Continual learning with direction-constrained optimization (Supplementary Material)

7 Supplementary references

We next briefly review dynamic architecture methods and reply techniques which, similarly to regularization-based methods, constitute continual learning strategies.

Dynamic architecture methods either expand the model architecture [19, 20, 21, 22] to allocate additional resources to accommodate new tasks (they are typically memory expensive) or exploit the network structure by parameter pruning or masking [23, 24]. Some techniques [25] interleaves the periods of network expansion with network compression, network pruning, and/or masking phases to better control the growth of the model.

Replay methods are designed to train the model on a mixture of samples from a new task and samples from the previously seen tasks. The purpose of replaying old examples is to counter-act the forgetting process. Many replay methods rely on the design of sampling strategies [26, 27]. Other techniques, such as GEM [28], A-GEM [18], and MER [29], use replay specifically to encourage positive transfer between the tasks (increasing the performance on preceding tasks when learning a new task). Replay methods typically require large memory. Deep generative replay technique [30, 31] addresses this problem and employs a generative model to learn a mixed data distribution of samples from both current and past tasks. Samples generated this way are used to support the training of a classifier. Finally, note that the setting considered in our paper does not rely on the replay mechanism.

In addition to the above discussed research directions, very recently authors started to look at task agnostic continual learning where no information about task boundaries or task identity is given to the learner [32, 33, 34]. These approaches lie beyond the scope of this work.

8 Experimental details for Section 3

We first extract images from MNIST data set [35] with labels of $\{0, 1\}$, then resize the original images to size $1 \times 8 \times 8$, and finally normalize each image by mean (0.1307) and standard deviation (0.3081).

We use a two-hidden-layer MLP to make prediction between these two classes. The numbers of neurons for each layer are (64-30-30-2) and no bias is applied. We use SGD optimizer [5] (learning rate = 1×10^{-3}) with batch size of 128.

We use cross-entropy loss in our experiments.

9 Additional experimental results for Section 3

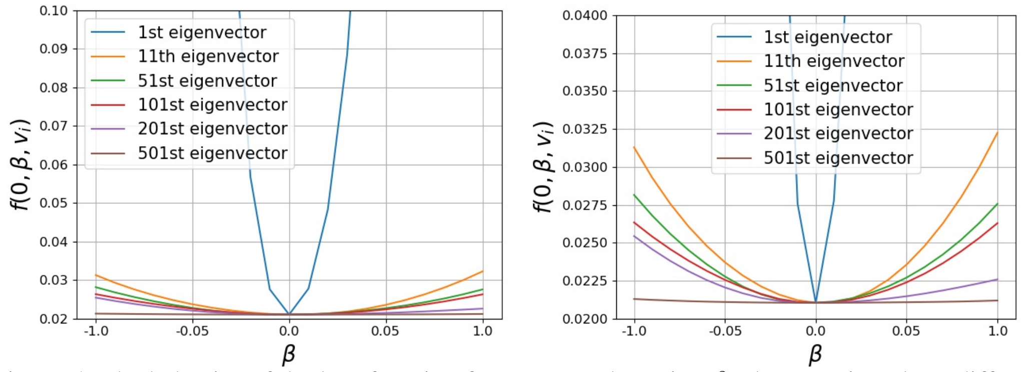

Figure 5: The behavior of the loss function for $\alpha = 0$ and varying β when moving along different eigenvectors (**left**: original plot **right**: zoomed plot).

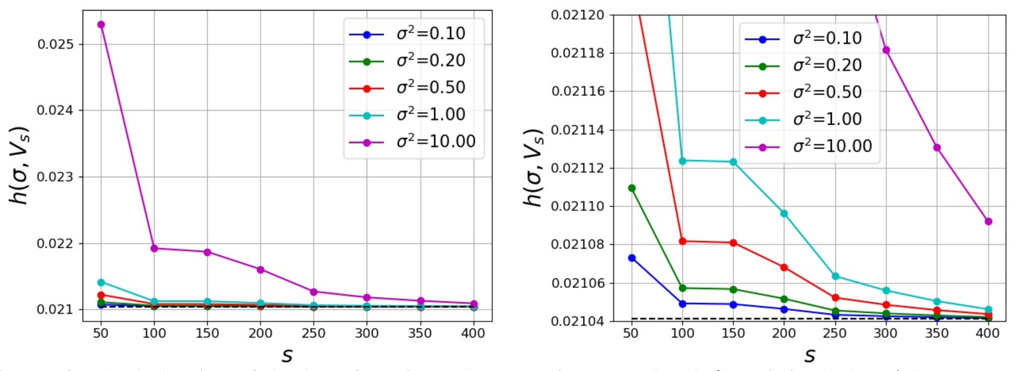

Figure 6: The behavior of the loss function when varying σ and s (**left**: original plot **right**: zoomed plot).

10 Experimental details for Section 5

10.1 Data sets

We consider three commonly used continual learning data sets: (1) **Permuted MNIST**. For each task we used a different permutation of the pixels of images from the original MNIST data set [35]. We generated 5 data sets this way corresponding to 5 tasks. (2) **Split MNIST**. We divide original MNIST data set into 5 disjoint subsets corresponding to labels $\{\{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}, \{8, 9\}\}$. (3) **Split CIFAR-100** data set. We divide original CIFAR-100 data set [36] into 10 disjoint subsets corresponding to labels $\{\{0 - 9\}, \cdots, \{90 - 99\}\}$.

In the experiments with Split MNIST and Split CIFAR-100 we use a multi-head setup [2, 9] and we provide task descriptors [28, 18] to the model at both *training* and *testing*.

10.2 Data processing

For Permuted MNIST and Split MNIST, we use the original image of size $1 \times 28 \times 28$. We then normalize each image by mean (0.1307) and standard deviation (0.3081). For Split CIFAR-100, we use the original image of size $3 \times 32 \times 32$. We then normalize each image by mean $(0.5071, 0.4867, 0.4408)$ and standard deviation $(0.2675, 0.2565, 0.2761)$.

10.3 Architectures

For Permuted MNIST, we use a Multi-Layer Perceptron (MLP) with two hidden layers, each having 256 units with ReLU activation functions. For Split MNIST, we use a MLP with two hidden layers each having 100 units with ReLU activation functions. For Split CIFAR-100, we use the same convolutional neural network as in [9]. In all architectures we turn off the biases. To train the model, we use Adam optimizer [37] (learning rate = 1×10^{-3}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$) with batch size set to 128, 128 and 64 respectively for Permuted MNIST, Split MNIST and Split CIFAR-100.

10.4 Hyperparameters

In Table 2 we summarize the setting of the regularization parameters explored for each method (except A-GEM which is classified as a replay method).

Table 2: Regularizations.

Name	Permuted MNIST	Split MNIST	Split CIFAR-100
EWC (λ)	$\{10, 20, 50, 100\}$	$\{10^4, 10^5, 10^6, 10^7\}$	$\{10^4, 10^5, 10^6, 10^7\}$
SI (c)	$\{0.01, 0.1, 1, 10\}$	$\{0.1, 1, 10, 100\}$	$\{0.01, 0.1, 1, 10\}$
RWALK (λ)	$\{0.01, 0.1, 1, 10\}$	$\{0.1, 1, 10, 100\}$	$\{10^1, 10^2, 10^3, 10^4\}$
DCO (λ)	$\{100\}$	$\{100\}$	$\{1000\}$

In Table 3 we summarize the training details for A-GEM. We denote the amount of episodic memory per task and the size of the batch used for the computation of reference gradients g_{ref} as episodic memory size and episodic batch size respectively.

Table 3: Training settings for A-GEM.

A-GEM	Permuted MNIST	Split MNIST	Split CIFAR-100
Episodic Memory Size	256	256	512
Episodic Batch size	256	256	1300

In Table 4 we summarize training details for DCO. When we train the linear autoencoders for DCO in step 3 of the Algorithm 1, we always scale $L_{mse}(W; G)$ by a factor ρ to avoid numerical issues.

Table 4: Training settings for DCO.

γ	N	m	τ	k	ρ
0.1	10	128	2	1000	100

11 Additional experimental results for Section 5

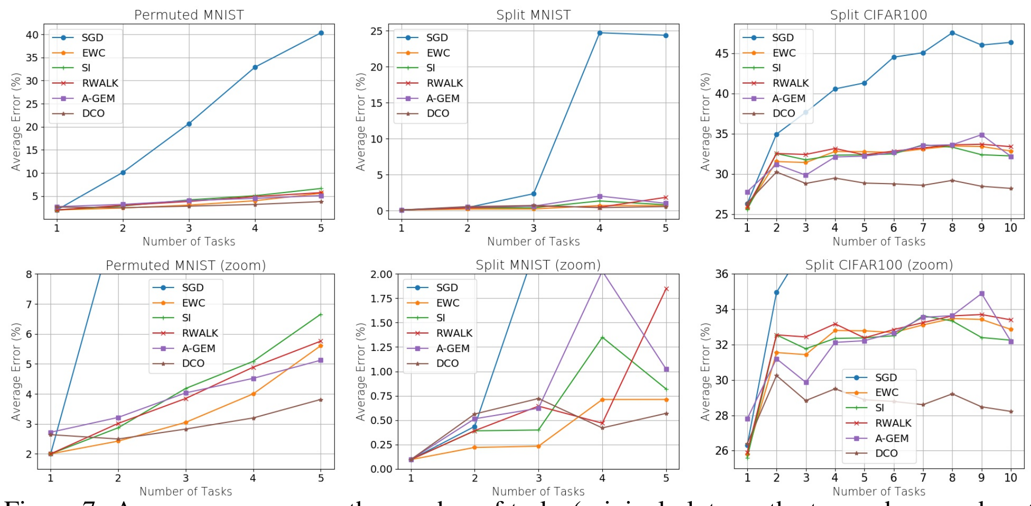

Figure 7: Average error versus the number of tasks (original plots on the top and zoomed on the bottom; **left:** Permuted MNIST **middle:** Split MNIST **right:** Split CIFAR-100).