SUPPLEMENTARY MATERIAL: CONTINUAL LEARNING IN RECURRENT NEURAL NETWORKS

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 $\mathcal{D}_k = \big\{(\mathbf{x}^{(n)}_{1:T^{(n)}_{\text{in}}}, \mathbf{y}^{(n)}_{1:T^{(n)}_{\text{out}}})\big\}_{n=1}^{N_k}$. A data sample $(\mathbf{x}_{1:T_{\text{in}}}, \mathbf{y}_{1:T_{\text{out}}})$ consists of a sequence of inputs $\mathbf{x}_{1:T_{\text{in}}} = (\mathbf{x}_1, \dots, \mathbf{x}_{T_{\text{in}}})$, $\mathbf{x}_t \in \mathcal{R}^{F_{\text{in}}}$, and a sequence of target outputs $\mathbf{y}_{1:T_{\text{out}}} = (\mathbf{y}_1, \dots, \mathbf{y}_{T_{\text{out}}})$, $\mathbf{y}_t \in \mathcal{R}^{F_{\text{out}}}$, 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 $\{\mathcal{D}_k\}_{k=1}^K$, is an RNN with parameters ψ. With an abuse of notation, we describe it by $\hat{\mathbf{y}}_{1:T_{\text{out}}} = f(\mathbf{x}_{1:T_{\text{in}}}, \psi)$. To express the step-by-step computation of the RNN we use $(\hat{\mathbf{y}}_t, \mathbf{h}_t) = f_{\text{step}}(\mathbf{x}_t, \mathbf{h}_{t-1}, \psi)$. Specifically, we denote by ψ_{hh} the hidden-to-hidden weights, which are a subset of ψ and are exclusively involved in the computation from $\mathbf{h}_{t-1}$ to $\mathbf{h}_t$. The **hypernetwork** is a feedforward neural network $\psi = h(\mathbf{e}_k, \theta)$ with parameters θ, that generates the parameters ψ of the main network given the task embedding $\mathbf{e}_k$ 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 $\mathcal{L}_{\text{task}}(\psi, \mathcal{D}_k)$ applied across all methods are described in Sec. B.5 (cf. Eq. 7 and Eq. 8).

B.1 FINE-TUNING

`Fine-tuning` (Li and Hoiem, 2017) refers to sequentially optimizing the task-loss $\mathcal{L}_{\text{task}}(\psi, \mathcal{D}_k)$ for $k = 1, \dots, K$ 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 $\mathcal{L}_{\text{task}}(\psi^{(k)}, \mathcal{D}_k)$.

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 (Li and Hoiem, 2017), refers to jointly training on all datasets at once: $\min_\psi \sum_{k=1}^K \mathcal{L}_{\text{task}}(\psi, \mathcal{D}_k)$. 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.

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 (von Oswald et al., 2020), 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. 2). The complete loss function for learning the K-th task is given by:[4]

$$\mathcal{L}(\theta, \mathbf{e}_1, \dots, \mathbf{e}_K, \mathcal{D}_K) = \mathcal{L}_{\text{task}}(\theta, \mathbf{e}_K, \mathcal{D}_K) + \frac{\beta}{K-1} \sum_{k=1}^{K-1} \| h(\mathbf{e}_k, \theta) - h(\tilde{\mathbf{e}}_k^{(K-1)}, \tilde{\theta}^{(K-1)}) \|_2^2 \quad (2)$$

where $\mathcal{D}_K$ is the dataset of task K, $\mathcal{L}_{\text{task}}(\cdot)$ is the loss function of the current task, β is the regularization strength and $\tilde{\theta}^{(K-1)}, \tilde{\mathbf{e}}_1^{(K-1)}, \dots, \tilde{\mathbf{e}}_{K-1}^{(K-1)}$ denote hypernetwork weights θ 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, thus preventing forgetting.

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. (2020), who showed that in the non-parametric limit a chunked hypernetwork can realize all possible continuous mappings between embedding and weight space. This method splits the vectorized main network weights ψ into equally sized chunks. Each chunk will be assigned a chunk embedding $\mathbf{c}_i$. The hypernetwork can then produce all weights ψ by processing a batch of chunk embeddings (utilizing parallelization on modern GPUs): $\psi = h(\mathbf{e}_k, \theta = \tilde{\theta} \cup \{\mathbf{c}_i\}) = \text{concat}([\dots, \tilde{h}(\mathbf{e}_k, \mathbf{c}_i, \tilde{\theta}), \dots])$. In our implementation chunk embeddings are considered to be part of θ and are therefore shared across tasks.

This approach to chunking is agnostic to the structure that ψ 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 $W_{\text{hh}} \in \mathbb{R}^{n_h \times n_h}$ and $W_{\text{ih}} \in \mathbb{R}^{n_h \times n_i}$ 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 $V_{\text{hh},i}, V_{\text{ih},i} = \tilde{h}(\mathbf{e}_k, \mathbf{c}_i, \tilde{\theta})$, with $V_{\text{hh},i} \in \mathbb{R}^{n_c \times n_h}$, $V_{\text{ih},i} \in \mathbb{R}^{n_c \times n_i}$ and $0 \equiv n_h \pmod{n_c}$. 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. (2020).

In addition, we would like to mention two properties of the hypernetwork approach that have been empirically verified (von Oswald et al., 2020). 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. (2020) seem to indicate that the learned embedding space possesses a structure that supports transfer. Second, von Oswald et al. (2020) noted and showed empirically that the regularizer in Eq. 2 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. 2), which is a sum of L2 norms of the hypernetwork output (of size $|\psi|$) over past tasks, yielding a time complexity of $\mathcal{O}(K|\psi|)$ if the regularizer is applied to all previous tasks, and $\mathcal{O}(C|\psi|)$ otherwise.

[4] We slightly modified the original regularizer by excluding the lookahead $\Delta\theta$ used in von Oswald et al. (2020) and by allowing fine-tuning of previous task embeddings, which requires us to additionally checkpoint these task embeddings before learning a new task.

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. 2) 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 $\left|\theta \cup \{\mathbf{e}_k\}_{k=1}^{K}\right| \approx |\psi|$, we simply denote this space complexity increase by $\mathcal{O}(|\psi|)$.

B.5 Elastic weight consolidation

Here, we quickly recapitulate the basic concepts behind elastic weight consolidation (EWC, Kirkpatrick et al. (2017a)). Since EWC is a prior-focused method (Farquhar and Gal, 2018), 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:

$$p(\psi \mid \mathcal{D}_1, \ldots, \mathcal{D}_K) \propto p(\psi \mid \mathcal{D}_1, \ldots, \mathcal{D}_{K-1})\, p(\mathcal{D}_K \mid \psi) \tag{3}$$

where $p(\psi \mid \mathcal{D}_1, \ldots, \mathcal{D}_{K-1})$ is the posterior from previous tasks and $p(\mathcal{D}_K \mid \psi)$ the likelihood of the current task. The precise derivation of the algorithm described here can be found in Huszár (2018), and has been termed `Online EWC` in Schwarz et al. (2018).

When learning task K, we aim to find a maximum a posteriori (MAP) solution of $p(\psi \mid \mathcal{D}_1, \ldots, \mathcal{D}_K)$ maximizing the following loss function:

$$\max_{\psi} \log p(\mathcal{D}_K \mid \psi) + \log p(\psi \mid \mathcal{D}_1, \ldots, \mathcal{D}_{K-1}) \tag{4}$$

We discuss the likelihood function for sequential data below. To obtain a tractable loss function, EWC utilizes an approximate posterior $q_\zeta^{(K-1)}(\psi) \approx p(\psi \mid \mathcal{D}_1, \ldots, \mathcal{D}_{K-1})$, whose parameters ζ are computed at the end of task $K-1$. Specifically, EWC first applies a Laplace approximation MacKay (1992) (using the MAP solution $\tilde{\psi}^{(K-1)}$ obtained at the end of training of task $K-1$) to obtain a Gaussian $q_\zeta^{(K-1)}(\psi)$ with mean $\tilde{\psi}^{(K-1)}$ and precision matrix $F = \sum_{k=1}^{K-1} F^{(k)}$, where $F^{(k)}$ denotes the empirical Fisher matrix.[5] As noted in Huszár (2018), this version of `Online EWC` still does not carry out the Laplace approximation correctly, as the precision matrix of $q_\zeta^{(K-1)}(\psi)$ 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 λ_{EWC}. As a second approximation, EWC considers all off-diagonal elements of F to be zero: $F_{i \neq j} = 0$. Taken together, while ignoring all terms independent of ψ, the loss described by Eq. 4 is approximated in `Online EWC` via (cf. Eq. 1):

$$\min_{\psi} -\log p(\mathcal{D}_K \mid \psi) + \lambda_{\text{EWC}} \sum_{i=1}^{|\psi|} F_{ii} (\psi_i - \tilde{\psi}_i^{(K-1)})^2 \tag{5}$$

where F_{ii} can be considered as weight-specific importance values and $\mathcal{L}_{\text{task}}(\psi, \mathcal{D}_K) \equiv -\log p(\mathcal{D}_K \mid \psi)$ 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: $\tilde{\psi}_i^{(1)} = \arg\max_\psi \log p(\mathcal{D}_1 \mid \psi) + \log p(\psi)$. However, we ignored the prior influence when obtaining $\tilde{\psi}_i^{(1)}$.

[5] Schwarz et al. (2018) introduced an additional hyperparameter $\gamma_F \leq 1$ to explicitly promote forgetting: $F = \sum_{k=1}^{K-1} \gamma_F^{K-1-k} F^{(k)}$. We left $\gamma_F = 1$ throughout this work.

Negative log-likelihood (NLL) for sequential data. Finally, we discuss how to implement $\mathcal{L}_{\text{task}}(\psi, \mathcal{D}_K) \equiv -\log p(\mathcal{D}_K \mid \psi)$ when applied to sequential data. Note that $p(\mathcal{D}_K \mid \psi) = \prod_{n=1}^{N_K} p(\mathbf{y}^{(n)}_{1:T^{(n)}_{\text{out}}} \mid \psi)$ and that $p(\mathbf{y}_{1:T_{\text{out}}} \mid \psi) = \prod_{t=1}^{T_{\text{out}}} p(\mathbf{y}_t \mid \mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \psi)$. Given the autoregressive structure of an RNN, we make the following assumption: $p(\mathbf{y}_t \mid \mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \psi) \approx p(\mathbf{y}_t \mid \mathbf{h}_{t-1}, \psi)$. Hence, we can decompose the NLL as follows:

$$-\log p(\mathcal{D}_K \mid \psi) = -\sum_{n=1}^{N_K} \sum_{t=1}^{T^{(n)}_{\text{out}}} \log p(\mathbf{y}^{(n)}_t \mid \mathbf{h}^{(n)}_{t-1}, \psi) \tag{6}$$

We first consider typical classification problems (cf. Sec. 5.2 and Sec. 5.3). In this case, $\mathbf{y}^{(n)}_t$ is a one-hot encoded representation of a label $y^{(n)}_t \in \{1, \ldots, F_{\text{out}}\}$, where F_{out} denotes the number of classes. We consider a sofmax output $\hat{\mathbf{y}}^{(\mathbf{n})}_{\mathbf{t}} = \text{softmax}(\tilde{\beta}_t \hat{\mathbf{z}}^{(\mathbf{n})}_{\mathbf{t}})$, where $\tilde{\beta}_t$ 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 $\tilde{\beta}_t = 0$ results in timestep t being ignored for the computation of the loss. Indeed, for the experiments in Sec. 5.2 and Sec. 5.3, the loss is evaluated solely based on the prediction of the last timestep $T^{(n)}_{\text{in}}$ of the given input sequence. Using this setting for classification problems leads to the well-known cross-entropy loss evaluated per timestep and summed over all timesteps:

$$-\sum_{n=1}^{N_K} \sum_{t=1}^{T^{(n)}_{\text{out}}} \log p(\mathbf{y}^{(n)}_t \mid \mathbf{h}^{(n)}_{t-1}, \psi) = -\sum_{n=1}^{N_K} \sum_{t=1}^{T^{(n)}_{\text{out}}} \sum_{c=1}^{F_{\text{out}}} [y^{(n)}_t = c] \log \left(\text{softmax}(\tilde{\beta}_t \hat{\mathbf{z}}^{(\mathbf{n})}_{\mathbf{t}})_c \right) \tag{7}$$

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

Lastly, we consider the NLL for the Copy Task and its variants (cf. Sec. 5.1), where the output has to match a binary target pattern. In this case, each pixel in the output pattern will be evaluated (independent of all other pixels) using a binary cross-entropy loss. Likelihood predictions of pixel values are obtained via a (tempered) sigmoid: $\hat{y}^{(n)}_{t,f} = \text{sigmoid}(\tilde{\beta}_{t,f} \hat{z}^{(n)}_{t,f})$, where $\hat{y}^{(n)}_{t,f}$ denotes the f-th entry of $\hat{\mathbf{y}}^{(n)}_t$, and $\tilde{\beta}_{t,f}$ can be interpreted as an inverse temperature that can be specified per timestep and feature. Taken together, the NLL loss for matching binary output patterns can be specified via:

$$-\sum_{n=1}^{N_K} \sum_{t=1}^{T^{(n)}_{\text{out}}} \log p(\mathbf{y}^{(n)}_t \mid \mathbf{h}^{(n)}_{t-1}, \psi) = \sum_{n=1}^{N_K} \sum_{t=1}^{T^{(n)}_{\text{out}}} \sum_{f=1}^{F_{\text{out}}} \left(-y^{(n)}_{t,f} \log \hat{y}^{(n)}_{t,f} - (1 - y^{(n)}_{t,f}) \log(1 - \hat{y}^{(n)}_{t,f}) \right) \tag{8}$$

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 $\arg\max_\psi p(\psi \mid \mathcal{D}_1, \ldots, \mathcal{D}_K)$ (which necessitates a certain compatibility across tasks), the hypernetwork-based approach allows task-specific solutions $\psi^{(k)} = \arg\max_\psi p(\psi \mid \mathcal{D}_k)$, 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 ψ. Assuming forward and backward computation only increases linearly with ψ, 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, Zenke et al. (2017)) 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 ψ_i on the loss change. Thus, at each optimization step s while training task k, an online importance estimate $\tilde{\omega}_i^{(k)}$ of ψ_i is updated via:

$$\tilde{\omega}_i^{(k)} \leftarrow \tilde{\omega}_i^{(k)} - \Delta\psi_i(s)\frac{\partial\mathcal{L}_{\text{task}}\big(\psi, \mathcal{B}(s)\big)}{\partial\psi_i} \tag{9}$$

where $\Delta\psi_i(s)$ is the weight change determined by the optimizer at step s, and $\mathcal{B}(s) \subseteq \mathcal{D}_k$ 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 $\mathcal{L}_{\text{task}}(\cdot)$ 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:

$$\Omega_i^{(k)} = \Omega_i^{(k-1)} + \frac{\tilde{\omega}_i^{(k)}}{\Delta\psi_i^{(k)} + \epsilon} \tag{10}$$

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

$$\min_{\psi} \mathcal{L}_{\text{task}}(\psi, \mathcal{D}_K) + \lambda_{\text{SI}} \sum_{i=1}^{|\psi|} \Omega_i^{(K-1)}(\psi_i - \tilde{\psi}_i^{(K-1)})^2 \tag{11}$$

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

The increase in space complexity arises due to the storage of $\Omega_i^{(K)}$, $\tilde{\psi}_i^{(K-1)}$, $\tilde{\omega}_i^{(K)}$, as well as a temporary copy of ψ from before the current optimizer step in order to compute $\Delta\psi_i(s)$: $\mathcal{O}(4|\psi|)$.

B.7 MASKING

Context-dependent gating (or `Masking`) is a mechanism to alleviate catastrophic interference that was introduced by Masse et al. (2018). 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 $\mathbf{h}_t$. For vanilla RNNs, which in our case are inspired by Elman networks, `Masking` affects the hidden state $\mathbf{h}_t$ as well as the RNN layer output.[6] 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.

[6]Note 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.

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 $\mathcal{O}(K|\psi|)$. 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 (Nguyen et al., 2018; Rebuffi et al., 2017). Rebuffi et al. (2017) discusses strategies on how to properly select coreset samples. Here, we simply take a random subset of N input samples from 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 $\tilde{\psi}_i^{(K-1)}$ before learning task K is generated and used to create *soft-targets* $\tilde{\mathbf{y}}_{1:T_{\text{out}}} = f(\mathbf{x}_{1:T_{\text{in}}}, \tilde{\psi}_i^{(K-1)})$, where $\mathbf{x}_{1:T_{\text{in}}}$ is a sample taken from a coreset (van de Ven and Tolias, 2019; Li and Hoiem, 2017). The soft-targets $\tilde{\mathbf{y}}_{1:T_{\text{out}}}$ are distilled Hinton et al. (2015) 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 $\mathcal{B}(s) \subseteq \mathcal{D}_K$, an additional mini-batch $\tilde{\mathcal{B}}(s)$ 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:

$$\min_{\psi} \mathcal{L}_{\text{task}}(\psi, \mathcal{B}(s)) + \lambda_{\text{distill}} \mathcal{L}_{\text{distill}}(\psi, \tilde{\mathcal{B}}(s)) \tag{12}$$

where λ_{distill} is a hyperparameter and $\mathcal{L}_{\text{distill}}(\cdot)$ denotes the distillation loss (Hinton et al., 2015).

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

B.9 GENERATIVE REPLAY

Conceptually, `Generative Replay` (Shin et al., 2017; van de Ven and Tolias, 2018) 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, Kingma and Welling (2014); Rezende et al. (2014)) 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: $p_\nu(\mathbf{x}) = \int_{\mathcal{Z}} p_\nu(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}$. Here, $\mathbf{z} \in \mathcal{Z}$ denotes a latent variable (or hidden cause), $p(\mathbf{z})$ is the prior and $p_\nu(\mathbf{x} \mid \mathbf{z})$ is a likelihood function defined via a decoder network whose parameters are denoted by ν. To learn the parameters ν given a dataset $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^N$, the corresponding hidden causes $\mathbf{z}_n$ have to inferred from the posterior $p_\nu(\mathbf{z} \mid \mathbf{x}) \propto p_\nu(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})$. However, the precise value of the posterior is in general intractable. Therefore, VAEs resort to variational inference (VI) to approximate the posterior using $q_\psi(\mathbf{z} \mid \mathbf{x}) \approx p_\nu(\mathbf{z} \mid \mathbf{x})$, where $q_\psi(\mathbf{z} \mid \mathbf{x})$

is realized through an encoder network with parameters ψ. VI utilizes the following inequality (cf. Kingma and Welling (2014) for a derivation):

$$\log p_\nu(\mathbf{x}) \geq -KL\big(q_\psi(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z})\big) + \mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{x})}\big[\log p_\nu(\mathbf{x} \mid \mathbf{z})\big] \tag{13}$$

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* $KL\big(q_\psi(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z})\big)$ and a *negative log-likelihood* (NLL) term $-\mathbb{E}_{q_\psi(\mathbf{z}|\mathbf{x})}\big[\log p_\nu(\mathbf{x} \mid \mathbf{z})\big]$.

Next, we discuss how to extend this framework to sequential data (also cf. Chung et al. (2015); Bayer and Osendorfer (2014)). We use an independence assumption when defining a prior for a sequence of hidden causes:

$$p(\mathbf{z}_{1:T}) = \prod_t p(\mathbf{z}_t) \tag{14}$$

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

$$p_\nu(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:T}) = \prod_t p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t}) \tag{15}$$

The decoder network is an RNN defined via $[\varphi_t, \mathbf{h}_t^{\text{dec}}] = f_{\text{dec,step}}(\mathbf{z}_t, \mathbf{h}_{t-1}^{\text{dec}}, \nu)$, where $\mathbf{h}_t^{\text{dec}}$ denotes the hidden state of the decoder network and $\varphi_t \in \Phi$ denotes the parameters of a parametric distribution (e.g., a Gaussian), which can be used to tractably compute densities $p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t})$ conditioned on $\mathbf{z}_{1:T}$.

As a last ingredient, we have to define the recognition model $q_\psi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})$. 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:

$$q_\psi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \overset{\text{chain rule of prob.}}{=} \prod_t q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{1:T}) \overset{\text{filtering assumption}}{\approx} \prod_t q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \tag{16}$$

Analogously to the likelihood, the components $q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t})$ of the approximate posterior are represented by an RNN encoder network $[\xi_t, \mathbf{h}_t^{\text{enc}}] = f_{\text{enc,step}}(\mathbf{x}_t, \mathbf{h}_{t-1}^{\text{enc}}, \psi)$, where $\xi_t \in \Xi$ are the parameters of a distribution over the latent space $\mathcal{Z}$.

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:

$$\begin{aligned}
&KL\big(q_\psi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \,||\, p(\mathbf{z}_{1:T})\big) \\
&= \int_{\mathbf{z}_{1:T}} \prod_{t'} q_\psi(\mathbf{z}_{t'} \mid \mathbf{z}_{<t'}, \mathbf{x}_{\leq t'}) \sum_t \log \frac{q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t})}{p(\mathbf{z}_t)} \, d\mathbf{z}_{1:T} \\
&= \sum_t \int_{\mathbf{z}_{1:T}} \prod_{t'} q_\psi(\mathbf{z}_{t'} \mid \mathbf{z}_{<t'}, \mathbf{x}_{\leq t'}) \log \frac{q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t})}{p(\mathbf{z}_t)} \, d\mathbf{z}_{1:T} \\
&= \sum_t \int_{\mathbf{z}_{1:t}} \prod_{t' \leq t} q_\psi(\mathbf{z}_{t'} \mid \mathbf{z}_{<t'}, \mathbf{x}_{\leq t'}) \log \frac{q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t})}{p(\mathbf{z}_t)} \, d\mathbf{z}_{1:t}
\end{aligned} \tag{17}$$

Note that the last manipulation is possible since the log-ratio does not depend on $\mathbf{z}_{t'}$ 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:

$$
\begin{aligned}
&KL\big(q_\psi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \,||\, p(\mathbf{z}_{1:T})\big) \\
&= \sum_t \int_{\mathbf{z}_{1:t-1}} \prod_{t'<t} q_\psi(\mathbf{z}_{t'} \mid \mathbf{z}_{<t'}, \mathbf{x}_{\leq t'}) \int_{\mathbf{z}_t} q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \log \frac{q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t})}{p(\mathbf{z}_t)} \, d\mathbf{z}_t \, d\mathbf{z}_{1:t-1} \\
&= \sum_t \int_{\mathbf{z}_{1:t-1}} \prod_{t'<t} q_\psi(\mathbf{z}_{t'} \mid \mathbf{z}_{<t'}, \mathbf{x}_{\leq t'}) KL\big(q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \,||\, p(\mathbf{z}_t)\big) \, d\mathbf{z}_{1:t-1} \\
&= \sum_t \int_{\mathbf{z}_1} q_\psi(\mathbf{z}_1 \mid \mathbf{x}_1) \int_{\mathbf{z}_2} \cdots \int_{\mathbf{z}_{t-1}} q_\psi(\mathbf{z}_{t-1} \mid \mathbf{z}_{<t-1}, \mathbf{x}_{\leq t-1}) KL(\dots) \, d\mathbf{z}_{t-1} \dots d\mathbf{z}_1
\end{aligned} \tag{18}
$$

Note that the KL divergence term $KL\big(q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \,||\, p(\mathbf{z}_t)\big)$ 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 $\mathbf{x}_{1:T}$, we use the recognition model $[\xi_t, \mathbf{h}_t^{\text{enc}}] = f_{\text{enc,step}}(\mathbf{x}_t, \mathbf{h}_{t-1}^{\text{enc}}, \psi)$ to compute a latent sequence $\mathbf{z}_{1:T}$ via $\mathbf{z}_t \sim q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \Leftrightarrow \mathbf{z}_t \sim p_{\xi_t}(\mathbf{z}_t)$, where $p_{\xi_t}(\cdot)$ is an explicit parametric distribution that we chose for the latent space (typically Gaussian), to evaluate the KL term. Note, ξ_t depends on $\mathbf{x}_t$ and $\mathbf{h}_{t-1}^{\text{enc}}$. However, in the implementation that we chose for this study, $\mathbf{h}_{t-1}^{\text{enc}}$ does not explicitly depend on $\mathbf{z}_{<t}$ (only implicitly through its distribution determined by ξ_t) even though $q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t})$ requires an explicit dependency.[7]

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

$$
KL\big(q_\psi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \,||\, p(\mathbf{z}_{1:T})\big) \approx \sum_t KL\big(p_{\xi_t}(\mathbf{z}_t) \,||\, p(\mathbf{z}_t)\big) \tag{19}
$$

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

$$
\begin{aligned}
\text{NLL} &= -\mathbb{E}_{q_\psi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})}\big[\log p_\nu(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:T})\big] \\
&= -\int_{\mathbf{z}_{1:T}} \prod_{t'} q_\psi(\mathbf{z}_{t'} \mid \mathbf{z}_{<t'}, \mathbf{x}_{\leq t'}) \sum_t \log p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t}) \, d\mathbf{z}_{1:T} \\
&= -\sum_t \int_{\mathbf{z}_{1:t}} \prod_{t' \leq t} q_\psi(\mathbf{z}_{t'} \mid \mathbf{z}_{<t'}, \mathbf{x}_{\leq t'}) \log p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t}) \, d\mathbf{z}_{1:t} \\
&= -\sum_t \int_{\mathbf{z}_1} q_\psi(\mathbf{z}_1 \mid \mathbf{x}_1) \cdots \int_{\mathbf{z}_t} q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \log p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t}) \, d\mathbf{z}_t \dots d\mathbf{z}_1 \\
&\overset{\text{MC sample size of 1}}{\approx} -\sum_t \log p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t})
\end{aligned} \tag{20}
$$

If $p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t})$ 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 φ_t of the decoder $[\varphi_t, \mathbf{h}_t^{\text{dec}}] = f_{\text{dec,step}}(\mathbf{z}_t, \mathbf{h}_{t-1}^{\text{dec}}, \nu)$ is the mean of a Gaussian distribution $\mathcal{N}(\mathbf{x}_t; \varphi_t, \tau^{-1} I)$, therefore $\mathcal{X} \equiv \Phi$. One could sample reconstructions from this distribution using the reparametrization trick (Kingma and Welling, 2014). However, at this level we do not introduce additional noise and instead aim to match encoder input $\mathbf{x}_t$ and decoder output φ_t directly:[8]

$$
\text{NLL} \approx \sum_{t=1}^{T_{\text{in}}} \frac{\tau}{2} \|\mathbf{x}_t - \varphi_t\|^2 \tag{21}
$$

[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 $[\xi_t, \mathbf{h}_t^{\text{enc}}] = f_{\text{enc,step}}(\mathbf{x}_t, \mathbf{z}_{t-1}, \mathbf{h}_{t-1}^{\text{enc}}, \psi)$ with $\mathbf{z}_{t-1} \sim p_{\xi_{t-1}}(\mathbf{z}_{t-1})$.

[8] Note, in contrast to the approximate posterior distribution $q_\psi(\mathbf{z}_t \mid \mathbf{z}_{<t}, \mathbf{x}_{\leq t})$ (which we crucially require to replay samples of prior tasks), we only require a sensible mean of the likelihood $p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t})$ to represent reconstructions.

In case of the Copy Task (and its variants), it makes sense to choose $p_\nu(\mathbf{x}_t \mid \mathbf{x}_{<t}, \mathbf{z}_{\leq t})$ to be a Bernoulli distribution (assuming the raw decoder output φ_t has been squeezed through a sigmoid):

$$\text{NLL} \approx \sum_{t=1}^{T_{\text{in}}} \sum_{f=1}^{F_{\text{in}}} -x_{t,f} \log \varphi_{t,f} - (1 - x_{t,f}) \log(1 - \varphi_{t,f}) \tag{22}$$

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` (van de Ven and Tolias, 2018) (referred to as `Generative Replay` in the main text) and `HNET+R` (von Oswald et al., 2020). In both cases, we use the main model simultaneously as classifier and VAE encoder $[\hat{\mathbf{y}}_t, \xi_t, \mathbf{h}_t] = f_{\text{step}}(\mathbf{x}_t, \mathbf{h}_{t-1}, \psi)$, where $\hat{\mathbf{y}}_t$ remains the model's prediction (cf. Sec. A) and ξ_t encodes a mean and diagonal covariance matrix of a Gaussian distribution used to sample latent representations of the VAE $\mathbf{z}_t \sim p_{\xi_t}(\mathbf{z}_t)$.

`RtF` (van de Ven and Tolias, 2018) refers 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 $\tilde{\nu}_i^{(K-1)}$ 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 $\mathcal{B}(s)$ with data from task K and a mini-batch $\tilde{\mathcal{B}}(s)$ with replayed data (using $\tilde{\nu}_i^{(K-1)}$) from tasks 1 to $K-1$ is assembled. In addition to the distillation loss (cf. Eq. 12; using hyperparameter λ_{distill}), that only affects the encoder f_{step}, the reconstruction loss $\mathcal{L}_{\text{rec}}(\cdot)$ (cf. Eq. 20) and prior-matching loss $\mathcal{L}_{\text{pm}}(\cdot)$ (cf. Eq. 19) are evaluated on $\mathcal{B}(s)$ and $\tilde{\mathcal{B}}(s)$:

$$\begin{aligned}\min_{\psi,\nu} \mathcal{L}_{\text{task}}(\psi, \mathcal{B}(s)) + \lambda_{\text{distill}} \mathcal{L}_{\text{distill}}(\psi, \tilde{\mathcal{B}}(s)) + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}}\big(\psi, \nu, \mathcal{B}(s) \cup \tilde{\mathcal{B}}(s)\big) \\ + \lambda_{\text{pm}} \mathcal{L}_{\text{pm}}\big(\psi, \nu, \mathcal{B}(s) \cup \tilde{\mathcal{B}}(s)\big)\end{aligned} \tag{23}$$

where $lambda_{\text{rec}}$ and λ_{pm} denote two new hyperparameters. Note, in order to train a multi-head main network f_{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 $\mathbf{z}_t$.

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 (von Oswald et al., 2020). The method `HNET+R` (von Oswald et al., 2020) circumvents this problem of `RtF` by training a task-specific decoder, where decoders of previous tasks are protected by a hypernetwork (cf. Eq. 2) 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) $\nu = h_{\text{dec}}(\mathbf{e}_k^{\text{dec}}, \theta^{\text{dec}})$. The loss in this case becomes (cf. Eq. 2 ad Eq. 23):

$$\begin{aligned}\min_{\psi,\theta^{\text{dec}}} \mathcal{L}_{\text{task}}(\psi, \mathcal{B}(s)) + \lambda_{\text{distill}} \mathcal{L}_{\text{distill}}(\psi, \tilde{\mathcal{B}}(s)) + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}}\big(\psi, \theta^{\text{dec}}, \mathcal{B}(s)\big) \\ + \lambda_{\text{pm}} \mathcal{L}_{\text{pm}}\big(\psi, \theta^{\text{dec}}, \mathcal{B}(s)\big) \\ + \frac{\beta_{\text{dec}}}{K-1} \sum_{k=1}^{K-1} \| h_{\text{dec}}(\mathbf{e}_k^{\text{dec}}, \theta^{\text{dec}}) - h_{\text{dec}}(\tilde{\mathbf{e}}_k^{(\text{dec},K-1)}, \tilde{\theta}^{(\text{dec},K-1)}) \|_2^2\end{aligned} \tag{24}$$

where β_{dec}, $\tilde{\mathbf{e}}_k^{(\text{dec},K-1)}$, $\tilde{\theta}^{(\text{dec},K-1)}$ are defined for the decoder hypernetwork h_{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_{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, $\mathcal{L}_{\text{rec}}(\cdot)$ and $\mathcal{L}_{\text{pm}}(\cdot)$, 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. 3 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 $\mathbf{h}_t$ of dimension n_h. The dynamics of the network are defined as follows:

$$\mathbf{h}_t = W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t \tag{25}$$

$$\hat{\mathbf{y}}_t = W_{hy}\mathbf{h}_t \tag{26}$$

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 $\mathbf{h}_t^{(k)} \in \mathbb{R}^{n_h}$ the content of $\mathbf{h}_t$ 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 $\mathbf{x}_t$ 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 $\mathbf{h}_t$ 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 $\mathbf{h}_t^{(k)}$ by W_{hh} overlaps with the processing of $\mathbf{h}_t^{(l\neq k)}$, 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 $\mathbf{h}_t$ 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 $\mathbf{h}_t^{(k)}$, the information within $\mathbf{h}_t$ relevant for solving task k, populates distinct and non-overlapping linear subspaces of $\mathbb{R}^{n_h}$ for each task across all $t > 0$:

$$\mathbf{h}_t^{(k)} \in \mathcal{S}_k \tag{27}$$

$$\mathcal{S}_k \cap \mathcal{S}_{l\neq k} = \{\mathbf{0}\} \tag{28}$$

If this wasn't the case and $\mathcal{S}_k$ overlapped with other subspaces, $\mathbf{h}_t^{(k)}$ could have components in $\mathcal{S}_{l\neq k}$, which would be influenced by the task-specific processing of other tasks.

Because W_{hh} is sequentially applied to $\mathbf{h}_t$, it must perform a subspace-retaining operation on $\mathbf{h}_t^{(k)} \in \mathcal{S}_k$ such that:

$$\mathbf{h}_{t+1}^{(k)} = W_{hh}\mathbf{h}_t^{(k)} \tag{29}$$

$$\mathbf{h}_{t+1}^{(k)} \in \mathcal{S}_k \tag{30}$$

The task-specific output head $W_{hy}^{(k)}$ can then select a linear subspace S_k of $\mathbf{h}_t$ that serves as output for task k. Hence, as long as the W_{hh} can represent a task-specific and subspace-retaining operation on $\mathbf{h}_t^{(k)}$, 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 $\mathbf{h}_t$ 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 $\mathcal{S}_k$ 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 $\mathcal{S}_k$ by the column space of a matrix U_k with orthonormal columns. Because $\mathbf{h}_t^{(k)}$ only has components in $\mathcal{S}_k$, it can be written as:

$$\mathbf{h}_t^{(k)} = U_k\mathbf{c}_t^{(k)} \tag{31}$$

with $\mathbf{c}_t^{(k)} \in \mathbb{R}^{p_k}$ the coordinates of $\mathbf{h}_t^{(k)}$ 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 $\bar{U} = \left[U_1 ... U_K \tilde{U}\right]^{n_h \times n_h}$ is an orthogonal basis for $\mathbb{R}^{n_h}$, with $\tilde{U}$ 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:

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

where Q_{ij} corresponds to the computation within W_{hh} that leads a subspace transformation from S_j to S_i. Then, $\mathbf{h}_{t+1}^{(k)}$ is given by

$$\mathbf{h}_{t+1}^{(k)} = W_{hh}\mathbf{h}_t^{(k)} \tag{33}$$

$$= W_{hh}U_k\mathbf{c}_t^{(k)} \tag{34}$$

$$= \bar{U}Q\bar{U}^T U_k\mathbf{c}_t^{(k)} \tag{35}$$

$$= \sum_{l=1}^{K} U_l Q_{lk}\mathbf{c}_t^{(k)} + \tilde{U}Q_{\sim k}\mathbf{c}_t^{(k)} \tag{36}$$

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

$$\mathbf{h}_{t+1}^{(k)} = U_k Q_{kk}\mathbf{c}_t^{(k)} \tag{37}$$

Therefore, one can easily design Q in such a way that W_{hh} performs a subspace-retaining transformation on $\mathbf{h}_t^{(k)}$, i.e. Q needs to have a block diagonal structure. Otherwise, $U_l Q_{lk}\mathbf{c}_t^{(k)}$ for $l \neq k$ is non-zero, and $\mathbf{h}_{t+1}^{(k)}$ will contain components in $\mathcal{S}_{l\neq k}$.

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 $\mathcal{S}_k$, 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. Even though we consider a simplified scenario where the recurrent processing cannot be shared across tasks, most commonly tasks will benefit from some form of shared processing. Crucially, whenever the subspaces associated with individual tasks may overlap, the overall dimensionality of the used hidden space can be less than the sum of dimensionalities of individual task-related subspaces. This frees up capacity in the recurrent weights to learn new tasks. Therefore, together with the working memory of individual tasks and the number of tasks, task similarity will also play a role in the effectiveness of weight-importance methods for RNNs. Interestingly, assuming it translates to nonlinear RNNs, one can use this intuition to design CL methods that avoid interference between tasks, but use shared computation whenever possible, as demonstrated by concurrent work (Duncker et al., 2020).

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 $W_{hh}^{(k)}$ can be generated for every new task, without forgetting $W_{hh}^{(l)}$ 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 Further Discussion of Related Work

The literature contains a number of studies that touch upon the problem of retaining and transferring past knowledge when dealing with sequential data. Some of these studies do not directly address the problem of catastrophic forgetting in the form of retaining performance on old tasks while learning new tasks, or they are not applicable to the experimental settings investigated in this study. The purpose of this section is to provide a brief overview over these studies and contrast them to the subject of this work.

Nowadays, real-world applications of RNNs are almost exclusively trained using backpropagation-through-time (BPTT) as underlying optimization algorithm. Even though most CL approaches are by design agnostic to this choice, they are typically tested and benchmarked against each other using BPTT. The study at hand is no exception in that regard. In contrast, recent work by Ororbia et al. (2020) has developed a biologically-inspired alternative to BPTT that enables zero-shot adaptation in a range of sequential generative modelling tasks. They perform `Fine-Tuning` experiments for a set of optimization techniques without utilizing explicit mechanisms supporting CL and show that the choice of optimization algorithm can lead to slight differences in terms of vulnerability to catastrophic forgetting. However, the applicability of this method outside the domain of token-level generative modelling remains unclear since, for example, it is not straightforward to adapt to classification problems.

Li et al. (2020) consider a specific sequential CL setting where it is assumed that the type of recurrent computation to be processed across tasks is identical. In their particular case, the recurrent component is frozen after the first task, i.e., recurrent computation is shared across tasks, and therefore catastrophic forgetting only needs to be prevented in the feedforward component of the architecture using an existing approach such as EWC Kirkpatrick et al. (2017a).

The work from Philps et al. (2019) focuses on financial time-series data. Their approach to CL is memory-based, where tuples of a representation of the training data and the model parameters are stored. Predictions can then be obtained via a weighted average of existing models, where a model is given more weight based on how similar its data representation is to the currently observed data. Such an approach is arguably expensive in terms of memory but also in terms of computation during inference, as final predictions are obtained by a weighted average of all models in the memory bank.

Learning without task-boundaries is another interesting challenge of CL and has been explored in the context of NLP tasks by Kruszewski et al. (2020). They provide datasets consisting of either four or five distinct tasks, which are randomly repeated such that the network observes 100 fragments, each consisting of samples from a single task. Their proposed method consists of an ensemble of models whose predictions are averaged using learned weighting factors. The updates of those weighting factors are faster than those of the ensemble members, implicitly allowing a mechanism for fast remembering (cf. He et al. (2019)) and therewith avoiding that ensemble members are completely overwritten upon task switches, which mitigates catastrophic interference. Note, the CL methods studied in our work can be adapted to an experimental setting where task-boundaries are not provided during training by either monitoring outliers in the loss or the model's predictive uncertainty (assuming tasks with non-overlapping input data distributions).

An approach for efficient knowledge transfer using ideas from CL for sentiment classification has been proposed by Lv et al. (2019). This approach contains two subnetworks, one that has an explicit mechanism against catastrophic forgetting (using a soft-masking approach) and a network that has full flexibility to adapt to the task at hand. The predictions of these two subnetworks are combined when forming a final decision.

Several studies have used methodologies from the CL literature for solving NLP problems (Wolf et al., 2018; Madasu and Rao, 2020; Thompson et al., 2019). For instance, Wolf et al. (2018) consider language modelling and employ a meta-model that can update the parameters of the language model such that it can focus on the local context. However, this approach is vulnerable to loosing the overall, pretrained language modelling skills, and therefore benefits from EWC to constrain the updates made by the meta-model.

E 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 four datasets. i and p refer to the input sequence and pattern lengths of the Copy Task, m refers to the number of digits in a SMNIST sequence. For the PoS dataset we report mean and standard deviation over tasks; the input feature size is given by the size of the word embeddings.

	Copy Task Variants	**SMNIST**	**AudioSet**	**PoS**
Classes	N/A	2	10	17
Training samples	100000	2 * 6000	10 * 750	9582 ± 4962
Validation samples	1000	2 * 500	10 * 50	1492 ± 1875
Test samples	1000	2 * 1000	10 * 200	1467 ± 2066
Input feature size	8	4	128	64
Number of timesteps	$i + 1 + p$	117*m	10	20 ± 6

E.1 Variations of the Copy Task

The Copy Task (Graves et al., 2014) 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.

E.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 $\mathbf{x}_{1:T}$ (with $T = i + 1 + p$) consists of a random binary pattern at timesteps $t = 1, \ldots, p$, where only feature dimensions 1 to $F_{\text{in}} - 1$ 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 $\mathbf{y}_{1:T}$ has no feature dimension reserved for the stop bit ($F_{\text{out}} = F_{\text{in}} - 1$). 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).

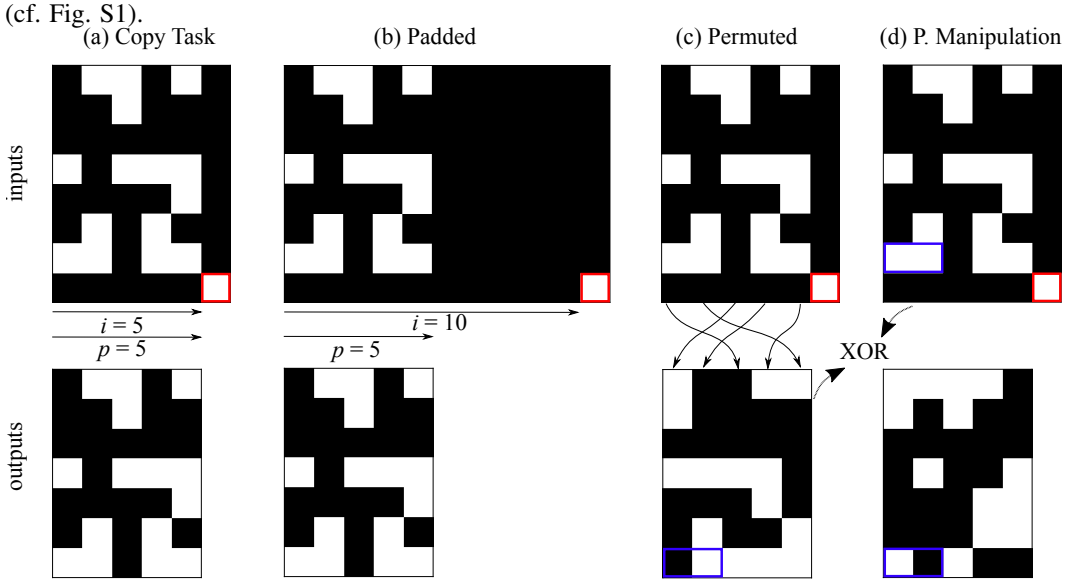

Figure S1: Variants of the Copy Task. The upper and lower rows show example input and output patterns, respectively. **(a)** In the basic Copy Task, the input sequence length i is equal to the actual pattern length p ($i = p = 5$ in this example), and the output is identical to the input except for the fact that the stop bit (signaled in red) is omitted. **(b)** In the Padded Copy Task, the pattern of length p to be copied is padded with zeros, yielding an input sequence length $i > p$. **(c)** In the Permuted Copy Task, the output corresponds to a time-permuted version of the input (indicated by the arrows). **(d)** In the Pattern Manipulation Task, the output is computed from the input pattern by applying a binary XOR operation iteratively with all of its r permutations, where r allows to control task difficulty. Here we illustrate the case where $r = 1$, and the XOR is computed between the input and its permuted version shown in (c). The blue rectangles highlight this operation for two specific bits.

E.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).

E.1.3 PERMUTED COPY TASK

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

E.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 $\mathbf{y}_{i+2:i+1+p}$ are generated from input patterns $\mathbf{x}_{1:p}$ by iterating the following procedure r times (where r determines the task difficulty). We start by assigning $\mathbf{y}_{i+2:i+1+p} \leftarrow \mathbf{x}_{1:p}$ and then iterate for $r' = 1 \dots r$

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

E.2 Sequential Stroke MNIST

Stroke MNIST (SMNIST, de Jong (2016)) represents MNIST images as a sequence of quadruples $\left\{dx_i, dy_i, eos_i, eod_i\right\}_{i=1}^{T}$. 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). The result of this procedure is available for download. [10] 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.

E.3 AudioSet

AudioSet (Gemmeke et al., 2017) 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.[11] Following Kemker et al. (2018), 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 ≥ 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.[12] 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.4 Multilingual Part-of-Speech Tagging

The Universal Dependencies dataset (Nivre et al., 2016) consists of grammar annotation treebanks from 92 different languages (version 2.6). For our multilingual Part-of-Speech (PoS) Tagging experiments we chose treebanks from 20 frequently used languages (cf. Plank et al., 2016). If multiple treebanks are available for a given language, we choose treebanks according to the available number of samples and whether they use the universal tagset (17 tags). We use the provided splits for training, test and validation samples. The data we use in our experiments is available for download[13],

[9] https://github.com/edwin-de-jong/mnist-digits-stroke-sequence-data/
[10] https://www.dropbox.com/s/sadzc8qvjvexdtx/ss_mnist_data?dl=1
[11] https://research.google.com/audioset/download.html
[12] https://www.dropbox.com/s/07dfeeuf5aq4w1h/audioset_data_balanced?dl=1
[13] https://www.dropbox.com/s/9xjrtprc2mfxcla/mud_data_2_6.pickle?dl=1

and the exact choice of treebanks as well as any other preprocessing steps can be reproduced with the code that accompanies this paper. Lastly, we use pretrained polyglot word embeddings. [14]

F 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.

F.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. E.1.1), where outputs are a copy of the inputs. We computed the hidden state activations $\mathbf{h}_{1:T}$ 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 obtained irrespective of the value of this threshold (we tested 20%, 30% ... 90%).

F.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. F.5. Table 5 shows all `during` and `final` accuracies of the SMNIST experiment described in Sec. 5.2.

Table 5: Task averaged `during` and `final` test accuracies for the SMNIST experiments (Mean $\pm$ SEM in %, $n = 10$).

	accuracy	$\mathbf{m = 1}$	$\mathbf{m = 2}$	$\mathbf{m = 3}$	$\mathbf{m = 4}$
`Online EWC`	during	98.52 ± 0.11	91.86 ± 1.13	83.62 ± 1.10	92.33 ± 2.49
	final	97.05 ± 0.59	76.65 ± 3.39	75.26 ± 1.85	73.16 ± 0.98
`HNET`	during	99.54 ± 0.02	97.87 ± 0.99	91.63 ± 1.54	95.49 ± 1.13
	final	99.52 ± 0.02	94.89 ± 3.81	91.63 ± 1.53	94.42 ± 1.85
`Fine-tuning`	during	99.67 ± 0.03	99.30 ± 0.04	99.14 ± 0.04	98.96 ± 0.03
	final	89.07 ± 1.22	75.23 ± 2.25	68.28 ± 0.72	69.04 ± 0.73
`Masking`	during	99.63 ± 0.02	99.25 ± 0.02	96.04 ± 1.35	87.78 ± 0.92
	final	99.23 ± 0.14	93.96 ± 1.29	86.20 ± 0.95	76.75 ± 1.32
`Masking + SI`	during	99.68 ± 0.02	99.02 ± 0.04	98.61 ± 0.22	96.42 ± 1.27
	final	99.68 ± 0.02	99.02 ± 0.04	98.62 ± 0.22	96.43 ± 1.26
`SI`	during	99.21 ± 0.04	89.78 ± 1.09	74.74 ± 0.14	69.58 ± 0.55
	final	97.08 ± 0.66	85.10 ± 1.57	74.72 ± 0.14	69.58 ± 0.55
`From scratch`	during	99.70 ± 0.02	95.86 ± 1.16	92.13 ± 1.25	88.48 ± 0.09
	final	99.70 ± 0.02	95.86 ± 1.16	92.13 ± 1.25	88.48 ± 0.09
`Coresets`	during	99.61 ± 0.01	99.05 ± 0.03	98.70 ± 0.05	85.46 ± 1.60
	final	99.40 ± 0.03	98.10 ± 0.07	97.60 ± 0.08	82.89 ± 1.80
`Multitask`	during	99.72 ± 0.02	99.19 ± 0.04	99.06 ± 0.04	98.72 ± 0.06
	final	99.72 ± 0.02	99.19 ± 0.04	99.06 ± 0.04	98.72 ± 0.06

F.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

[14] `https://sites.google.com/site/rmyeid/projects/polyglot`

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.

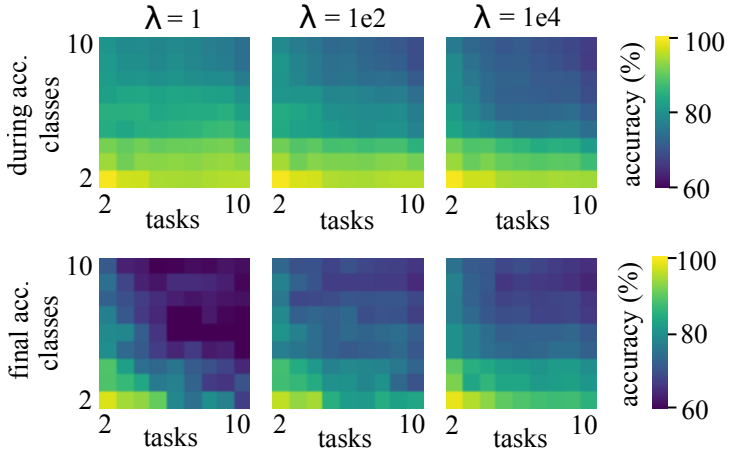

Figure S2: Task-averaged `during` and `final` test accuracies for `Online EWC` AudioSet experiments with varying numbers of classes per task, performed with different values for λ_{EWC} (cf. Fig. 5).

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 λ 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 λ values. For the lowest λ 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 λ 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 λ.

F.4 POS TAGGING

We use bidirectional LSTM main networks with 32 units and a fully connected output head per task. We decided to not fine-tune word embeddings. Further parameter choices and hyperparameter searches are detailed in Sec. F.5.

F.5 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.

F.5.1 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

F.5.2 PERMUTED COPY TASK

For Permuted Copy Task experiments with five tasks and $p = i = 5$, the hyperparameter searches are described in Table 7.

Table 7: 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_{EWC}	1e2, 1e3, ..., 1e10
`SI`	λ_{SI}	1e-3, 1e-2, 1e-1, 1, 1e2, 1e3
`Masking`	masked fraction	0.2, 0.4, 0.6, 0.8
`Masking + SI`	λ_{SI}	1e-3, 1e-2, 1e-1, 1, 1e2, 1e3
	masked fraction	0.2, 0.4, 0.6, 0.8
`Generative Replay`	strength of the prior-matching term (λ_{pm})	1, 10
	strength of the reconstruction term (λ_{rec})	1, 10
	strength of the soft-target distillation loss ($\lambda_{distill}$)	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

F.5.3 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 8.

F.5.4 PATTERN MANIPULATION TASK

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

Table 8: 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`	λ_{EWC}	1e2, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9, 1e10
`HNET`	β	5, 10, 50
	`HNET` hidden layers	"60,60,30"
	`HNET` output size	4000
	task embedding size	16, 32
	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_{EWC}	1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3
	orthogonal regularization strength	1, 10
`HNET`	β	1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3
	`HNET` hidden layers	"64,64,64", "64,64,32"
	`HNET` output size	2000, 4000
	chunk embedding size	32
	task embedding size	32

F.5.5 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.

F.5.6 AUDIOSET

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

F.5.7 POS TAGGING

The hyperparameter searches for the PoS-Tagging experiments are described in Table 12.

Table 10: Hyperparameter search for Sequential Stroke MNIST

Method	Hyperparameter	Searched values
All	batch size	64, 128
	number of hidden units of the main network	256
	main network activation function	*tanh*
	learning rate	1e-3, 5e-3, 1e-4
	clip gradient norm	None, 1, 100
	number of iterations for $m = 1$	2000, 3000, 4000
	number of iterations for $m = 2$	6000, 8000, 10000
	number of iterations for $m = 3$	8000, 12000, 16000
	number of iterations for $m = 4$	15000, 20000, 25000
`Online EWC`	λ_{EWC}	1e1, 1e2, ..., 1e10
`SI`	λ_{SI}	1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3
`Masking`	masked fraction	0.2, 0.4, 0.6, 0.8
`Masking + SI`	λ_{SI}	1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3
	masked fraction	0.2, 0.4, 0.6, 0.8
`HNET`	β	1e-1, 1e0, 1e1
	`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
`Coreset`	$\lambda_{distill}$	1e-1, 1e0, 1e1
`Multitask`	batch size	64, 128, 256, 512

G SUPPLEMENTARY EXPERIMENTS AND FURTHER REMARKS

G.1 PROCESSING SEQUENTIAL DATA WITH RNNS

Although recent results suggest that feedforward networks, which have parallelization and optimization benefits during training (Oord et al., 2016a), can successfully process sequential data (Devlin et al., 2019; Oord et al., 2016b; Radford et al., 2019), RNNs still have theoretical benefits compared to their feedforward alternatives (Oord et al., 2016b; Vaswani et al., 2017), 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.

G.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 (Vorontsov et al., 2017) 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

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`	λ_{EWC}	1e-1, 1e0, ..., 1e10
`SI`	λ_{SI}	1e-4, 1e-3, 1e-2, 1e-1, 1, 1e2, 1e3, 1e4
`Masking`	masked fraction	0.2, 0.4, 0.6, 0.8
`Masking + SI`	λ_{SI}	1e-4, 1e-3, 1e-2, 1e-1, 1, 1e2, 1e3, 1e4
	masked fraction	0.2, 0.4, 0.6, 0.8
`HNET`	β	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

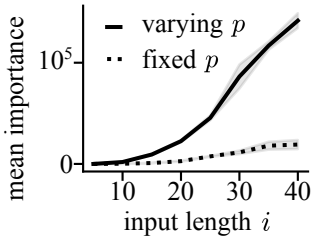

Figure S3: Mean importance values (as computed in `SI`) of hidden-to-hidden weights after learning the Copy Task (solid line, $p = 5, 10, ...40$) or the Padded Copy Task (dotted line, $p = 5$) independently for an increasing set of sequence lengths i (Mean $\pm$ SD, $n = 5$). Cf. Fig. 3.

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, Miyato et al. (2018)). 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.

Table 12: Hyperparameter search for PoS Tagging

Method	**Hyperparameter**	**Searched values**
All	number of hidden units of the main network	32
	main network activation function	*tanh*
	batch size	64
	number of iterations	2500, 5000
	learning rate	5e-3, 1e-3, 1e-4
	clip gradient norm	None, 100
	orthogonal regularization strength	0, 1
`Online EWC`	λ_{EWC}	1e1, 1e2, ..., 1e10
`SI`	λ_{SI}	1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3
`HNET`	β	5e-2, 5e-1, 5e0, 5e1
	`HNET` hidden layers	"10,10", "25,25,25", "75,125"
	`HNET` output size	190, 1600, 4000
	`HNET` activation function	*sigmoid*, *relu*
	chunk embedding size	8, 32
	task embedding size	8, 32
`Coreset`	$\lambda_{distill}$	1e-1, 1e0, 1e1
`Masking`	masked fraction	0.2, 0.4, 0.6, 0.8
`Masking + SI`	λ_{SI}	1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3
	masked fraction	0.2, 0.4, 0.6, 0.8

G.3 Analysis of Synaptic Intelligence

We repeated the analysis of weight-importance values for the Copy Task (cf. 4), but this time computing the weight-importance values as prescribed by `SI`. We obtained qualitatively similar results to `Online EWC` (Fig. S3). Specifically, we observe that weight importance values noticeably increase when the length of the sequence to be recalled increases (varying p), whereas the increase is comparatively negligible when the pattern length remains constant (varying i with constant p). Thus, we were able to empirically validate the hypotheses derived from our linear analysis (cf. SM C) in nonlinear RNNs for two different weight-importance methods, indicating that the described mechanisms might in general influence the performance of weight-importance methods in RNNs.

G.4 Analysis of Weight-Importance Methods in LSTMs

To test whether our results from the Copy Task (cf. Sec. 4) hold for different types of RNNs, we extend our analysis by using LSTMs instead of vanilla RNNs. Specifically, we use networks with an LSTM layer of 256 units, and a fully connected layer of 128 units before the actual output layer. Because we found it difficult to train LSTMs (cf. Sec. G.2) to recall long sequences in the Copy Task, we limit our analysis to inputs of length $i = 5, 10, ..25$. All other details are identical to those explained in Sec. 4.

The results of this analysis are shown in Fig. S4. We observe that the intrinsic dimensionality of the LSTM hidden space increases with the length p of the pattern to be copied (Fig. S4a) but not necessarily with the input length i (Fig. S4). Furthermore, we observe that an increase in the intrinsic dimensionality of the hidden space relates to an increase in mean importance values as calculated with `Online EWC`. Therefore, we observe with LSTMs the same trends we observed for vanilla RNNs. This validates the observation that working memory requirements, and not necessarily sequence length, lead to an increase in importance values as computed by weight-importance methods when sequentially learning a set of tasks with a recurrent network.

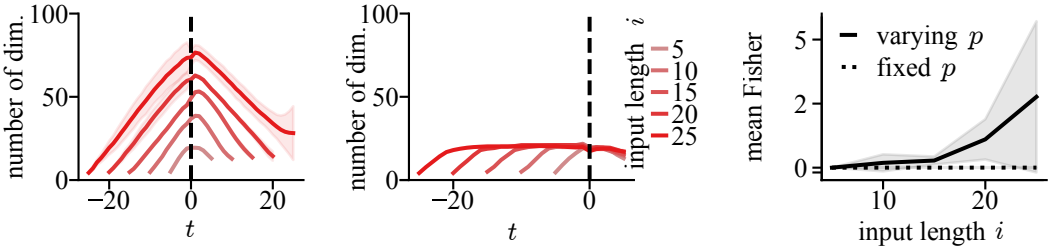

Figure S4: Same analysis as in Fig. 3 but for LSTM instead of vanilla RNNs. **(a)** Intrinsic dimensionality per timestep of the 256-dimensional LSTM hidden space $\mathbf{h}_t$ for the basic Copy Task, where input and pattern lengths are tied ($i = p$). The stop bit (dotted black line) is shown at time $t = 0$ (Mean $\pm$ SD, $n = 5$). **(b)** Same as (a) for the Padded Copy Task, where the pattern length is fixed ($p = 5$) but input length i varies. In (a) and (b), dimensionality of the hidden state space increases only during input pattern presentation. **(c)** Mean Fisher values (weight-importance values in `Online EWC`) of recurrent weights after learning the Copy Task (solid line, $p = 5, 10, ...25$) or the Padded Copy Task (dotted line, $p = 5$) independently for an increasing set of sequence lengths i (Mean $\pm$ SD, $n = 5$).

G.5 Determining task-relevant intrinsic dimensionality

The experiments in Sec. 4 and SM G.3 to analyse the intrinsic task-specific dimensionality of the hidden-state for varying task complexities relied on unsupervised dimensionality reduction methods. As such, these methods cannot differentiate between task-unrelated (e.g., random) information encoded in the hidden state and information extracted by the output heads from those hidden states in order to perform inference. To overcome this limitation we perform a simple type of supervised dimensionality reduction, which allows us determine the subspace of the hidden state that contains the necessary information to carry out predictions. In particular, we are interested to perform this type of analysis in a CL setting, i.e., studying the evolution of the intrinsic hidden-state dimensionality when successively learning tasks with comparable complexity.

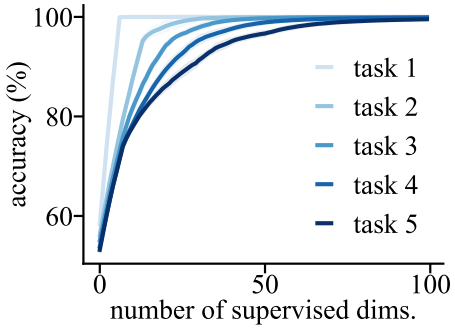

Figure S5: To complement our analysis of intrinsic hidden-state dimensionality for tasks with varying complexity from Sec. 4 and SM G.3, we provide here the evolution of the estimated hidden-state dimensionality when continually learning five subtasks of the *Permuted Copy Task* using `Online EWC` (cf. 5.1). The plot shows, for any given dimensionality, what predictive performance can be achieved when choosing the respective hidden-state subspace that maximizes this performance (cf. SM G.5 for details). The task value indicates the number of tasks that have been learned up to that point (Mean $\pm$ SD in %, $n = 5$).

In the following, we briefly sketch how we perform supervised dimensionality reduction. We require the checkpointed RNN model parameters $\tilde{\psi}^{(k)}$ obtained after training on task k. We can use these models to collect the hidden states for samples of all tasks trained so far (i.e., from $\mathcal{D}_1, \ldots, \mathcal{D}_k$). We gather those hidden states into the rows of a matrix $H^{(k)}$ and perform a reconstruction from an $n_{\text{red}} \leq n_h$ dimensional subspace using an approximately orthonormal matrix $U^{(k)} \in \mathbb{R}^{n_h \times n_h}$ via

$\tilde{H}^{(k)} = H^{(k)} U^{(k)}_{:n_{\text{red}}} (U^{(k)}_{:n_{\text{red}}})^T$, where $U^{(k)}_{:n_{\text{red}}} \in \mathbb{R}^{n_h \times n_{\text{red}}}$ denotes the matrix obtained by taking the first n_{red} columns from $U^{(k)}$, and n_h is the number of hidden neurons. We can then continue obtaining predictions from the RNN with parameters $\tilde{\psi}^{(k)}$ using the reconstructions $\tilde{H}^{(k)}$ and compare these to the ground-truth targets in order to judge the quality of the subspace.

We now describe how we obtain $U^{(k)}$ for each task k. As this process is identical for every k, we omit the superscript k from our notation. The matrix U is retrieved by an iterative procedure (column-by-column), where every added column is optimized to extract a new subspace dimension that maximizes predictive performance. Let $\hat{\mathbf{y}}_{1:T} = f_{\text{out}}(\mathbf{h}_{1:T}, \tilde{\psi})$ denote the part of the RNN f that generates predictions from a given hidden state sequence $\mathbf{h}_{1:T}$ (note, f_{out} only performs time-independent feed-forward computation, thus processing hidden states independently). In general, different timesteps may use different subspaces, so the matrix U should be specific to a predefined timestep s, such that reconstructed hidden states can be obtained via $\tilde{\mathbf{h}}_s^T = \mathbf{h}_s^T U_{:n_{\text{red}}} U_{:n_{\text{red}}}^T$, $\hat{\mathbf{y}}_s = f_{\text{out}}(\tilde{\mathbf{h}}_s, \tilde{\psi})$ and $(\hat{\mathbf{y}}_t, \tilde{\mathbf{h}}_t) = f_{\text{step}}(\mathbf{x}_t, \tilde{\mathbf{h}}_{t-1}, \tilde{\psi})$ for $t = s+1, \dots, T$. The network predictions $\hat{\mathbf{y}}_{s:T}$ from the *reconstructed* hidden states $\tilde{\mathbf{h}}_{s:T}$ can then be compared to the ground-truth $\mathbf{y}_{s:T}$ via some loss $\mathcal{L}_{\text{task}}(\hat{\mathbf{y}}_{s:T}, \mathbf{y}_{s:T})$ and U can be optimized to minimize this loss. More precisely, we minimize the following loss function iteratively for $n = 1, \dots, n_h$:

$$\min_{\mathbf{u}_n} \|U_{:n}^T U_{:n} - I_{:n}\|^2 + \gamma \mathcal{L}_{\text{task}}(\tilde{\mathcal{Y}}, \mathcal{Y}) \tag{38}$$

where γ is a hyperparameter, $\mathbf{u}_n$ denotes the n-th column of U and $\tilde{\mathcal{Y}}$ consists of sequences $\hat{\mathbf{y}}_{s:T}$ computed from reconstructed hidden states using U with input samples taken from the training sets of all previously learned tasks, while $\mathcal{Y}$ contains the corresponding ground-truth sequences $\mathbf{y}_{s:T}$.

As this process is computationally quite demanding and it is not a priori clear which timesteps to focus on, we decided to take a simpler approach and learn a shared matrix U for all timesteps, such that for every t we compute a reconstruction using $\tilde{\mathbf{h}}_t^T = \mathbf{h}_t^T U_{:n_{\text{red}}} U_{:n_{\text{red}}}^T$ as well as the corresponding predictions $\hat{\mathbf{y}}_{1:T} = f_{\text{out}}(\tilde{\mathbf{h}}_{1:T}, \tilde{\psi})$. With this procedure, we might overestimate the actual intrinsic dimensionality required for solving all tasks seen so far, but we can still assess relative differences, e.g., an increase in intrinsic dimensionality if more tasks are learned.

The results of this analysis on an instance of the *Permuted Copy Task* are depicted in Fig. S5. We observe that the number of dimensions required to achieve high accuracy levels increases as more tasks are being learned. In other words, the task-relevant dimensionality of the hidden space increases with the number of tasks that has been learned so far, which aligns with the predictions from SM C.

It is worth noting that if f_{out} is a linear operation, one could directly obtain the dimensionality of task-specific subspaces read by each output head by applying singular value decomposition to the hidden-to-output weight matrix, and thresholding the singular values. This procedure allows one to obtain a task-specific basis $U^{(k)}$, which can be compared to other tasks k' by computing the subspace similarity $\text{sim}(k, k') = \|U^{(k)}(U^{(k')})^T\|_F$ and can therefore serve as a proxy for task similarity.

G.6 Increased difficulty of the Permuted Copy Task

We empirically observed that the Permuted Copy Task (cf. 5.1) 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.

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., Zaremba and Sutskever, 2014), providing more evidence that there are indeed intrinsic differences between solving the basic and Permuted Copy Task.

G.7 Replay for Split-SMNIST experiments

To complement our investigations of the Split-SMNIST experiments in Sec. 5.2, 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. 5.2).

Table 13: Mean `during` and `final` accuracies for Split-SMNIST rehearsal experiments (Mean $\pm$ 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.

	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

The results are shown in Table 13. 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.[15] 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).

G.8 Sequence length and working memory requirements in real world tasks

Analyzing how sequence length and working memory requirements influence CL in RNNs requires a setting in which these two factors can be clearly disentangled. The synthetic Copy Task provides the flexibility to independently manipulate the two aspects and thus isolate their effects. However, it is unclear whether the observations made on this simple task generalize to more complex, real-world scenarios. Here we show further experiments that support our Copy Task results in a more practically relevant scenario related to the concept of temporal resolution.

Sequential data often contains redundant information, e.g. when the sampling process is much faster than the data generation process. Thus, we can manipulate the length of a sequence without changing the information it contains by temporal upsampling. In the same way, we can change the sequence length of a task's input, without noticeably affecting the working memory required to solve it.

In our sequential SMNIST experiments (cf. Sec 5.2), input sequence length and working memory requirements both increase with increasing digit sequence length m. Here, we contrast this setting with Split-SMNIST experiments with fixed working memory requirements ($m = 1$), but increasing input sequence length (as dictated by an upsampling factor $u \in \{2, 3, 4\}$).

We use `Online EWC` to train networks on five tasks for three different upsampling factors and display the results in Table 14. Increasing the upsampling factor u does not yield a drop in performance. This is contrasted by the sequential SMNIST results, where an increase in the number of digits m leads to substantially worse results. This difference indicates that the drop in performance observed for increasing m is caused by higher task difficulty, and is independent of the length of input sequences. Taken together, these results support our conclusions from the Copy Task experiments and show that they hold in less artificial settings.

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

Table 14: Mean `final` accuracies for Split-SMNIST experiments under different task difficulty scenarios. l is the factor by which input sequence length is increased w.r.t the original sequence composed of one digit. This factor can be increased by either increasing the number of digits m (top row, copied from sequential SMNIST experiments), or by increasing the upsampling factor u (bottom row). (Mean $\pm$ SEM in %, $n = 10$)

Setting	$\mathbf{l = 1}$	$\mathbf{l = 2}$	$\mathbf{l = 3}$	$\mathbf{l = 4}$
Varying m	97.05 ± 0.59	76.65 ± 3.39	75.26 ± 1.85	73.16 ± 0.98
Varying u	97.05 ± 0.59	95.35 ± 1.06	94.65 ± 0.85	96.27 ± 0.66

G.9 Part-of-Speech Tagging

Part-of-Speech Tagging is a classical NLP task in which words in a sentence are tagged according to their grammatical properties (e.g. nouns, verbs, etc.). The Universal Dependencies dataset (Nivre et al., 2016) contains large annotated text corpora from a wide range of languages and has previously been used in a multitask setting (Plank et al., 2016; Heinzerling and Strube, 2019). We cast this dataset to a CL setting, by sequentially training RNNs one language at a time. The results of training bidirectional LSTMs (BiLSTM) on 20 languages are displayed in Table 15.

We can observe a similar trend to other experiments, where `HNET` outperforms other regularization methods. The performance of `Masking+SI` indicates that no good trade-off has been found between small, distinct but plastic subnetworks and large, overlapping and thus rigid subnetworks. Again the simple method `Coresets` exhibits the strongest performance among CL approaches, making it the preferable choice when data storage is feasible.

G.10 Task Similarity and Forward Transfer

Our analysis on linear RNNs learning multiple tasks relies on the assumption that the tasks are so different that they cannot share any recurrent computation. This is, however, an extreme scenario and in practice some level of similarity will exist between the tasks. CL approaches can in theory exploit this similarity by using some form of shared processing across tasks, and therefore facilitating knowledge transfer between tasks.

In this section, we provide additional analyses and insights regarding task similarity and transfer of knowledge. For this, we consider a Permuted Copy Task experiment with ten tasks, where we present the same set of five tasks to the network twice. A thorough comparison of how each method might benefit from knowledge transfer is beyond the scope of our work. However, we provide here some pointers on how weight-importance methods and the hypernetwork approach might reuse previously acquired knowledge, noting that the two approaches transfer knowledge in a fundamentally different way.

Weight-importance methods. Since weight-importance methods use the same set of weights for all tasks, computation relevant for multiple tasks can directly be reused. In the experiment mentioned above, `Online EWC` achieves a mean final accuracy of 99.01%. To compare the subspaces read out

Table 15: Mean `during` and `final` accuracies for the PoS experiments (Mean $\pm$ SEM in %, $n = 10$).

	during	**final**
`Multitask`	N/A	92.52 ± 0.02
`From-scratch`	N/A	95.04 ± 0.01
`Fine-tuning`	91.63 ± 0.01	48.62 ± 0.58
`HNET`	89.83 ± 0.09	89.30 ± 0.09
`Online EWC`	87.49 ± 0.04	86.89 ± 0.03
`SI`	85.66 ± 0.06	84.62 ± 0.08
`Masking`	91.29 ± 0.02	46.76 ± 0.81
`Masking+SI`	82.60 ± 0.09	82.34 ± 0.09
`Coresets-100`	91.64 ± 0.02	90.05 ± 0.03
`Coresets-500`	91.60 ± 0.03	90.63 ± 0.03

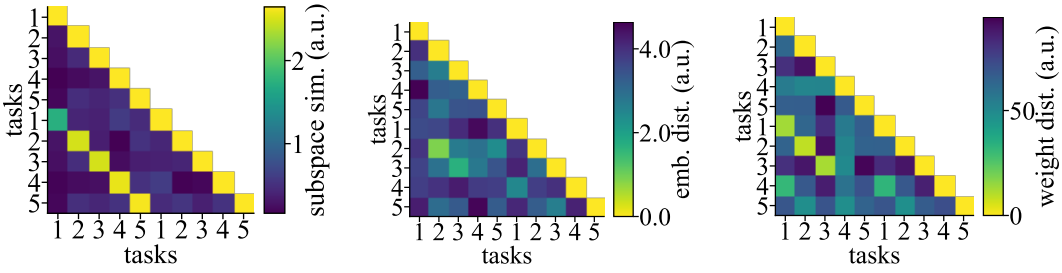

Figure S6: Mechanisms for knowledge transfer in a Permuted Copy Task setting with similarity across tasks (tasks 1 to 5 are identical to tasks 6 to 10). **(a)** Subspace similarity between tasks for `Online EWC`. **(b)** Euclidean distance between task-embeddings for `HNET`. **(c)** Euclidean distance in task-specific `HNET` outputs (i.e. main network weights).

by the task-specific linear output heads, we compute subspace similarities between pairs of tasks by performing SVD on the head-specific weight matrices, as described at the end of Sec. G.5. Fig. S6a shows that the subspaces used to solve the first set of tasks are being reused when the network solves the tasks for the second time, demonstrating that subspace reuse is a possible mechanism for transfer in weight-importance methods.

Hypernetworks. von Oswald et al. (2020) showed that hypernetworks can benefit from forward transfer between tasks. Since in this approach a new set of weights is generated for every task, transfer has to occur in the mapping from task embeddings to main network weights. Indeed, von Oswald et al. (2020) also showed that this can be mediated by a suitably structured embedding space. For the Permuted Copy Task version described above, where the hypernetwork approach achieves a mean final accuracy of 98.74%, transfer could occur by simply reusing the embeddings of the first set of tasks. However, the pairwise distances between task embedding vectors, as well as between the generated main network weights do not strongly support this (i.e. only the embeddings and weights of the tasks 2 and 3 being learned for the second time show some similarity to the original solutions). This could indicate that, in this setting, `HNET` finds a different solution when solving a task for the second time. Alternatively, this can be explained by the fact that Euclidean distances are not an appropriate measure to capture transfer in this scenario, since proximity in the embedding and weight spaces does not necessarily reflect similarity in the functional space.

One way to incentivize the reuse of previously found solutions, could be to selectively increase the learning rate of the embedding optimization when starting to train on a new task. This could allow for fast exploration of the space of existing solutions and opens interesting avenues for future work.

G.11 Discussion on task-specific solutions in CL

In this section, we further discuss differences between CL regularization approaches, specifically between weight-importance methods and `HNET`.

We start by reiterating our main motivation for proposing the use of `HNET` for RNNs. Because RNNs, just like feedforward networks, have a static set of weights, the regularization employed by `HNET` is not affected by the recurrent processing occurring in the main network (cf. Eq. 2). Therefore, the CL regularization is independent of the choice between an RNN or feedforward network as main network. Note however, that individual tasks are initially learned through the recurrent main network while the data is available (thus before CL regularization is required); therefore the sequential nature of individual tasks does influence hypernetwork optimization (Sec. G.2).

Another benefit of the method `HNET` becomes apparent when considering the non-parametric limit. If the hypernetwork is considered a universal function approximator, the generation of a new set of weights per task can in theory allow CL without any forgetting. In contrast, weight-importance methods cannot provide the same guarantees even if the main network is considered a universal function approximator. For instance, it is not guaranteed that the posterior mode found via `Online EWC` for the first task contains viable solutions for all upcoming tasks.

In this work, we show that these intuitions transfer to practical applications and that `HNET` often exhibits strong performance advantages over weight-importance methods. To guide future research in this area, we devote the rest of the section to the main conceptual difference between these methods, namely that `HNET` allows task-specific solutions whereas weight-importance methods aim for a trade-off solution (cf. Fig. 1b). We draw intuition from Bayesian inference and therefore confine ourselves to a subclass of weight-importance methods that can be interpreted as prior-focused methods (cf. Farquhar and Gal (2018), e.g., `Online EWC`). In addition, we ignore knowledge transfer through the shared meta-model and consider `HNET` as an approximation to the `From-Scratch` baseline. In this simplified setting, `From-Scratch`/`HNET` have the ability to gather a sample from a task-specific posterior (e.g., the maximum-a-posteriori (MAP)), i.e., for tasks $k = 1, \dots, K$:

$$\psi_{\text{MAP}}^{(k)} = \arg\max_{\psi} p(\psi \mid \mathcal{D}_k) = \arg\min_{\psi} -\big(\log p(\mathcal{D}_k \mid \psi) + \log p(\psi)\big) \tag{39}$$

Note, that the right-hand side of the equation above often matches the optimization criterion applied to the current task, where $-\log p(\mathcal{D}_k \mid \psi)$ matches the negative log-likelihood (cf. Eq. 6) and the prior influence can be realized via weight-decay if $p(\psi)$ is assumed to be an isotropic Gaussian distribution.

In contrast, a prior-focused method aims to find a single shared solution:

$$\psi_{\text{MAP}}^{(1:K)} = \arg\max_{\psi} p(\psi \mid \mathcal{D}_1, \dots, \mathcal{D}_K) = \arg\min_{\psi} -\big(\log p(\mathcal{D}_K \mid \psi) + \log p(\psi \mid \mathcal{D}_1, \dots, \mathcal{D}_{K-1})\big) \tag{40}$$

For completeness, if the main network weights ψ are split into a shared body ψ_{shared} and task-specific output-head weights $\psi_{\text{specific}}^{(1)}, \dots, \psi_{\text{specific}}^{(K)}$, the equation above becomes

$$\begin{aligned}
\psi_{\text{MAP}}^{(1:K)} &= \arg\max_{\psi} p(\psi \mid \mathcal{D}_1, \dots, \mathcal{D}_K) && (41)\\
&= \arg\min_{\psi} -\Bigg(\log p(\mathcal{D}_K \mid \psi_{\text{shared}}, \psi_{\text{specific}}^{(K)}) && (42)\\
&\quad + \log p(\psi_{\text{shared}}, \psi_{\text{specific}}^{(1)}, \dots, \psi_{\text{specific}}^{(K-1)} \mid \mathcal{D}_1, \dots, \mathcal{D}_{K-1}) + \log p(\psi_{\text{specific}}^{(K)}) \Bigg) && (43)
\end{aligned}$$

However, for the sake of readability, we ignore the multi-head setting in the remainder of the section.

As a way to analyze the stability of the MAP estimates, we perform a Laplace approximation of the posterior parameter distributions using the solutions obtained in Eq. 39 and Eq. 40 (cf. Sec. B.5 and Huszár (2018)), yielding:

$$p(\psi \mid \mathcal{D}_k) \approx \mathcal{N}\big(\psi_{\text{MAP}}^{(k)}, (\Omega^{(k)})^{-1}\big) \quad \text{and} \quad p(\psi \mid \mathcal{D}_1, \dots, \mathcal{D}_K) \approx \mathcal{N}\big(\psi_{\text{MAP}}^{(1:K)}, (\Omega^{(1:K)})^{-1}\big) \tag{44}$$

where the precision matrices are given by:

$$\Omega^{(k)} = \frac{1}{\sigma_{\text{prior}}^2} I + N_k F^{(k)} \tag{45}$$

$$\Omega^{(1:K)} = \frac{1}{\sigma_{\text{prior}}^2} I + \sum_{k=1}^{K} N_k F^{(k)} \tag{46}$$

assuming an isotropic Gaussian prior $p(\psi) = \mathcal{N}(0, \sigma_{\text{prior}}^2 I)$. In the equations above, N_k denotes the dataset size of task k and $F^{(k)}$ the Fisher information matrix estimated on the model after learning task k.

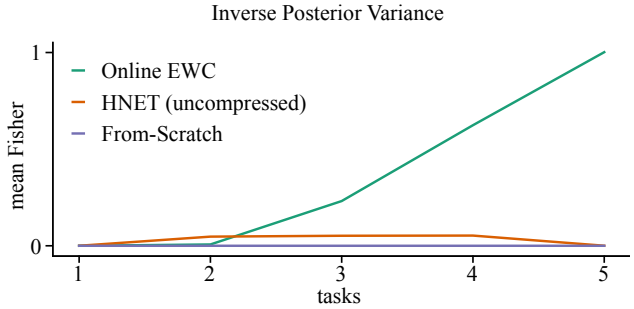

Figure S7: Inverse posterior variance for some CL methods. The plot depicts the mean of the diagonal Fisher matrix as obtained from the `during` solutions found by different methods. For methods using task-specific solutions, such as `From-Scratch` and `HNET`, robustness only has to be measured with respect to the current task and therefore the Fisher has been evaluated on the corresponding task only. For `Online EWC`, the solution always has to be robust with respect to all tasks seen so far.

Since the Gaussian posterior approximation contains viable solutions for the data to be explained, high entries in the covariance matrix indicate a flat solution around the MAP estimate. Analogously, the Fisher information matrix is connected to the expected Hessian of the log-likelihood. Therefore, low Fisher values correspond approximately to low curvature of the likelihood loss landscape and are thus indicative of flat minima, which are often considered desirable and have been linked to better generalization (Hochreiter and Schmidhuber, 1997b; Chaudhari et al., 2019). Incidentally, flat minima can also be considered desirable for the `HNET` approach. Because flatness indicates robustness of the found solutions against parameter perturbations, it loosens the pressure on the L2 regularization applied in Eq. 2, and therefore decreases sensitivity to the regularization strength β.

To test these insights empirically, we studied the mean of the diagonal Fisher matrix obtained in the `during` models for the Permuted Copy task with $p = i = 5$ (cf. Sec. 5.1). The results are plotted in Fig. S7. The `From-Scratch` baseline consists of independently trained models and therefore its solution only has to be stable with respect to the current task. Since the hypernetwork approach aspires to this behavior while allowing transfer, we treat it identically. Thus, for both methods we characterize the flatness of the solution using the mean of the corresponding task-specific diagonal Fisher matrix $F^{(k)}$. As expected, the `From-Scratch` baseline exhibits low Fisher values that are similar across tasks, illustrating the flatness of the separately found solutions. For the compressed hypernetworks used in this work, we do not see a clear trend. Instead, the mean Fisher values are highly variable and depend on the hyperparameter setting. Therefore, we instead considered an uncompressed version, where the hypernetwork is allowed to be five times as big as the corresponding main model (i.e., it consumes roughly the same amount of model parameters as the `From-Scratch` baseline). In this case, and as shown in Fig. S7, the mean Fisher values are similar across tasks and the flatness of solutions is comparable to those found by `From-Scratch`. This observation is interesting as it opens up possible avenues for future work. As indicated above, it would be desirable to obtain flat, task-specific solutions with the `HNET` approach. Since this does not seem to occur automatically (at least in the compressed regime), one could enforce flat solutions explicitly. One could use, for instance, methods such as Entropy-SGD (Chaudhari et al., 2019) that are applicable to the hypernetwork approach.

Lastly, the solution found by `Online EWC` has to be robust with respect to all previously seen tasks and therefore its curvature for task k can be characterized via the sum over all tasks seen so far $\sum_{k'<k} N_{k'} F^{(k')}$. As expected, mean Fisher values are progressively increasing (Fig. S7). This has the well known side effect of increasing rigidity and harming flexibility for learning new tasks. Furthermore, and connecting back to the literature on flat minima, it can be argued that the final solution found by `Online EWC` is less desirable as it resides in a sharper minimum of the loss landscape and empirical evidence suggests that such minima are harmful to generalization.

G.12 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

inferring task identity based on the input sequence.[16] 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.[17]

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 (van de Ven and Tolias, 2018) that is trained analogously as described in Sec. B.8 and B.9.[18] 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. (2020) 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 (Snoek et al., 2019), it would be an interesting direction for future work to investigate this approach for RNNs.

Another alternative, utilized in Cossu et al. (2020a;b) 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.[19] 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).

G.13 The effect of modifying the experimental setup

To further highlight the importance of comparing CL methods within a clearly defined experimental setting, we perform a few controls where we vary this setting and study the impact on performance, focusing on `Online EWC`.

Despite an even distribution of hyperparameter-optimization resources among methods, a comparison might still be inconclusive if certain experimental variables are misaligned, such as the number of output heads, the number of model parameters or the availability of task identity information. Such factors can positively or negatively influence the performance of each method and therefore need to be carefully controlled for.

Using a single output head. Throughout this study all methods are trained in a multi-head setting, where a different set of output weights is used for each task, consistent with the first CL scenario described by van de Ven and Tolias (2019). Another possibility is to use a single head, i.e. a common set of output weights across all tasks. One can expect that a method producing task-conditional weights, such as `HNET`, might be better suited for a single-head setting than a method that has to progressively adapt its output weights, especially if the output is normalized (e.g., by using a softmax, since individual weight changes without co-adaptation of other weights in the output layer can drastically alter all predictions on prior tasks).

To empirically assess whether `HNET` and `Online EWC` are differently affected by the type of output layer used, we rerun the hyperparameter searches for the *Permuted Copy Task* experiment $i = p = 5$ from Sec. 5.1 using a single shared output head for all tasks. For fair comparison (cf. paragraph below), task identity in the form of a one-hot vector is provided as an additional input to the networks trained with `Online EWC`.

[16] This kind of CL scenario was termed `CL3` in van de Ven and Tolias (2019) and von Oswald et al. (2020).

[17] It is however always possible to design an auxiliary system that infers task identity from a given and appropriately chosen context (von Oswald et al., 2020; He et al., 2019)

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

[19] Note that regularized autoencoders have been shown to elicit properties of the data-generating density function (Alain and Bengio, 2014). Hence, this method of task inference can be loosely linked to proper out-of-distribution detection.

While the performance for HNET is only slightly affected (95.78 ± 2.13 mean final accuracy), Online EWC drops to 68.41 ± 1.45 % mean final accuracy, which drastically differs from the multi-head result obtained in Sec. 5.1. This comparison highlights the importance of having task-specific output weights in a CL setting with dissimilar tasks.

Weight-importance methods with task-conditional computation. This study focuses on comparing methods in a CL scenario where task identity is known during inference (cf. SM G.12). Traditionally, weight-importance methods solely use task identity to select the correct output head in a multi-head setting. This implies that computation within the RNN is shared among all tasks up to the output layer, and that all tasks need to be solved in parallel if task identity cannot be inferred from the provided inputs. To overcome this constraint and allow for task-conditional computation, task identity can be provided to the network as an additional input (e.g., in the form of a one-hot encoding).

Table 16: Mean during and final accuracies across several experiments for Online EWC either without providing the task identity as additional input (left column) or by explicitly enabling task-conditional computation by giving the task identity as additional input (right column). (Mean $\pm$ SEM in %, $n = 10$).

	w/o task identity		with task identity	
	during	**final**	**during**	**final**
Permuted Copy $p = i = 5$	99.93 ± 0.01	98.66 ± 0.14	97.61 ± 0.31	97.54 ± 0.30
Sequential Split-SMNIST $m = 1$	98.52 ± 0.11	97.05 ± 0.59	99.39 ± 0.08	98.36 ± 0.37
Sequential Split-SMNIST $m = 2$	91.86 ± 1.13	76.65 ± 3.39	95.71 ± 0.53	88.51 ± 1.26
Sequential Split-SMNIST $m = 3$	83.62 ± 1.10	75.26 ± 1.85	96.63 ± 0.89	93.14 ± 1.24
Sequential Split-SMNIST $m = 4$	92.33 ± 2.49	73.16 ± 0.98	90.65 ± 1.28	88.94 ± 1.32
Audioset	68.82 ± 0.20	65.56 ± 0.35	71.74 ± 0.20	66.35 ± 0.36
PoS tagging	87.49 ± 0.04	86.89 ± 0.03	89.78 ± 0.03	89.67 ± 0.04

To evaluate whether a weight-importance method (namely Online EWC) can benefit from such task-conditioning, we rerun the hyperparameter-searches for some of our main experiments while providing the task-identity as an additional input. The results are depicted in Table 16. We observe that in some experiments where the main network is an LSTM or BiLSTM, performance can be greatly improved by introducing task-conditioning. Performance gains are particularly striking in the Sequential Split-SMNIST experiment. This is somewhat surprising since in this dataset task identity can be inferred from the inputs alone (even though this might require observing a significant portion of the input sequence first), which should enable task-conditional processing without the need to explicitly provide the task identity.

When moving into more difficult CL scenarios, task-identity is not explicitly given to the system and has to be first inferred from the inputs to allow task-conditional processing. SM G.12 details several approaches for how an auxiliary system can provide a task-identity signal. However, a more natural approach for adapting task-conditional EWC to these CL scenarios would be to use predictive uncertainty (which was suggested by von Oswald et al. (2020) in the context of CL with hypernetworks). Therefore, one could use the approximate parameter posterior distribution, that has been obtained during training, also during inference to quantify uncertainty of output heads when considering all possible task-identity inputs. Note, however, that this increases computational efforts significantly as (1) an MC estimate of the posterior predictive distribution needs to be obtained and (2) this process needs to be repeated for every task identity.

In conclusion, this unorthodox but justified use of Online EWC reveals great potential of weight-importance methods for sequential processing. Because of its performance, ease of implementation and elegant mathematical derivation, Online EWC might be preferable in simple use cases over more elaborate methods such as HNET, even though HNET remains the better performing option in most situations. Therefore, an important contribution of our work is to identify Online EWC as a strong contender for sequential processing, despite having been questioned by existing work (Asghar et al., 2020; Cossu et al., 2020a; Li et al., 2020) and despite the shortcomings revealed by our own study.