Appendix

The supplementary materials include

- **Appendix A**: Full algorithmic formulation of CoPE.
- **Appendix B**: Setup details with all hyperparameters for reproducibility.
- **Appendix C**: Resource analysis of CoPE compared to other data incremental replay methods.
- **Appendix D**: Extended ablation study of prototype momentum and the PPP-loss components with t-SNE visualization.
- **Appendix E**: Additional experiments in low capacity benchmarks Split-MNIST-mini and Split-CIFAR10-mini.

A. Algorithm

Our proposed algorithm is fully formalized in this section, as well as in our code that will be made publicly available on acceptance of this paper. Algorithm 1 and Algorithm 2 describe the learner for CoPE, whereas the evaluator uses $c^* = \arg\max_{c \in \mathcal{Y}} \mathbf{f}_i^T\ \mathbf{p}^c$, classifying $\mathbf{x}_i$ as category c^* with the most similar prototype $\mathbf{p}^{c^*}$. As for a true continually progressing system, the evaluator can urge prediction at any point in time, while the learner keeps acquiring knowledge from the data stream.

Algorithm 1 The CoPE learner in the data incremental learning setup.

```
Require: data stream $S$, prototype momentum $\alpha$, memory capacity $M$, learning rate $\eta$
Initialize  operational memory $\mathcal{M} = \emptyset$, observed classes $\mathcal{Y} = \emptyset$, sample count per class $N = \emptyset$, model parameters $\theta$
 1: for $B_n = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_{|B_n|}, \mathbf{y}_{|B_n|})\} \sim S$ do     ▷ Data stream batch w/o task information
 2:     $B_{\mathcal{M}} \leftarrow$ RandomSample$(\mathcal{M}_r, |B_n|)$     ▷ Randomly sample $|B_n|$ exemplars from $\mathcal{M}_r$
 3:     $\mathcal{B} = \emptyset$
 4:     for $(\mathbf{x}_i, \mathbf{y}_i) \in B_n \cup B_{\mathcal{M}}$ do
 5:         if $y_i \notin \mathcal{Y}$ then
 6:             InitClass$(\mathcal{M}, N, \mathcal{Y}, y_i)$     ▷ Initialize memory and prototype
 7:         end if
 8:         $\mathcal{B} \leftarrow \mathcal{B} \cup f_\theta(\mathbf{x}_i)$     ▷ Collect features
 9:     end for
10:     $\mathcal{L} \leftarrow 0$     ▷ Initialize loss
11:     for $\mathbf{f}_i^c \in \mathcal{B}$ do
12:         $\mathcal{L} \leftarrow \mathcal{L} - \frac{1}{|\mathcal{B}|}\left[\log P(c|\mathbf{x}_i^c) + \sum_{\mathbf{x}_j^k} \log(1 - P(c|\mathbf{x}_j^k))\right]$     ▷ Sum al instances PPP-loss
13:     end for
14:     $\theta \leftarrow \theta + \eta\, \nabla \mathcal{L}$     ▷ Optimize objective with SGD
15:     PrototypeUpdate$(\mathcal{M}_p, \mathcal{B}, N, \alpha)$     ▷ Update prototypes in $\mathcal{M}_p$
16:     MemoryUpdate$(\mathcal{M}_r, B_n, N)$     ▷ Update memory $\mathcal{M}_r$ with new input samples
17: end for
```

Algorithm 2 Memory Management of the replay memory and prototypes. $\text{UNIFORM}^{\mathbb{R}^d}(s_1, s_2)$ samples elements in a d-dimensional vector with uniform probability in range $[s_1, s_2] \in \mathbb{R}$.

```
Require: memory capacity $M$
1: function INITCLASS($\mathcal{M}, N, \mathcal{Y}, y$)
2:     $N \leftarrow N \cup \{N^y = 0\}$                          ▷ Sample counts
3:     $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{y\}$                          ▷ Observed classes
4:     $m = M/|\mathcal{Y}|$                                   ▷ Capacity per class
5:     for $\mathcal{M}_r^c = (\mathbf{x}_1, ..., \mathbf{x}_{|\mathcal{M}_r^c|}) \in \mathcal{M}_r$ do
6:         $\mathcal{M}_r^c \leftarrow (\mathbf{x}_1, ..., \mathbf{x}_m)$                ▷ Keep first $m$
7:     end for
8:     $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mathcal{M}^y = \emptyset\}$
9:     $\mathbf{p}^y \leftarrow \text{UNIFORM}^d(0, 1)$
10:    $\mathcal{M}_p^y \leftarrow \{\mathbf{p}^y / ||\mathbf{p}^y||_2\}$                  ▷ Init prototype
11: end function
```

```
1: function PROTOTYPEUPDATE($\mathcal{M}_p, \mathcal{B}, N, \alpha$)
2:     for $\mathbf{p}^c \in \mathcal{M}_p$ do
3:         $N^c \leftarrow N^c + |\mathcal{B}^c|$
4:         $\bar{\mathbf{p}}^c = \frac{1}{|\mathcal{B}^c|} \sum_{\mathbf{f}^c \in \mathcal{B}^c} \mathbf{f}^c$
5:         $\mathbf{p}^c \leftarrow \alpha \mathbf{p}^c + (1 - \alpha) \bar{\mathbf{p}}^c$
6:         $\mathbf{p}^c \leftarrow \mathbf{p}^c / ||\mathbf{p}^c||_2$                    ▷ Normalize
7:     end for
8: end function
9: function MEMORYUPDATE($\mathcal{M}_r, B_n, N$)
10:    for $\mathbf{x}_i^c \in B_n$ do                          ▷ Class Reservoir
11:        $j = \text{UNIFORM}^{\mathbb{N}^1}(1, N^c)$
12:        if $j \leq |\mathcal{M}_r^c|$ then
13:            $\mathcal{M}_r^c[j] \leftarrow \mathbf{x}_i^c$                    ▷ Replace exemplar
14:        end if
15:    end for
16: end function
```

B. Setup

A gridsearch in the online continual learning setup was adopted, selecting the setup with highest performance, similar to [5]. All methods are prone to learning rate gridsearch $[0.05, 0.01, 0.005, 0.001]$. iCaRL knowledge distillation strength is set to 1, and GEM bias is set to 0.5, following [5, 8, 2]. GSS and MIR follow their original setup from their codebase in [2] and [1], with our additional learning rate gridsearch. CURL [7] and DN-CPM [4] results, and the best imbalanced Split-MNIST results out of the greedy/IQP versions for GSS [2] are reported from their original works. CoPE searched for a suitable temperature $\tau = [0.1, 0.2, ..., 1, 2]$ which was set to 0.1 for all balanced and imbalanced Split-MNIST and Split-CIFAR10 experiments, similar to [9]. Based on the ablation study in Appendix D, we set the prototypical momentum fixed to 0.99. For the challenging Split-CIFAR100 setting methods are allowed multiple iterations per batch as in [5], from which the best results are selected (baselines, reservoir, CN-DPM perform 1 iteration, others 5). The CIFAR100 temperature required higher concentration with $\tau = 0.05$ and prototypical momentum 0.9. For the balanced setups, the latent dimensionality d is fixed to 100 for Split-MNIST as in [7], and selected 256 in a gridsearch $[128, 256]$ and $[128, 256, 512]$ for Split-CIFAR10 and Split-CIFAR100 respectively. The imbalanced benchmarks follow the low capacity setup in Appendix E.1, with $d \in [16, 32, 64]$ set to 64 for Split-MNIST and $d \in [128, 256]$ set to 128 for Split-CIFAR10 and 256 for Split-CIFAR100, with $|\mathcal{M}_r|$ set to 0.3k, 1k and 5k respectively. We found slightly better results without L2 re-normalization of the prototypes. The CIFAR10 labels in the confusion matrices from 0 to 10 stand for the indices in the following list: [*airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*]. Code is publicly available to ensure reproducibility.

C. Resource analysis data incremental replay methods

In this section we compare usage of computational and memory resources for the replay methods fitted for the task-free online data incremental learning paradigm.

Reservoir is a powerful baseline for balanced data streams [3], with only minimal computational cost by keeping count n of how many samples have been observed. This count is then used relative to the buffer size M to define the probability M/n to store the new sample. As shown in the imbalanced data stream experiments, Reservoir is not fit for more real-world scenarios with typically varying frequency of occurrence per class. Improving this simple experience replay has led to research focusing on more complex strategies, discussed in the following.

MIR [1] replaces the random retrieval from the buffer in Reservoir with a loss-based approach. They store a momentary update of the network optimized for the new incoming batch and calculate the change in loss for a random subset of replay memories $\tilde{B}$, which is larger than the batch size (ideally five times the batch size for their experiments [1]). Besides a copy of the full model, this also requires calculating the loss twice in a sequential manner for the full subset $\tilde{B}$ and an extra temporary model update using only the new batch B_n, both significantly increasing processing time for the learner.

GSS [2] resides with Reservoir to use random retrieval of the buffer, but proposes a gradient-based population strategy.

They introduce two variants, in which the first solves an Integer Quadratic Problem (IQP) with polynomial complexity w.r.t. the replay memory. As this is not scalable, they also propose a stochastic GSS-greedy variant. This more efficient GSS-greedy approach requires an additional forward pass, loss calculation, and backwards pass to obtain the gradients for the full considered subset $\tilde{B}$ in the memory. Additionally, it uses similarities of the gradients for stochastic sample selection in the replay memory $\mathcal{M}_r$, straining memory requirements as batch B_n requires for each sample $|\tilde{B}| + 1$ gradients to be accessed simultaneously to calculate $|\tilde{B}|$ cosine similarities in the high-dimensional gradient-space.

CoPE (ours) resembles Reservoir's memory population by keeping count of the samples per class-specific replay memory subset. The PPP-loss requires calculation of a similarity matrix with all the d-dimensional representations in the batch B. Using a normalized cosine similarity, this implies efficient matrix multiplication with the low-dimensional vectors. This is in high contrast to GSS, which calculates cosine similarity in the full high-dimensional gradient space for additional samples that are not present in current batch B, and therefore requires additional costly forward and backward passes. Furthermore, in our prototypical approach the prototype momentum updates also rely solely on samples that are in the current batch B, hence requiring only minimal additional computation. Comparing to both MIR and GSS, we don't require storing model copies or additional gradients, but merely store low-dimensional prototypes for each class, saving a significant amount of required storage space. For example, a Resnet18 model requires 11.7 million parameters to enable model copies or gradients, whereas our method even for 1000-way classification with $d = 1024$ would require only 9% of the model capacity in memory for the prototypes. Note that new categories require an additional prototype, which would be the same size of the weight vector of an additional output unit in a linear classifier. However, in this work we consider the features on the level of the output space. Therefore, additional d-dimensional prototypes are stored, but as mentioned in the previous example, these are limited in size and the set of categories is typically limited as well.

D. Extended ablation study

D.1. Ablation prototype momentum

In all experiments, a high momentum is employed to update prototypes with the latent mean of the batch. Table 1 illustrates the influence of higher momentum (≥ 0.9). Compared to low momentum of 0.1, Split-MNIST only gains a small margin of 0.45%, whereas Split-CIFAR10 and Split-CIFAR100 significantly improve with at least 3.0% and 4.2% respectively. Using momentum prevents the prototype to rely solely on the current batch instances, and higher momentum values attain a more gradual change of the prototypes by stabilizing its trajectory in the ever-evolving latent space.

	Prototype Momentum			
	0.1	0.9	0.95	0.99
Split-MNIST	93.49 ± 0.70	94.11 ± 0.34	93.96 ± 0.30	93.94 ± 0.20
Split-CIFAR10	44.48 ± 3.19	48.02 ± 2.49	47.98 ± 3.14	48.92 ± 1.32
Split-CIFAR100	15.79 ± 1.16	21.62 ± 0.69	21.56 ± 0.58	20.01 ± 1.81

Table 1: Ablation study changing momentum strength for prototype updates, reported in average accuracy (%) over 5 runs. Higher momentum values (≥ 0.9) obtain better performance, especially for the CIFAR sequences, compared to low momentum (0.1).

D.2. Ablation inter and intra-class variance terms PPP-loss

In this section, the importance is scrutinized of the two loss components to enhance inter and intra-class variance in the PPP-loss. Table 2 compares using only positive pairs from the batch in the attractor ($\mathcal{L}_{pos}$) or only negative pairs in the repellor ($\mathcal{L}_{neg}$) to the full-fledged PPP-loss ($\mathcal{L}$). The attractor term shows competitive performance to the full PPP-loss for Split-MNIST, but deteriorates as the data streams become harder for the CIFAR setups. The repellor term is on par with the full PPP-loss for Split-MNIST and Split-CIFAR10, but collapses for Split-CIFAR100. The latter is challenging due to the high number of classes with only a batch size of 10, which impedes having pseudo-prototypes of all classes in the same batch. The PPP-loss incorporates both reduction of intra-class variance with the attractor term and increases inter-class variance with the repellor term, attaining state-of-the-art performance.

Besides isolating the attractor and repellor terms of the PPP-loss in the ablation study, we further investigate the weighing of the two terms during the lifetime of the learner in Figure 1. We average results over 5 runs for balanced Split-MNIST, finding the repellor to dominate. This trend is to be expected as the repellor term has per instance a summation over all other class instances. The attractor term has minimal influence especially for data presented for the first task. This indicates the samples in the binary latent space (having observed only two classes) majorly repelling rather than attracting samples. The embedding network is still learning the initial features, and overlap in the two latent class distributions summed over the other class samples results in a prevailing repellor term.

	PPP-loss		
	$\mathcal{L}$	$\mathcal{L}_{pos}$	$\mathcal{L}_{neg}$
Split-MNIST	$\mathbf{93.94 \pm 0.20}$	93.25 ± 0.22	$\mathbf{93.84 \pm 0.48}$
Split-CIFAR10	$\mathbf{48.92 \pm 1.32}$	30.96 ± 3.58	$\mathbf{49.30 \pm 3.57}$
Split-CIFAR100	$\mathbf{21.62 \pm 0.69}$	15.85 ± 0.34	9.43 ± 0.94

Table 2: Ablation using solely the attractor ($\mathcal{L}_{pos}$) or repellor ($\mathcal{L}_{neg}$) compared to using both terms in the PPP-loss ($\mathcal{L}$).

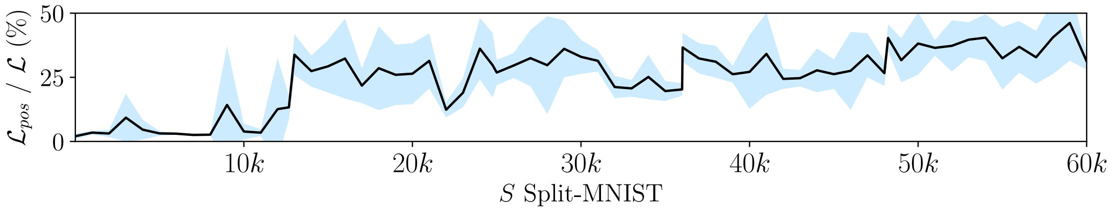
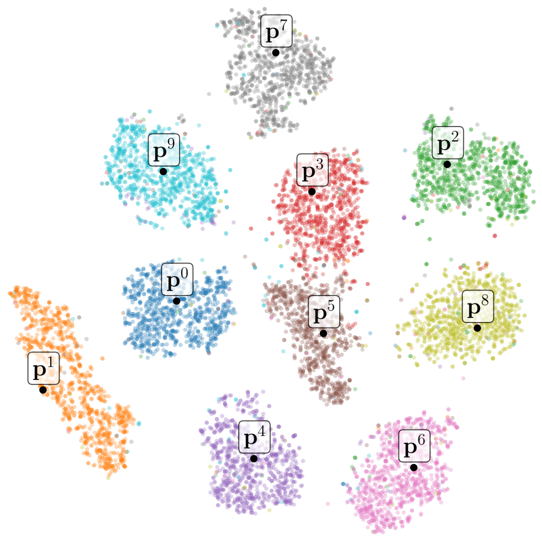
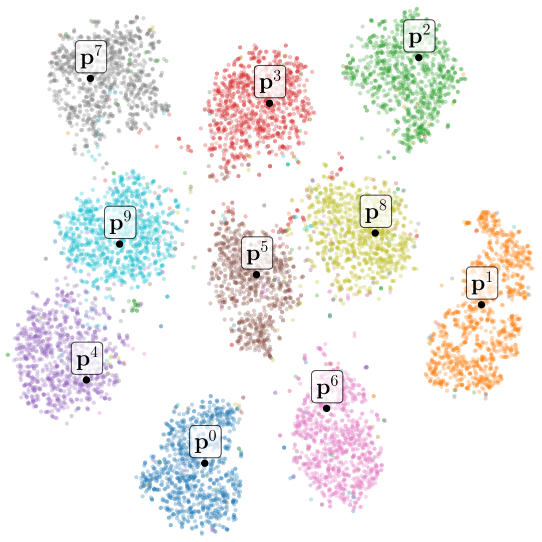


Figure 1: Weighing (%) between the attractor loss term $\mathcal{L}_{pos}$ compared to the full PPP-loss $\mathcal{L}$, averaged over 5 runs of balanced Split-MNIST with standard deviation in blue.



(a) PPP-loss – *incl.* $\hat{\mathbf{p}}$



(b) PPP-loss – *excl.* $\hat{\mathbf{p}}$

Figure 2: Split-MNIST first seed t-SNE representation of the test data S_{eval}, including (a) and excluding (b) the pseudo-prototypes $\hat{\mathbf{p}}$ in the PPP-loss.

D.3. Pseudo-prototype ablation visualization

In the main paper we find in an ablation study that using pseudo-prototypes $\hat{\mathbf{p}}$ as proxy for the class-mean has significant improvements for the PPP-loss. Additionally, Figure 2 shows this in a 2-dimensional t-SNE [6] space for the first seed of the balanced Split-MNIST experiment. Including the pseudo-prototypes (*incl.* $\hat{\mathbf{p}}$) illustrates a striking degree of inter-class variance in Figure 2a, whereas more interference occurs when excluding the pseudo-prototypes in Figure 2b. This is reflected in the performance, as including prototypes results in 94.52% accuracy, whereas excluding them has only 90.86% for the first seed.

E. Additional experiments

E.1. Balanced data streams with low capacity

In these experiments we scrutinize performance of CoPE with less capacity in the memory and model, and with shorter data streams. All methods are allowed multiple iterations (maximal 5) as in [2]. Results are averaged over 5 seeds. Similar to the setup of GSS [2], we adopt two data sequences with truncated data per task:

- **Split-MNIST-mini** is similar to the Split-MNIST data stream with 5 tasks, but each task is confined to 1k training samples. Evaluation considers the full test subset. The network is an MLP with two hidden layers of 100 units, with total memory size of 0.3k exemplars. Latent dimensionality d is selected 32 from $[16, 32, 64]$.
- **Split-CIFAR10-mini** is similar to the Split-CIFAR10 data stream with 5 tasks, but each task comprises 2k training samples, with a total subset of 10k samples out of the 50k available. The full test subset is used for evaluation. The network used is the same ResNet18 as in the main paper, with total memory size of 1k exemplars. Latent dimensionality d is selected 128 from $[128, 256]$.

Analysis. Table 3 shows the results for Split-MNIST-mini and Split-CIFAR10-mini, with GSS and DN-CPM results reported from their original works in a corresponding setup. In Split-MNIST-mini our method approaches the iid-online baseline up to 1%, and outperforms its closest competitors GEM and MIR with at least 1.45%. In Split-CIFAR10-mini CoPE saliently surpasses the iid-online baseline with 2.25%, hence outperforming online training over an iid datastream. Moreover, CoPE surpasses CN-DPM by 3%. Reservoir proves a strong baseline, with in this case the additional MIR loss-based retrieval decreasing performance. Similar to our findings in the main paper and [1, 2], GEM encounters difficulties in a CIFAR10 based setup, for which we find the bias hyperparameter $\gamma \geq 0$ in the gradient projection to have insignificant influence. These results confirm CoPE outperforming both GSS and CN-DPM in this low capacity setting established in their original work.

	Split-MNIST-mini	**Split-CIFAR10-mini**
iid-offline	94.58 ± 0.17	67.41 ± 1.37
iid-online	87.57 ± 3.54	42.50 ± 2.15
finetune	21.74 ± 3.38	16.65 ± 0.24
GEM	85.09 ± 0.52	22.31 ± 1.37
iCaRL	83.23 ± 0.92	26.54 ± 2.73
DN-CPM [4]	–	41.78
reservoir	82.73 ± 2.39	38.21 ± 3.39
MIR	84.40 ± 0.91	37.20 ± 2.74
GSS [2]	82.60 ± 2.90	33.56 ± 1.70
CoPE	$\mathbf{86.54 \pm 1.41}$	$\mathbf{44.75 \pm 2.68}$

Table 3: Split-MNIST-mini and Split-CIFAR10-mini results, with respectively only 1k and 2k samples per task. GSS and DN-CPM results reported from original work in these setups.

E.2. Imbalanced Benchmark Results

The graphs in the main paper visualize the numbers in Table 4, which we fully report here as a reference for future work. Each $S(T_i)$ data stream performance is averaged over five different initial seeds. The '*Avg.*' results average over all mean performances of the dataset variants $S(T_i)$.

Dataset	Imbalanced Sequence	CoPE	CoPE-CE	GSS	MIR	Reservoir
Split-MNIST	$S(T_1)$	83.4 ± 2.0	81.8 ± 1.2	75.9 ± 3.2	64.8 ± 5.1	64.2 ± 2.3
	$S(T_2)$	84.5 ± 1.6	80.1 ± 1.9	78.5 ± 2.7	67.4 ± 3.2	65.5 ± 4.6
	$S(T_3)$	85.1 ± 0.6	79.6 ± 2.0	81.5 ± 2.3	72.4 ± 3.0	72.1 ± 4.0
	$S(T_4)$	84.8 ± 1.0	80.0 ± 3.1	79.5 ± 0.6	72.6 ± 3.1	73.6 ± 2.4
	$S(T_5)$	84.0 ± 1.3	80.7 ± 1.8	79.1 ± 0.7	77.2 ± 3.4	73.2 ± 4.0
	Avg.	$\mathbf{84.4 \pm 0.7}$	80.4 ± 0.9	78.9 ± 2.0	70.9 ± 4.9	69.7 ± 4.5
Split-CIFAR10	$S(T_1)$	39.0 ± 1.3	36.4 ± 3.0	32.3 ± 3.0	32.6 ± 3.6	35.5 ± 3.4
	$S(T_2)$	35.3 ± 2.6	34.1 ± 2.8	28.3 ± 0.4	27.2 ± 1.8	29.3 ± 2.8
	$S(T_3)$	36.2 ± 2.5	34.6 ± 2.5	29.5 ± 1.5	29.6 ± 2.1	31.4 ± 2.1
	$S(T_4)$	39.1 ± 2.4	33.5 ± 4.2	34.6 ± 1.3	31.0 ± 2.3	32.1 ± 0.6
	$S(T_5)$	37.3 ± 3.3	33.9 ± 2.9	28.3 ± 2.4	27.6 ± 2.7	28.8 ± 1.9
	Avg.	$\mathbf{37.4 \pm 1.7}$	34.5 ± 1.1	30.6 ± 2.8	29.6 ± 2.3	31.4 ± 2.7
Split-CIFAR100	$S(T_1)$	18.2 ± 0.6	11.7 ± 0.6	10.2 ± 0.8	18.4 ± 0.9	11.1 ± 0.6
	$S(T_5)$	18.5 ± 1.3	12.6 ± 1.2	10.7 ± 0.5	17.6 ± 0.9	11.5 ± 1.4
	$S(T_{10})$	19.2 ± 0.9	11.1 ± 0.7	11.1 ± 0.3	17.8 ± 0.7	11.9 ± 0.7
	$S(T_{15})$	18.7 ± 0.6	11.2 ± 0.8	11.1 ± 0.9	17.8 ± 0.9	12.1 ± 0.8
	$S(T_{20})$	18.5 ± 1.5	12.8 ± 1.3	11.1 ± 0.4	17.6 ± 0.4	12.5 ± 1.1
	Avg.	$\mathbf{18.6 \pm 0.4}$	11.9 ± 0.8	10.8 ± 0.4	17.8 ± 0.3	11.8 ± 0.5

Table 4: Numeric results for imbalanced Split-MNIST, Split-CIFAR10 and Split-CIFAR100 sequences.

References

[1] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems*, pages 11849–11860, 2019. 2, 5

[2] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*, pages 11816–11825, 2019. 2, 5

[3] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486*, 2019. 2

[4] Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A neural dirichlet process mixture model for task-free continual learning. In *International Conference on Learning Representations*, 2019. 2, 5

[5] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017. 2

[6] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 5

[7] Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual unsupervised representation learning. In *Advances in Neural Information Processing Systems*, pages 7645–7655, 2019. 2

[8] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 2

[9] Mang Ye, Xu Zhang, Pong C Yuen, and Shih-Fu Chang. Unsupervised embedding learning via invariant and spreading instance feature. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6210–6219, 2019. 2