed [30].

Finally, complementary memory systems have also been applied to the retention of sequential information. In an early work, Ans et al. [31] proposed a secondary network that generates patterns for rehearsing previously learned information. Asghar et al. [19] suggested using an external memory that is progressively increased when new information is encountered. Sodhani et al. [32] combined an external memory with Net2Net [33], such that the network capacity can be extended while maintaining memories. The major drawback of complementary memory systems is that they either violate CL desiderata by storing past data, or rely on the ability to learn a generative model, a task that arguably scales poorly to complex data.
Hypernetworks. The term "hypernetwork" was introduced by Ha et al. [14] and refers to a neural network that generates the weights of another neural network. The idea can be traced back to Schmidhuber [34], who already suggested that a recurrent hypernetwork could be used in the context of learning to learn [35]. An important advantage of hypernetworks is that they can make use of the fact that parameters in a neural network possess compressible structure [36, 37]. Indeed, Ha et al. [14] showed that the number of trainable weights of feed-forward architectures can be reduced via the importance method (such as EWC). Both methods start learning a second task from a common solution \( \psi_1 \). EWC is rigid along certain directions in weight space, which leads to a trade-off solution \( \psi_2 \) when seeking good optima for the upcoming task (cf. [10]). The hypernetwork-based approach (Eq. [1]) has no such trade-off build into its objective; it is only limited by the optimization algorithm and network capacity, and is capable to output the task-specific solutions \( \psi_1 \) and \( \psi_2^{\text{HNET}} \) (figure adapted from Kirkpatrick et al. [8]).

3 Methods

Recurrent Neural Networks. We consider discrete-time RNNs. At timestep \( t \), the network’s output \( \tilde{y}_t \) and hidden state \( h_t \) are given by \( \tilde{y}_t, h_t = f_{\text{step}}(x_t, h_{t-1}, \psi) \), where \( x_t \) denotes the input at time \( t \) and \( \psi \) the parameters of the network \([39, 40, 41]\). In this work, we consider either vanilla RNNs (based on Elman networks \([40]\)) or LSTMs \([41]\).

Hypernetworks. We formalize hypernetworks \([14]\) as neural networks \( \psi = h(e, \theta) \) with parameters \( \theta \) and input embeddings \( e \). They can be seen as meta-models that generate the weights of a main network: \( f_{\text{step}}(x_t, h_{t-1}, \psi) = f_{\text{step}}(x_t, h_{t-1}, h(e, \theta)) \) (Fig. 1a). We exclusively study hypernetworks with feedforward architectures, and discuss in Sec. 5 how our work can be extended to recurrent hypernetworks.

Continual learning with hypernetworks. Introduced by von Oswald et al. [15], this method sidesteps the problem of finding a compromise between tasks with a shared model \( \psi \), by generating a task-specific model \( \psi^{(k)} \) from a low-dimensional embedding space via a shared, continually-learned hypernetwork.

The CL approach is based on a simple L2-regularization of the hypernetwork output. The complete loss function for learning the \( K \)-th task is given by\(^1\)

\[
\mathcal{L}(\theta, e_1, \ldots, e_K, D_K) = \mathcal{L}_{\text{task}}(\theta, e_K, D_K) + \frac{\beta}{K-1} \sum_{k=1}^{K-1} \| h(e_k, \theta) - h(e_{(K-1)}, \tilde{\theta}^{(K-1)}) \|^2 \]

\(^1\)We slightly modified the original regularizer by excluding the lookahead \( \Delta \theta \) used in [15] and by allowing fine-tuning of previous task embeddings, which requires us to additionally checkpoint these task embeddings before learning a new task.
where \( \mathcal{D}_K \) is the dataset of task \( K \), \( \mathcal{L}_{\text{task}}(\cdot) \) is the loss function of the current task, \( \beta \) is the regularization strength and \( \hat{\theta}^{(K-1)}, \hat{e}_1^{(K-1)}, \ldots, \hat{e}_{\psi}^{(K-1)} \) denote hypernetwork weights \( \theta \) and task embeddings that were checkpointed after learning task \( K - 1 \). These checkpointed weights are fixed and needed to compute the regularization targets, which ensure that the output of the network stays constant for previously learned tasks.

Von Oswald et al. [15] exclusively studied the effectiveness of Eq. 1 for main networks with feedforward architecture and for non-sequential data. Here, we focus instead on RNNs as main networks. A naive implementation of a hypernetwork as a fully-connected network that outputs all parameters \( \psi \) of the main network at once typically has many more parameters than \( |\psi| \). Therefore, we focus on chunked hypernetworks (for details see supplementary materials, SM B.4, and [15]), denoted by HNET, and ensure, for fair comparison, that the number of trainable parameters is comparable to other baselines: \( |\theta \cup \{e_k\}_{K=1}^\infty| \approx |\psi| \).

**Baselines.** In our experiments we consider the following baselines. **Fine-tuning** refers to training an RNN sequentially on all tasks without any CL protection. Each task has a different output head (multi-head), and the heads of previously learned tasks are kept fixed. **From-scratch** refers to training a separate neural network for each task (which excludes potential knowledge transfer). **Multitask** describes the parallel training on all tasks (no CL). To keep approaches comparable, the multitask baseline uses a multi-head output. **Online EWC** [12, 8, 13] and **SI** [9] are different weight-importance CL methods. A simple weighted L2 regularization ensures that the neural network is more rigid in weight directions that are considered *important* for previous tasks, i.e., the loss for the \( K \)-th task is given by

\[
\mathcal{L}(\psi, \mathcal{D}_K) = \mathcal{L}_{\text{task}}(\psi, \mathcal{D}_K) + \lambda \left| \left| \sum_{i=1}^{[\psi]} \omega_i (\psi_i - \hat{\psi}_i^{(K-1)})^2 \right| \right| \quad (2)
\]

where \( \lambda \) is the regularization strength, \( \omega_i \) is the *importance* associated with \( \psi_i \) (cf. SM B.5 and B.6) and \( \hat{\psi}_i^{(K-1)} \) denotes the main network weights \( \psi \) that were checkpointed after learning task \( K - 1 \). **Masking** (or context-dependent gating) [7] applies a binary random mask per task for all hidden units of a multi-head network, and can be seen as a simple method for selecting a different subnetwork per task. Since catastrophic interference might still occur because of the overlap between subnetworks, this method can be combined with other CL methods such as SI (Masking+SI). We also consider methods based on replaying input data from previous tasks, either via a sequentially trained generative model [42, 43], denoted by **Generative Replay**, or by maintaining a small subset of previous training data [44, 46], denoted by **Coresets-N**, where \( N \) denotes the number of samples stored for each task. Target outputs for replayed data are obtained via a copy of the main network, stored before training on the current task (detailed baseline descriptions in SM B).

**Task Identity.** Throughout the main text of this paper, we assume that task identity is provided to the system during training and inference time. This is realized either by selecting the correct output head or by feeding the correct task embedding \( e_k \) into the hypernetwork. In SM F.1 we elaborate on how to overcome this limitation.

### 4 Experiments

To highlight strengths and weaknesses that different CL methods might have in various settings, we performed experiments on one synthetic and two real-world sequential datasets, using different types of RNNs.\(^2\) We distinguish between during and final accuracies. The during accuracy of a CL experiment is obtained by taking the mean over the test accuracy from each task right after it has been trained on, i.e., when tasks have not yet been subject to forgetting. The final accuracy describes the mean test accuracy over all tasks obtained after the last task has been learned.

We obtained the reported results via an extensive hyperparameter search on each method, where the hyperparameter configuration of the run with best final accuracy was selected and subsequently tested on multiple random seeds (experimental details in SM E).

\(^2\)Source code for all experiments (including all baselines) is available at [https://github.com/mariacer/cl_in_rnns](https://github.com/mariacer/cl_in_rnns).
4.1 Copy Task

First, we consider the Copy Task \[16\], a synthetic dataset which allows us to investigate the particularities of weight-importance methods, represented by Online EWC, in a controlled setting. The basic Copy Task consists of a random binary input pattern that has to be recalled by the network after a stop bit is observed. In this section we make use of several variations of this setting (see SM D.1 for details). We use vanilla RNNs combined with orthogonal regularization (see SM F.2) for all Copy Task experiments.

Factors contributing to weight importance. We hypothesize that weight importance values rise due to working memory requirements (storage and manipulation of data) and are not necessarily linked to sequence length, and provide experimental and theoretical evidence validating this hypothesis. Let us denote the length (number of timesteps) of the binary input pattern to be copied by \( p \), and the actual number of timesteps until the stop bit is encountered by \( i \) (examples can be found in SM Fig. S1). This distinction allows us to consider two variants, the basic Copy Task where \( p = i \), and another one where \( i > p \), which we name Padded Copy Task. In this variant, we zero-pad a binary input pattern of length \( p \) for \( i - p \) timesteps until the occurrence of the stop bit, resulting in an input sequence with \( i \) timesteps.

Specifically, we consider a set of Copy Tasks with varying input lengths \( i \) and, either a fixed pattern length \( p = 5 \), or a pattern length tied to the input length \( (p = i) \). This allows us to disentangle how sequence length and memory load affect weight importance. As in Online EWC, we calculate weight importance as the diagonal elements of the empirical Fisher information matrix (see SM B.3). To quantify memory load, we study the intrinsic dimensionality of the hidden state of the RNN using principal component analysis (PCA), once all networks have been trained to achieve near optimal performance (above 99%). We define the intrinsic dimensionality as the number of principal components that are needed to explain 75% of the variance.

As expected, the intrinsic dimensionality of the hidden space increases while the binary input pattern is presented and peaks after \( p \) timesteps, which coincides with the stop bit for tasks with \( p = i \) (Fig. 2a) but occurs \( i - p \) timesteps before the stop bit if \( p = 5 \) remains fixed (Fig. 2b). As Fig. 2c shows, weight importance values rapidly increase with memory requirements \( (p) \), but not with sequence length (increasing \( i \) for fixed \( p \)). Note that large Fisher values lead to weight rigidity and cause insufficient plasticity for learning new tasks. This toy analysis reveals that Online EWC is not sufficient for tasks that need to non-linearly encode information into the hidden state for large \( i \).

Note, these tasks are learned independently and not continually in order to investigate the effects of \( i \) and \( p \) on weight importance values.

Note, Fig. 2a shows a decreased dimensionality at \( t = 0 \) for \( i = 35 \) or \( i = 40 \) compared to \( i = 30 \). We hypothesize that this is due to a need to non-linearly encode information into the hidden state for large \( i \), and verified that the dimensionality indeed increases with \( i \) when using Kernel PCA (data not shown).
affected by the sequential nature of the data, even though the same set of weights is reused for many timesteps, but rather by the processing and storage required by the task.

To further investigate this phenomenon, we theoretically analyse the learning of a distinct set of tasks with linear RNNs (see SM [C]). We show that the intrinsic dimensionality of the hidden state is directly related to the minimal capacity of hidden-to-hidden weights that is required to solve a given task. This quickly results in the dilemma that plasticity for learning new tasks can only be provided at the cost of decreased stability. Interestingly, this problem can be sidestepped by hypernetworks, which are able to provide a distinct set of hidden-to-hidden weights per task (cf. SM [C]).

Continual learning on variations of the Copy Task. After exposing the challenges that weight-importance methods face when dynamically processing data, we explore how these are manifested in a CL scenario derived from the Copy Task. We compare weight-importance methods against other CL approaches, with a particular focus on the HNET, which can in principle bypass those challenges. To transform the Copy Task into a set of tasks, we apply a random time-permutation per task, that translates each timestep \( t \in \{1, \ldots, p\} \) from the input pattern to a corresponding output timestep \( t_o \in \{1, \ldots, i\} \) at which the binary feature vector presented at \( t \) should be recalled. We denote this setting as Permuted Copy Task. First, we evaluate all methods in a relatively simple setting with five tasks using \( p = i = 5 \), with results reported in Table 1. Online EWC achieves very high performance, and HNET reaches close to 100% accuracy. The random subnetworks in Masking, which per task used 20% of the network’s full capacity, can learn individual tasks to perfection. However, weight changes within subnetworks, which result from random overlaps, cause severe performance drops and show the need to add stabilization mechanisms, e.g., Masking+SI. Since the input data distribution is relatively simple and identical across tasks, learning a generative model is feasible, which is illustrated by the performance of Generative Replay.

In the following, we focus on a comparison between Online EWC and HNET to further investigate how these methods are affected by sequence length and working memory requirements. We first test whether EWC is affected by sequence length by investigating the Permuted Copy Task at \( p = 5, i = 25 \) using 5 tasks. As Table 2 shows, the performance of both methods is not markedly affected by sequence length. Interestingly, the results are slightly better for longer sequences with both methods, which can be due to an increased processing time between input presentation and recall.

Next, we compare the performance of Online EWC and HNET in a set of tasks for which working memory requirements can be easily controlled. In this setting, referred to as Pattern Manipulation Task, difficulty is controlled by a set of \( r \) task-specific random permutations along the time axis. The output is computed from the input pattern by applying a binary XOR operation iteratively with all of its \( r \) permutations. Note that this variant substantially differs from previous Copy Task variations, since the processing of input patterns is now both input- and task-dependent. As shown in Table 2, Online EWC experiences a larger drop with increased task difficulty than HNET, confirming that it is more severely affected by working memory requirements.

### Table 1: Mean during and final accuracies for the Permuted Copy Task with \( p = i = 5 \) using 5 tasks each (Mean ± SEM in %, \( n = 10 \)).

| Method          | during          | final           |
|-----------------|-----------------|-----------------|
| Multitask       | N/A             | 99.87 ± 0.05    |
| From-scratch    | N/A             | 100.00 ± 0.00   |
| Fine-tuning     | 99.99 ± 0.00    | 71.05 ± 0.13    |
| HNET            | 99.98 ± 0.00    | 99.96 ± 0.01    |
| Online EWC      | 99.93 ± 0.01    | 98.66 ± 0.14    |
| SI              | 98.41 ± 0.06    | 94.03 ± 0.24    |
| Masking         | 99.53 ± 0.26    | 72.31 ± 0.82    |
| Masking+SI      | 99.40 ± 0.25    | 99.40 ± 0.25    |
| Gen. Replay     | 100.00 ± 0.00   | 100.00 ± 0.00   |
| Coresets-100    | 100.00 ± 0.00   | 99.94 ± 0.00    |

### Table 2: Detailed comparison of Online EWC and HNET on three additional Copy Task versions (Mean ± SEM in %, \( n = 5 \)).

| Task                      | during          | final           |
|---------------------------|-----------------|-----------------|
| Padded Copy Task          |                 |                 |
| HNET                      | 100.00 ± 0.00   | 100.00 ± 0.00   |
| Online EWC                | 97.94 ± 0.09    | 97.89 ± 0.10    |
| Pattern Manipulation Task  |                 |                 |
| r = 1                     |                 |                 |
| HNET                      | 99.98 ± 0.00    | 99.84 ± 0.15    |
| Online EWC                | 98.52 ± 0.27    | 95.45 ± 0.17    |
| Pattern Manipulation Task  |                 |                 |
| r = 5                     |                 |                 |
| HNET                      | 99.73 ± 1.44    | 93.87 ± 1.24    |
| Online EWC                | 87.40 ± 4.53    | 81.80 ± 3.25    |
### 4.2 Sequential Stroke MNIST

To test whether the results from the synthetic Copy Task hold true for real world data, we turned to a variation of MNIST [46]. In the Stroke MNIST (SMNIST) dataset [47], MNIST images are represented as sequences of pen displacements, that result in the original digits when drawn in order. We adapt this dataset to a CL scenario by splitting it into five binary classification problems (digits 0 vs 1, 2 vs 3, etc.), reminiscent of the popular Split-MNIST experiment commonly used to benchmark CL methods on static data [9]. To test how the performance of different CL methods depends on the difficulty of individual tasks, we generalized the notion of Split-SMNIST to sequences of \( m \) SMNIST samples [cf. 17], where each sequence contains only two digits (e.g. 2332 or 7767 for \( m = 4 \)). To obtain a binary decision problem, we randomly group all \( 2^m \) possible sequences within a task into two classes. This ensures that despite increasing levels of task difficulty, as determined by \( m \), chance level is not affected.

We train LSTM networks on five tasks for four different difficulty levels and observe that methods are differently affected (see Fig. 3). Indeed, for four digits per sequence (\( m = 4 \)) the performance of Online EWC and SI drops to 68.67 ± 1.48% and 70.20 ± 0.92%, while the hypernetwork approach successfully classifies 85.68 ± 3.31% of all inputs. Masking+SI shows good results up to three digits per sequence (\( m = 3 \)), but fails for \( m = 4 \), probably as a result of task-specific subnetworks having insufficient capacity to solve individual tasks (during accuracy of 67.48 ± 2.62%). We discuss the use of replay for Split-SMNIST in SMF.4

### 4.3 AudioSet

AudioSet [18] is a dataset of manually annotated audio events. It consists of 10-second audio snippets which have been preprocessed by a VGG network to extract 128-dimensional feature vectors at 1 Hz. This dataset has been previously adapted for CL by [48, 49], whose particular split has not been made public, and by Cossu et al. [50], for which the test set size largely differed across classes. We therefore created a new variant, which we call Split-AudioSet-10, containing 10 tasks with 10 classes each (see SM E.3 for details).

The results obtained in this dataset using LSTMs are listed in Table 3. HNET is the strongest among regularization based methods, and is only outperformed by Coresets, which rely on storing past data. Masking during accuracies indicate that random subnetworks have enough capacity to learn individual tasks. However, low final accuracies suggest that catastrophic forgetting occurs, presumably because of the existing overlap between subnetworks. This is partly solved in Masking+SI by introducing stabilization which, however, reduces plasticity for learning new tasks. The Multitask baseline underperforms several CL methods, indicating that the latter may have

---

#### Table 3: Mean during and final accuracies for the Split-AudioSet-10 experiments (Mean ± SEM in %, \( n = 10 \)).

| Method      | during       | final        |
|-------------|--------------|--------------|
| Multitask   | N/A          | 69.13 ± 0.29 |
| From-scratch| N/A          | 79.06 ± 0.11 |
| Fine-tuning | 71.95 ± 0.24 | 49.02 ± 1.00 |
| HNET        | 73.05 ± 0.45 | 71.76 ± 0.62 |
| Online EWC  | 68.82 ± 0.20 | 65.56 ± 0.35 |
| SI          | 67.66 ± 0.10 | 66.92 ± 0.04 |
| Masking     | 70.81 ± 0.25 | 49.54 ± 1.24 |
| Masking+SI  | 57.94 ± 0.47 | 57.90 ± 0.48 |
| Coresets-100| 74.25 ± 0.11 | 72.30 ± 0.11 |
| Coresets-500| 77.03 ± 0.08 | 73.90 ± 0.07 |
optimization benefits that arise from separately training on different subtasks, instead of learning them all at once. The From-scratch baseline outperforms other methods, which is explained by the fact that it trains a separate model per task, leading to 10 times more network capacity. Notably, we were not able to successfully train a Generative Replay model on this dataset despite extensive hyperparameter search. Together with the results in Sec. 4.1 this highlights that the performance of Generative Replay depends on the complexity of the input data distribution, and not necessarily on the CL nature of the problem. To further investigate the stability-plasticity trade-off, we tested HNet and Online EWC across a range of difficulty levels in individual tasks. This can be controlled by the number of classes to be learned within each task, which we varied from two to ten. For both methods we used the best hyperparameters found for Split-AudioSet-10 (cf. Fig. 5b for varying regularization strengths). Since the performance of Online EWC strongly depends on the regularization parameter $\lambda$, we tuned this value to achieve optimal results in each setting (cf. Fig. 5b). Fig. 4 shows the task-averaged final accuracies for the different task-difficulty settings. While HNet performance is primarily affected by task difficulty but not by the number of tasks, results for Online EWC show an interplay between task difficulty and the ability to retain good performance on many tasks. These results provide further evidence that the hypernetwork-based approach can resolve the limitations of weight-importance CL methods for sequential data.

5 Discussion

The stability-plasticity dilemma with sequential data. Weight-importance methods address CL by progressively constraining a network’s weights, directly trading plasticity for stability. In the case of RNNs, weights are subject to additional constraints, since the same set of weights is reused across time to dynamically process an input stream of data. We show that increased working memory requirements, resulting from more complex processing within individual tasks, lead to high weight-importance values and can hinder the ability to learn future tasks (cf. Sec. 4.1). On the contrary, we find that longer sequence lengths do not impact performance for a fixed level of task complexity (cf. Fig. 2b Table 2), suggesting that weight reuse doesn’t interfere with the RNN’s ability to retain previous knowledge. These observations are consistent with our theoretical analysis of linear RNNs (SM C), which predicts that more challenging processing within individual tasks leads to increased interference between tasks. This aggravates the stability-plasticity dilemma in weight-importance based methods, which we confirm through a range of experiments.

Benefits of a hypernetwork-based CL approach for sequential data. To alleviate the stability-plasticity dilemma when continually learning with RNNs, we propose the use of hypernetworks [15]. As seen in Eq. 1, stability is outsourced to a regularizer that does not directly limit the plasticity required for finding new solutions. In particular, if tasks require different or even conflicting processing strategies, hypernetworks possess more flexibility than weight-importance methods and can achieve better results, as demonstrated by our experiments on the Pattern Manipulation Task. In addition, we show that these advantages translate to real world applications in our SMNIST and AudioSet experiments.

Future avenues for CL with hypernetworks. Despite the discussed advantages, hypernetworks introduce additional optimization challenges, especially in conjunction with vanilla RNNs (cf. SM F.2), which leaves room for future improvements. Another interesting direction is the use of a recurrent hypernetwork to generate timestep-specific weights in the main RNN [14, 51]. A naive application of this combination for CL using Eq. 1 would come at the cost of a linear increase of the required computation with the number of timesteps. However, this problem can be elegantly sidestepped by the use of a feed-forward hypernetwork that generates the weights of the recurrent hypernetwork. Eq. 1 can then simply be applied to the static output of this hyper-hypernetwork, protecting a set of timestep-specific weights per task without the need to increase the regularization budget.

6 Conclusion

Our work advances the CL community in three ways. First, by systematically evaluating the performance of established CL methods when applied to RNNs, we provide extensive baselines that can serve as reference for future studies on CL with sequential data. Second, we use theoretical arguments derived from linear RNNs to hypothesize limitations of weight-importance based CL in the
context of recurrent computation, and provide empirical evidence to support these statements. Third, derived from these insights, we propose to use an approach based on hypernetworks that mitigates the stability-plasticity dilemma, and outperforms weight-importance methods on synthetic as well as real-world data. Finally, our work discusses several future improvements and directions of the hypernetwork-based CL approach for sequential data.

**Broader Impact**

Since research on CL is still at an early stage, it is not directly clear how our work will lead to applications with broad impact across society. However, a thorough understanding of the approaches that address CL for sequential data is a prerequisite for developing related applications, such as processing audio or video streams. These could for example arise in settings where autonomous agents need to interact with varying sensory streams using limited hardware capacities. Our work makes a valuable contribution by analyzing the advantages and disadvantages of different CL approaches and can help finding tailored solutions for context-specific applications. For example, the hypernetwork-based approach that we propose could be particularly powerful in a setting where tasks are unambiguously defined, as it provides fully task-conditioned solutions. On the other hand, our approach will experience difficulties when task signals are ambiguous. An immediate negative impact of this type of research is its vast power demand. Even though the models we consider in this work are relatively small and cheap to train, we systematically evaluated the performance of a variety of methods, in different datasets. This required extensive use of GPUs, resulting in significant power consumption. However, this could have a positive impact for future work, as researchers have the opportunity to utilize our open-source codebase and benchmarks together with all reported baselines, potentially saving energy resources in the long run.

**Acknowledgements**

This work was supported by the Swiss National Science Foundation (B.F.G. CRSII5-173721 and 315230_189251), ETH project funding (B.F.G. ETH-20 19-01) and funding from the Swiss Data Science Center (B.F.G, C17-18, J. v. O. P18-03). We especially thank João Sacramento for insightful discussions, ideas and feedback. We would also like to thank Nikola Nikolov for helpful advice and James Runnalls for proofreading our manuscript.

**References**

[1] Stephen Grossberg. Consciousness CLEARS the mind. *Neural Networks*, 20(9):1040–1053, Nov 2007. ISSN 0893-6080. doi: 10.1016/j.neunet.2007.09.014.

[2] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Front. Psychol.*, 4, Aug 2013. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00504.

[3] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, May 2019. ISSN 0893-6080. doi: 10.1016/j.neunet.2019.01.012.

[4] Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *arXiv*, Apr 2019.

[5] Monika Schak and Alexander Gepperth. A Study on Catastrophic Forgetting in Deep LSTM Networks. *SpringerLink*, pages 714–728, Sep 2019. doi: 10.1007/978-3-030-30484-3_56.

[6] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[7] Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115(44):E10467–E10475, 2018.

[8] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis
Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017.

[9] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML’17, pages 3987–3995. JMLR.org, 2017.

[10] Sebastian Farquhar and Yarin Gal. A unifying bayesian view of continual learning. *Bayesian Deep Learning Workshop at NeurIPS*, 2018.

[11] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. U.S.A.*, 114(13):3521–3526, Mar 2017. ISSN 0027-8424. doi: 10.1073/pnas.1611835114.

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

[13] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4528–4537, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[14] David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. *arXiv*, Sep 2016.

[15] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

[16] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[17] Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

[18] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[19] Nabiha Asghar, Lili Mou, Kira A. Selby, Kevin D. Pautasdo, Pascal Poupart, and Xin Jiang. Progressive memory banks for incremental domain adaptation. In *International Conference on Learning Representations*, 2020.

[20] Andrea Cossu, Antonio Carta, and Davide Bacciu. Continual Learning with Gated Incremental Memories for sequential data processing. *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN 2020)*, April 2020.

[21] Guangyu Robert Yang, Madhura R Joglekar, H Francis Song, William T Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature neuroscience*, 22(2):297–306, 2019.

[22] Sungjin Lee. Toward Continual Learning for Conversational Agents. *arXiv*, Dec 2017.

[23] Robert Coop and Itamar Arel. Mitigation of catastrophic interference in neural networks using a fixed expansion layer. In *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 726–729. IEEE, 2012.

[24] Robert Coop and Itamar Arel. Mitigation of catastrophic forgetting in recurrent neural networks using a fixed expansion layer. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2013.

[25] Robert M. French. Semi-distributed Representations and Catastrophic Forgetting in Connectionist Networks. *Connection Science*, 4(3-4):365–377, Jan 1992. ISSN 1360-0494. doi: 10.1080/09540099208946624.

[26] Robert M. French. Dynamically Constraining Connectionist Networks to Produce Distributed, Orthogonal Representations to Reduce Catastrophic Interference. *ResearchGate*, Aug 1994.

[27] Robert M. French. Using Pseudo-Recurrent Connectionist Networks to Solve the Problem of Sequential Learning. *ResearchGate*, Feb 1970.
[28] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. CoRR, abs/1606.04671, 2016.

[29] Rahaf Aljundi, Purnaraj Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3366–3375, 2017.

[30] Ben Tsuda, Kay M. Tye, Hava T. Siegelmann, and Terrence J. Sejnowski. A modeling framework for adaptive lifelong learning with transfer and savings through gating in the prefrontal cortex. bioRxiv, page 2020.03.11.984757, Mar 2020. doi: 10.1101/2020.03.11.984757.

[31] Bernard Ans, Stéphane Rousset, Robert M. French, and Serban Musca. Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting. Connection Science, 16(2):71–99, Jun 2004. ISSN 0954-0091. doi: 10.1080/0954090412331271199.

[32] Shagun Sodhani, Sarath Chandar, and Yoshua Bengio. Toward training recurrent neural networks for lifelong learning. Neural computation, 32(1):1–35, 2020.

[33] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating Learning via Knowledge Transfer. In 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[34] Jürgen Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. Neural Comput., 4(1):131–139, Jan 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.1.131.

[35] Jürgen Schmidhuber. A ‘self-referential’ weight matrix. In International Conference on Artificial Neural Networks, pages 446–450. Springer, 1993.

[36] Misha Denil, Babak Shakibi, Laurent Dinh, Marc’Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. In Advances in neural information processing systems, pages 2148–2156, 2013.

[37] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In Advances in neural information processing systems, pages 1135–1143, 2015.

[38] Xu He, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu. Task agnostic continual learning via meta learning. arXiv preprint arXiv:1906.05201, 2019.

[39] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 2014.

[40] Jeffrey L. Elman. Finding structure in time. Cognitive Science, 14(2):179 – 211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[42] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual Learning with Deep Generative Replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 2990–2999. Curran Associates, Inc., 2017.

[43] Gido M. van de Ven and Andreas S. Tolias. Generative replay with feedback connections as a general strategy for continual learning. arXiv preprint arXiv:1809.10635, 2018.

[44] Sylvester-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pages 2001–2010, 2017.

[45] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In International conference on artificial neural networks, pages 583–588. Springer, 1997.
[46] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[47] Edwin D. de Jong. Incremental Sequence Learning. *arXiv*, Nov 2016.

[48] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018.

[49] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[50] Andrea Cossu, Antonio Carta, and Davide Bacciu. Continual learning with gated incremental memories for sequential data processing, 2020.

[51] Joseph Suarez. Language modeling with recurrent highway hypernetworks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3267–3276. Curran Associates, Inc., 2017.

[52] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[53] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

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

[55] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[56] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[57] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2980–2988. Curran Associates, Inc., 2015.

[58] Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

[59] Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D Sculley, Joshua Dillon, Jie Ren, and Zachary Nado. Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, pages 13969–13980, 2019.

[60] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593, 2014.

[61] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3570–3578. JMLR. org, 2017.

[62] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[63] Wojciech Zaremba and Ilya Sutskever. Learning to execute, 2014.

[64] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1747–1756. New York, New York, USA, 20–22 Jun 2016. PMLR.

[65] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human
[66] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499, 2016.

[67] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017.
Supplementary Material:
Continual Learning in Recurrent Neural Networks with
Hypernetworks

Benjamin Ehret*, Christian Henning*, Maria R. Cervera*, Alexander Meulemans, Johannes von Oswald, Benjamin F. Grewe

A Summary of notation

In this section we define the mathematical notation that we consistently use throughout the paper. We consider the successive learning of $K$ datasets $D_k = \{(x^{(n)}_1: T_{in}^{(n)}, y^{(n)}_1: T_{out}^{(n)})\}_{n=1}^{N_k}$. A data sample $(x_1:T_{in}, y_1:T_{out})$ consists of a sequence of inputs $x_1:T_{in} = (x_1, \ldots, x_T)$, $x_t \in \mathbb{R}^{F_{in}}$, and a sequence of target outputs $y_1:T_{out} = (y_1, \ldots, y_T)$, $y_t \in \mathbb{R}^{F_{out}}$, where $T_{in}/T_{out}$ denote the time dimension and $F_{in}/F_{out}$ the feature dimension, respectively. In general, the number of timesteps is sample-dependent and not constant.

The main network, which processes data from the datasets $\{D_k\}_{k=1}^{K}$, is an RNN with parameters $\psi$. With an abuse of notation, we describe it by $\hat{y}_{1:T_{out}} = f(x_{1:T_{in}}, \psi)$. To express the step-by-step computation of the RNN we use $(\hat{y}_t, h_t) = f_{\text{step}}(x_t, h_{t-1}, \psi)$. Specifically, we denote by $\psi_{hh}$ the hidden-to-hidden weights, which are a subset of $\psi$ and are exclusively involved in the computation from $h_{t-1}$ to $h_t$. The hypernetwork is a feedforward neural network $\psi = h(e_k, \theta)$ with parameters $\theta$, that generates the parameters $\psi$ of the main network given the task embedding $e_k$ of task $k$.

B Detailed description of all methods

Here, we provide a mathematical description of all methods mentioned in Sec. 3, together with an estimate of their time and space complexity increase when compared to the naive Fine-tuning baseline.

The task-specific loss functions $L_{\text{task}}(\psi, D_k)$ applied across all methods are described in Sec. B.5 (cf. Eq. 7 and Eq. 8).

B.1 Fine-tuning

Fine-tuning [52] refers to sequentially optimizing the task-loss $L_{\text{task}}(\psi, D_k)$ for $k = 1, \ldots, K$ without any explicit protection against catastrophic forgetting. However, since each task has its own output head, the output head weights are task-specific and fixed for past tasks.

Even though Fine-tuning has no built-in mechanism to prevent forgetting, we selected the hyperparameter configuration based on the best final accuracy. This ensured consistency with other methods, and allowed directly assessing improvements when employing CL methods.

B.2 Training from scratch

From-scratch refers to the independent training of a set of network parameters $\psi^{(k)}$ per task, i.e., $K$ separate networks are trained by minimizing $L_{\text{task}}(\psi^{(k)}, D_k)$.

Complexity estimation. This approach does not add time complexity, but leads to a linear increase in the memory requirements with the number of tasks.

B.3 Multitask

Multitask, or joint training [52], refers to jointly training on all datasets at once: $\min_\psi \sum_{k=1}^{K} L_{\text{task}}(\psi, D_k)$. We performed joint training by assembling a mini-batch of size $B$ using samples equally distributed across all $K$ datasets. Note that in order to provide a fair comparison to our CL baselines, the main network is still a multi-head network with a task-specific fully-connected
output layer per task. Thus, the task identity has to be provided during inference in order to select the correct output head.

Joint training does not necessarily lead to better solutions, since the optimization process may benefit from the divide-and-conquer strategy taken by CL approaches, i.e., optimizing one task after another rather than directly looking for a combined solution.

**Complexity estimation.** Even though this approach does not lead to time or memory complexity increases, it requires all data to be available at all times.

### B.4 Hypernetwork-protected models

The hypernetwork-based CL approach, HNET [15], is described in Sec. 3. It is an L2-regularization technique that, in contrast to weight-importance methods, aims to fix certain input-output mappings of a secondary neural network, instead of directly fixing the weights of a main network (cf. Eq. 1).

To establish a fair comparison to other methods (in terms of number of trainable weights), we used the chunking approach described in von Oswald et al. [15], who showed that the non-parametric limit of a chunked hypernetwork can realize all possible continuous mappings between embedding and weight space. This method splits the vectorized main network weights $\psi$ into equally sized chunks. Each chunk will be assigned a chunk embedding $c_i$. The hypernetwork can then produce all weights $\psi$ by processing a batch of chunk embeddings (utilizing parallelization on modern GPUs): $\psi = h(e_k, \theta = \hat{\theta} \cup \{c_i\}) = \text{concat}([\ldots, \hat{h}(e_k, c_i, \hat{\theta}), \ldots])$. For details please refer to [15].

This approach to chunking is agnostic to the structure that $\psi$ takes in the main network through $f$’s architectural design. Therefore, we investigated other approaches to chunking that respect the architecture of $f$. For instance, if $W_{ih} \in \mathbb{R}^{n_h \times n_h}$ and $W_{ih} \in \mathbb{R}^{n_h \times n_i}$ denote the weights of a recurrent layer, where $n_h$ and $n_i$ are the number of hidden and input units respectively, the hypernetwork can be designed to produce chunks $V_{ih,i}$, $V_{ih,i} = \hat{h}(e_k, c_i, \hat{\theta})$, with $V_{ih,i} \in \mathbb{R}^{n_h \times n_h}$, $V_{ih,i} \in \mathbb{R}^{n_h \times n_i}$ and $0 \equiv n_h \pmod{n_v}$. However, since we didn’t observe any improvements in a set of exploratory experiments, all reported results were obtained using the approach suggested in von Oswald et al. [15].

In addition, we would like to mention two properties of the hypernetwork approach that have been empirically verified [15]. First, the approach supports positive forward transfer, as the knowledge of previous tasks is entangled in the shared meta-model. Experiments on a low-dimensional task embedding space in von Oswald et al. [15] seem to indicate that the learned embedding space possesses a structure that supports transfer. Second, von Oswald et al. [15] noted and showed empirically that the regularizer in Eq. 1 does not have to increase linearly with the number of tasks $K$, but can instead be subsampled using a random set of $C$ tasks for each loss evaluation. We verified this in the Permuted Copy Task, where computing the regularizer for a single randomly chosen task ($C = 1$) at each loss evaluation did not lead to a performance decrease for patterns of length $p = 5$ (data not shown).

**Complexity estimation.** Independent of its application to CL, the use of a hypernetwork increases time complexity because weights need to be generated before being used for the forward computation of the main network. Another factor contributing to the increase in time complexity is the regularizer (Eq. 1), which is a sum of L2 norms of the hypernetwork output (of size $|\psi|$) over past tasks, yielding a time complexity of $O(K|\psi|)$ if the regularizer is applied to all previous tasks, and $O(C|\psi|)$ otherwise.

Space complexity also increases due to two factors. First, a second network object (i.e., the hypernetwork) has to be maintained in memory. Second, the computation of the regularizer (Eq. 1) requires storing a set of checkpointed hypernetwork weights and task embeddings when training on a new task. Since we restrict here our analyses to settings where $|\theta \cup \{e_k\}|_k^{K} \approx |\psi|$, we simply denote this space complexity increase by $O(|\psi|)$.

### B.5 Elastic weight consolidation

Here, we quickly recapitulate the basic concepts behind elastic weight consolidation (EWC) [8]. Since EWC is a prior-focused method [10], solutions of upcoming tasks must lie inside the posterior parameter distribution of previous tasks. To achieve this, EWC approximates the posterior via a
Gaussian distribution with diagonal covariance matrix. Note that this restriction does not apply to task-specific weights, which may be restricted by an arbitrary choice of the prior. However, to avoid overly cluttered notation, we explicitly ignore the multi-head setting in this section, where parameters ψ can be split into task-specific (the corresponding output head’s weights) and task-shared (all weights excluding the output layer) weights.

EWC makes use of the fact that Bayes rule allows the following decomposition of the posterior parameter distribution:

\[ p(\psi \mid D_1, \ldots, D_K) \propto p(\psi \mid D_1, \ldots, D_{K-1}) p(D_K \mid \psi) \]  

(3)

where \( p(\psi \mid D_1, \ldots, D_{K-1}) \) is the posterior from previous tasks and \( p(D_K \mid \psi) \) the likelihood of the current task. The precise derivation of the algorithm described here can be found in Huszár [12], and has been termed Online EWC in Schwarz et al. [13].

When learning task \( K \), we aim to find a maximum a posteriori (MAP) solution of \( p(\psi \mid D_1, \ldots, D_K) \) maximizing the following loss function:

\[ \max_{\psi} \log p(D_K \mid \psi) + \log p(\psi \mid D_1, \ldots, D_{K-1}) \]  

(4)

We discuss the likelihood function for sequential data below. To obtain a tractable loss function, EWC utilizes an approximate posterior \( q(\psi^{(K-1)}) \approx p(\psi \mid D_1, \ldots, D_{K-1}) \), whose parameters \( \zeta \) are computed at the end of task \( K - 1 \). Specifically, EWC first applies a Laplace approximation [33] (using the MAP solution \( \hat{\psi}^{(K-1)} \) obtained at the end of training of task \( K - 1 \)) to obtain a Gaussian \( q(\psi^{(K-1)}) \) with mean \( \hat{\psi}^{(K-1)} \) and precision matrix \( F = \sum_{k=1}^{K-1} F^{(k)} \), where \( F^{(k)} \) denotes the empirical Fisher matrix. As noted in Huszár [12], this version of Online EWC still does not carry out the Laplace approximation correctly, as the precision matrix of \( q(\psi^{(K-1)}) \) misses the prior influence and the individual terms \( F^{(k)} \) are not properly scaled. However, if the prior influence on the precision matrix is ignored and dataset sizes are identical, then the proper scaling can be absorbed into the regularization strength \( \lambda_{EWC} \). As a second approximation, EWC considers all off-diagonal elements of \( F \) to be zero: \( F_{ij} = 0 \). Taken together, while ignoring all terms independent of \( \psi \), the loss described by Eq. 4 is approximated in Online EWC via (cf. Eq. 2):

\[ \min_{\psi} -\log p(D_K \mid \psi) + \lambda_{EWC} \sum_{i=1}^{\mid \psi \mid} F_{ii}(\psi_i - \hat{\psi}^{(K-1)}_i)^2 \]  

(5)

where \( F_{ii} \) can be considered as weight-specific importance values and \( L_{task}(\psi, D_K) \equiv -\log p(D_K \mid \psi) \) describes the negative log-likelihood (NLL) detailed below.

Note that the correct deployment of Eq. 4 requires obtaining a MAP estimate for the first task: \( \hat{\psi}^{(1)}_i = \arg \max_{\psi} \log p(D_1 \mid \psi) + \log p(\psi) \). However, we ignored the prior influence when obtaining \( \hat{\psi}^{(1)}_i \).

**Negative log-likelihood (NLL) for sequential data.** Finally, we discuss how to implement \( L_{task}(\psi, D_K) \equiv -\log p(D_K \mid \psi) \) when applied to sequential data. Note that \( p(D_K \mid \psi) = \prod_{n=1}^{N_K} p(y^{(n)}_{1:T_{\text{train}}^{(n)}} \mid \psi) \) and that \( p(y_{1:T} \mid y_{1:1}, \ldots, y_{T-1}, \psi) = \prod_{t=1}^{T_{\text{train}}} p(y_t \mid y_{1:1}, \ldots, y_{T-1}, \psi) \). Given the autoregressive structure of an RNN, we make the following assumption: \( p(y_t \mid y_{1:1}, \ldots, y_{T-1}, \psi) \approx p(y_t \mid h_{t-1}, \psi) \). Hence, we can decompose the NLL as follows:

\[ -\log p(D_K \mid \psi) = -\sum_{n=1}^{N_K} \sum_{t=1}^{T_{\text{train}}^{(n)}} \log p(y_t^{(n)} \mid h_{t-1}^{(n)}, \psi) \]  

(6)

\(^3\text{Schwarz et al. [13] introduced an additional hyperparameter } \gamma_F \leq 1 \text{ to explicitly promote forgetting: } F = \sum_{k=1}^{K-1} \gamma_F^{k-1} F^{(k)} \text{. We left } \gamma_F = 1 \text{ throughout this work.} \)
We first consider typical classification problems (cf. Sec. 4.2 and Sec. 4.3). In this case, \( y^{(n)} \) is a one-hot encoded representation of a label \( y_t^{(n)} \in \{1, \ldots, F_{\text{out}}\} \), where \( F_{\text{out}} \) denotes the number of classes. We consider a softmax output \( \hat{y}_t^{(n)} = \text{softmax}(\beta_t \hat{z}_t^{(n)}) \), where \( \beta_t \) denotes a timestep-specific inverse temperature that may be used to bias the loss such that it puts more emphasis on certain timesteps. For instance, setting \( \beta_t = 0 \) results in timestep \( t \) being ignored for the computation of the loss. Indeed, for the experiments in Sec. 4.2 and Sec. 4.3 the loss is evaluated solely based on the prediction of the last timestep \( T \). Results in timestep \( t \) being ignored for the computation of the loss.

Lastly, we consider the NLL for the Copy Task and its variants (cf. Sec. 4.1), where the output has to match a binary target pattern. In this case, each pixel in the output pattern will be evaluated for each sample is performed, while accumulating importance values for each entry in the output pattern. Taken together, the NLL loss for matching binary output patterns can be specified via:

\[
- \sum_{n=1}^{N_K} \sum_{t=1}^{T_{\text{out}}^{(n)}} \log p(y_t^{(n)} | h_{t-1}^{(n)}, \psi) = - \sum_{n=1}^{N_K} \sum_{t=1}^{T_{\text{out}}^{(n)}} \sum_{c=1}^{F_{\text{out}}} [y_t^{(n)} = c] \log \left( \text{softmax}(\beta_t \hat{z}_t^{(n)})_c \right)
\]  

(7)

where \([\cdot]\) denotes the Iverson bracket and \( \text{softmax}(\cdot)_c \) refers to the \( c \)-th entry of the softmax output vector.

Conceptual differences to a hypernetwork-based approach. An important conceptual difference between EWC (and prior-focused methods in general) and the hypernetwork-based approach (cf. Sec. B.4) lies in the nature of Eq. 3. Whereas prior-focused methods aim to find \( \arg \max_{\psi} p(\psi | D_1, \ldots, D_K) \) (which necessitates a certain compatibility across tasks), the hypernetwork-based approach allows task-specific solutions \( \psi^{(k)} = \arg \max_{\psi} p(\psi | D_k) \), where knowledge transfer between tasks (to exploit compatibilities) is implicitly outsourced to a meta-model (the hypernetwork).

Complexity estimation. The regularization introduced in Eq. 5 leads to a time complexity increase of \( \mathcal{O}(|\psi|) \) when computing the loss. Additionally, the computation of Fisher values at the end of each of the \( K \) tasks leads to a further increase in time complexity. Indeed, a forward and backward computation for each sample is performed, while accumulating importance values for each entry in \( \psi \). Assuming forward and backward computation only increases linearly with \( \psi \), we can summarize this contribution via \( \mathcal{O}(|\psi| \sum_k N_k) \), where \( N_k \) is the number of samples in task \( k \).

The increase in space complexity arises due to the storage of the diagonal Fisher elements as well as the most recent MAP solution: \( \mathcal{O}(2|\psi|) \).

B.6 Synaptic intelligence

Synaptic intelligence (SI) [9] is another weight-importance method that, in contrast to EWC, computes the importance values online, i.e., during training rather than at the end of training. The method is based on a first-order Taylor approximation to estimate the loss change after an optimizer update step. This allows estimating the influence of each individual weight \( \psi_i \) on the loss change. Thus, at each optimization step \( s \) while training task \( k \), an online importance estimate \( \hat{\omega}_i^{(k)} \) of \( \psi_i \) is updated via:

\[
\hat{\omega}_i^{(k)} \leftarrow \hat{\omega}_i^{(k)} - \Delta \psi_i(s) \frac{\partial \mathcal{L}_{\text{task}}(\psi, B(s))}{\partial \psi_i}
\]  

(9)
where $\Delta \psi_i(s)$ is the weight change determined by the optimizer at step $s$, and $B(s) \subseteq D_k$ is the $s$-th minibatch. Importantly, we compute both the optimizer update $\Delta \psi_i(s)$ and the gradient based on the task-specific loss $L_{\text{task}}(\cdot)$ only, ignoring potential regularizers such as the SI regularizer itself. To do so, we compute the update step $\Delta \psi_i(s)$ that would be taken by the optimizer without actually taking it. Interestingly, we did not observe a noticeable difference between this variant, where importance is solely based on task-specific influences, and one where the full loss is taken into consideration.

After training of task $k$ is completed, the final importance values $\Omega_i^{(k)}$ are computed as follows:

$$
\Omega_i^{(k)} = \Omega_i^{(k-1)} + \frac{\omega_i^{(k)}}{\Delta \psi_i^{(k)}} + \epsilon
$$

(10)

where $\Delta \psi_i^{(k)}$ is the complete weight change (of weight $\psi_i$) between before and after training on task $k$, and $\epsilon (= 1e-3)$ ensures numerical stability. If $\omega_i^{(k)} < 0$, we clamp its value to zero to avoid negative importance values. The SI loss function for training task $K$ is:

$$
\min_\psi L_{\text{task}}(\psi, D_K) + \lambda_{\text{SI}} \sum_{i=1}^{\mid \psi \mid} (\Omega_i^{(K-1)}(\psi_i - \tilde{\psi}_i^{(K-1)})^2
$$

(11)

**Complexity estimation.** The increase in time complexity due to the regularization introduced in Eq. (11) can be summarized as $O(|\psi|)$ per loss evaluation. An additional increase arises due to the online estimation of importance values (cf. Eq. 9). The contribution is bounded by $O(|\psi|)$ per training iteration.

The increase in space complexity arises due to the storage of $\Omega_i^{(K)}$, $\tilde{\omega}_i^{(K)}$, $\omega_i^{(K)}$, as well as a temporary copy of $\psi$ from before the current optimizer step in order to compute $\Delta \psi_i(s): O(4|\psi|)$.

### B.7 Masking

Context-dependent gating (or Masking) is a mechanism to alleviate catastrophic interference that was introduced by Masse et al. [7]. The method stores a random binary mask per task, which is used to gate all hidden activations. For LSTM layers, this method masks the hidden state $h_t$. For vanilla RNNs, which in our case are inspired by Elman networks, Masking affects the hidden state $h_t$ as well as the RNN layer output. Throughout all experiments, we masked 80% of the hidden activations. Due to the independent and random generation of masks, small overlaps across tasks may occur (or if activations are computed using shared weights such as in CNNs). To prevent catastrophic interference within those overlaps, one may combine Masking with, for instance, SI (cf. Sec. B.6). If subnetworks are sufficiently task-specific, SI will only influence the overlaps with subnetworks of previous tasks, without introducing rigidity for the remainder of the current subnetwork.

**Complexity estimation.** Masking does not introduce an increase in time complexity. On the contrary, if efficiently implemented, it may decrease time complexity since only activations of the active subnetwork need to be computed.

Since a binary mask per task needs to be stored, there is an increase in space complexity of $O(K|\psi|)$. However, binary masks can be stored efficiently, as only one bit per task/activation is required. If combined with SI, the space and time complexity considerations mentioned in Sec. B.6 also apply.

### B.8 Coresets

Coresets refers to CL methods that store subsets of past data that can be mixed with new data in order to prevent catastrophic interference [6, 44]. Rebuffi et al. [44] discusses strategies on how to properly select coreset samples. Here, we simply take a random subset of $N$ input samples from

---

6Note that for LSTMs the hidden state is also the layer output, whereas a vanilla RNN layer (an Elman network) has an additional linear readout of the hidden state. If Masking would only affect this readout, then there would be unhampered catastrophic interference in the crucial hidden-to-hidden computation.
each previous dataset, denoted by Coresets-N, for which we aim to keep the network predictions fixed when learning new tasks. Therefore, a copy of the network \( q^{(K-1)}_i \) before learning task \( K \) is generated and used to create soft-targets \( \tilde{y}_{1:T_{in}} = f(x_{1:T_{in}} \nu^{(K-1)}) \), where \( x_{1:T_{in}} \) is a sample taken from a coreset \([4,52]\). The soft-targets \( \tilde{y}_{1:T_{in}} \) are distilled \([54]\) into the network while training on the current task. This can be viewed as a form of regularization that incorporates past data. In addition to the current mini-batch \( B(s) \subseteq D_K \), an additional mini-batch \( \tilde{B}(s) \) is assembled from inputs randomly distributed across all \( K - 1 \) coresets together with their corresponding soft-targets. We chose to always assume that both of these mini-batches have the same size. The total loss for task \( K \) can then be described as follows:

\[
\min_\psi \mathcal{L}_{\text{task}}(\psi, B(s)) + \lambda_{\text{distill}}\mathcal{L}_{\text{distill}}(\psi, \tilde{B}(s))
\]

(12)

where \( \lambda_{\text{distill}} \) is a hyperparameter and \( \mathcal{L}_{\text{distill}}(\cdot) \) denotes the distillation loss \([54]\).

**Complexity estimation.** The time complexity of the loss evaluation roughly doubles (the time complexities of \( \mathcal{L}_{\text{task}}(\cdot) \) and \( \mathcal{L}_{\text{distill}}(\cdot) \) are comparable).

Storage increases by \( \mathcal{O}(|\psi|) \) due to the network copy \( \psi^{(K-1)}_i \). However, the critical storage increase is due to the storage of past data, which can be summarized by \( \mathcal{O}(KNF_{in}T_{in}) \), assuming all samples within coresets have the same temporal dimension \( T_{out} \).

**B.9 Generative replay**

Conceptually, Generative Replay \([42,43]\) is similar to Coresets (cf. Sec. B.8), i.e., it is based on the rehearsal of past input data whose soft-targets are subsequently distilled into the network (cf. Eq. [12]). The major difference is that Coresets directly store past data, while Generative Replay relies on the ability to learn a generative model of past input data. In this study, we consider Variational Autoencoders (VAE) \([55,56]\) as generative models. We first recap the workings of a VAE on sequential data in Sec. B.10 before explaining in Sec. B.11 how catastrophic interference can be mitigated in a VAE when learning a set of tasks sequentially.

**B.10 Sequential variational autoencoder**

The traditional VAE (for static data) defines a generative model via marginalization of a hidden variable model: \( p_\nu(x) = \int_Z p_\nu(x \mid z)p(z)dz \). Here, \( z \in Z \) denotes a latent variable (or hidden cause), \( p(z) \) is the prior and \( p_\nu(x \mid z) \) is a likelihood function defined via a decoder network whose parameters are denoted by \( \nu \). To learn the parameters \( \nu \) given a dataset \( D = \{x_n\}_{n=1}^N \), the corresponding hidden causes \( z_n \) have to inferred from the posterior \( p_\nu(z \mid x) \propto p_\nu(x \mid z)p(z) \). However, the precise value of the posterior is in general intractable. Therefore, VAEs resort to variational inference (VI) to approximate the posterior using \( q_\psi(z \mid x) \approx p_\nu(z \mid x) \), where \( q_\psi(z \mid x) \) is realized through an encoder network with parameters \( \psi \). VI utilizes the following inequality (cf. \([55]\) for a derivation):

\[
\log p_\nu(x) \geq -KL(q_\psi(z \mid x) \parallel p(z)) + \mathbb{E}_{q_\psi(z \mid x)}[\log p_\nu(x \mid z)]
\]

(13)

where the right-hand side is commonly known as evidence lower bound (ELBO). VAE training proceeds by maximizing the ELBO or equivalently by minimizing the negative ELBO which decomposes into a prior-matching term \( KL(q_\psi(z \mid x) \parallel p(z)) \) and a negative log-likelihood (NLL) term \(-\mathbb{E}_{q_\psi(z \mid x)}[\log p_\nu(x \mid z)] \).

Next, we discuss how to extend this framework to sequential data (also cf. \([57,58]\)). We use an independence assumption when defining a prior for a sequence of hidden causes:

\[
p(z_{1:T}) = \prod_t p(z_t)
\]

(14)
In addition, we consider the following decomposition of the likelihood function:

\[ p_\nu(x_{1:T} \mid z_{1:T}) = \prod_t p_\nu(x_t \mid x_{<t}, z_{\leq t}) \]  

(15)

The decoder network is an RNN defined via \( [\phi, h_{\text{dec}}] = f_{\text{dec,step}}(z_t, h_{\text{dec},t-1}, \nu) \), where \( h_{\text{dec}} \) denotes the hidden state of the decoder network and \( q_{\nu} \in \Phi \) denotes the parameters of a parametric distribution (e.g., a Gaussian), which can be used to tractably compute densities \( p_\nu(x_t \mid x_{<t}, z_{\leq t}) \) conditioned on \( z_{1:T} \).

As a last ingredient, we have to define the recognition model \( q_\psi(z_{1:T} \mid x_{1:T}) \). If the prior and likelihood defined above are inserted into Bayes rule, there is no obvious way to simplify the dependency structure of the true posterior such that the autoregressive nature of an RNN recognition model is not violated. We therefore apply an additional assumption when defining the decomposition applied to our recognition model:

\[ q_\psi(z_{1:T} \mid x_{1:T}) \quad \text{chain rule of prob} \prod_t q_\psi(z_t \mid z_{<t}, x_{1:T}) \quad \text{filtering assumption} \prod_t q_\psi(z_t \mid z_{<t}, x_{\leq t}) \]  

(16)

Analogously to the likelihood, the components \( q_\psi(z_t \mid z_{<t}, x_{\leq t}) \) of the approximate posterior are represented by an RNN encoder network \( [\xi, h_{\text{enc}}^\psi] = f_{\text{enc,step}}(x_t, h_{\text{enc},t-1}, \psi) \), where \( \xi \in \Xi \) are the parameters of a distribution over the latent space \( Z \).

At this point, we have all ingredients of the ELBO (cf. Eq. [13]) defined and can now focus our discussion on how to tractably evaluate the ELBO for the case of sequential data. We will start with decomposing the prior-matching term:

\[
KL(q_\psi(z_{1:T} \mid x_{1:T}) \mid | p(z_{1:T})) = \int_{z_{1:T}} \prod_t q_\psi(z_t \mid z_{<t}, x_{\leq t}) \log \frac{q_\psi(z_t \mid z_{<t}, x_{\leq t})}{p(z_t)} dz_{1:T} \\
= \sum_t \int_{z_{1:T}} \prod_t q_\psi(z_t \mid z_{<t}, x_{\leq t}) \log \frac{q_\psi(z_t \mid z_{<t}, x_{\leq t})}{p(z_t)} dz_{1:T} \\
= \sum_t \int_{z_{1:T}} \prod_t q_\psi(z_t \mid z_{<t}, x_{\leq t}) \log \frac{q_\psi(z_t \mid z_{<t}, x_{\leq t})}{p(z_t)} dz_{1:T} 
\]

(17)

Note that the last manipulation is possible since the log-ratio does not depend on \( z_{t'} \) when \( t' > t \) and, therefore, the log-ratio can be moved outside the respective integrals which evaluate to 1. We can further simplify the expression as follows:

\[
KL(q_\psi(z_{1:T} \mid x_{1:T}) \mid | p(z_{1:T})) = \sum_t \int_{z_{1:T}} \prod_{t' \leq t} q_\psi(z_{t'} \mid z_{<t'}, x_{\leq t'}) \int_{z_{\leq t}} q_\psi(z_t \mid z_{<t}, x_{\leq t}) \log \frac{q_\psi(z_t \mid z_{<t}, x_{\leq t})}{p(z_t)} dz_{t} dz_{1:t-1} \\
= \sum_t \int_{z_{1:T}} \prod_{t' \leq t} q_\psi(z_{t'} \mid z_{<t'}, x_{\leq t'}) KL(q_\psi(z_t \mid z_{<t}, x_{\leq t}) \mid | p(z_t))) dz_{1:t-1} \\
= \sum_t \int_{z_{1:T}} q_\psi(z_t \mid x_t) \cdots \int_{z_{\leq t-1}} q_\psi(z_{t-1} \mid z_{<t-1}, x_{\leq t-1}) KL(\cdots) dz_{t-1} \cdots dz_1 
\]

(18)

Note that the KL divergence term \( KL(q_\psi(z_t \mid z_{<t}, x_{\leq t}) \mid | p(z_t)) \) in Eq. [18] is analytically solvable based on a proper choice of prior and likelihood. The surrounding integrals can be estimated via Monte-Carlo (MC) sampling. In the simplest case, they are estimated by taking one sample per integral, i.e., given an input sequence \( x_{1:T} \), we use the recognition model \( [\xi, h_{\text{enc}}^\psi] = f_{\text{enc,step}}(x_t, h_{\text{enc},t-1}, \psi) \)

\[ \sum_t \int_{z_{1:T}} \prod_{t' \leq t} q_\psi(z_{t'} \mid z_{<t'}, x_{\leq t'}) \int_{z_{\leq t}} q_\psi(z_t \mid z_{<t}, x_{\leq t}) \log \frac{q_\psi(z_t \mid z_{<t}, x_{\leq t})}{p(z_t)} dz_{t} dz_{1:t-1} \\
= \sum_t \int_{z_{1:T}} \prod_{t' \leq t} q_\psi(z_{t'} \mid z_{<t'}, x_{\leq t'}) KL(q_\psi(z_t \mid z_{<t}, x_{\leq t}) \mid | p(z_t))) dz_{1:t-1} \\
= \sum_t \int_{z_{1:T}} q_\psi(z_t \mid x_t) \cdots \int_{z_{\leq t-1}} q_\psi(z_{t-1} \mid z_{<t-1}, x_{\leq t-1}) KL(\cdots) dz_{t-1} \cdots dz_1 
\]
to compute a latent sequence \( z_{1:T} \) via \( z_t \sim q_\psi(z_t \mid z_{<t}, x_{<t}) \Leftrightarrow z_t \sim p_\xi(z_t) \), where \( p_\xi(\cdot) \) is an explicit parametric distribution that we chose for the latent space (typically Gaussian), to evaluate the KL term.

Note, \( \xi_t \) depends on \( x_t \) and \( h_{enc}^{\text{step}} \). However, in the implementation that we chose for this study, \( h_{enc}^{\text{step}} \) does not explicitly depend on \( z_{<t} \) (only implicitly through its distribution determined by \( \xi_t \)) even though \( q_\psi(z_t \mid z_{<t}, x_{<t}) \) requires an explicit dependency.\(^7\)

Taken together, we approximate the prior-matching term as follows:

\[
KL(q_\psi(z_{1:T} \mid x_{1:T}) || p(z_{1:T})) \approx \sum_t KL(p_\xi(z_t) || p(z_t))
\]  

(19)

Similarly, we can handle the negative log-likelihood (NLL):

\[
\text{NLL} = -E_{q_\psi(z_{1:T} \mid x_{1:T})} \left[ \log p_\nu(x_{1:T} \mid z_{1:T}) \right]
\]

\[
= -\int_{z_{1:T}} \prod_t q_\psi(z_t \mid z_{<t}, x_{<t}) \sum_t \log p_\nu(x_t \mid x_{<t}, z_{<t}) \, dz_{1:T}
\]

\[
= -\sum_t \int_{z_1} \int_{z_{<t}} q_\psi(z_t \mid z_{<t}, x_{<t}) \log p_\nu(x_t \mid x_{<t}, z_{<t}) \, dz_{1:t}
\]

\[
= -\sum_t \int_{z_t} q_\psi(z_t \mid x_t) \cdots \int_{z_1} q_\psi(z_1 \mid x_1) \prod_t \log p_\nu(x_t \mid x_{<t}, z_{<t}) \, dz_t \cdots dz_1
\]

\[
\text{MC sample size of 1} \approx -\sum_t \log p_\nu(x_t \mid x_{<t}, z_{<t})
\]  

(20)

If \( p_\nu(x_t \mid x_{<t}, z_{<t}) \) is a Gaussian distribution (which we assume for the SMNIST and AudioSet experiments), Eq. (20) becomes a sum over mean-squared error (MSE) losses (after dropping constant terms and assuming the covariance matrix to be a scaled identity matrix \( \tau^{-1} I \)). Thus, we assume the output \( \varphi_t \) of the decoder \( [\varphi_t, h_{dec}^{\text{step}}] = f_{\text{dec,step}}(x_t, h_{enc}^{\text{step}}, \nu) \) is the mean of a Gaussian distribution \( \mathcal{N}(\varphi_t; \varphi_t, \tau^{-1} I) \), therefore \( \mathcal{X} \equiv \Phi \). One could sample reconstructions from this distribution using the reparameterization trick \([55] \). However, at this level we do not introduce additional noise and instead aim to match encoder input \( x_t \) and decoder output \( \varphi_t \) directly.\(^8\)

\[
\text{NLL} \approx \sum_{t=1}^{T_n} \frac{\tau}{2} \| x_t - \varphi_t \|^2
\]  

(21)

In case of the Copy Task (and its variants), it makes sense to choose \( p_\nu(x_t \mid x_{<t}, z_{<t}) \) to be a Bernoulli distribution (assuming the raw decoder output \( \varphi_t \) has been squeezed through a sigmoid):

\[
\text{NLL} \approx \sum_{t=1}^{T_n} \sum_{f=1}^{F_n} -x_{t,f} \log \varphi_{t,f} -(1 - x_{t,f}) \log (1 - \varphi_{t,f})
\]  

(22)

### B.11 Generative replay using a sequential VAE

Above, we describe how we train a VAE on sequential data. In order to use it as a generative model for CL, we have to employ strategies that mitigate catastrophic interference when training consecutively on multiple tasks. We therefore explore two strategies inspired by related work on static data, RtF \([43] \) (referred to as Generative Replay in the main text) and HNet+R \([15] \). In both cases, we use the main model simultaneously as classifier and VAE encoder \([\hat{y}_t, \xi_t, h_t] = f_{\text{step}}(x_t, h_{enc, t-1}, \psi) \), where

---

\(^7\)This limitation could be overcome if the RNN definition would be slightly adapted. For instance, if the definition of the encoder would change to \( [\xi_t, h_{enc}^{\text{step}}] = f_{\text{enc,step}}(x_t, z_{t-1}, h_{enc}^{\text{step}}) \) with \( z_{t-1} \sim p_{\xi_{t-1}}(z_{t-1}) \).

\(^8\)Note, in contrast to the approximate posterior distribution \( q_\psi(z_t \mid z_{<t}, x_{<t}) \) (which we crucially require to replay samples of prior tasks), we only require a sensible mean of the likelihood \( p_\nu(x_t \mid x_{<t}, z_{<t}) \) to represent reconstructions.
We empirically showed in Fig. 2 that increasing pattern lengths \( p \) where \( \beta \) is referred to training the VAE on data from all tasks seen so far. However, in a CL setting, data from previous tasks is not available. Therefore, a checkpointed decoder \( \rho_i^{(K-1)} \) is used to replay data from tasks 1 to \( K - 1 \) while training on task \( K \). In summary, similar to Coresets (cf. Sec. B.8), a mini-batch \( B(s) \) with data from task \( K \) and a mini-batch \( \tilde{B}(s) \) with replayed data (using \( \tilde{\rho}_i^{(K-1)} \)) from tasks 1 to \( K - 1 \) is assembled. In addition to the distillation loss (cf. Eq. 12 using hyperparameter \( \lambda_{\text{distill}} \)), that only affects the encoder \( f_{\text{step}} \), the reconstruction loss \( L_{\text{rec}}(\cdot) \) (cf. Eq. 20) and prior-matching loss \( L_{\text{pm}}(\cdot) \) (cf. Eq. 19) are evaluated on \( B(s) \) and \( \tilde{B}(s) \):

\[
\min_{\psi, \nu} L_{\text{task}}(\psi, B(s)) + \lambda_{\text{distill}} L_{\text{distill}}(\psi, \tilde{B}(s)) + \lambda_{\text{rec}} L_{\text{rec}}(\psi, \nu, B(s) \cup \tilde{B}(s))
\]

\[
+ \lambda_{\text{pm}} L_{\text{pm}}(\psi, \nu, B(s) \cup \tilde{B}(s))
\]

(23)

where \( \lambda_{\text{rec}} \) and \( \lambda_{\text{pm}} \) denote two new hyperparameters. Note, in order to train a multi-head main network \( f_{\text{step}} \) with replayed data, the output head (task identity) of replayed data has to be known. To achieve this, task identity has to be provided as a one-hot encoding to the decoder in addition to the latent variable \( z_t \).

The generative model used by RtF is therefore continuously retrained on its own replayed data. Hence, distributional shifts and mismatches accumulate over time, leading to a decrease in quality of replayed samples [15]. The method HNET+R [15] circumvents this problem of RtF by training a task-specific decoder, where decoders of previous tasks are protected by a hypernetwork (cf. Eq. 1) and only the current task’s decoder is trained on actual data. To do so, a hypernetwork is introduced for the decoder (and not the main network) \( \nu = h_{\text{dec}}(e_k^{\text{dec}}, \theta^{\text{dec}}) \). The loss in this case becomes (cf. Eq. 1 and Eq. 23):

\[
\min_{\psi, \theta^{\text{dec}}} L_{\text{task}}(\psi, B(s)) + \lambda_{\text{distill}} L_{\text{distill}}(\psi, \tilde{B}(s)) + \lambda_{\text{rec}} L_{\text{rec}}(\psi, \theta^{\text{dec}}, B(s))
\]

\[
+ \lambda_{\text{pm}} L_{\text{pm}}(\psi, \theta^{\text{dec}}, B(s))
\]

\[
+ \frac{\beta_{\text{dec}}}{K - 1} \sum_{k=1}^{K-1} \| h_{\text{dec}}(e_k^{\text{dec}}, \theta^{\text{dec}}) - h_{\text{dec}}(e_k^{(\text{dec,K-1})}, \tilde{\theta}^{(\text{dec,K-1})}) \|^2_2
\]

(24)

where \( \beta_{\text{dec}} \), \( e_k^{(\text{dec,K-1})} \), and \( \tilde{\theta}^{(\text{dec,K-1})} \) are defined for the decoder hypernetwork \( h_{\text{dec}} \) analogously as described for the main network’s hypernetwork in Sec. 3.

**Complexity estimation.** RtF and HNET+R are affected from the same complexity considerations as Coresets (cf. Sec. B.8) except for storing past data (which are instead replayed from a checkpointed decoder \( f_{\text{step,dec}} \) resp. decoder-hypernetwork \( h_{\text{dec}} \)). Method HNET+R has the additional complexity increases mentioned in Sec. B.4. Both methods require maintaining an additional decoder network. Both also require the evaluation of two extra loss terms, \( L_{\text{rec}}(\cdot) \) and \( L_{\text{pm}}(\cdot) \), whereas this cost is doubled for RtF as it always evaluates these terms on current and replayed data.

### C A theoretical view on CL in linear RNNs

In this section, we provide theoretical insights on why high working memory requirements might be problematic when weight-importance methods such as EWC are applied to RNNs.

We empirically showed in Fig. 2 that increasing pattern lengths \( p \) of Copy Task inputs lead to increasing weight-importance values as calculated by EWC. We also observed that higher working memory requirements (resulting from increasing pattern length) force the networks to utilize more of their capacity, which leads to a higher intrinsic dimensionality of the hidden state. These observations led us to the prediction that high working memory requirements can lead to a saturation of weight-importance values, thus decreasing performance of methods such as EWC when sequentially learning multiple tasks.
Here, we examine these statements from a theoretical perspective for the case of linear RNNs. More specifically, we explore why using a shared set of recurrent weights for several tasks can be problematic when the intrinsic dimensionality of the hidden state increases. Note that this framework is therefore applicable to any method that uses a single set of recurrent weights for several tasks, no matter whether these are learned sequentially or not (which includes weight-importance methods but also, for instance, replay methods and the multitask setting).

**Model.** We consider a linear RNN with one recurrent hidden layer $h_t$ of dimension $n_h$. The dynamics of the network are defined as follows:

$$h_t = W_{hh} h_{t-1} + W_{xh} x_t$$

$$\hat{y}_t = W_{hy} h_t$$

(25)  

(26)

with $W_{hh}$, $W_{xh}$ and $W_{hy}$ weight matrices. We consider a setting in which the network has to learn $K$ different tasks using the shared weights $W_{hh}$ and $W_{xh}$, and a set of task-specific output heads $W_{hy}^{(k)}$. We denote by $h_t^{(k)} \in \mathbb{R}^{n_h}$ the content of $h_t$ that is utilized for the task-specific processing of task $k$.

**Task.** We consider a variant of the Copy Task, in which at timesteps $t = 1 : p$ the network needs to output a manipulated copy of the network inputs at timesteps $t = -p : -1$. The input $x_t$ is zero for $t > 0$, and the specific manipulation of the input is different for all $K$ tasks.

**Simplifying assumptions.** To make the analysis as clear as possible we make the following simplifying assumptions:

1. Task-specific recurrent processing on $h_t$ via $W_{hh}$ is still required for $t > 0$ in order to solve task $k$ (i.e. the task-specific output heads $W_{hy}^{(k)}$ are not rich enough to model all task variabilities).

2. Each task $k$ needs a completely distinct processing mechanism from other tasks. There is thus no possibility of transfer-learning across tasks, and if the processing of $h_t^{(k)}$ by $W_{hh}$ overlaps with the processing of $h_t^{(l \neq k)}$, the two tasks will interfere with each other, leading to a drop in performance.

**Theoretical analysis of the linear toy problem.** Our PCA analyses show that the hidden state $h_t$ is embedded in a lower-dimensional linear subspace of $\mathbb{R}^{n_h}$. Based on the above simplifying assumptions, the only way for a linear RNN to ensure a task-specific processing is that $h_t^{(k)}$, the information within $h_t$ relevant for solving task $k$, populates distinct and non-overlapping linear subspaces of $\mathbb{R}^{n_h}$ for each task across all $t > 0$: 

$$h_t^{(k)} \in S_k$$

$$S_k \cap S_{l \neq k} = \{0\}$$

(27)  

(28)

If this wasn’t the case and $S_k$ overlapped with other subspaces, $h_t^{(k)}$ could have components in $S_{l \neq k}$, which would be influenced by the task-specific processing of other tasks.

Because $W_{hh}$ is sequentially applied to $h_t$, it must perform a subspace-retaining operation on $h_t^{(k)} \in S_k$ such that:

$$h_{t+1}^{(k)} = W_{hh} h_t^{(k)}$$

$$h_{t+1}^{(k)} \in S_k$$

(29)  

(30)

The task-specific output head $W_{hy}^{(k)}$ can then select a linear subspace $S_k$ of $h_t$ that serves as output for task $k$. Hence, as long as the $W_{hh}$ can represent a task-specific and subspace-retaining operation on $h_t^{(k)}$, it is possible for the RNN to represent $K$ different tasks that do not interfere with each other, and the task-specific output head $W_{hy}^{(k)}$ can select the appropriate subspace of $h_t$ to present at the output.
In the following, we show that it is possible to have such task specific processing of the hidden-state vectors by \( W_{hh} \) if the subspaces \( S_k \) are orthogonal to each other and if the intrinsic dimensionality of task-relevant information within the hidden space is sufficiently small across tasks. We use this finding as an intuition to justify why increasing intrinsic dimensionality of the hidden state can lead to interference across tasks when a single matrix \( W_{hh} \) is used.

Let’s represent each subspace \( S_k \) by the column space of a matrix \( U_k \) with orthonormal columns. Because \( h_t^{(k)} \) only has components in \( S_k \), it can be written as:

\[
\sum_{c_t^{(k)}} = U_k c_t^{(k)} \tag{31}
\]

with \( c_t^{(k)} \in \mathbb{R}^{p_k} \) the coordinates of \( h_t^{(k)} \) in the basis \( U_k \). If subspaces are orthogonal and \( \sum_k p_k \leq n_h \), we can state that: \( U_k \) is orthogonal to all other \( U_{l \neq k} \), and that \( \bar{U} = \left[ U_1 \ldots U_K \bar{U} \right]_{n_h \times n_h} \) is an orthogonal basis for \( \mathbb{R}^{n_h} \), with \( \bar{U} \) orthogonal to all \( U_k \).

Now we can define \( Q = \bar{U}^T W_{hh} \bar{U} \), a change of basis of \( W_{hh} \) under \( \bar{U} \). \( Q \) can be structured in the following blocks:

\[
Q = \begin{bmatrix}
Q_{11} & Q_{12} & \cdots & Q_{1\sim} \\
Q_{21} & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & Q_{K\sim} \\
Q_{\sim 1} & \cdots & Q_{\sim K} & Q_{\sim \sim}
\end{bmatrix} \tag{32}
\]

where \( Q_{ij} \) corresponds to the computation within \( W_{hh} \) that leads a subspace transformation from \( S_j \) to \( S_i \). Then, \( h_t^{(k)} \) is given by

\[
\begin{align*}
    h_t^{(k)} & = W_{hh} h_t^{(k)} \\
    & = W_{hh} U_k c_t^{(k)} \\
    & = \bar{U} Q \bar{U}^T U_k c_t^{(k)} \\
    & = \sum_{l=1}^{K} U_l Q_{lk} c_t^{(k)} + \bar{U} Q_{\sim k} c_t^{(k)} \tag{36}
\end{align*}
\]

We can easily see that, if \( Q_{ij} = 0 \) for \( i \neq j \), we obtain:

\[
    h_t^{(k)} = U_k Q_{kk} c_t^{(k)} \tag{37}
\]

Therefore, one can easily design \( Q \) in such a way that \( W_{hh} \) performs a subspace-retaining transformation on \( h_t^{(k)} \), i.e. \( Q \) needs to have a block diagonal structure. Otherwise, \( U_l Q_{lk} c_t^{(k)} \) for \( l \neq k \) is non-zero, and \( h_t^{(k)} \) will contain components in \( S_{l \neq k} \).

To summarize, we see that it is possible for the RNN to have a task-specific processing of the hidden-state vector for each task, without interfering with the other tasks, as long as \( \sum_k p_k \leq n_h \). If \( \sum_k p_k > n_h \), it is not possible anymore to have \( K \) orthogonal linear subspaces \( S_k \), which can lead to interference between tasks and a resulting drop in performance.

**Implications for CL.** We showed that it is possible to build a linear RNN that doesn’t suffer from interference across tasks despite using a single set of recurrent weights, as long as the intrinsic dimensionality of the hidden space is not too large. This observation has clear implications for weight-importance methods in CL, which progressively restrict the plasticity of a single set of recurrent weights when sequentially learning different tasks. Theoretically, weight-importance methods can encourage task-relevant information of the hidden state to be encoded in orthogonal subspaces, such that the learning of new tasks does not interfere with the previously learned tasks. However, if the subspace dimensionality \( p_k \) increases (e.g., for increasing pattern lengths in the Copy Task) or if the number of tasks is too large, leading to \( \sum_k p_k > n_h \), the various tasks will start interfering with each other, and the performance will drop.
Theoretical benefits of hypernetworks for CL. Following the above analysis, we conclude that hypernetworks provide a theoretical advantage over weight-importance methods. With hypernetworks, a task-specific \( W^{(k)}_{hh} \) can be generated for every new task, without forgetting \( W^{(l)}_{hh} \) of previous tasks \( l < k \). Hence, because \( W_{hh} \) does not need to represent subspace-retaining operations, a hypernetwork-based CL approach exhibits more flexibility for mitigating the stability-plasticity dilemma.

D Datasets and tasks

Here we provide details on the datasets and tasks used in this study. All details on preprocessing or generating data, as well as links for downloading the precise datasets can also be found in the accompanied code repository.

Table 4: Summary of the data used to train and evaluate one subtask for each of the three datasets. \( i \) and \( p \) refer to the input sequence and pattern lengths of the Copy Task, and \( m \) refers to the number of digits in a SMNIST sequence.

| Copy Task Variants | Split Sequential SMNIST | AudioSet |
|--------------------|-------------------------|----------|
| Classes           | N/A                     | 2        | 10       |
| Training samples  | 100000                  | 2 * 2000 | 10 * 750 |
| Validation samples| 1000                    | 2 * 50   | 10 * 50  |
| Test samples      | 1000                    | 2 * 500  | 10 * 200 |
| Input feature size| 8                      | 4        | 128      |
| Number of timesteps| \( i + 1 + p \)       | 117*m    | 10       |

D.1 Variations of the Copy Task

The Copy Task [16] is a synthetic dataset that we use to investigate different aspects of CL with sequential data. In this section, we first explain the basic Copy Task, and subsequently give details about the different manipulations we introduced to create variations of this task. For all variants, we used the training / validation / testing scheme described in Table 4.

D.1.1 Basic Copy Task

In the basic version of the Copy Task, networks are trained to memorize and reproduce random sequences, whose input sequence length \( i \) is equal to the length of the pattern \( p \) to be copied (\( i = p \), cf. Fig. [S1]). An input sample \( x_{1:T} \) (with \( T = i + 1 + p \)) consists of a random binary pattern at timesteps \( t = 1, \ldots, p \), where only feature dimensions 1 to \( F_{in} - 1 \) are used for the binary pattern, while feature dimension \( F_{in} \) is reserved for the stop bit. It contains zeroes at timesteps \( t = i - p, \ldots, i \), a stop flag at timestep \( t = i + 1 \) and zeroes at timesteps \( t = i + 2, \ldots, i + 1 + p \). The target output sequence \( y_{1:T} \) has no feature dimension reserved for the stop bit (\( F_{out} = F_{in} - 1 \)). It consists of zeroes up to timestep \( i + 1 \) and contains a copy of the random input pattern at timesteps \( t = i + 2, \ldots, i + 1 + p \) (cf. Fig. [S1]).

D.1.2 Padded Copy Task

The Padded Copy Task is a simple extension of the basic version described above where \( i > p \). This variant allows us to assess the effects of increasing sequence length, realized through increasing \( i \) while keeping the complexity of the underlying task constant (i.e., keeping \( p \) fixed).

D.1.3 Permuted Copy Task

To adapt the Copy Task to a CL setting, we introduce the Permuted Copy Task. Here, the output sequence \( y_{1:T} \) corresponding to an input sequence \( x_{1:T} \) is obtained by permuting the random input pattern \( x_{1:p} \) along the time dimension before assigning it as target to \( y_{i+2:i+1+p} \) (cf. Sec. D.1.1). In our CL experiments, the subtasks differ in the random permutation which is used to generate these input-output mappings.
D.1.4 Pattern Manipulation Task

The main challenge of the Copy Task is the memorization and recall of the presented input sequences. However, we additionally wanted to test how CL methods are affected by data processing requirements that go beyond simple memorization and which are different across tasks. The Pattern Manipulation Task offers a way to gradually increase the difficulty of this processing. Here we exclusively consider the case where $p = i$. Target patterns $y_{i+2:i+1+p}$ are generated from input patterns $x_{1:p}$ by iterating the following procedure $r$ times (where $r$ determines the task difficulty). We start by assigning $y_{i+2:i+1+p} \leftarrow x_{1:p}$ and then iterate for $r' = 1 \ldots r$

1. Permute $x_{1:p}$ along the time dimension using the $r'$-th permutation to generate a pattern $x_{1:p}^{(r')}$.  
2. Update $y_{i+2:i+1+p}$ by computing the logical XOR operation between the current $y_{i+2:i+1+p}$ and $x_{1:p}^{(r')}$.  

D.2 Sequential Stroke MNIST

Stroke MNIST (SMNIST) [47] represents MNIST images as a sequence of quadruples $\{dx_i, dy_i, eos_i, eod_i\}_{i=1}^T$. The length $T$ of the sequence corresponds to the number of pen displacements needed to define the digit, $(dx_i, dy_i)$ correspond to the relative offset from the previous pen position, $eos_i$ is a binary feature denoting the end of a stroke, and $eod_i$ denotes the end of a digit. We downloaded the dataset [9] and split the 70000 sample digits into training, validation and test sets (50000, 10000 and 10000 samples respectively). Since samples have different sequence lengths $T$, we zero padded the samples to obtain a uniform input length of 117 (maximal $T$). For our Split Sequential SMNIST experiments, we generated training, validation and test sample sequences from the corresponding digit sets. For experiments with $m$ digits per sequence, we generated the same number of samples for all of the possible $2^m$ binary sequences (e.g. 22, 23, 32 and 33 for $m = 2$ in the split containing only 2s and 3s). Finally we randomly assigned the $2^m$ possible sequences to two classes to create a binary decision problem.

[9] https://github.com/edwin-de-jong/mnist-digits-stroke-sequence-data/
## D.3 AudioSet

AudioSet [18] consists of more than two million 10-second audio samples, that are hierarchically ordered into 632 classes. To generate a set of classification tasks for CL, we selected and preprocessed a subset of the available data. Following Kemker et al. [49], we selected classes and samples according to the following criteria. We only considered classes that have (1) no restrictions according to the AudioSet ontology, (2) no parent-child relationship with any of the other classes and (3) a quality estimate provided by human annotators of \( \geq 70\% \). Samples were excluded if they did not contain data for the entire 10 seconds, or if they belonged to multiple of the considered classes. This procedure yielded a set of 189 classes, out of which 106 had a number of samples \( \geq 1000 \). To generate a balanced dataset, we randomly selected 1000 samples from each of the 100 classes with the highest number of samples. Finally, we split the 1000 samples per class into 800 samples for training and 200 samples for testing. The result of this procedure is available for download.\(^{11}\)

For our Split-AudioSet-10 experiments, we randomly grouped the 100 classes into 10 subtasks with 10 classes each. Validation samples were randomly selected from the training data, while maintaining the balance between classes.

## E Experimental details

Here we give further details on the results provided in the main text, and describe the procedures that we used to obtained these results.

### E.1 Copy Task

The analyses on the intrinsic dimensionality of the RNN’s hidden space were performed when learning a single task of the basic Copy Task setting (cf. Sec. D.1.1), where outputs are a copy of the inputs. We computed the hidden state activations \( h_{1:T} \) on the test set after learning the task. Then, we performed principal component analysis (PCA) on these activations, independently for each timestep. Specifically, for each timestep we performed PCA on a matrix of size \( \mathbb{R}^{N \times n_h} \), where \( N \) is the number of samples in the test set, and \( n_h \) is the number of hidden neurons. We then defined the intrinsic dimensionality of the hidden space as the number of principal components needed to explain 75\% of the variance. Qualitatively similar results can be obtained irrespective of the value of this threshold (we tested 20\%, 30\% ... 90\%).

### E.2 Sequential Stroke MNIST

We use LSTM main networks with 256 hidden units and a fully connected output head per task for all SMNIST experiments. Further parameter choices and hyperparameter searches are detailed in Sec.

| Method       | m = 1       | m = 2       | m = 3       | m = 4       |
|--------------|-------------|-------------|-------------|-------------|
| Online EWC   | 96.76 ± 0.68| 83.65 ± 1.79| 80.89 ± 0.86| 88.96 ± 4.19|
|              | 96.72 ± 0.67| 81.08 ± 1.71| 76.92 ± 1.24| 68.67 ± 1.48|
| HNET         | 99.30 ± 0.05| 96.06 ± 1.97| 95.60 ± 0.90| 87.04 ± 3.30|
|              | 99.20 ± 0.06| 95.52 ± 2.45| 87.97 ± 5.69| 85.68 ± 3.31|
| Fine-tuning  | 99.37 ± 0.08| 98.16 ± 0.17| 94.64 ± 1.83| 95.15 ± 1.17|
|              | 88.58 ± 1.36| 61.07 ± 1.61| 61.27 ± 2.19| 60.67 ± 1.16|
| Masking + SI | 99.37 ± 0.06| 94.24 ± 1.94| 91.05 ± 2.42| 67.48 ± 2.62|
|              | 99.38 ± 0.06| 94.25 ± 1.93| 91.05 ± 2.42| 67.47 ± 2.60|
| SI           | 98.01 ± 0.24| 93.82 ± 2.09| 75.89 ± 0.48| 93.55 ± 1.00|
|              | 95.23 ± 1.88| 80.67 ± 1.35| 75.53 ± 0.52| 70.20 ± 0.92|

\(^{10}\)https://research.google.com/audioset/download.html

\(^{11}\)https://www.dropbox.com/s/07dfeeuf5aq4w1h/audioset_data_balanced?dl=0
E.4 Table 5 shows all during and final accuracies of the SMNIST experiment described in Sec. 4.2.

E.3 Split-AudioSet-10

The experiments are performed using a main network with one LSTM layer with 32 units and a fully-connected output head per task. We initially used larger LSTM layers but observed extensive overfitting. Therefore, we ran a fine-tuning hyperparameter search for LSTM layer sizes: 8, 16, 32, 64, 128 and 256 and chose 32 as it resulted in the least amount of overfitting, while not leading to significant drops in maximum during accuracy. We also increased the hyperparameter search grid of the Multitask baseline compared to other reported results, incorporating larger batch sizes since all tasks are trained at once.

Figure S2: Task-averaged during and final test accuracies for Online EWC AudioSet experiments with varying numbers of classes per task, performed with different values for $\lambda_{EWC}$ (cf. Fig. 4).

For our AudioSet experiments with varying levels of difficulty, we used the best hyperparameter configurations from our Split-AudioSet-10 experiments (cf. Table 3). For Online EWC, we ran each experiment with multiple $\lambda$ values, as this parameter directly controls the trade-off between stability and plasticity. Fig. S2 shows during and final accuracies in the different settings for three $\lambda$ values. For the lowest $\lambda$ value, during accuracies are highest because few restrictions apply when solving individual tasks, but the performance drops when testing after all tasks are learned. For higher $\lambda$ values, final accuracies get closer to the during performance. This, however, comes at the cost of decreased during accuracies due to the restrictions imposed by the strong regularization controlled by $\lambda$.

E.4 Hyperparameter searches

We performed extensive hyperparameter searches for all methods in all experiments. Because of computational reasons, we limited the number of explored configurations to 100 per method and experiment (taking a random subset of all possible combinations defined by the search grid). By default, we tested the run with the best final accuracy on multiple random seeds. If however, the best run did not prove to be random seed robust, we additionally evaluated the second and third best runs on multiple random seeds and selected the configuration with the best results across a set of random seeds. For the HNET, we only searched feedforward fully-connected architectures that yielded a compression ratio of approximately 1, meaning that the number of weights in the hypernetwork is approximately equal to the number of weights of the main RNN. All experiments where conducted using the Adam optimizer. For all results, exact command line calls are provided in the README files of the published code base.

All experiments were performed with access to 32 GPUs of type NVIDIA RTX 2080 TI and NVIDIA QUADRO RTX 6000.
E.4.1 Basic Copy Task

For basic Copy Task experiments of a single task, used for the analyses on the intrinsic dimensionality of the RNN’s hidden space, the hyperparameters searches are described in Table 6.

Table 6: Hyperparameter search for the Basic Copy Task

| Hyperparameter                                | Searched values         |
|-----------------------------------------------|-------------------------|
| number of iterations                          | 20000                   |
| number of hidden units of the main network    | 256                     |
| main network activation function              | tanh                    |
| batch size                                    | 68, 128                 |
| learning rate                                 | 5e-4, 1e-3, 5e-3, 1e-2  |
| clip gradient norm                            | None, 1, 100            |
| orthogonal initialization                     | True, False             |
| orthogonal regularization strength            | 0, 1                    |

E.4.2 Padded Copy Task

For Padded Copy Task experiments with five tasks and \( p = 5, i = 25 \), the hyperparameter searches for the different methods are specified in Table 7.

Table 7: Hyperparameter search for the Padded Copy Task

| Method          | Hyperparameter                                | Searched values         |
|-----------------|-----------------------------------------------|-------------------------|
| All             | number of iterations                          | 20000                   |
|                 | batch size                                    | 128                     |
|                 | number of hidden units of the main network    | 256                     |
|                 | main network activation function              | tanh                    |
|                 | learning rate                                 | 5e-4, 1e-3, 5e-3, 1e-2  |
|                 | clip gradient norm                            | None, 1, 100            |
|                 | orthogonal initialization                     | True, False             |
|                 | orthogonal regularization strength            | 0, 1                    |
| Online EWC      | \( \lambda_{EWC} \)                            | 1e2, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9, 1e10 |
|                 | \( \beta \)                                   | 5, 10, 50               |
| HNET            | HNET hidden layers                            | "60,60,30"              |
|                 | HNET output size                              | 4000                    |
|                 | task embedding size                           | 16, 32                  |
|                 | chunk embedding size                          | 16, 32                  |

E.4.3 Permuted Copy Task

For Permuted Copy Task experiments with five tasks and \( p = i = 5 \), the hyperparameter searches are described in Table 8.

E.4.4 Pattern Manipulation Task

For Pattern Manipulation Task experiments, the hyperparameter searches are described in Table 9.

E.4.5 Sequential Stroke MNIST

The hyperparameter searches for Sequential Stroke MNIST experiments are described in Table 10.

The number of iterations was set according to the the number of digits in the sequences used in a given SMNIST experiment. We used 2000, 5000, 10000 and 20000 iterations for sequence lengths of 1, 2, 3 and 4 respectively.
Table 8: Hyperparameter search for the Permuted Copy Task

| Method      | Hyperparameter                                      | Searched values                  |
|-------------|-----------------------------------------------------|----------------------------------|
| **All**     | number of iterations                                 | 20000                            |
|             | batch size                                           | 128                              |
|             | number of hidden units of the main network           | 256                              |
|             | main network activation function                     | tanh                             |
|             | learning rate                                        | 5e-4, 1e-3, 5e-3, 1e-2            |
|             | clip gradient norm                                   | None, 1, 100                     |
|             | orthogonal initialization                            | True, False                       |
|             | orthogonal regularization strength                   | 0, 0.01, 1                       |
| **Online EWC** | λ<sub>EWC</sub>                                      | 1e2, 1e3, ..., 1e10              |
| **SI**      | λ<sub>SI</sub>                                       | 1e-3, 1e-2, 1e-1, 1, 1e2, 1e3    |
| **Generative Replay** | strength of the prior-matching term (λ<sub>prm</sub>) | 1, 10                           |
|              | strength of the reconstruction term (λ<sub>rec</sub>) | 1, 10                           |
|              | strength of the soft-target distillation loss (λ<sub>distill</sub>) | 1, 10                           |
|              | dimensionality of the VAE latent space               | 8, 100                           |
| **HNET**    | β                                                    | 1e-2, 1e-1, 1, 1e1, 1e2           |
|              | SD for the initialization of the task embeddings     | .1, 1                            |
|              | SD for the initialization of the chunk embeddings    | .1, 1                            |
|              | HNET hidden layers                                   | "", "25,25", "50,50"            |
|              | HNET output size                                     | 2000, 5000                       |
|              | chunk embedding size                                 | 16, 32                           |

Table 9: Hyperparameter search for the Pattern Manipulation Task

| Method      | Hyperparameter                                      | Searched values                  |
|-------------|-----------------------------------------------------|----------------------------------|
| **All**     | number of iterations                                 | 20000                            |
|             | batch size                                           | 128                              |
|             | number of hidden units of the main network           | 256                              |
|             | main network activation function                     | tanh                             |
|             | learning rate                                        | 5e-3, 1e-3, 5e-4                 |
|             | clip gradient norm                                   | None, 1, 100                     |
| **Online EWC** | λ<sub>EWC</sub>                                      | 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3  |
| **HNET**    | β                                                    | 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3   |
|              | HNET hidden layers                                   | "", "64,64", "64,64,32"         |
|              | HNET output size                                     | 2000, 4000                       |
|              | chunk embedding size                                 | 32                               |
|              | task embedding size                                  | 32                               |

E.4.6 Audioset

The hyperparameter searches for Audioset experiments are described in Table 11.

F Supplementary experiments and further remarks

F.1 Inferring task identity at test time

In this study, we only consider the case where task identity is known to the system during test time. A more challenging but arguably also more interesting CL scenario overcomes this constraint by
### Table 10: Hyperparameter search for Sequential Stroke MNIST

| Method          | Hyperparameter                        | Searched values |
|-----------------|----------------------------------------|-----------------|
| **All**         | batch size                             | 64              |
|                 | number of hidden units of the main network | 256             |
|                 | main network activation function       | tanh            |
|                 | learning rate                           | 1e-2, 1e-3, 1e-4 |
| **Online EWC**  | $\lambda_{EWC}$                        | 1e-1, 1e-2, ..., 1e10 |
| **SI**          | $\lambda_{SI}$                         | 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3 |
| **Masking + SI**| $\lambda_{SI}$                         | 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3 |
| **HNET**        | $\beta$                                | 1e-1, 1e0, 1e1   |
|                 | clip gradient norm                      | 1, 100          |
|                 | HNET hidden layers                      | "32,32", "32,16", "64,32,16", "32,32,32" |
|                 | HNET output size                        | 8000, 16000     |
|                 | chunk embedding size                    | 32, 64          |
|                 | task embedding size                     | 32, 64          |

### Table 11: Hyperparameter search for Audioset

| Method          | Hyperparameter                        | Searched values |
|-----------------|----------------------------------------|-----------------|
| **All**         | number of hidden units of the main network | 32              |
|                 | main network activation function       | tanh            |
|                 | batch size                             | 64, 128         |
|                 | number of iterations                   | 10000, 15000, 25000, 50000 |
|                 | learning rate                           | 1e-3, 1e-4, 1e-5 |
|                 | clip gradient norm                      | None, 1         |
|                 | orthogonal initialization               | False, True     |
|                 | orthogonal regularization strength      | 0, .1           |
| **Online EWC**  | $\lambda_{EWC}$                        | 1e-1, 1e0, ..., 1e10 |
| **SI**          | $\lambda_{SI}$                         | 1e-4, 1e-3, 1e-2, 1e-1, 1, 1e2, 1e3, 1e4 |
| **HNET**        | $\beta$                                | 1e-2, 1e-1, 1, 1e1   |
|                 | SD for the initialization of the task embeddings | .1, 1 |
|                 | SD for the initialization of the chunk embeddings | .1, 1 |
|                 | HNET hidden layers                      | "10,10", "20,20" |
|                 | HNET output size                        | 1000, 2000      |
|                 | chunk embedding size                    | 32              |
|                 | task embedding size                     | 32              |
| **Coreset**     | $\lambda_{distill}$                    | 1e-1, 1e0, 1e1   |
| **Multitask**   | batch size                             | 64, 128, 256, 512 |

inferring task identity based on the input sequence\footnote{This kind of CL scenario was termed CL3 in \cite{8, 15}.}. However, this is only possible for task sets where the data input distributions are sufficiently dissimilar to allow discrimination. For instance, the Copy Task and its variants would not be applicable to this scenario, as all tasks share the same input data distribution. Thus, inferring the task identity from the input alone is impossible in such a case.\footnote{It is however always possible to design an auxiliary system that infers task identity from a given and appropriately chosen context \cite{15, 38}.}

One possible way to achieve this is by sequentially turning the CL problem into a multitask problem via replay. For classification problems, the multi-head output could be replaced by a growing softmax.
that is trained analogously as described in Sec. B.8 and B.9. However, this solution relies on successfully training generative models or on storing a sufficient amount of past data. It also successively turns the CL problem into a multitask problem leading to an undesirable increase of computational demands.

An alternative approach suggested in von Oswald et al. [15] relies on outlier detection via predictive uncertainty. For instance, in a multi-head setting, one could choose the output head with the lowest predictive uncertainty for classification, as the input sample can be considered “in-distribution” for this head. Even though proper out-of-distribution detection is a challenging and in itself still unresolved problem of machine learning [59], it would be an interesting direction for future work to investigate this approach for RNNs.

Another alternative, utilized in [20, 50] on sequential data, is the use of a different autoencoder per task. The autoencoder with the lowest reconstruction error for a given input sample will determine the task identity. Such an approach also relies on the ability to successfully train generative models. In addition, the naive implementation requires one autoencoder per task. However, this last problem can be sidestepped using a hypernetwork-protected autoencoder (cf. method HNET+R in Sec. B.11).

F.2 Notes on optimization for the Copy Task

We observed better empirical results with vanilla RNNs than with LSTMs in the variants of the Copy Task. We also observed that throughout all CL methods, the Copy Task with vanilla RNNs can only be solved when using orthogonal regularization [61] for all hidden-to-hidden weight matrices, whereas orthogonal initialization did not seem to play an important role.

The requirement of using orthogonal regularization poses a particular problem in combination with hypernetworks. In contrast to all other methods, orthogonal regularization will regularize the output of a neural network and not the weight matrix itself. We consistently observed that the orthogonal regularization loss is harder to optimize and usually plateaus at higher values when used in combination with hypernetworks. We unsuccessfully experimented with several potential resolutions to overcome this problem, but did not use any of them for the results reported in this paper.

We first tried an annealing schedule for the orthogonal regularization strength, starting at very high values putting the emphasis of the optimizer on producing orthogonal hidden-to-hidden matrices via the hypernetwork. This can be also viewed as a pretraining phase, where the hypernetwork is pretrained to produce orthogonal matrices (to sidestep the limitation that we cannot initialize hidden-to-hidden weights orthogonally when using a hypernetwork).

In another attempt, we periodically measured the highest singular value of the hypernetwork-produced hidden-to-hidden matrix, and divided the outputted matrix by it (inspired by spectral normalization [62]). The purpose of this approach is to mitigate exploding activations/gradients and therefore to avoid the saturation of the tanh nonlinearity, which would lead to vanishing gradients.

However, we did not see consistent improvements using any of the aforementioned approaches and therefore neglected them for all our experiments.

F.3 Increased difficulty of the Permuted Copy Task

We empirically observed that the Permuted Copy Task (cf. 4.1) is harder to solve (for both vanilla RNNs and LSTMs, data not shown). Intuitively, such increase in difficulty can already be anticipated by analyzing a linear RNN (cf. Eq. 25). The basic Copy Task can be manually implemented as linear RNN by realizing a queue-like mechanism (i.e., the input-to-hidden weights write inputs into a subspace of the hidden space, while the hidden-to-hidden weights shift these chunks consecutively through subspaces until they reach an output subspace which is read out by the hidden-to-output weights). This specific implementation cannot be trivially extended to the time-permuted case (where the order in the queue needs to change before elements are shifted to the output subspace), which indicates why an increase in difficulty may occur.

Distillation targets have to be zero-padded as the softmax dimension is growing with each task.

Note that regularized autoencoders have been shown to elicit properties of the data-generating density function [60]. Hence, this method of task inference can be loosely linked to proper out-of-distribution detection.
We hypothesize that the increase in difficulty can also be linked to optimization, and more specifically to the large variation in backpropagation through time (BPTT) path lengths from each output timestep to its corresponding input timestep. Note that the mean BPTT path length is the same for the permuted and unpermuted case, but the standard deviation is zero for the unpermuted case. We observed that this variability in BPTT path lengths creates an optimization bias towards pairs of input/output timesteps that lie closer together in time (data not shown). Furthermore, previous work suggested in similar sets of experiments that the order of recall matters [e.g., 63], providing more evidence that there are indeed intrinsic differences between solving the basic and Permuted Copy Task.

F.4 Replay for Split-SMNIST experiments

To complement our investigations of the Split-SMNIST experiments in Sec. 4.2, we provide further experimental results on rehearsal methods in this section. As the training of generative models is challenging on real-world data, we restrict our exploration in this section to the original Split-SMNIST experiment, i.e., difficulty \( m = 1 \) (cf. Sec. 4.2).

The results are shown in Table 12. As can be seen, hypernetwork-protected replay HNET+R outperforms other rehearsal approaches and performs on par with Multitask training. However, when analysing results obtained from methods based on generative replay, namely RtF and HNET+R, we realized that even though reconstruction is feasible, rehearsal via samples obtained from the prior did not lead to visually meaningful digits. Aside from the difficulty of training a generative model, we hypothesize that this behavior is due to the coarse approximations made in Sec. B.11. Interestingly, we did not observe these difficulties for the Copy Task, where input samples are sequences without direct temporal dependencies (aside from the correct placement of the stop bit).

Table 12: Mean during and final accuracies for Split-SMNIST rehearsal experiments (Mean ± SEM in %, \( n = 10 \)). Method RtF was denoted Generative Replay in the main text. Both, RtF and HNET+R, are introduced in Sec. B.11. Methods denoted with a * use a decoder architecture that has an additional fully-connected layer of size 256 before and after the LSTM layer.

| Method         | during  | final   |
|----------------|---------|---------|
| Multitask      | N/A     | 99.18 ± 0.05 |
| Coresets-10    | 99.64 ± 0.02 | 96.44 ± 0.25 |
| Coresets-100   | 99.51 ± 0.01 | 98.85 ± 0.05 |
| RtF            | 98.95 ± 0.08 | 95.01 ± 0.88 |
| RtF*           | 99.51 ± 0.02 | 98.41 ± 0.22 |
| HNET+R         | 99.67 ± 0.01 | 99.34 ± 0.04 |
| HNET+R*        | 99.44 ± 0.03 | 99.10 ± 0.13 |

F.5 Processing sequential data with RNNs

Although recent results suggest that feedforward networks, which have parallelization and optimization benefits during training [64], can successfully process sequential data [65, 66, 67], RNNs still have theoretical benefits compared to their feedforward alternatives [66, 68], including an unlimited receptive field in time, and a linear time complexity in sequence length. We therefore consider research on RNNs as vital and hope that future works utilizes the insights and baselines provided in this study to develop CL algorithms tailored to RNNs.

Note that a sequence of pen-strokes (SMNIST sample) can easily be converted into an image.