Clustering-based Domain-Incremental Learning - Appendix

Christiaan Lamers [1]
NORCE
Grimstad, 4879, Norway
chla@norceresearch.no

René Vidal
Institute for Data Engineering and Science
University of Pennsylvania
Philadelphia, USA
vidalr@seas.upenn.edu

Nabil Belbachir
NORCE
Grimstad, 4879, Norway
nabe@norceresearch.no

Niki van Stein
Leiden Institute of Advanced Computer Science
Leiden, 2333 CA, The Netherlands
n.van.stein@liacs.leidenuniv.nl

Thomas Bäck
Leiden Institute of Advanced Computer Science
Leiden, 2333 CA, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

Paris Giampouras
Johns Hopkins University
Baltimore, MD 21218, US
parisg@jhu.edu

A. Averaged Gradient Episodic Memory

A-GEM uses a projected gradient descent-type approach for updating the weights of the model. It first calculates the gradient of the loss function $\nabla L_t(w) \in \mathbb{R}^p$ and then compares this to a reference gradient $\nabla L_{ref}(w) \in \mathbb{R}^p$, of the loss function $L_{ref}(w)$ as defined by

$$L_{ref}(w) = \frac{1}{|\tilde{M}|} \sum_{(x,y)\in\tilde{M}} L_{(x,y)}(w), \tag{1}$$

where $\tilde{M}$ is a randomly selected batch of data sampled from the episodic memory M which contains label training data from tasks seen before time t i.e., $M = \cup_{k<t} M_k$, where M_k is the episodic memory for task k. When the inner product between the gradient of the loss function $\nabla L_t(w)$ and the reference gradient $\nabla L_{ref}(w)$ is non-negative, the gradient update boils down to a vanilla gradient descent step without projections. However, when this inner product is less than zero, the weights of the model are updated using a projected gradient-type update, i.e.,

$$w \leftarrow w - \eta\tilde{g} \tag{2}$$

where η denotes the step size and $\tilde{g}$ is defined as,

$$\tilde{g} = \nabla L_t(w) - \frac{\nabla L_t(w)^\top \nabla L_{ref}(w)}{\nabla L_{ref}(w)^\top \nabla L_{ref}(w)} \nabla L_{ref}(w). \tag{3}$$

Note that $\tilde{g}$ is now orthogonal to the reference gradient $\nabla L_{ref}(w)$; thus, if $\nabla L_{ref}(w)$ successfully represents a non-forgetting direction of previously sees tasks, then the update rule of the weights of the model in (2), decreases the loss on the current task without incurring any forgetting of previous tasks [1].

Algorithm 1 shows pseudo-code for A-GEM [1]. It uses both a loss gradient $\nabla L(w)$ and a reference gradient $\nabla L_{ref}(w)$ that is computed using a sub sample of the episodic memory, namely $\tilde{M}$. If the dot product between the loss gradient and the reference gradient is greater or equal to zero, gradient descent proceeds as normal. However, if it is smaller than zero, the loss gradient is projected onto the reference gradient and the resulting component is subtracted from the loss gradient. The weights w are then updated using this newly obtained gradient and the step size η. After the training on task k finishes, s samples are taken from the data of task T_k and added to the episodic memory M.

B. Orthogonal Gradient Descent

Orthognal Gradient Descent (OGD) departs from A-GEM, which uses the gradient of the loss function, and performs projections leveraging gradients of the output $f(x;w)$ of the model denoted as $\nabla f(x;w) \in \mathbb{R}^{p\times c}, x \in T_{k_t}$. We also denote as

$$\nabla f_j(x;w) \in \mathbb{R}^p, (x,y) \in T_{k_t} \tag{4}$$

[1]Corresponding author.

Algorithm 1 Averaged Gradient Episodic Memory

```
    Input: Task sequence $T_1, T_2, T_3, \ldots$ learning rate $\eta$
    Output: Model
 1: Initialize $M \leftarrow \{\}; w \leftarrow w_0$
 2: for Task ID $k = 1, 2, 3, \ldots$ do
 3:     repeat
 4:         $L(w) \leftarrow \frac{1}{|T_{k_t}|} \sum_{(x,y) \in T_{k_t}} L_{(x,y)}(w)$
 5:         $L_{ref}(w) \leftarrow \frac{1}{|\tilde{M}|} \sum_{(x,y) \in \tilde{M}} L_{(x,y)}(w)$
 6:         if $\nabla L(w)^\top \nabla L_{ref}(w) \geq 0$ then
 7:             $\tilde{g} \leftarrow \nabla L(w)$
 8:         else
 9:             $g_{proj} \leftarrow \frac{\nabla L(w)^\top \nabla L_{ref}(w)}{\nabla L_{ref}(w)^\top \nabla L_{ref}(w)} \nabla L_{ref}(w)$
10:             $\tilde{g} \leftarrow \nabla L(w) - g_{proj}$
11:         end if
12:         $w \leftarrow w - \eta \tilde{g}$
13:     until converge
14:     for $i = \{1, \ldots, s\}$ do
15:         $(\mathbf{x}, y) \sim T_k$
16:         $M \leftarrow M \cup (\mathbf{x}, y)$
17:     end for
18: end for
```

the gradient associated with the output logit j:

$$f_j(x; w) \in \mathbb{R}, (x, y) \in T_{k_t}, \tag{5}$$

i.e. the j-th output of the final layer, where j again is the class label. For simplicity, from now on, we refer to (4) with the term *model gradient*.

OGD uses projected gradient-type updates in order to update the model parameters for a new task in directions that minimally increase the loss for samples from previously seen tasks. Specifically, after training on task k is finished, an update takes place that stores the relevant model gradients for task k in the model gradient pool.

Assume our model has been trained on task T_1 using SGD and denote as w_1^* the weights learned through the training process. The gradient pool after the training on task T_1 is the set $\mathcal{S}_1$, defined as

$$\mathcal{S}_1 = \{\nabla f_j(x; w_1^*) \mid j : y_j = 1, (x, y) \in T_1\}. \tag{6}$$

When training on the next task T_2, rather than training using standard SGD, OGD first projects the gradient of the loss for each batch onto the orthogonal complement of the subspace spanned by all elements of the gradient pool $\mathcal{S}_1$. More specifically, the updates of the weights of OGD for given batches of task T_2 have the following form,

$$w \leftarrow w - \eta \hat{g} \tag{7}$$

where η is the stepsize and $\hat{g} = g - \sum_{u \in \mathcal{S}_1} \text{proj}_u(g)$ with g denoting the gradient $\nabla L_t(w)$ of the loss $L_t(w)$ corresponding to the batch T_{k_t}. When training on T_2 ends with weights w_2^*, OGD augments the gradient pool $\mathcal{S}_1$ with model gradients for T_2 and so on. After training on task T_{k-1}, the gradient pool is redefined as

$$\mathcal{S}_{1:k-1} = \{\nabla f_j(x; w_i^*) \mid j : y_j = 1, 1 \leq i < k, (x, y) \in T_i\}. \tag{8}$$

Hence, when training on task T_k using the update in (7), we have that $\forall_{u \in \mathcal{S}_{1:k-1}}, \; \hat{g} \perp u$.

The stored model gradients are interpreted as the directions of most change on the previous task. Thus, by projecting orthogonal to these, the resulting projected gradient steps move the weights towards the region of low error on the new task (Task B), while minimally changing the performance on previous tasks (Task A).

Algorithm 2 shows pseudo-code for OGD [2]. The loss gradient $\nabla L(w)$ is projected on an orthogonal basis S and the resulting component is subtracted from the loss gradient $\nabla L(w)$. The weights w are then updated using this newly obtained gradient and the step size η. After the training on task T_k is finished, samples (x, y) from the task T_k are used to calculated the model gradient $f_j(x; w)$, which represents the gradient of the jth output of the network, where j corresponds to the label y. This model gradient is then added to the basis S, but it is first made orthogonal to all elements in S, to assure that S remains an orthogonal basis.

Algorithm 2 Orthogonal Gradient Descent

```
    Input: Task sequence $T_1, T_2, T_3, \ldots$ learning rate $\eta$
    Output: Model
 1: Initialize $S \leftarrow \{\}; w \leftarrow w_0$
 2: for Task ID $k = 1, 2, 3, \ldots$ do
 3:     repeat
 4:         $L(w) \leftarrow \frac{1}{|T_{k_t}|} \sum_{(x,y) \in T_{k_t}} L_{(x,y)}(w)$
 5:         $g_{proj} \leftarrow \sum_{\mathbf{v} \in S} \frac{\nabla L(w)^\top \mathbf{v}}{\mathbf{v}^\top \mathbf{v}} \mathbf{v}$
 6:         $\tilde{g} \leftarrow \nabla L(w) - g_{proj}$
 7:         $w \leftarrow w - \eta \tilde{g}$
 8:     until converge
 9:     for $(x, y) \in T_k$ and $j \in [1, c]$ s.t. $y_j = 1$ do
10:         $f_{proj} \leftarrow \sum_{\mathbf{v} \in S} \frac{\nabla f_j(x;w)^\top \mathbf{v}}{\mathbf{v}^\top \mathbf{v}} \mathbf{v}$
11:         $u \leftarrow \nabla f_j(x; w) - f_{proj}$
12:         $S \leftarrow S \cup \{u\}$
13:     end for
14: end for
```

C. Clustering Mechanism

Algorithm 3 shows the clustering mechanism used by TA-A-GEM and TA-OGD, when adding a sample $\mathbf{z}$ to the pool. First, a pool C is initialized as an empty set. The maximum number of clusters is denoted with Q. The maximum cluster size is P. The fist Q samples $\mathbf{z}$ initialize a cluster q, be being added to it and by setting the cluster's mean $\boldsymbol{\mu}_q$ to $\mathbf{z}$. After all clusters are initialized, the ℓ_2 norm is used to determine the cluster q^*, that has its mean closest to $\mathbf{z}$. $\mathbf{z}$ is then added to this cluster. The size of q^* is kept below the maximum cluster size P, by removing its oldest member $\mathbf{z}^{q^*}_{\text{oldest}}$. The mean of the cluster q^*, namely $\boldsymbol{\mu}_{q^*}$, is recalculated by taking the average of all its members.

Algorithm 3 Clustering Mechanism

```
1:  $C \leftarrow \{\}$
2:  $Q \leftarrow 100$
3:  $P \leftarrow 3$
4:  procedure ADD($\mathbf{z}$)
5:      if $|C| < Q$ then
6:          $q \leftarrow \{\mathbf{z}\}$
7:          $C \leftarrow C \cup q$
8:          $\boldsymbol{\mu}_q \leftarrow \mathbf{z}$
9:      else
10:         $q^* \leftarrow \text{argmin}_{q \in \{1,2,\ldots,Q\}} \|\mathbf{z} - \boldsymbol{\mu}_q\|_2^2$
11:         $q^* \leftarrow q^* \cup \mathbf{z}$
12:         if $|q^*| > P$ then
13:             $q^* \leftarrow q^* \setminus \mathbf{z}^{q^*}_{\text{oldest}}$
14:         end if
15:         $\boldsymbol{\mu}_{q^*} \leftarrow \frac{1}{|q^*|} \sum_{p=1}^{|q^*|} \mathbf{z}^{q^*}_p$
16:     end if
17: end procedure
```

D. Metric definition

The **accuracy** A_l is defined as $A_l = \frac{M_{k_l}}{N_{k_l}}$, where M_{k_l} is the number of correctly classified samples in dataset T_{k_l} and N_{k_l}, is the total number of samples corresponding to task T_{k_l}. Note that the index k_l is the index of samples from task k at epoch l. The **forgetting** F_l on the validation accuracy is calculated from the validation accuracy A_l, at epoch l using,

$$F_l = \max(\{A_i | i \leq l\}) - A_l \tag{9}$$

E. Learning Rate Scheduler

Algorithm 4 shows the pseudo-code of the learning rate scheduler as used by the TA-OGD method. It *lowers* the learning rate, by multiplying it with a **Factor** < 1, when the learning process *stagnates*, i.e. when the loss value reaches a plateau. It *increases* the learning rate, by resetting it to **LR init**, when a *large spike* in the loss value overshoots the **Best** loss value by more than the **Reset Threshold** value. The **Patience** value, together with the counters **N_Stagnant** and **N_Spike**, make sure that the learning rate scheduler does not react to irrelevant outlier loss values. The **Min LR** determines a minimum learning rate, which the learning rate should never fall below. The values mentioned in the pseudo-code are used in the experiments.

Algorithm 4 Learning Rate Scheduler

```
1:  Factor : 0.9999
2:  Min LR : $10^{-5}$
3:  LR init: $10^{-3}$
4:  Patience : 5
5:  Best : $\infty$
6:  N_Stagnant : 0
7:  N_Spike : 0
8:  Threshold: $10^{-4}$
9:  Reset Threshold: 1
10: procedure STEP(Loss, LR)
11:     if Loss $<$ Best $*$ (1$-$ Threshold) then
12:         Best $\leftarrow$ Loss
13:         N_Stagnant $\leftarrow 0$
14:     else
15:         N_Stagnant $\leftarrow$ N_Stagnant $+1$
16:     end if
17:     if Loss $>$ Best + Reset Threshold then
18:         N_Spike $\leftarrow$ N_Spike $+1$
19:     else
20:         N_Spike $\leftarrow 0$
21:     end if
22:     if N_Spike $>$ Patience then
23:         LR $\leftarrow$ LR init
24:         Best $\leftarrow$ Loss
25:         N_Spike $\leftarrow 0$
26:     end if
27:     if N_Stagnant $>$ Patience then
28:         LR $\leftarrow$ max(LR $*$ Factor, Min LR)
29:         N_Stagnant $\leftarrow 0$
30:     end if
31:     return LR
32: end procedure
```

F. Experiments

Two main classes of experiments are described in the main paper, the *"Disjoint tasks experiments"* and the *"Continuous change experiments"*. In this section, figures and results are given, for which no space existed in the main paper.

F.1. Best practices

In order to test the effectiveness of a continual learning method, different tasks need to be presented to it over time. These tasks can be synthesized from standard datasets, but it is crucial that the produced tasks give a good representation of a real world continual learning setting. In designing our experiment, we abide by all the desiderata described in [3], i.e., 1) cross-task resemblances between tasks, 2) a shared output head, 3) no test-time assumed task labels, 4) no unconstrained retraining on old tasks, and 5) the use of more than just two tasks. By conforming to these, we ensure that the continual learning task is not trivially easy and thus a method does not seem to be more effective than it would truly be in real-case scenarios. Note that for the case of the experiments on permutation for task generation, we "violate" the cross-task resemblances since such a requirement is impossible to be enforced.

F.2. Task generation details

Separate tasks were created from existing datasets by means of *permutation*, *rotation* and *class splitting*. The task permutation mechanism picks one permutation that shuffles all pixels at random per task and then applies this one permutation to all images in the dataset. The task rotation mechanism rotates every image in one task with a given number of degrees. We chose to increment the number of degrees by 20 for each new task. The class splitting mechanism splits the dataset on disjoint sets of labels. As subsets we chose the labels (0,1), (2,3), (4,5), (6,7) and (8,9).

To make the tasks more disjoint, for the task permutation and task rotation, the data was first split into disjoint subsets (one for each task). After this, the permutation or rotation was applied. For class splitting, the different tasks are already completely disjoint, so no extra separation step was required. Note that we do not use head swapping, meaning that we have one fixed output layer for all tasks. For the class split scenario, only two classes are given to the network during one task. Therefore we chose to give the network just two output nodes, instead of ten. To accommodate this, we changed the labels. All even labels were set to 0 and all odd labels were set to 1.

F.3. Environment

Each experiment instance ran on a single NVIDIA Tesla T4 card with 15 Gb of video memory. Up to fifteen NVIDIA Tesla T4 cards were available in parallel, together with 32 physical processor cores and 252 GB of RAM.

F.4. Disjoint tasks experiments

Figures 1, 2, 3, 4 and 5 show the average validation accuracy, the validation accuracy on the first task and the forgetting on the validation accuracy during training for different task split methods, using the MNIST, Fashion MNIST, NOT MNIST, CIFAR10 and SVHN datasets. Table 1 and 2 summarize the results on the validation accuracy on the first task and the forgetting on the validation accuracy on the first task, respectively. A summary for the average validation accuracy is included in the main paper.

Specific settings for the task splitting can be seen in table 6. The settings specific to TA-A-GEM and TA-OGD can be seen in table 7 and 8, respectively. All results depicted are the average of five independent runs.

F.5. Continuous change experiments

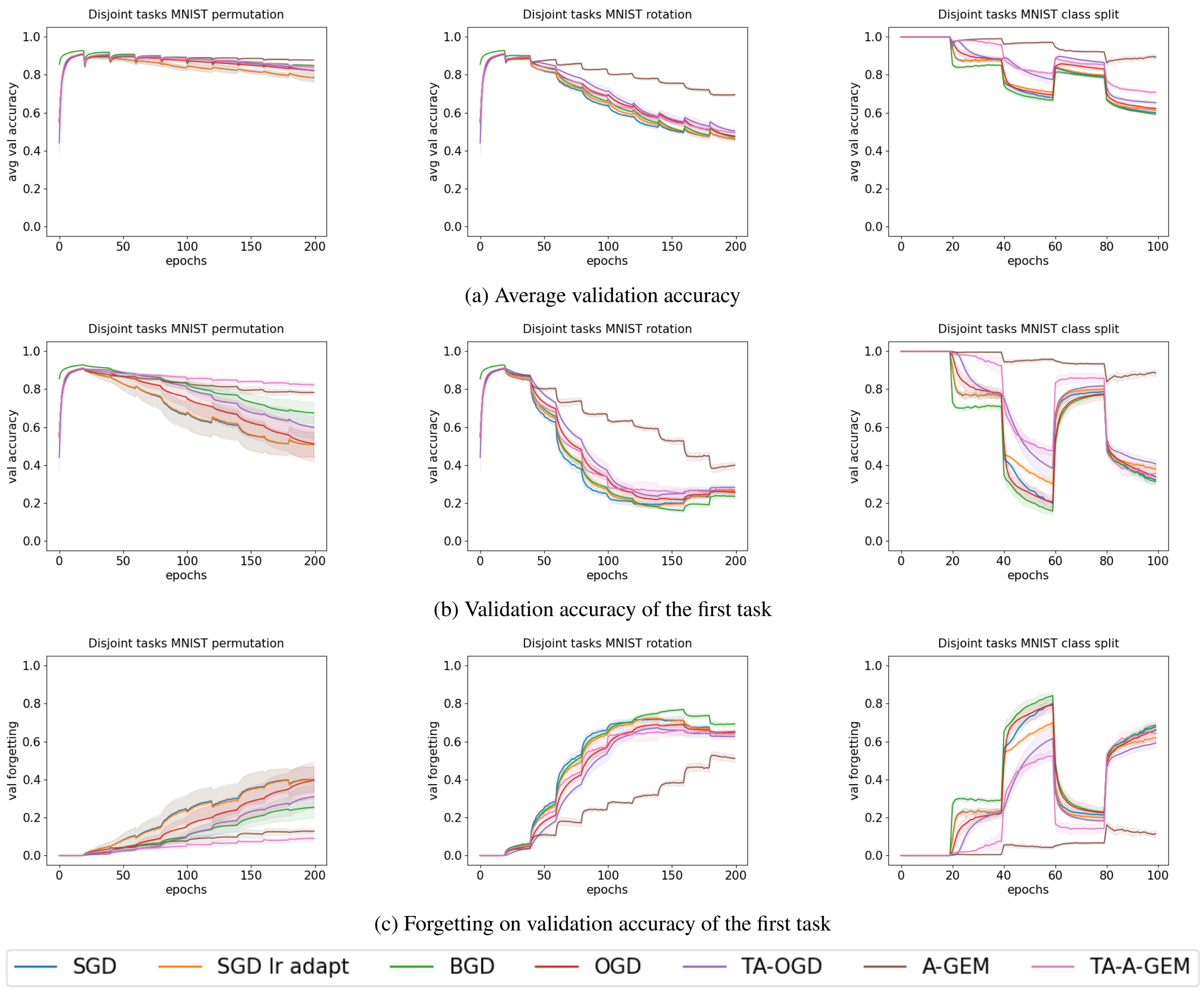
In the *Continuous change experiments*, tasks slowly change into the next task halfway through the training process. For the first 10 epochs (0-9), the network is trained on purely the current task. For epoch 10 to 19, for every batch, slowly more and more samples of the next task are mixed in. So the first batch of epoch 10 will contain 100% samples from the current tasks and 0% of the next task, while the last batch of epoch 19 will contain 0% samples of the current task and 100% samples of the next task. These percentages gradually change by linear interpolation. Table 3 shows the result of the *continuous change experiments*. In these experiments, tasks gradually change into the next task. This is accomplished by first loading two datasets A and B, where dataset A represent the current task and dataset B represents the next task. For each epoch, A is randomly shuffled and B is randomly shuffled. A fraction $q \in (0, 1)$ determines how many samples of A and how many samples of B should be mixed into one batch. A batch of size z is taken from both A and B. From the batch of A, the first $\lfloor q * z \rfloor$ samples are selected. From the batch of B, we select the last $\lceil (1 - q) * z \rceil$ samples. Since datasets A and B are shuffled every epoch, this is equivalent to random sampling without replacement. The selected samples are joined in a mixed batch. This mixed batch is then shuffled again, so no distinction between the tasks can be made. This mixed batch is then used for training.

For this experiment, training commences for each pair of datasets for 20 epochs. For the first 10 epochs (0-9), q is set to 1.0. This means that for these epochs, only the first of the datasets is presented to the learner. For the next 10 tasks (10-19), q gradually changes for each batch from 1.0 to 0.0. This means that for the first batch of epoch 10, q equals 1.0 and for the last batch of epoch 19, q equals 0.0. Between these batches, q changes linearly.

Figures 6, 7, 8, 9 and 10 show the average validation accuracy during training for different task split methods, using the MNIST, Fashion MNIST, NOT MNIST, CIFAR10 and SVHN datasets, for the "Continuous change" experiments. Table 4 and 5 summarize the results on the validation accuracy on the first task and the forgetting on the validation accuracy on the first task, respectively. Again, all results are the average of five independent runs.

G. Sampling Rate

For simplicity, we simply always add one sample per batch and set a maximum pool size. Once the pool is full, for every sample that is added, one is removed. If we remove one sample at random however, samples that are relevant to previous tasks quickly disappear from the pool. In



(a) Average validation accuracy

(b) Validation accuracy of the first task

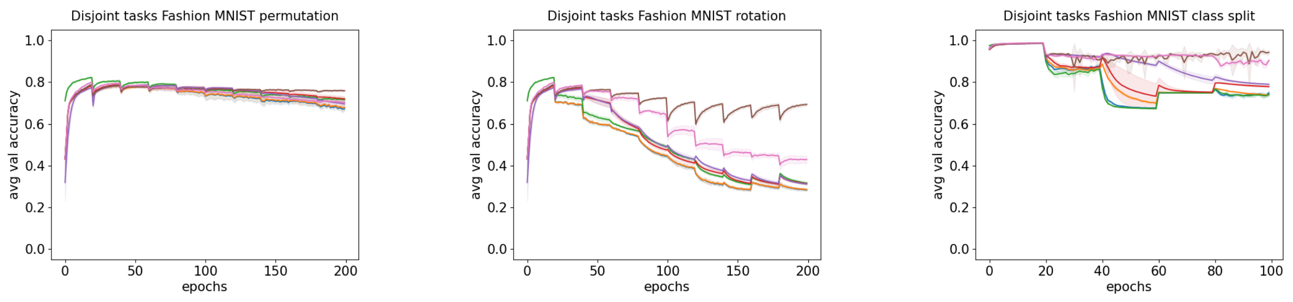
(c) Forgetting on validation accuracy of the first task

Figure 1: Results of the "**Disjoint tasks experiments**" on the **MNIST** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

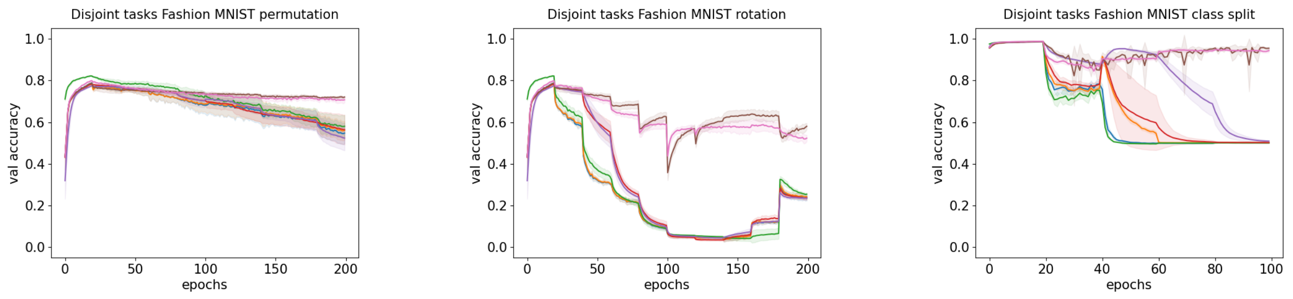
this case, the pool itself suffers from catastrophic forgetting. The easiest way to fix this is to lower the sampling rate, i.e. the frequency at which samples are added to the pool. In pre-experiments, we found out that setting the sampling rate to 0.01 gave optimal results in negating the forgetting. This effectively means that one sample was added to the pool every 100 batches. As it turned out, this gave the algorithm enough time to take a decent amount of samples for a task, while it did not take so many samples as to completely eliminate samples from older tasks from the pool. We then realized however, that this "optimal" sampling rate was completely dependent on the fact that every task was presented to the model for 20 epochs. It thus completely depends on the knowledge that a task will change after 20 epochs. In a truly task-agnostic setting, this information is not known. We therefore consider this method that solely relies on the "optimal" sampling rate to be a trivial case that is useless in a task-agnostic setting.

H. Clustering versus Random sampling

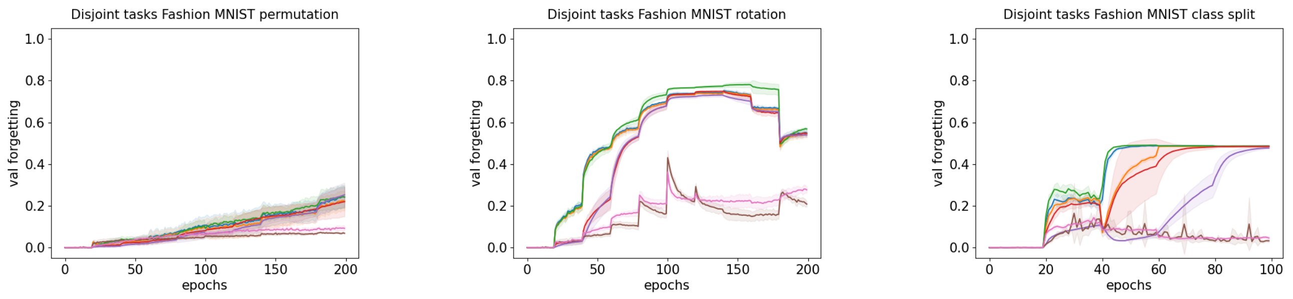
Figure 14 demonstrates that using clustering is a necessity. It shows that without the use of clustering (TA-A-



(a) Average validation accuracy



(b) Validation accuracy of the first task

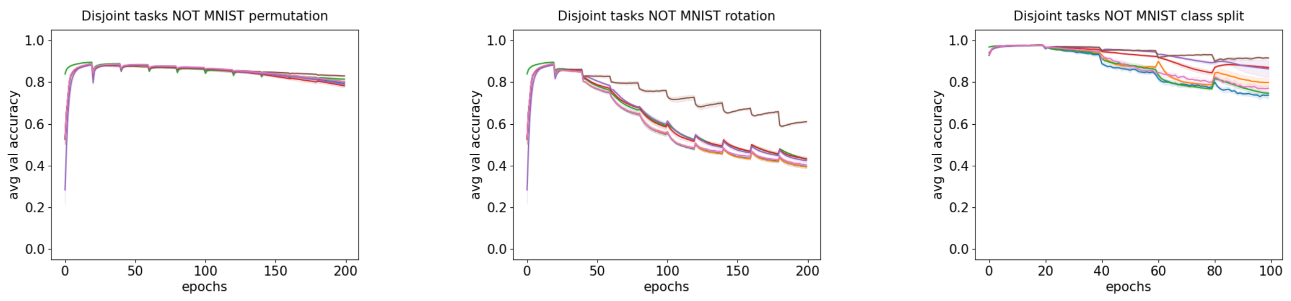


(c) Forgetting on validation accuracy of the first task

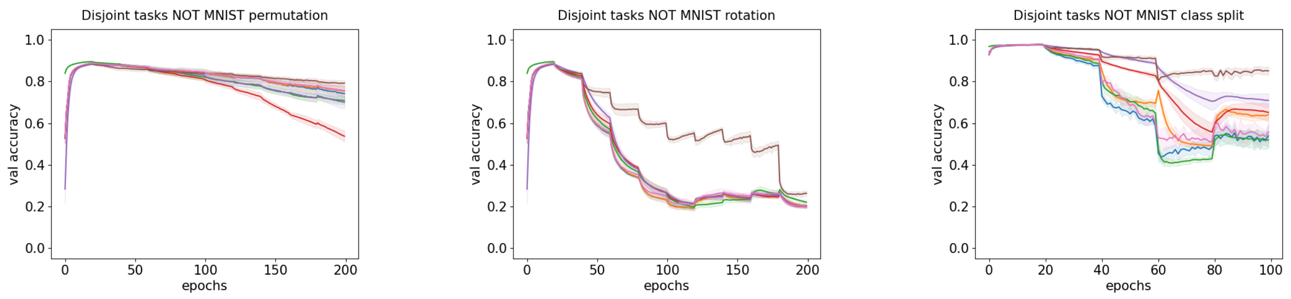
Figure 2: Results of the "**Disjoint tasks experiments**" on the **Fashion MNIST** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

GEM random), TA-A-GEM can only counteract the forgetting with an "optimal" sampling rate (sr) of 0.01, that is completely dependent on the frequency of task change; once every 20 epochs in this case. When clustering is introduced (TA-A-GEM), a wide variety of sampling rates, 0.01, 0.1 and 1 can effectively negate forgetting.

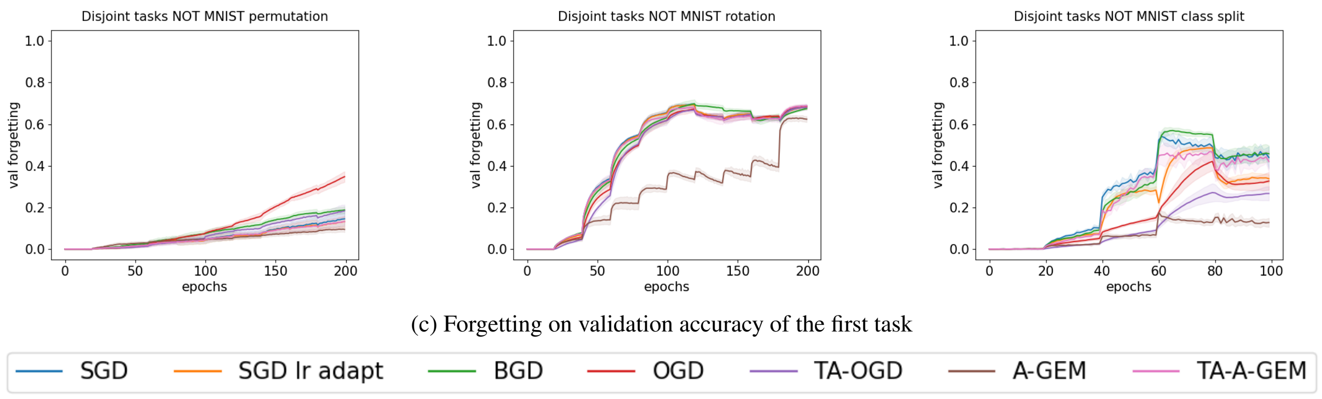
To further investigate the effect of clustering on the model gradient pool, the content of the pool is plotted over time for TA-A-GEM, that uses clustering and TA-A-GEM with random cluster assignment. In the latter, a newly sampled data point is added to a randomly selected cluster instead of being added to the cluster that has its mean closest to it (as is the case in the standard clustering-based TA-A-GEM approach). We test the performance of the two methods for three different values of the sampling rate i.e., 0.01, 0.1 and 1. Figures 11, 12 and 13 show the content of the gradient pool for TA-A-GEM (left) and TA-A-GEM with random cluster assignment (right), for a sampling rate of 0.01, 0.1 and 1. Looking at the cluster contents of TA-A-GEM, it can be seen that the amount of task variety is highly affected by the sampling rate. The sampling rate can therefore be tweaked to an optimal value, in this case 0.01, given



(a) Average validation accuracy



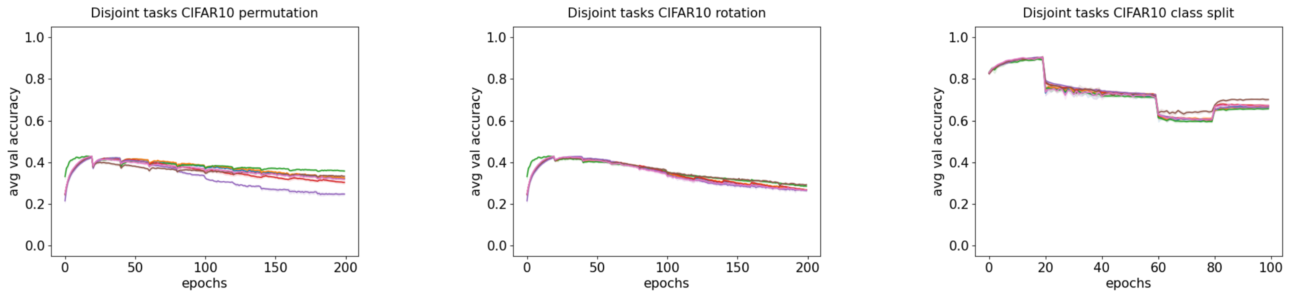
(b) Validation accuracy of the first task



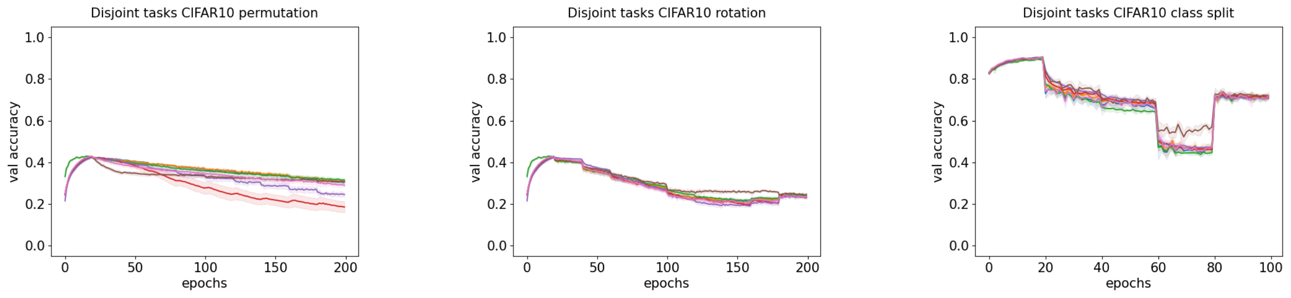
(c) Forgetting on validation accuracy of the first task

Figure 3: Results of the "**Disjoint tasks experiments**" on the **NOT MNIST** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

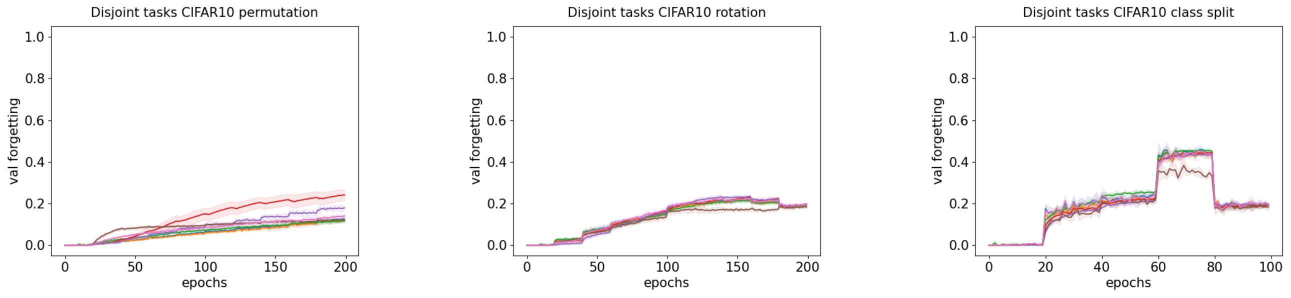
that a new task arrives every 20 epochs. This is a trivial case that we consider not to be truly task-agnostic, since it is tweaked on 20 epochs. By using clustering, TA-A-GEM manages to keep a wide variety of task information in the pool for all sampling rates. It therefore improves on the trivial case, which arises by fine-tuning the sampling rate.



(a) Average validation accuracy



(b) Validation accuracy of the first task



(c) Forgetting on validation accuracy of the first task

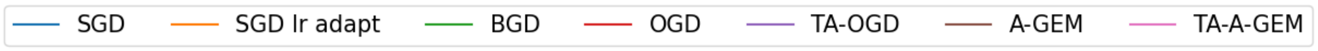
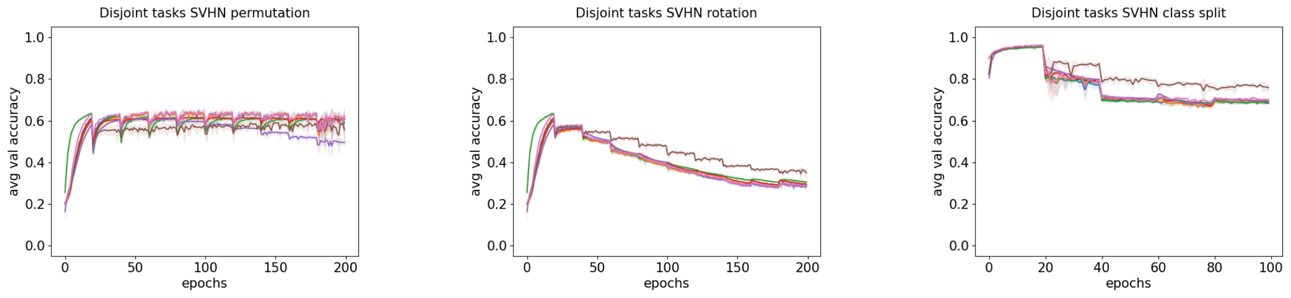
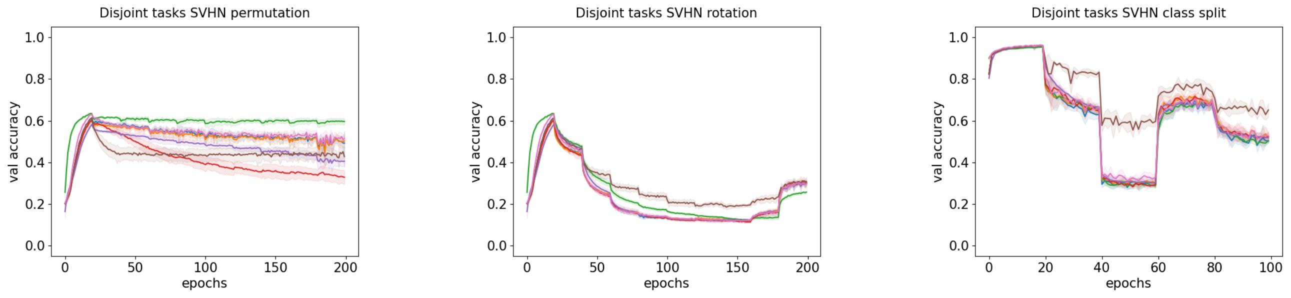


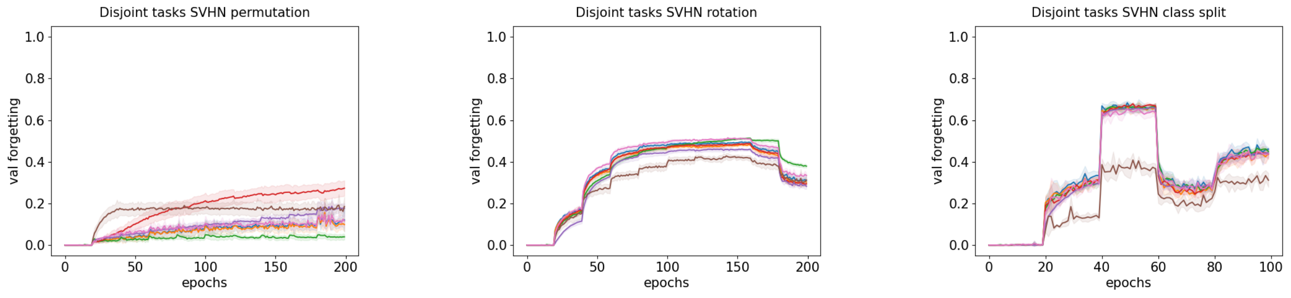
Figure 4: Results of the "**Disjoint tasks experiments**" on the **CIFAR10** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.



(a) Average validation accuracy



(b) Validation accuracy of the first task



(c) Forgetting on validation accuracy of the first task

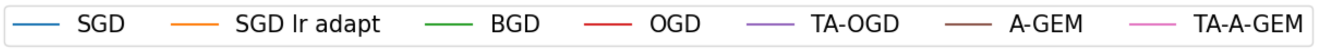


Figure 5: Results of the "**Disjoint tasks experiments**" on the **SVHN** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

	MNIST			Fashion MNIST			NOT MNIST			CIFAR10			SVHN		
	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class
SGD	0.691	0.419	0.643	0.682	0.254	0.657	**0.825**	0.408	0.707	**0.364**	0.288	0.692	0.527	0.231	0.622
SGD lr adapt	0.693	0.429	0.669	0.686	0.255	0.684	**0.827**	0.408	0.761	**0.365**	0.288	**0.699**	0.520	0.231	**0.635**
BGD	0.812	0.431	0.606	**0.716**	0.270	0.646	**0.813**	**0.425**	0.711	**0.365**	**0.297**	0.682	**0.596**	**0.261**	0.623
TA-OGD	0.779	**0.487**	**0.721**	0.686	0.302	0.831	0.804	**0.429**	**0.866**	0.332	0.283	**0.708**	0.473	0.235	**0.637**
TA-A-GEM	**0.858**	**0.465**	**0.748**	**0.734**	**0.625**	**0.931**	**0.830**	0.415	0.738	0.347	0.285	0.695	0.536	0.236	**0.640**
OGD	0.737	0.461	0.645	0.688	0.306	0.699	0.766	0.426	0.817	0.293	0.285	0.703	0.414	0.229	0.634
A-GEM	0.830	0.654	0.95	0.735	0.640	0.934	0.834	0.605	0.906	0.338	0.308	0.725	0.443	0.292	0.757

Table 1: **Validation accuracy** of the first task, averaged over all epochs, then averaged over five runs, for the **disjoint tasks experiments** when using a MLP. Per column, the best result for the *task-agnostic* methods are written in bold. In case a task-agnostic method's result is less optimal and not significantly different from the best result, with a confidence of 99%, it is also written in bold. The results for the *task-aware* methods OGD and A-GEM are given for context. Since these algorithms benefit from knowing task identities and changes, we just use them here as baselines for indicating the best performance we can achieve.

	MNIST			Fashion MNIST			NOT MNIST			CIFAR10			SVHN		
	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class
SGD	0.213	0.485	0.356	0.099	0.527	0.329	**0.056**	0.473	0.270	**0.060**	**0.136**	0.209	0.068	0.364	0.337
SGD lr adapt	0.209	0.473	0.330	**0.091**	0.521	0.302	**0.053**	0.471	0.215	**0.056**	**0.134**	0.201	0.064	0.353	**0.322**
BGD	0.115	0.496	0.393	0.104	0.550	0.340	0.081	0.469	0.266	**0.065**	**0.133**	0.212	**0.032**	0.367	**0.328**
TA-OGD	0.123	**0.415**	**0.278**	**0.081**	0.464	0.154	**0.070**	**0.444**	**0.109**	0.086	**0.135**	**0.193**	0.091	**0.330**	**0.319**
TA-A-GEM	**0.049**	0.442	**0.251**	**0.058**	**0.167**	**0.057**	**0.053**	0.468	0.239	0.078	**0.139**	0.207	0.078	0.378	**0.322**
OGD	0.165	0.441	0.354	0.089	0.470	0.286	0.113	0.453	0.160	0.128	0.137	0.198	0.170	0.354	0.323
A-GEM	0.074	0.250	0.049	0.046	0.142	0.052	0.048	0.276	0.072	0.086	0.116	0.176	0.153	0.303	0.202

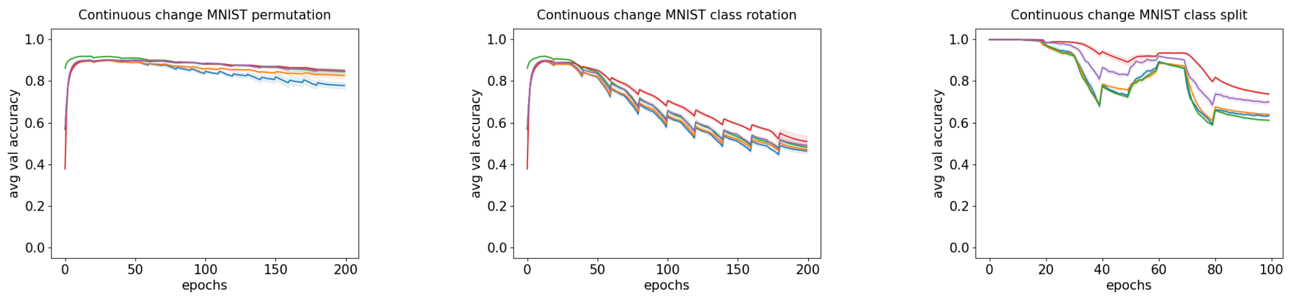
Table 2: **Forgetting** on validation accuracy of the first task, averaged over five runs, then averaged over all epochs, for the **disjoint tasks experiments** when using a MLP. Per column, the best result for the *task-agnostic* methods are written in bold. In case a task-agnostic method's result is less optimal and not significantly different from the best result, with a confidence of 99%, it is also written in bold. The results for the *task-aware* methods OGD and A-GEM are given for context. Since these algorithms benefit from knowing task identities and changes, we just use them here as baselines for indicating the best performance we can achieve.

	MNIST			Fashion MNIST			NOT MNIST			CIFAR10			SVHN		
	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class
SGD	0.841	0.647	0.816	0.738	0.471	0.814	0.849	0.588	0.866	0.378	0.348	0.728	0.597	0.396	0.755
SGD lr adapt	0.859	0.654	0.819	0.748	0.487	0.821	**0.854**	0.592	0.893	0.380	0.349	0.731	0.592	0.394	0.772
BGD	**0.884**	0.678	0.808	**0.765**	0.515	0.812	**0.860**	**0.621**	0.865	**0.390**	**0.358**	0.719	**0.605**	**0.419**	0.760
TA-OGD	0.875	**0.706**	**0.912**	0.748	0.522	0.902	0.846	**0.622**	**0.937**	0.335	0.343	**0.735**	0.567	0.392	**0.792**
TA-A-GEM	0.876	0.670	0.87	0.745	**0.594**	**0.929**	**0.850**	0.592	0.876	0.369	0.346	0.715	**0.606**	0.403	0.765

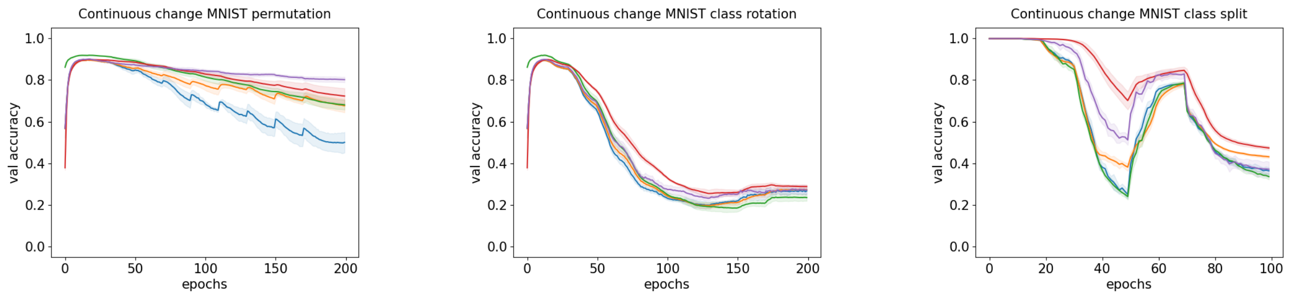
Table 3: **Average validation accuracy**, averaged over all tasks trained thus far, then averaged over all epochs, then averaged over five runs, for the **continuous change experiments** when using a MLP. Per column, the best result is written in bold. In case a result is less optimal and not significantly different from the best result, with a confidence of 99%, it is also written in bold.

	MNIST			Fashion MNIST			NOT MNIST			CIFAR10			SVHN		
	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class
SGD	0.702	0.414	0.657	0.659	0.267	0.665	**0.810**	0.395	0.694	**0.359**	0.289	0.707	0.490	0.230	0.626
SGD lr adapt	0.785	0.420	0.672	0.693	0.278	0.684	**0.825**	0.404	0.769	**0.361**	0.289	**0.717**	0.479	0.228	0.658
BGD	0.812	0.426	0.645	**0.702**	0.297	0.663	**0.804**	0.409	0.675	**0.366**	**0.299**	0.695	**0.588**	**0.264**	0.636
TA-OGD	0.819	**0.476**	**0.803**	0.682	0.317	0.842	0.789	**0.422**	**0.851**	0.325	0.284	**0.719**	0.457	0.228	**0.681**
TA-A-GEM	**0.842**	0.442	0.726	**0.729**	**0.600**	**0.926**	**0.813**	0.398	0.712	0.331	0.286	0.704	0.509	0.233	0.634

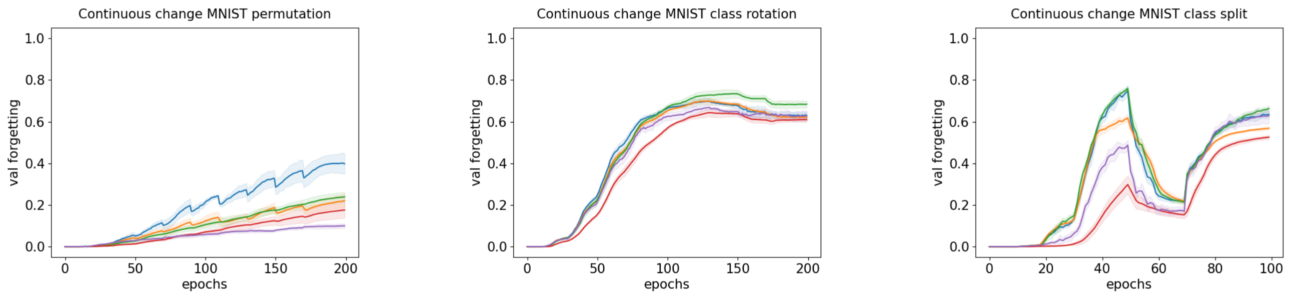
Table 4: **Validation accuracy** of the first task, averaged over all epochs, then averaged over five runs, for the **continuous change experiments** when using a MLP. Per column, the best result is written in bold. In case a result is less optimal and not significantly different from the best result, with a confidence of 99%, it is also written in bold.



(a) Average validation accuracy



(b) Validation accuracy of the first task



(c) Forgetting on validation accuracy of the first task

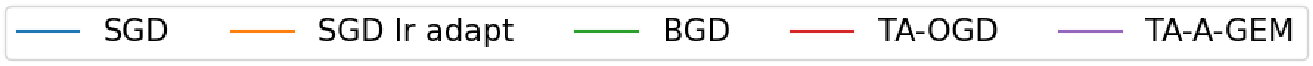
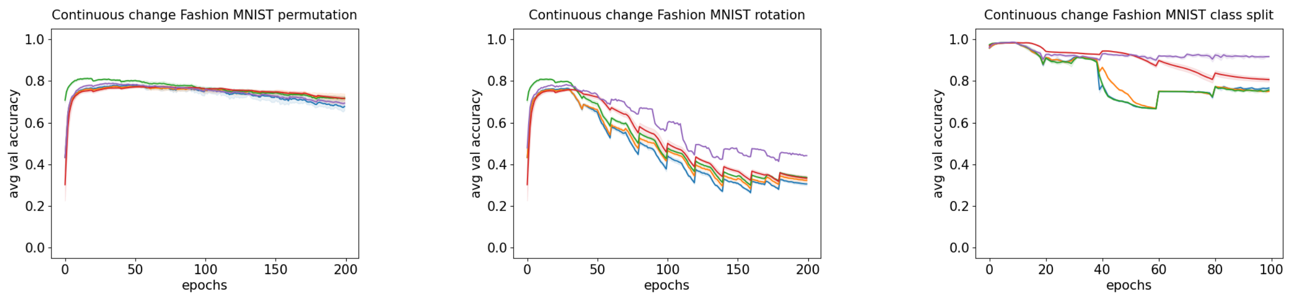


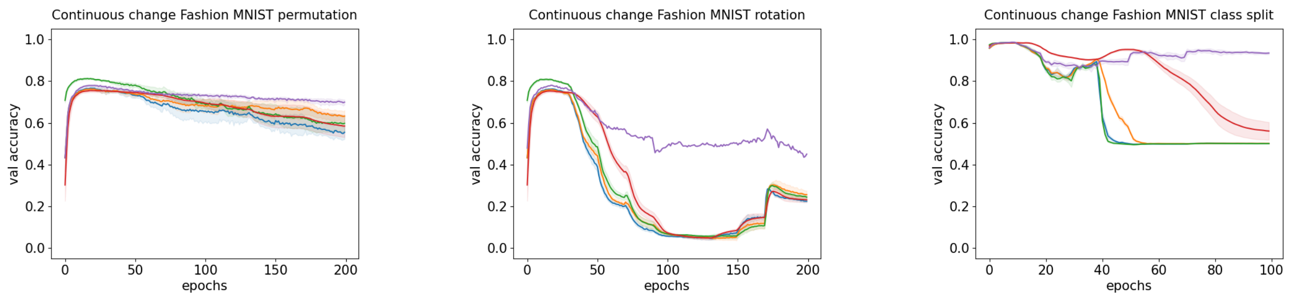
Figure 6: Results of the "**Continuous change experiments**" on the **MNIST** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with ± one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

	MNIST			Fashion MNIST			NOT MNIST			CIFAR10			SVHN		
	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class	perm	rot	class
SGD	0.192	0.478	0.342	0.104	0.492	0.319	**0.066**	0.475	0.281	**0.046**	**0.127**	0.185	0.057	0.312	0.325
SGD lr adapt	0.107	0.470	0.327	**0.067**	0.478	0.300	**0.049**	0.464	0.207	**0.041**	**0.124**	0.174	0.053	0.299	0.290
BGD	0.106	0.492	0.355	0.110	0.512	0.323	0.088	0.480	0.302	0.058	**0.132**	0.190	**0.027**	0.351	0.310
TA-OGD	**0.073**	**0.416**	**0.196**	**0.067**	0.429	0.142	0.077	**0.441**	**0.124**	0.069	**0.127**	0.171	0.050	**0.283**	**0.268**
TA-A-GEM	**0.055**	0.454	0.273	**0.047**	**0.172**	**0.061**	**0.064**	0.473	0.265	0.078	**0.131**	**0.147**	0.064	0.335	0.322

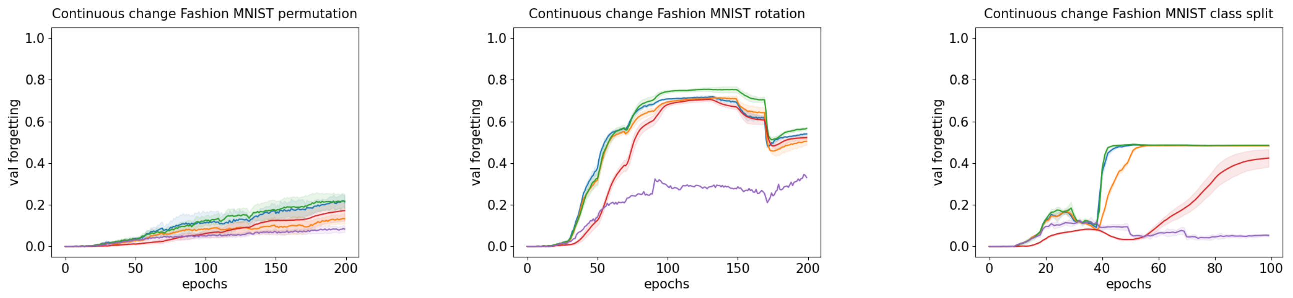
Table 5: **Forgetting** on validation accuracy of the first task, averaged over five runs, then averaged over all epochs, for the **continuous change experiments** when using a MLP. Per column, the best result is written in bold. In case a result is less optimal and not significantly different from the best result, with a confidence of 99%, it is also written in bold.



(a) Average validation accuracy



(b) Validation accuracy of the first task



(c) Forgetting on validation accuracy of the first task

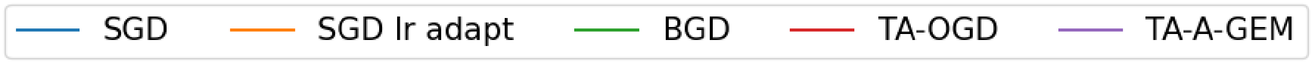
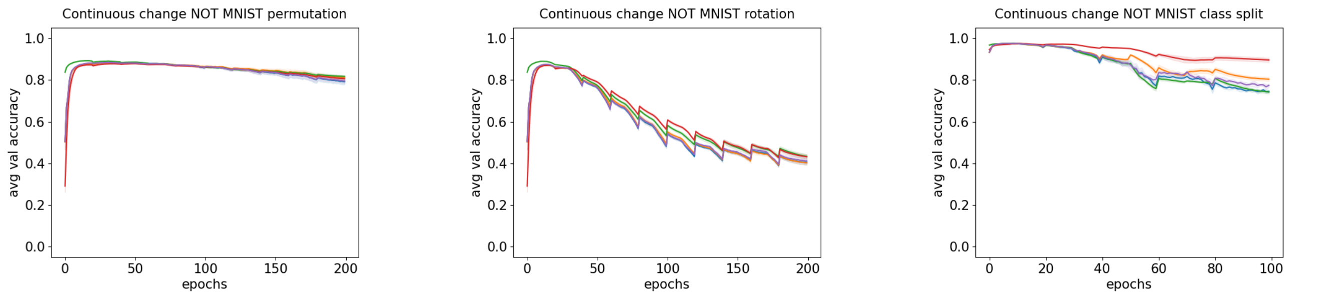


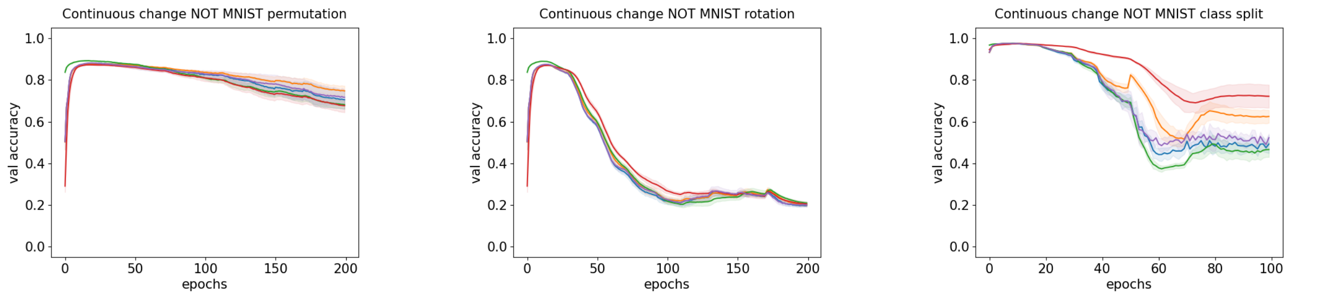
Figure 7: Results of the “**Continuous change experiments**” on the **Fashion MNIST** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with ± one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

Task generation	Mechanism	Parameters
Permutation	Number of tasks	10
Rotation	Rotation angles	[0, 20, 40, 60, 80, 100, 120, 140, 160, 180]
Class split	Subset labels	[[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]]

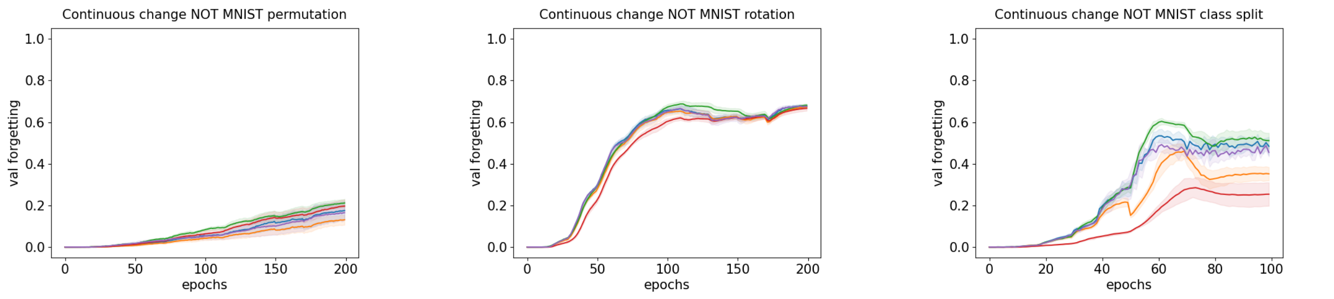
Table 6: Summary of the settings used for the task split for the “Disjoint tasks” experiment and the “Continuous change” experiment



(a) Average validation accuracy



(b) Validation accuracy of the first task

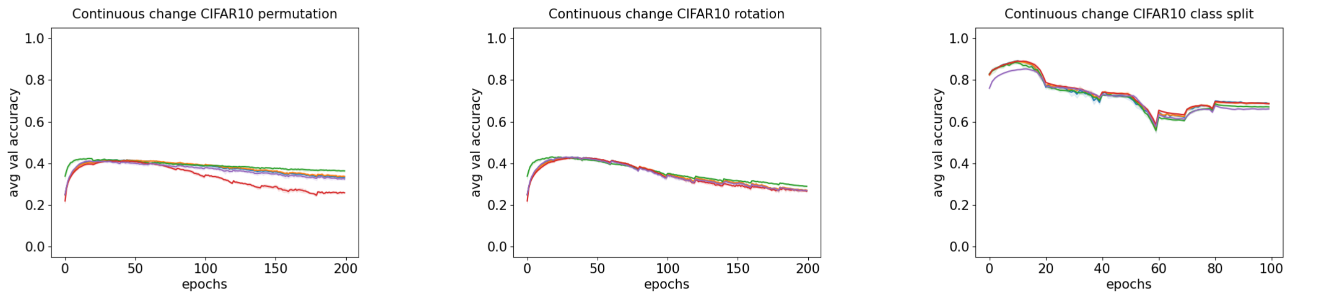


(c) Forgetting on validation accuracy of the first task

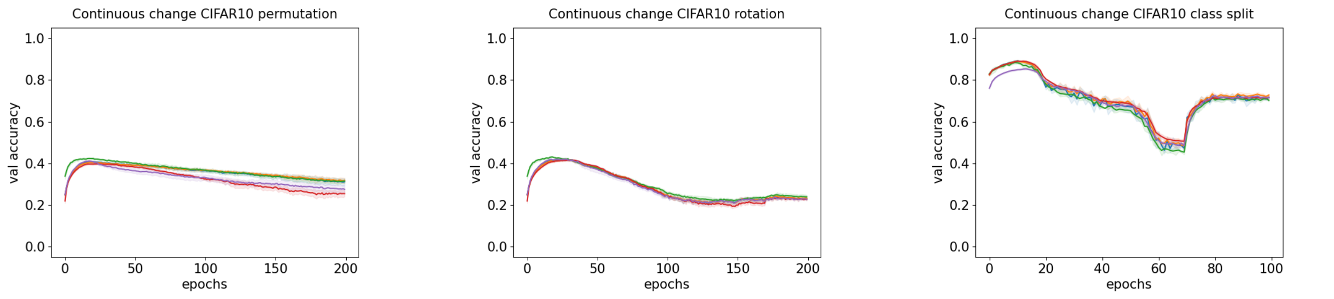
Figure 8: Results of the "**Continuous change experiments**" on the **NOT MNIST** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

Parameter name	Value	Explanation
Learning rate	10^{-3}	The same as for TA-OGD
Batch size	10	The same batch size as TA-OGD is chosen.
Sampling rate	1	The number of labeled data points sampled per batch, so in this case, one labeled data point is sampled every batch.
Number of pools	10 or 2	The number of separate pools; one for each class, so 10 classes for permutation and rotation and 2 for class split.
Number of clusters	10 or 50	Each pool contains 10 clusters in case there are 10 classes, or 50 clusters in case there are 2 classes.
Cluster size	3	The maximum number of sampled data points per cluster, if this number is exceeded, the oldest sample is removed.

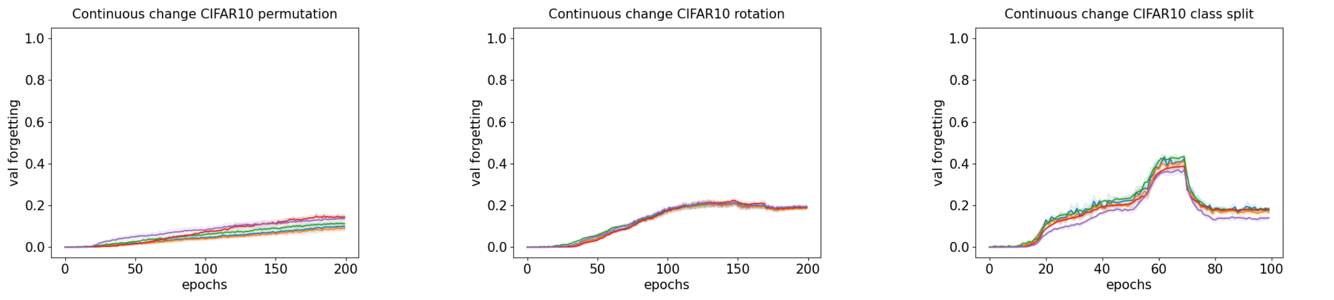
Table 7: TA-A-GEM uses these parameters. Their values were hand picked to maximize the average accuracy.



(a) Average validation accuracy



(b) Validation accuracy of the first task



(c) Forgetting on validation accuracy of the first task

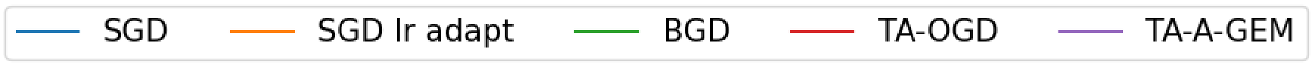
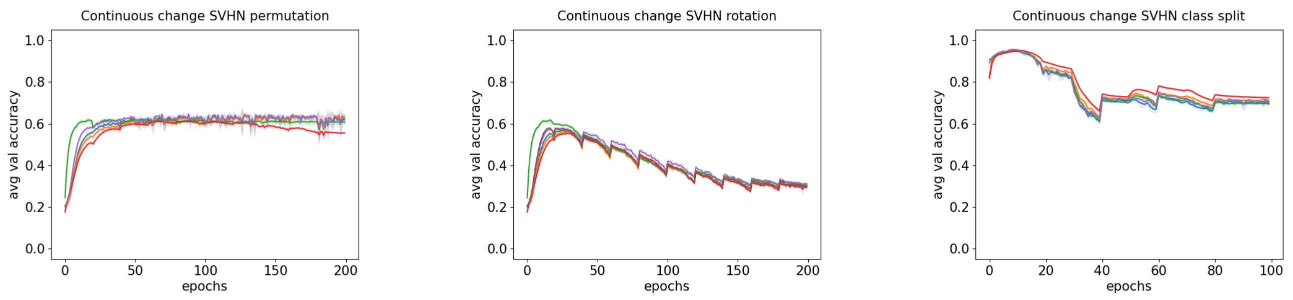


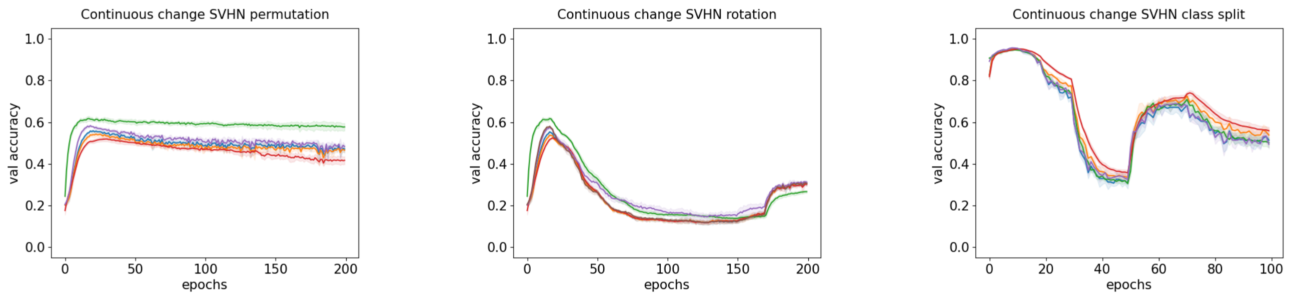
Figure 9: Results of the "**Continuous change experiments**" on the **CIFAR10** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

Parameter name	Value	Explanation
Learning rate	10^{-3}	The same as in the OGD paper of Farajtabar et al. [2]
Batch size	10	TA-OGD needs small batch sizes. Larger batch sizes will not yield good results.
Sampling rate	1	The number of gradients sampled per batch, so in this case, one gradient is sampled every batch.
Number of pools	1	TA-OGD uses one pool for all sampled model gradients.
Number of clusters	99	The number of distinct model gradient clusters.
Cluster size	3	The maximum number of model gradients per cluster, if this number is exceeded, the oldest model gradient is removed.

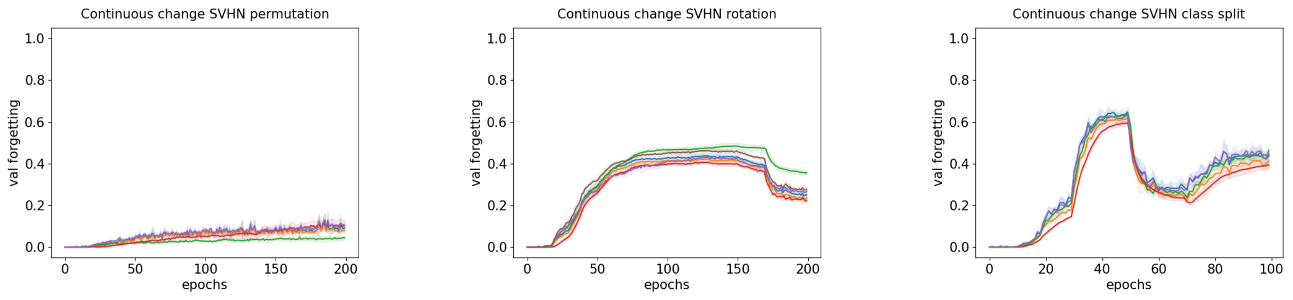
Table 8: TA-OGD uses these parameters. Their values were hand picked to maximize the average accuracy.



(a) Average validation accuracy



(b) Validation accuracy of the first task



(c) Forgetting on validation accuracy of the first task

SGD
SGD lr adapt
BGD
TA-OGD
TA-A-GEM

Figure 10: Results of the "**Continuous change experiments**" on the **SVHN** dataset. (a) Average validation accuracy, (b) Validation accuracy of the first task, (c) Forgetting on validation accuracy of the first task, averaged over all tasks trained thus far, then averaged over five runs, depicted by a solid line plot with $\pm$ one standard deviation as a shaded area. From left to right: task separation by permutation, task separation by rotation and task separation by class split.

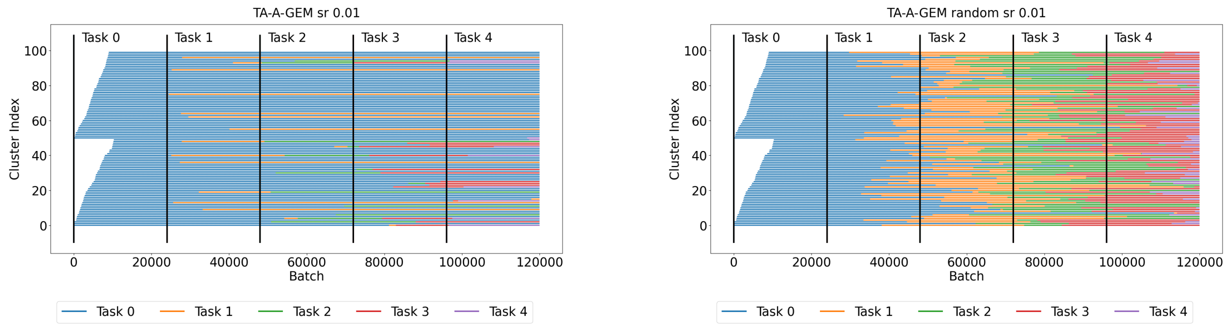


Figure 11: The horizontal lines depict the content of clusters when using class split as a task segmentation method on Fashion MNIST with a sampling rate of 0.01. Each task is associated with a unique color. The color of the vertical lines represents the oldest task information that is present in the cluster. The moment that a new task starts is indicated by a black vertical line. The clusters with index 0 to 49 correspond to the pool of class label 0. The clusters with index 50 to 99 correspond to the pool of class label 1. Left: Clustering helps in keeping a greater variety of task information in the gradient pool, but this effect is hardly noticable due to the optimal sampling rate being used. Right: Using random cluster assignment can result in a good amount of variety, but it is completely dependent on the optimal sampling rate of 0.01. but it is still just as effective at counteracting forgetting, as compared to clustering. This is only ideal when every task presents itself for exactly 20 epochs each time. For this, the frequency of task switching need to be known, thereby destroying the task-agnostic nature of the method.

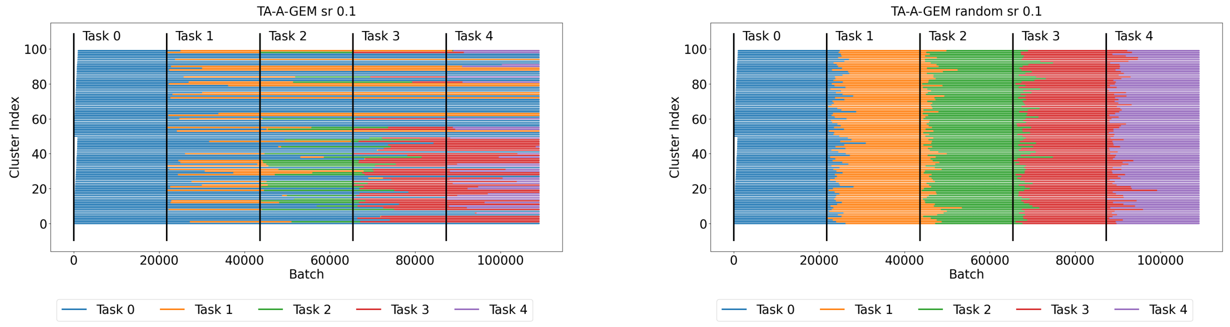


Figure 12: The content of clusters using when using class split as a task segmentation method on Fashion MNIST with a sampling rate of 0.1. Left: The use of clustering assures a good variety of task information is kept in the pool. Right: Information from previous tasks is quickly lost with this sampling rate setting.

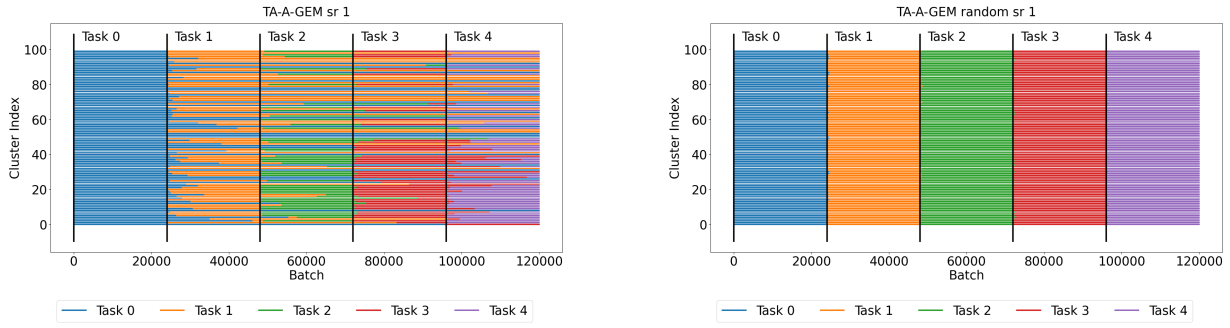


Figure 13: The content of clusters when using class split as a task segmentation method on Fashion MNIST with a sampling rate of 1. Left: Even with a high sampling rate, the clustering assures a good variety of task information in the pool. Right: Information from previous tasks is immediately lost after the moment of task change.

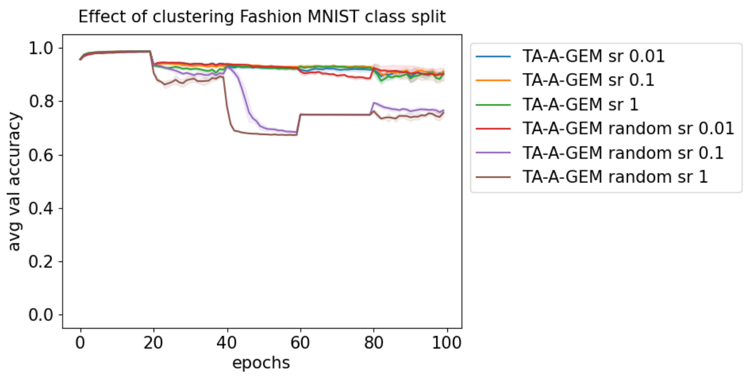


Figure 14: TA-A-GEM without its clustering mechanism (TA-A-GEM random) can only successfully negate forgetting with an “optimal” sampling rate (sr) of 0.01, that can only be known if the task change frequency is known, thus undoing its task-agnostic nature. On the other hand, when clustering is introduced to TA-A-GEM (TA-A-GEM), a wide variety of sampling rates (sr) (0.01, 0.1 and 1) can effectively reduce the amount of forgetting.

References

[1] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. *arXiv preprint arXiv:1812.00420*, 2018. 1

[2] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, volume 108, pages 3762–3773. PMLR, 26–28 Aug 2020. 2, 14

[3] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018. 3