A EXPERIMENT SETUP

We perform experiments on six commonly used classification datasets: PERMUTED MNIST, ROTATED MNIST (LeCun et al., 1998), SPLIT SVHN (Netzer et al., 2011), SPLIT CIFAR10 (Krizhevsky et al., 2009), SPLIT CIFAR100 (Krizhevsky et al., 2009), and SPLIT MINIIMAGENET (Vinyals et al., 2016).

- PERMUTED MNIST (Goodfellow et al., 2013) is a variant of the MNIST dataset of handwritten digits (LeCun et al., 1998), where each task applies a fixed random pixel permutation to the original dataset. The benchmark dataset consists of 20 tasks, each with 1000 samples from 10 different classes.
- ROTATED MNIST (Lopez-Paz & Ranzato, 2017) is another variant of the MNIST dataset of handwritten digits (LeCun et al., 1998), where each task applies a fixed random image rotation to the original dataset. The benchmark dataset consists of 20 tasks, each with 1000 samples from 10 different classes.
- SPLIT SVHN is a variant of the SVHN dataset (Netzer et al., 2011) that consists of 5 tasks, each with two consecutive classes. Since the benchmark dataset is much more challenging than the MNIST variants, we use all of its 73,257 training samples (i.e. 14,650 samples per task) to train our model and the baselines.
- SPLIT CIFAR10 is a variant of the CIFAR-10 dataset (Krizhevsky et al., 2009). Similar SPLIT SVHN, the benchmark dataset consists of 5 tasks, each with two consecutive classes. We use all of its 50,000 training samples (i.e. 10,000 samples per task) to train our model and the baselines.
- SPLIT CIFAR100 is a variant of the CIFAR-100 dataset (Krizhevsky et al., 2009). The benchmark dataset consists of 20 tasks, each with 5 consecutive classes. We use all of its 50,000 training samples (i.e. 2,500 samples per task) to train our model and the baselines.
- SPLIT MINIIMAGENET is a variant of the MINIIMAGENET dataset (Krizhevsky et al., 2009). The benchmark dataset consists of 20 tasks, each with 5 consecutive classes. We use all of its 50,000 training samples (i.e. 2,500 samples per task) to train our model and the baselines. Each image is resized to 84×84 pixels.

B MODEL ARCHITECTURES

As mentioned, while most of previous work uses multi-head architectures and assumes knowledge of task boundaries at test time, we employ a shared classifier head for all tasks. For the MNIST datasets, the image encoders f_{θ_1} (for graph construction) and f_{θ_2} (for latent computation) share a multi-layered perceptron with two hidden layers of 256 ReLU neurons, followed by two separate linear mappings, one for each of the encoders. For SPLIT SVHN, SPLIT CIFAR10, SPLIT CIFAR100, and SPLIT MINIIMAGENET, the image encoders share a simple convolutional network with the following structure: `conv 64` $\rightarrow$ `conv 64` $\rightarrow$ `maxpool` $\rightarrow$ `conv 64` $\rightarrow$ `conv 64` $\rightarrow$ `maxpool` $\rightarrow$ `conv 64` $\rightarrow$ `conv 64` $\rightarrow$ `maxpool`, where `conv NF` is a 3×3 convolution with `NF` output filters, BatchNorm, and ReLU activations. For all datasets, another linear mapping follows the image encoder f_{θ_1} before a Gaussian kernel computes the similarities between image embeddings. Finally, the classifier head consists of a RELU activation and a single linear mapping.

C BASELINE ARCHITECTURES

We use the same neural network architectures for all the baselines described in the paper: a multi-layered perceptron with two hidden layers of 400 ReLU neurons on PERMUTED MNIST and ROTATED MNIST, following (Hsu et al., 2018), and a ResNet-18 (He et al., 2016) with 20 filters across all layers on SPLIT SVHN and SPLIT CIFAR10, following (Lopez-Paz & Ranzato, 2017). For all datasets, the baselines consist of more parameters than our corresponding models. The numbers of parameters in each experiment are detailed in Table 4.

We adopt the implementations of EWC (Kirkpatrick et al., 2017), GEM (Lopez-Paz & Ranzato, 2017), and MER (Riemer et al., 2018) from the authors' repositories [1] [2]. Our implementation of GCL is available at https://bit.ly/2S5kQ2t.

Table 4: Number of trainable parameters in continual learning models.

Method	Finetune	EWC	GEM	ER	MER	**GCL**
SPLIT MNIST	478K	478K	478K	478K	478K	406K
PERMUTED MNIST	478K	478K	478K	478K	478K	406K
ROTATED MNIST	478K	478K	478K	478K	478K	406K
SPLIT SVHN	1.09M	1.09M	1.09M	1.09M	-	326K
SPLIT CIFAR10	1.09M	1.09M	1.09M	1.09M	-	326K
SPLIT CIFAR100	1.09M	1.09M	1.09M	1.09M	-	326K
SPLIT MINIIMAGENET	1.09M	1.09M	1.09M	1.09M	-	343K

D ADDITIONAL TASK-FREE BASELINES

We also note that despite our attempts to tune parameters for MER (Riemer et al., 2018) on SPLIT SVHN and SPLIT CIFAR10, the baseline does not perform reasonably well. The model uses a batch size of 1 and requires multiple passes through the episodic memory per batch, so it is much slower than our model and all other baselines. Due to limited time and computational resources, we do not further investigate the baseline and therefore avoid reporting immature results for fairness.

However, we include results of CN-DPM (Lee et al., 2020), a competitive task-free model based on Dirichlet process mixture models in Table 5. Our setup for SPLIT CIFAR10 is analogous to that of Lee et al. (2020), so we directly quote the numbers for CN-DPM from the paper. Although CN-DPM performs favorably among task-free approaches to continually learning, including GSS (Aljundi et al., 2019c), our model outperforms CN-DPM by a significant margin, even when using a smaller memory size.

Table 5: GCL results and CN-DPM results with different memory sizes.

Method	SPLIT SVHN		SPLIT CIFAR10	
	250	500	500	1000
ER (Chaudhry et al., 2019)	45.51 $\pm$ 3.03	57.51 $\pm$ 2.77	36.08 $\pm$ 1.09	45.75 $\pm$ 1.82
CN-DPM (Lee et al., 2020)	–	–	43.07 $\pm$ 0.16	45.21 $\pm$ 0.18
GCL (Ours)	**60.68** $\pm$ 1.67	**65.79** $\pm$ 1.54	**53.87** $\pm$ 0.97	**57.26** $\pm$ 0.28

E MEMORY USAGE

Both GCL and ER (Chaudhry et al., 2019) uses an episodic memory to store images and labels from past tasks. The only additional memory usage of GCL comes from the context graph $\mathbf{G}$, which is represented by a square matrix whose entries intuitively describe pairwise similarities between such images. Given a memory consisting of $|\mathcal{M}|$ images of size $C \times H \times W$, it only requires $|\mathcal{M}|^2$ floating points to store the matrix.

[1] https://github.com/facebookresearch/GradientEpisodicMemory

[2] https://github.com/mattriemer/mer

Table 6: Memory usage of ER and GCL for various datasets.

DATASET	$\lvert\mathcal{M}\rvert$	Image Size	ER	GCL
PERMUTED MNIST	1000	$1 \times 28 \times 28$	3.284 MB	7.284 MB
ROTATED MNIST	1000	$1 \times 28 \times 28$	3.284 MB	7.284 MB
SPLIT CIFAR10	250	$3 \times 32 \times 32$	3.109 MB	3.359 MB
SPLIT SVHN	250	$3 \times 32 \times 32$	3.109 MB	3.359 MB
SPLIT CIFAR100	500	$3 \times 32 \times 32$	6.219 MB	7.199 MB
SPLIT MINIIMAGENET	500	$3 \times 84 \times 84$	42.408 MB	43.389 MB

As seen from Table 6, the memory usage of GCL are very similar the same as that of ER, except when both are very small as in the case of PERMUTED MNIST and ROTATED MNIST, because (1) continual learning algorithms are often required to use a very small $|\mathcal{M}|$ and (2) the cost for storing natural images are often much higher than that of the context graph.

As the number of tasks increases, it is perhaps essential to expand the episodic memory, in which case the quadratic growth of the latter might dominate the linear increase of the former (e.g. $|\mathcal{M}| = 5000$ and images are of size $3 \times 32 \times 32$). Although we have not practically encountered such a problem with GCL, we note that the quadratic growth of the number of entries in the context graph can be reduced to a linear growth in memory requirements. More specifically, each entry is the output of the kernel function κ_τ (see Section 3, e.g. $\kappa_\tau(\mathbf{u}_i, \mathbf{u}_j) = \exp\left(-\frac{\tau}{2}\|\mathbf{u}_i - \mathbf{u}_j\|_2^2\right)$), so we could easily store $|\mathcal{M}|$ intermediate embeddings $\{\mathbf{u}_i\}$ at each step and apply the kernel function on the fly, which is especially beneficial when $\mathbf{u}_i$ are much lower dimensional than the original images.

F ADDITIONAL EXPERIMENT RESULTS

Table 7: Classification results (%) on SPLIT CIFAR100 with different number of epochs. When used, episodic memories contain 5 samples per class on average. The symbol ↑ (↓) indicates that a higher (lower) number is better.

TRAINING	1 EPOCH		10 EPOCHS	
Method	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)
Finetune	47.47 ± 2.77	10.94 ± 2.29	55.39 ± 1.94	25.94 ± 1.89
EWC	48.39 ± 0.99	6.49 ± 1.28	55.60 ± 1.11	23.53 ± 1.19
ICARL	52.18 ± 0.27	8.63 ± 2.71	58.74 ± 0.74	9.02 ± 2.49
GEM	57.90 ± 1.13	7.51 ± 1.52	65.66 ± 0.70	15.52 ± 0.41
ER	65.61 ± 1.28	5.05 ± 2.23	69.40 ± 1.21	11.25 ± 1.24
GCL	**69.82** ± 1.05	**4.27** ± 2.35	**74.51** ± 0.99	**6.54** ± 1.26

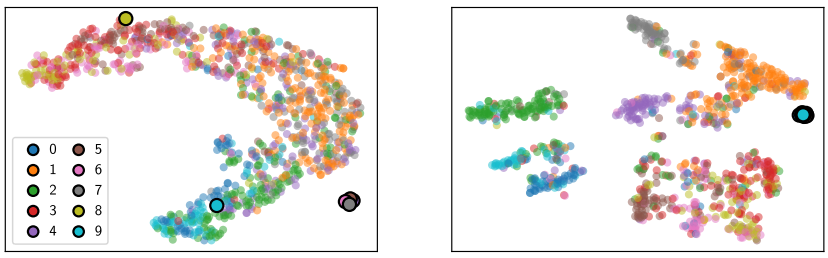

Figure 7: t-SNE visualization of image embeddings (small circles) from the penultimate layers and class embeddings (large circles) from the weights of the last layers on SPLIT SVHN. The *left* figure shows that **Finetune**, a model naively trained on the data stream, fails to recognize the class-based clustering structure and bias the image embeddings toward the last task (class 8 & 9). In contrast, the *right* figure shows that **GCL** (our model) maintains the relational structure and is more robust to the distributional shifts incurred by task changes.

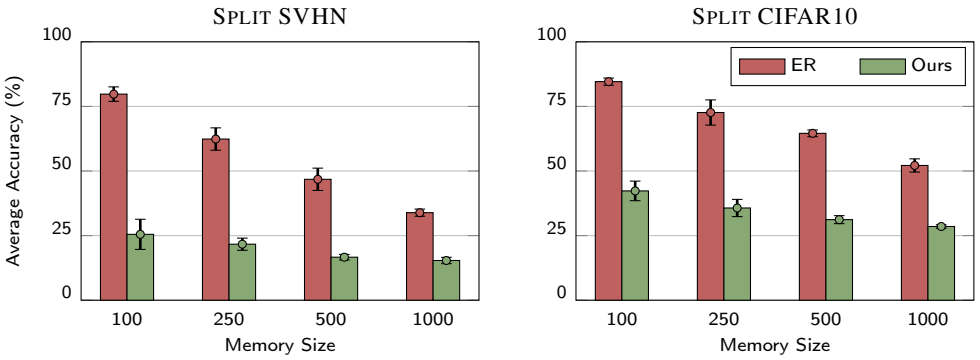

Figure 8: Average forgetting as a function of memory size on SPLIT SVHN and SPLIT CIFAR10.

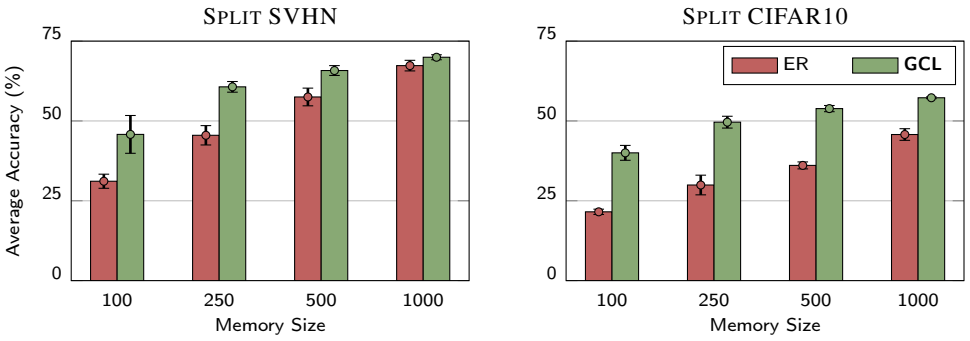

Figure 9: Average accuracy as a function of memory size.

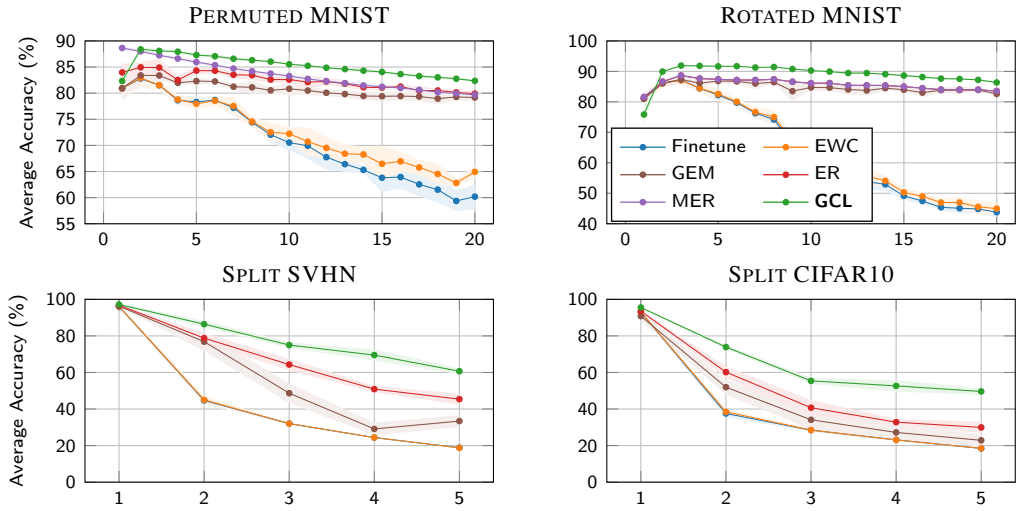

Figure 10: Average accuracy as a function of the number of tasks trained.

G HYPER-PARAMETERS

Following Lopez-Paz & Ranzato (2017), we report the hyper-parameter grids considered in our experiments. These hyper-parameters are selected independently for each model, and the best values are given in parenthesis.

- Finetune

  - optimizer: [Adam (Split SVHN, Split CIFAR10), SGD (Permuted MNIST, Rotated MNIST)]
  - learning rate: [0.0002, 0.001 (Split SVHN, Split CIFAR10), 0.01, 0.1 (Permuted MNIST, Rotated MNIST), 0.3, 1.0]
- EWC (Kirkpatrick et al., 2017)
  - optimizer: [Adam, SGD (Permuted MNIST, Rotated MNIST, Split SVHN, Split CIFAR10)]
  - learning rate: [0.0002 (Split SVHN, Split CIFAR10), 0.001, 0.01, 0.1 (Permuted MNIST, Rotated MNIST), 0.3, 1.0]
  - regularization: [0.1, 1, 10 (Permuted MNIST, Rotated MNIST), 100 (Split SVHN, Split CIFAR10), 1000]
- GEM (Lopez-Paz & Ranzato, 2017)
  - optimizer: [Adam (Split SVHN, Split CIFAR10), SGD (Permuted MNIST, Rotated MNIST)]
  - learning rate: [0.0002, 0.001 (Split CIFAR10), 0.01, 0.1 (Permuted MNIST, Rotated MNIST), 0.3, 1.0]
  - margin: [0.0, 0.1, 0.5 (Permuted MNIST, Rotated MNIST), 1.0 (Split SVHN, Split CIFAR10)]
- MER (Riemer et al., 2018)
  - optimizer: [SGD (Permuted MNIST, Rotated MNIST)]
  - learning rate: [0.0002, 0.001, 0.01, 0.1 (Permuted MNIST, Rotated MNIST), 0.3, 1.0]
  - within batch meta-learning rate (beta): [0.01 (Permuted MNIST, Rotated MNIST, Split CIFAR10), 0.03, 0.1, 0.3, 1]
- ER (Chaudhry et al., 2019)
  - optimizer: [Adam (Split SVHN, Split CIFAR10), SGD (Permuted MNIST, Rotated MNIST)]
  - learning rate: [0.0002, 0.001 (Split SVHN, Split CIFAR10), 0.01, 0.1 (Permuted MNIST, Rotated MNIST), 0.3, 1.0]
- GCL
  - optimizer: [Adam (Permuted MNIST, Rotated MNIST, Split SVHN, Split CIFAR10), SGD]
  - learning rate: [0.0002, 0.001 (Permuted MNIST, Rotated MNIST, Split SVHN, Split CIFAR10), 0.01, 0.1, 0.3, 1.0]
  - graph regularization: [0, 10, 50 (Split SVHN, Split CIFAR10), 100, 1000, 5000 (Rotated MNIST)]
  - context temperature: [0.1 (Permuted MNIST, Rotated MNIST), 0.3, 1 (Split SVHN, Split CIFAR10), 5, 10]
  - target temperature: [0.1, 0.3, 1, 5 (Permuted MNIST, Rotated MNIST, Split SVHN, Split CIFAR10), 10]