Preventing distribution divergence in generative replay using unsimilarity in CURL

Tobi Carvalho
Department of Computer Science and Operations Research
University of Montreal
Montreal, Quebec H3T 1J4
tobi.carvalho@gmail.com

Abstract

Continual learning aim to overcome changes in tasks distributions overtime without catastrophically forgetting previous learned knowledge. Generative replay (GR) is a commonly used strategy to overcome this challenge. However, GR depend on the performance of a generator. Any difference between the generator distribution and the true distribution is at risk to propagate at each training step, leading to a divergence between the two distributions and thus catastrophic forgetting. In this work, we attempt to prevent this using selective sampling in order to build a replay buffer based on un-similarity between samples. We investigated un-similarity in the sample space and the latent space and based our work on the CURL algorithm[1]. We found that using a replay-buffer seems to help with catastrophic forgetting, even when composed of less then 300 samples for a dataset of 5 tasks with 5000 samples each. Specially when the selection of candidates is done from the poor sample buffer, right after the generation of a new expert. We also found out that un-similarity in the latent space seem to be more effective then in the sample space as a selection criteria. While we found that selective sampling using un-similarity could provide task diversity, we couldn't conclude whether it was more effective then using random sampling, making it unclear if it is really useful.

1 Introduction

Machine learning is a growing field where the boundary of what is doable and the performance at which it is done keep getting pushed. However, most of the currently used algorithms are restricted to IID setting and perform poorly otherwise. What's more, most of these algorithm can't retain previously learned knowledge when trained on a new task. This phenomena is called catastrophic forgetting in the field of continual learning.

One simple strategy that have been shown to be effective is experience replay. This strategy consist of storing previously seen sample into a replay buffer in order to play those back later on. However, this strategy can be memory expensive, especially as the number of tasks goes up and where the total number of samples required for each task to be well represented increases.

Leveraging the ability of current generative model to reproduce sample from complex distribution, a similar technique have been developed to solve the limiting cost of replay buffer. This technique is called generative replay and consist of using a generative algorithm to generate, on the go, samples from previous task distributions instead of storing them. Doing so, a virtually infinite number of samples can be created at a reduced memory cost. However, generative replay rely heavily on the accuracy of the generator since it's retrain on the sample it generated. Because of this, any difference between the real distribution and the generator distribution is likely to propagate with training, leading to divergence between them and to catastrophic forgetting.

Preprint. Under review.

Since in this case the forgetting happens because of the propagation of difference between the real and the learned distributions, we investigate in this paper if it is possible to link this generator to a replay buffer of real samples in order to prevent the divergence. When it could seem counter productive to use a replay buffer since GR aim to replace these, our aim is to sustain a replay buffer smaller then what would normally be needed in order to do effective experience replay.

1.1 Continual unsupervised representation learning

Our work has been done in the framework of continual unsupervised representation learning (CURL). CURL is a mixture of experts algorithm based on the concept of variational auto-encoder (VAE)[1]. It is a model that aim to learn to do representational learning by doing images reconstruction. An illustration of it can be seen in figure 1. It is composed of mainly four components. The first one is the shared encoder, which is responsible to encode the input to a representational space ($q(h|x)$) and to predict to which expert the current sample belongs ($q(y|x)$). The second part is the mixture of latent encoders (the experts). They are responsible to output the parameters of the distribution from which the latent vector z is gonna be sampled ($q(z|x, y = k)$). The third one is the latent decoder. It is only used during training since it is used as a regularization that restricts each expert to a point in the parameter space ($p(z|y)$). Finally the fourth component is the shared decoder($p(x|z)$) which is responsible to reconstruct the input image from the latent vector z.

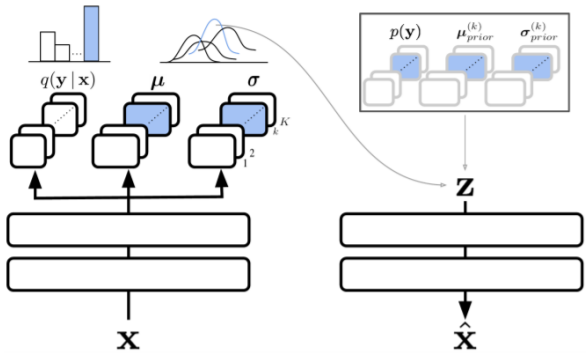

Figure 1: Curl architecture[1]

In their paper, [1] defined two loss functions, one for unsupervised training and another one for supervised training. The unsupervised loss function is the one that is used by default to train the algorithm. It is also used as a criteria for the expansion of experts. Whenever the loss of a sample exceed a defined threshold, this sample is added to a "poor" buffer. When the number of samples exceed a certain limit, a new expert is created and trained on these using the supervised loss function.

Because of its VAE nature and its expansion mechanism, we believe that the CURL algorithm is the perfect framework to investigate the use of selective sampling using un-similarity. We also believe that our approach could be generalized to any type of generative replay and that working under agnostic's setting only adds a constraint making the task more difficult. Any algorithm using un-agnostic settings could use the task labels instead of using CURL in order to detect a change of task.

1.2 Selective sampling using unsimilarity

We believe that the samples taken from the replay buffer will act as thumbtacks that will fix and prevent the generator distribution to drift over time. In this sense, spreading those thumbtacks should lower the risk of overfitting since the other samples that are in the center of the distribution will naturally be favoured by the generative replay. We believe that using selective sampling based on un-similarity could not only favor task diversity but also inner-task diversity.

In this work, we investigate un-similarity in two spaces, the sample space and the latent space of the shared encoder. We believed that both spaces have their advantages over the other and that a mix of both might be the best approach.

1.2.1 Similarity in the sample space

While pixels to pixels similarity in the sample space seems an obvious choice, we believe it does have major drawbacks that are worth investigating. While it is often the metric use for auto-encoders, it suffers from a lack of representation, which makes it susceptible to simple transformations as basic as translations. We can see in figure 2 an example of pixels translation. While it is obviously the same number, pixels to pixels difference would be big because of the lack of overlap between the activated pixels.

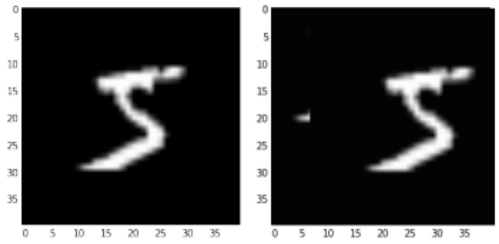

Figure 2: Pixels translation (image taken from [2]).

However, in our work, pixels to pixels should still favor task diversity, since we are working with the MNIST dataset and thus the samples are usually centered. Despite this disadvantage, pixels to pixels comparison has the advantage of being invariant to the algorithm training, which can lower the risk of catastrophic forgetting.

1.2.2 Similarity in the latent space

The second type of similarity we considered was the similarity in the latent space. Since the objective of the CURL algorithm is to do representative learning, we believe that this space could be effectively use for selective sampling since sample that are more distant in this space, are more likely to be from distinct tasks.What's more, considering that the experts are centered in the latent space by the KL-divergence, we believe that selecting sample at the edge of the latent space will favor the preservation of the whole sample distribution since the other samples will naturally be favored by the generative replay, see figure 3. We consider that using only the latent space might be dangerous since it's dynamic, making it prone to catastrophic forgetting.

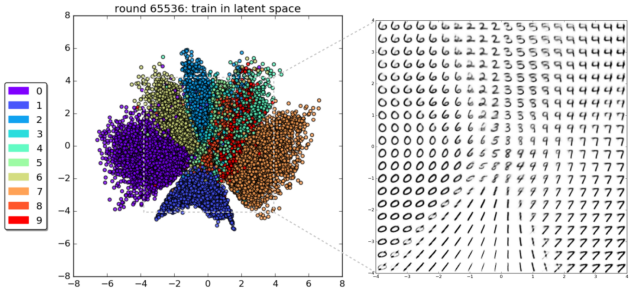

Figure 3: VAE latent space visualization (image taken from [3]).

However, we just realized that we might have investigated the wrong latent space since there is two principal latent spaces. There is the latent space of the shared encoder (h) and the one before the shared decoder (z). In this work, we have investigated the similarity in the latent space h. However, the clustering of the samples is enforced in the z space, not the h space. While the transformation from the h space to the z space is mostly linear (the transformation to the mean parameter of the distribution is), there is no guaranty that maximization of the un-similarity in the h space will yield the same results as doing it in the z space. The h space is still a representational space, making it worthy of investigation.

2 Experiments

2.1 Dataset

For these experiments, we splited the MNIST dataset into different tasks for each classes. We limited ourselves to only the first 5 classes for computational power reason. We trained our algorithms in a sequential manner by training for 5 epochs for each task before going to the next one. The frontier between the tasks where then clear, but not given to the algorithm. Since the number of experts can be interpreted as the number of tasks seen by the algorithm, each batch was repeated n+1 time, where n is number of experts. For each of these repetition, $|X|$ sample where randomly selected from the set

$$\{X, RB\left[\lambda n_e |X|\right], GR\left[(1-\lambda) n_e |X|\right]\} \tag{1}$$

where X is the current batch, $|X|$ the size of the batch, RB samples from the replay buffer, GR samples generated and λ a selection coefficient between RB and GR. Doing so, the importance given to each task is the same and the probability that a given sample in the batch is seen is still 1. This way of creating batch of data is different then the one used by [1]. In their paper, they do not mix the real data with the generated one. Making this work, we have found however that mixing them seem to be more stable as the parameters' update during a real batch seems to be detrimental to the generation of previously seen samples. They also trained for many more time steps per class than us, which could explained why this wasn't affecting them as much, as we also observed that using more time steps seemed to give better results. Our choice of using less time steps was motivated by a lack of computational power.

Otherwise, except the use of a replay buffer and the threshold that was set to 240 instead 200, all the other hyper-parameters/architecture features where the same as in the CURL paper[1] and we would then refer to it for other details.

2.2 Selective sampling criteria

In order to investigate the use of un-similarity between samples, the following criteria has been used

$$S_j = \alpha * \frac{||x_j - \mu_X||^2}{\sum_i ||x_i - \mu_X||^2} + (1-\alpha) * \frac{||h_j - \mu_H||^2}{\sum_i ||h_i - \mu_H||^2} \tag{2}$$

where S is the un-similarity measure, x the data in the sample space, h the data in the latent space. The capital letters are their equivalents over the whole set on which the similarity is evaluated and α the coefficient of importance between the sample space and the latent space. Each element of the similarity measure is normalized so that the importance between the two space is only parametrized by α.

2.3 Selective sampling mechanism

In this work, we tested two strategies to add samples to the replay buffer. The first consisted of selecting samples from the poor replay buffer right after an expansion of experts happened. Since this is suppose to happen when there is a shift in the input distribution, adding samples at this moment should ensure that the buffer is multi and inner task diversified. The reason we added the samples after the training of the new expert is to allow the algorithm to learn a useful representation of the new task before. At each expansion, a total of 30 samples were added to the replay buffer, which correspond to 30% of the poor replay.

The second mechanism we investigated was to allow the selection of samples at the last epoch of each task. Doing so, we once again allow for the algorithm to learn good representation of each task before selecting samples based on the representation space. It is important to notice that this technique is not task agnostic at training anymore since it requires knowledge of when the last epoch of each task happened. Under this mechanism, we tested two ways of scaling the number of samples in the replay buffer. The first one scaled with the number of experts and was reevaluating the whole replay buffer. The maximal number of samples allowed in the replay buffer was equal to 30 time the number of experts, being equivalent to the limit of the first sampling mechanism (at expansion).

The other scaling was simply to take only 5 samples from each batch, regardless of the number of samples.

We tested each of these experiments using the sampling criteria defined at equation 2 with α equal to 0, 0.5 and 1. To compare these criteria, we also used random sampling as a selection method. Finally, we used the normal CURL algorithm without the use of any replay buffer as a baseline.

3 Results

3.1 Selection at expansion (A)

As mentioned earlier, the first strategy consisted to add samples to the replay buffer at expansion time, right after having trained the new expert on the poor dataset. The results can be seen in figure 4. Looking at the first task, we can see that using a replay buffer seems to be effective. However, sampling only using un-similarity in the latent space and sampling randomly seems to perform better than the two other options. Not much can be said from the other tasks since the difference between each experience doesn't seem meaningful enough. We notice that the second task has a reconstruction loss that is way smaller then all the others. We believe that this stem from the simplicity of the number one, which usually requires less pixels to be activated. While we haven't done it ourselves, we believe that some strategies could be used in order to adapt the threshold value to consider a sample as badly reproduced. We would suggest using either task probing or, more simply, scaling the reconstruction criteria to the number of activated pixels.

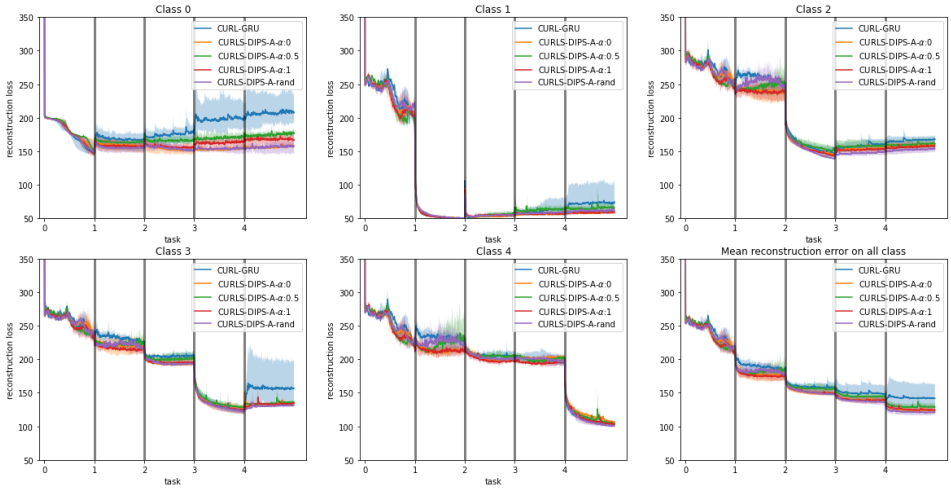

Figure 4: Reconstruction error when samples are selected after the expansion of experts for each class and overall.

Looking at each experiment, it seems that selective sampling using the latent space and random selection perform better then the other criteria. We suspect that since we are sampling a considerable amount of samples from the poor buffer (30% in our setting), the use of a selection criteria might not be necessary since a pre-selection is naturally made by the CURL algorithm. We can see in figure 5 the number of experts created overtime. We can see that when we used a replay buffer the number of experts was usually smaller then 10, meaning that the size of the replay buffer was less then 300. We consider this size to be fairly small compared to the full dataset that contained 25000 different samples. The number of experts created while using a replay buffer seems to be lower compared to the baseline. We believe it might be due to the replay buffer having some kind of stabilizing effect, leading to less experts being created needlessly.

To confirmed this, we have looked at the generation done by each expert around the center of their personal cluster. To do so, we used the latent decoder ($p(z|y = k)$) with a one-hot encoding in order to have the central parameter associated with each expert. We then used these parameters to explore the space in the range $[\mu - 1.4\sigma, \mu + 1.4\sigma]$. We notice that some of the execution for the baseline ended with experts that seem to all produce the same kind of samples uniformly across their

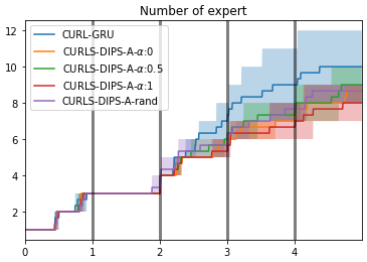

Figure 5: Number of expert overtime when samples are selected after the expansion of expert.

associated cluster. While this have also been observe when using a replay buffer, it seem to happen less often as it can be see in figure 6.

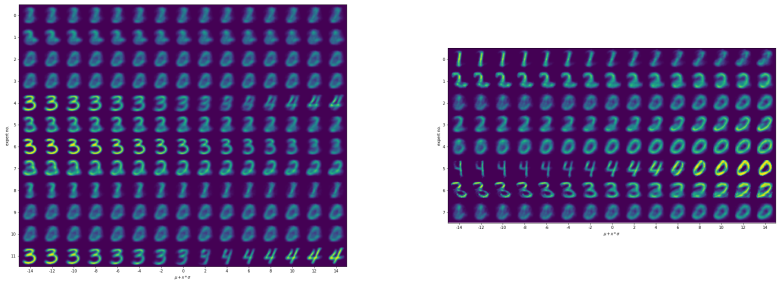

Figure 6: Exploration of the z latent space of CURL a) using only generative replay and b) using selective sampling with $\alpha = 0$. (I forgot to dive the xticks label by 10, the exploration is in the range $[\mu - 1.4\sigma, \mu + 1.4\sigma]$).

3.2 Selection at last epoch (B-C)

The second technique we tested was to add samples during the last epoch of each task. We tested this using two different approaches. The first one consisted of selecting 5 samples from each new batch. Doing so guarantied task diversity since once a sample was in the replay buffer it stayed there. The second one consisted to select from both the new batches and the existing replay buffer, meaning that a sample could be remove from the replay buffer. While the later is not guarantied to have task diversification, we believe it is interesting in order to evaluate the capability of using un-similarity for task diversification. What's more, this strategy could be of interest in case where a cap size of the replay buffer is pre-determined. In this case, different number of exemplars for each task might be preferable and reevaluation of the whole replay buffer could be favorable.

We can see in figure 7 that, as expected, doing random selection seems to be more prone to catastrophic forgetting as the performance didn't seem to be better for the first task then the baseline. On the other hand, we observe less forgetting when using selective sampling.

By looking at the examples of the replay buffers from figure 8, we can see that un-similarity in the latent space do manage to capture task diversification across all tasks, even in this setting. While un-similarity in the sample space seem to do it, it doesn't seem to be as good since we can notice a lack of 1 and we can see that, as expected, there is no task diversified sample in the replay buffer when sampling randomly. We think that it is a sign that of the possible usefulness of un-similarity in the latent space, specially since we think that using the z latent space could be even more promising since it is where the clustering is enforced.

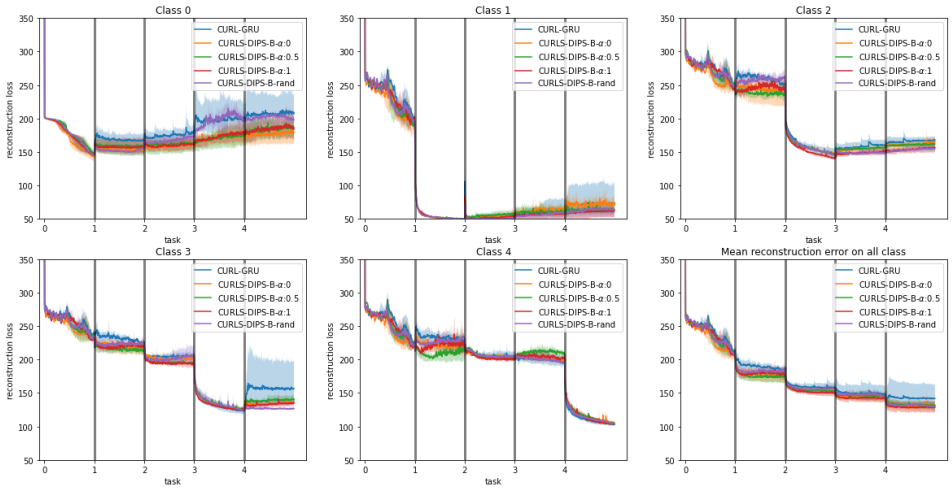

Figure 7: Reconstruction error when the whole replay buffer is reevaluated at the last epoch of each task for each class and overall.

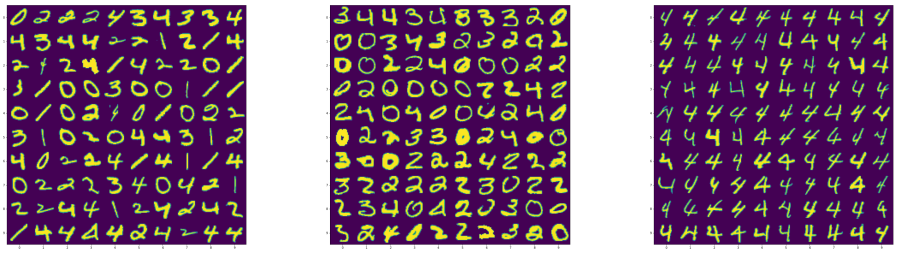

Figure 8: Replay buffer when reevaluating the whole replay buffer using a) selective sampling in the latent space, b) selective sampling in the sample space and c) random sampling.

We can see in figure 9 the results when only 5 samples were added per batch without ever removing samples from the replay buffer. Surprisingly, when looking at the results for the zeros, it seems like selecting samples randomly is better then using selective sampling. However, looking at figure 10, we can see that this technique created more experts then the others, suggesting worse generalization from each expert during training. We haven't considered the influence of the number of experts created up to this point. While we still think the use of selective sampling could be useful to build the replay buffer, since it is capable of promoting task diversity, we can not conclude that it is the case at the moment. In order to do so, we believe a similar experience could be done while limiting the maximum number of experts and by doing an ablation study on this parameter. While once again using only the latent space seems to perform better, using the sample space have created less experts in all three settings, raising the same uncertainty toward using these results as a complete metric in order to compare them.

3.3 Comparative study

We compared all three management strategies of the replay buffer when using only selective sampling with the latent space in figure 11. We can see that while selective sampling at the last epoch without removing samples from the replay buffer seems to perform better, selective sampling at expansion time seems to be on par with it. What's more, looking at figure 12, we can see that it also produce less experts. Considering that selective sampling at expansion time finish with a replay buffer more then 4 times smaller then doing so at last epoch without removing sample, we believe that it is in fact the best management strategy.

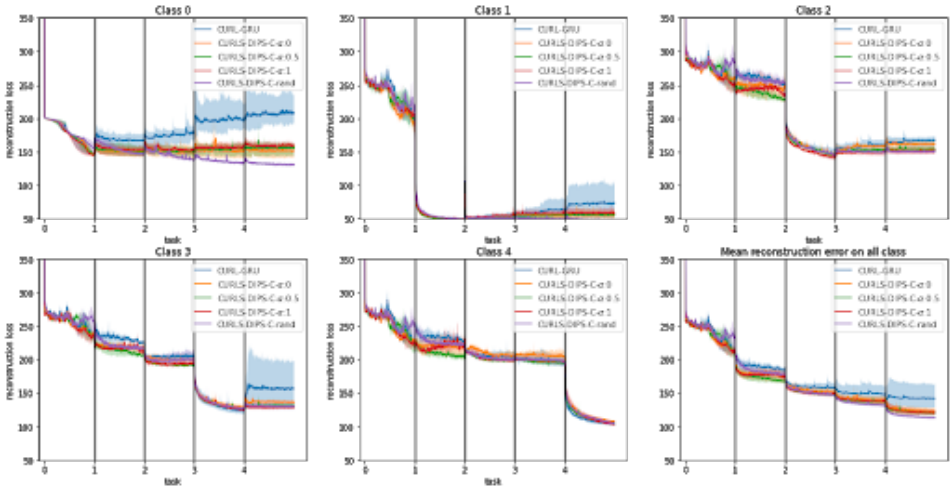

Figure 9: Reconstruction error when samples are added during the last epoch of each task without reevaluating those in the replay buffer for each class and overall.

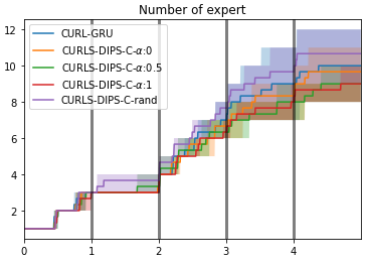

Figure 10: Number of expert overtime when samples are added during the last epoch of each task without reevaluating those in the replay buffer.

3.4 Influence of the replay buffer and the risk of overfitting

Finally, we wanted to evaluate if the improvement seen was only stemming from the use of a replay buffer compared to only using generative replay. To do so, we used selective sampling at expansion time using the latent space and tried three different modifications to favor sampling from the replay buffer. First, we changed the λ parameter from 0.1 to 0.9 in order to sample mostly from the replay buffer then from the generative replay 1. Secondly, we removed the use of the generative replay completely and tried to use $\lambda = 0.1$ and $\lambda = 0.3$ which will now affect the ratio between the real samples and those in the replay buffer (since generative replay is removed).

We can see in figure 13 that in all three cases the results are worst then when using generative replay as defined earlier and that catastrophic forgetting is either similar to the baseline or worst, suggesting overfitting on the selected samples. We believe that these results are demonstrative that using generative replay is essential when using such a small replay buffer.

4 Conclusion

We have investigated in this paper the use of a replay buffer formed using selective sampling in order to alleviate catastrophic forgetting in generative replay. We have found that the use of a replay buffer could help prevent catastrophic forgetting in generative replay even when this replay buffer would be enough by itself because of its size. Based on our results, it seems that from all the replay buffer management techniques proposed, selective sampling at expansion time is the most memory cost-effective. We also showed that un-similarity can provide task diversity when doing selective sampling. However, we could not conclude whether it is useful as we didn't took into account the influence of the number of experts generated. While the creation of experts doesn't necessarily

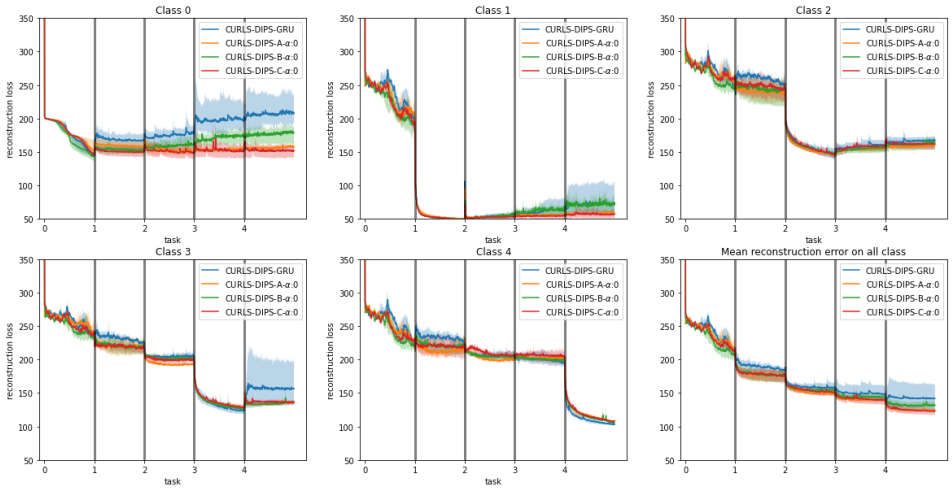

Figure 11: Reconstruction error for each replay buffer management mechanism using un-similarity in the latent space.

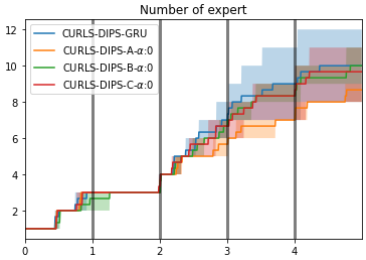

Figure 12: Number of expert overtime for each replay buffer management mechanism using un-similarity in the latent space.

translate to better results, it does affect the capacity of the algorithm and we believe that it should be taken into account before drawing any conclusion. We also consider that selective sampling might not be necessary when the replay buffer management strategy already enforces task diversity.

While we believe to have shed the light on the usefulness of combining generative replay with a replay buffer, a lot of questions are left open-ended and we are left with the belief that we barely graze the surface of this topics. We think that other kinds of similarities should be investigated, such as similarity in the z latent space since it's where the clustering is enforced, or such as similarity in the representation space of a pre-trained network using either a subset of the dataset or a similar dataset. We think that doing so could provide a metric that is both leveraging representation and independent of the training, making it less prone to catastrophic forgetting.

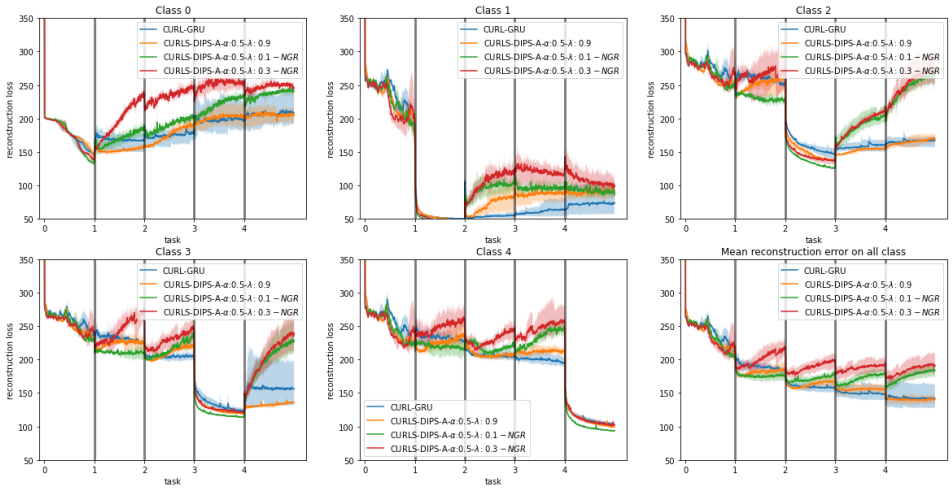

Figure 13: Reconstruction error when favoring mainly the replay buffer instead of generative replay.

References

[1] Dushyant Rao, Francesco Visin, Andrei A. Rusu, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Continual unsupervised representation learning, 2019.

[2] Raghul Asokan. Neural networks intuitions: 8. translation invariance in object detectors, May 2020.

[3] Miriam, Sean Lorenz, and Shioulin. Introducing variational autoencoders (in prose and code).