Deep Learning Project

Gallo Matteo

September 18, 2023

Contents

1 Introduction

Why is continual learning needed? To answer this question, it is first necessary to understand what continual learning is. Continual learning is an approach to machine learning in a dynamic setting, where the tasks that the machine learning model needs to solve or the datasets that it uses to solve them are changing over time. Using a standard approach would lead to problems which are later discussed in this report.
Continual learning can also solve problems that are not immediately obvious when using a standard model. One example is the amount of data that is needed to store in order to train all the networks out there. When a model is built, it is implicitly assumed that it will have the data available to it if it ever needs to be retrained. However, this might not be true in the future. International Data Corporation (IDC) recently released a white paper in which they put forth the argument that by the year 2025, the rate of data generation will surge from the current 16 zettabytes per year (that's equivalent to a trillion gigabytes) to a staggering 160 zettabytes. However, it's projected that there will only be the capacity to store a mere 3% to 12% of this colossal amount. To put it plainly, data will need to be processed in real-time, as failing to do so would result in its permanent loss due to storage technology struggling to keep pace with the rate of data production. So this is why continual learning is needed today.

But let's delve deeper in the classification problems.
Real-world classification tasks are complex and make our lives harder. While it is possible to find comfort in confining oneself indoors, shutting out external influences, and attempting to classify the MNIST dataset, this tranquillity shatters when venturing into the outside world, exposing the stark realities that were overlooked.
However, it's in these uncharted territories beyond the comfort zones that the true essence of the challenge comes alive. This is where the true story begins. The ambitions draw one towards real-world scenarios, where tangible situations are confronted.
Consider a young child just beginning to understand what the world is made of. They observe a sequence of unfamiliar objects and gradually learn the ability to distinguish between them. Now, imagine if this young mind could only absorb ten distinct items at any given time, much like a neural network constructed to classify the MNIST dataset. Then, unexpectedly, an eleventh object enters the scene. Regrettably, a dilemma arises, as the child must discard something from their existing knowledge to accommodate the new notion. The best solution would be to have the ability to learn without forgetting prior knowledge. An intriguing possibility, isn't it?
Now, let's shift the focus back to reality. These are the very issues that this work is about to delve into specifically, the intricate challenges of the 'New Classes' problem.

To study the problems, a simple convolutional neural network (CNN) called MoodyRoosterNet of five layers is built. This network has a variable output layer that is useful for analysing class incremental examples. The chosen loss function is cross entropy, and the optimisation method is stochastic gradient descent, with hyper parameters that may vary from case to case. The chosen dataset is the MNIST dataset.

To evaluate the performances of the presented approaches, different metrics are used. The metrics are the experience accuracy, the experience confusion matrix, and the experience forgetting. With the term experience is referred to the portion of the dataset that the network can use to train and test at specific time. The ExperienceForgetting metric, describes the accuracy loss detected for a certain experience. This plugin metric, computed separately for each experience, is the difference between the accuracy result obtained after first training on a experience and the accuracy result obtained on the same experience at the end of successive experiences.

2 Experiments and Discussion

Every strategy discussed works on the problem of New Classes. This is the case where new classes appear over time, and it is wanted to avoid retraining the model from scratch whenever a new class is added.

2.1 Naive Approach

Let's suppose that today a model to recognise handwritten digits is built. At the moment, only the dataset of zeros is accessible, so it is fed to the network. The next day, the dataset of ones becomes available. What is the correct method to update the network? The simplest method would be to feed the network the new dataset and see what happens, this is why is called naive. A priori, there might be no good reason not to do this.
So, in this approach, the experiences are created by splitting the digits into different sets for training, while the test dataset contains all of the digits from the previous experiences. This means that the model while training can only see one class at a time, instead while testing it can see all the previously seen classes.

What it is seen in Fig2, Fig1, Fig3 is that for this naive approach the results are awful. The network completely forgets what it has already learned when it is trained on a new class, and only achieves a high accuracy on the last class it has seen. This is also confirmed by the confusion matrix, which shows that the network classifies every image as the last class it has learned. The reason for this is simple. The manifold for a single handwritten digit is different from the manifold of all the handwritten digits. When the network reaches a minimum for the cost function with SGD, this minimum is different for each class, and it is also different from the minimum for all the ten classes together. Therefore, because of how the experiences for this approach are built, the best weights reached in experience 0 are just random starting weights for experience 1. This is the process known as Catastrophic Forgetting.

2.2 Cumulative Approach

Another simple idea that may quickly come to mind is to put all the datasets together and train and test the model on the buildup of the different classes. Here, the experiences are built by gradually stacking together the datasets of the digits that the model sees. Every time a new digit is added, the network is retrained and tested on all of the previous experiences.

The results here are very good, the model is doing well on all the classes across all the experiences. This can be seen from all the proposed metrics.
So, is this a cure for all the problem that continual learning is trying to solve? Absolutely no, because this is not continual learning, this approach is what one want to avoid when building continual learning strategies, but it is possible to use it as mean of comparison for the other approaches.

2.3 Replay Approach

The replay approach is called that because it stores a small amount of data from previous experiences in a buffer memory. The specific saving strategy can vary, but this work uses the balanced strategy, which divides the available space equally between the experiences. As a result, each experience will contain the original experience as well as the data stored in the buffer memory.

The results here are acceptable. While they are not as good as the cumulative approach, it is possible to achieve reasonable metric values with a little effort. One important thing to note is that the buffer memory size is crucial. If the buffer memory is too small, the model will produce worse results because it will store fewer examples from previous experiences. On the other hand, if the buffer memory is large enough, it will become more similar to the cumulative approach, resulting in better performances but losing the purpose of the approach.

2.4 Elastic Weight Consolidation Approach

Elastic weight consolidation (EWC) is a regularisation technique that helps neural networks avoid catastrophic forgetting. It does this by computing the importance of each weight in the network for each task, and then adding a penalty to the loss function if those weights are updated too much. The experiences are built in the same way as the naive approach, but instead of working on the dataset, EWC works on the parameters of the model.

After a neural network has been trained to perform a task, the sensitivity of the model to changes in each of its parameters can be estimated by looking at the curvature of the loss surface along the direction of those changes. In particular, high curvature means that a small change in a parameter can result in a large increase in the loss. The diagonal of the Fisher information matrix, which can be computed from the first partial derivatives of the loss function, is equivalent to the curvature of the loss surface near a minimum. Therefore, the importance of the k-th weights is directly proportional to the k-th element of the diagonal of the fisher information matrix. When fine-tuning a model on new tasks, it is important to avoid changing the weights that are most important for the original task too much. This can be done by adding a regularisation term to the loss function when training on a second task. $L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A_i})$, where L_B is the loss function relative to the task B, λ is a parameter to control the importance of the previous task, A refers to the previous task, and i labels each parameter.
The figures shows that the results obtained with the proposed approach are not as good as those obtained with the replay approach. In fact, they are quite poor. The model learns without completely forgetting at the beginning, but by the last experience, only two classes have non-zero accuracy. The forgetting metric shows that the model forgets the most in the last half of the experiences. This is likely because the model has to keep track of too many parameters that were important for the previous classes, but have no value for the new ones, while also trying to learn the new experience. To obtain these results, the learning rate was changed to a smaller value, and momentum was not used. The most important hyper parameter in this approach is λ, which is the weight of the penalty term in the loss function. A higher value of λ means that more importance is given to past experiences, while a lower value means that more importance is given to recent experiences. This hyper parameter was found to be very sensitive, as the network could transition from producing the results presented here to working just like the naive approach with only a small change in the value of λ.

2.5 Copy-weights with Re-init Approach

This approach is similar to the previous one in that it does not work on the data. However, it differs from EWC in that it is an architectural approach rather than a regularisation method.

This strategy works by incrementally adding neurons in the output layer as the model see new classes. In particular at the beginning of training, a set of consolidated weights cw is set to zero; then there is a *past* variable, also set equal to zero, used to count how many classes the model has already seen. The algorithm proceeds by randomly initialising all the weights before the output layer, these weights are called Θ, then for each training batch it expands the output layer by adding the neurons related to the specific classes presented in the batch that the model has never seen. The weights of the output layer are called temporary weights tw, and at this point are set to zero if the class is new, otherwise are set to the last value of cw. Then the model can train on the batch with SGD, and it updates all the Θ and tw if it is learning the first class, otherwise it just updates tw while freezing Θ. A the the end of the batch for each class j contained in the batch: $wpast_j = \sqrt{\frac{past_j}{cur_j}}$ where cur_j is the number of classes in the current batch, $cw[j] = \frac{cw[j]wpast_j + tw[j] - avg(tw)}{wpast_j + 1}$, $past_j = past_j + cur_j$.

The results show that the model does not learn as it progresses through the experiences. This may be because the representation of the data that the model learns is just based on the first class, which presents only a few features shared with other classes, and also because just the last layer is class specific. Another problem may be the selection of the hyper parameters for SGD. A lot of tests were conducted, and the boundary between not learning and learning only the last class is very thin. Another parameter that slightly improved the results was the one related to freezing the weights except for the last layer. This can be useful in the case of pre-training, where the model already knows the correct representation of the data but still needs to learn how to classify it. Even without pre-training, this yielded slightly better results, suggesting that moving with SGD using data from only one class completely ruins the model's performance.

2.6 Images

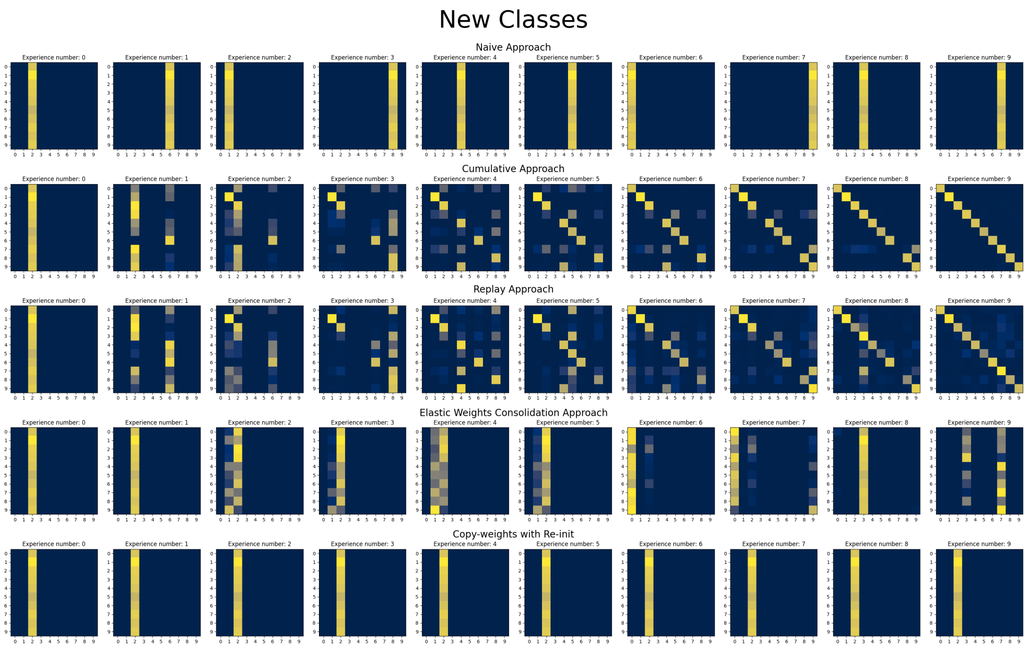

Figure 1: Confusion matrix across experiences

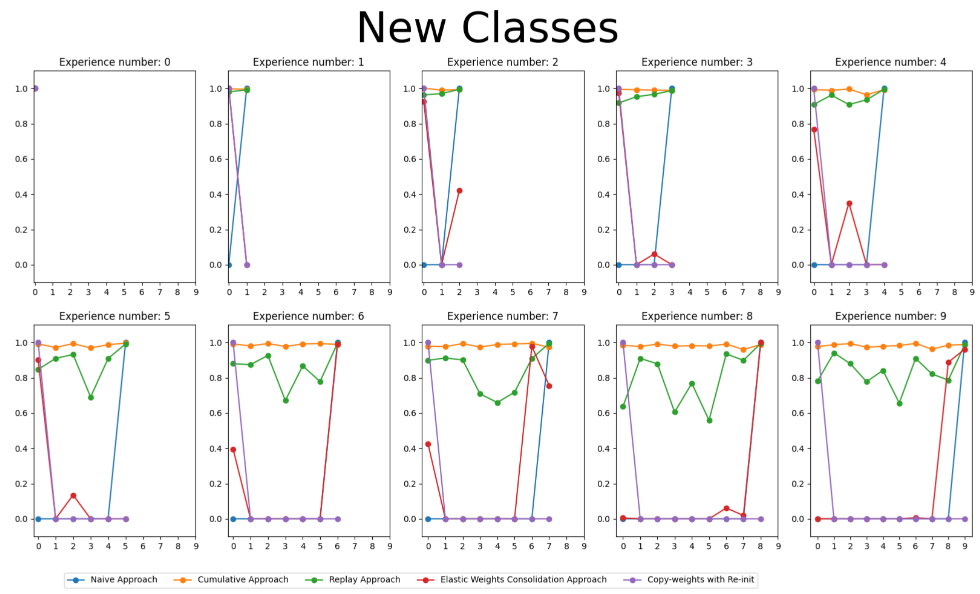

Figure 2: Accuracy across experiences

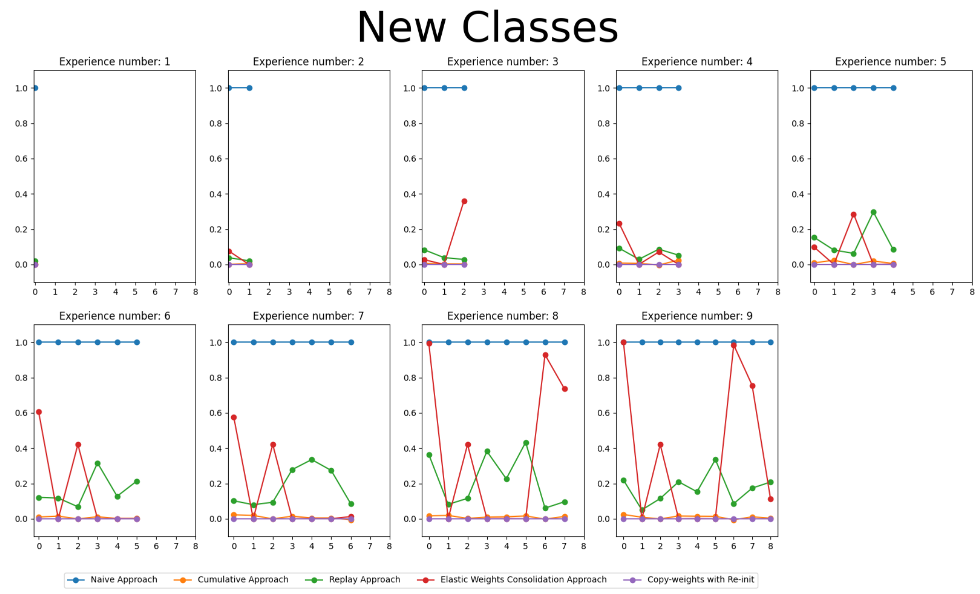

Figure 3: Forgetting metric across experiences

3 Conclusions

In conclusion, the first approach to continual learning was not much bright. The best results were yield by the replay approach even if it was the most simple one, aspect that should not lead to the conclusion that the other approaches were worse in general, but that maybe a more refined approach to the application of EWC and CWR should be taken in order to get out the most from them. The project could be further developed by implementing pretraining for the CWR approach, or implementing some kind of greed search for the EWC one. Another possible aspect to explore could be the new instances problem, where instead of having a dynamic setting in which the number of classes grows, the representation of the data is changing.

4 Appendix

4.1 New Instances Introduction

Alternatively, let's ponder this: what if the curious toddler mentioned before could solely recognise Spider-Man by the front part of his mask? What if he encountered Spider-Man with his back turned? The outcome would be an inability to identify the superhero, and worse, a potential compromise to Earth's safety. A truly distressing thought!
This second problem is the 'New Instances' problem, where the data representation changes over time. This is done by using the PermutedMNIST dataset, where the pixels of the images are randomly permuted in a specific way for each experience. This means that if there are 5 experiences, each experience will have all 10 classes, but each experience will have a different representation of the numbers.

4.2 Naive Approach

For the application on the new instances problem each train experience has only one representation of the ten classes but the model is tested on all the previously seen representations.
The new instances problem shows similar results as the new classes one, although with slightly better performance at the end. This may be because the random patterns created in permuted MNIST may disrupt the intrinsic structure of handwritten digits, resulting in representations that are not entirely different from previous ones. However, this may not be a negative thing, as different representations of a random object may not be very diverse either.

4.3 Cumulative Approach

The experiences are built in the same as in new classes, but here the build-up of classes is replaced with a build-up of representations.
The results and the conclusions are the same as the new classes section.

4.4 Replay Approach

The results and the conclusions are the same as the new classes section.

4.5 Elastic Weight Consolidation Approach

This strategy achieves better results on the new instances problem than on the new classes one. This may be because the model is able to learn global features when it sees the entire dataset. This allows EWC to better compute the importance of the weights, leading to less forgetting of previous experiences.

4.6 Copy-weights with Re-init Approach

This is also working better on this second problem, the reason is again imputable to the fact that the network is allowed to see all the classes and hence can learn features that have a global importance.

4.7 Images

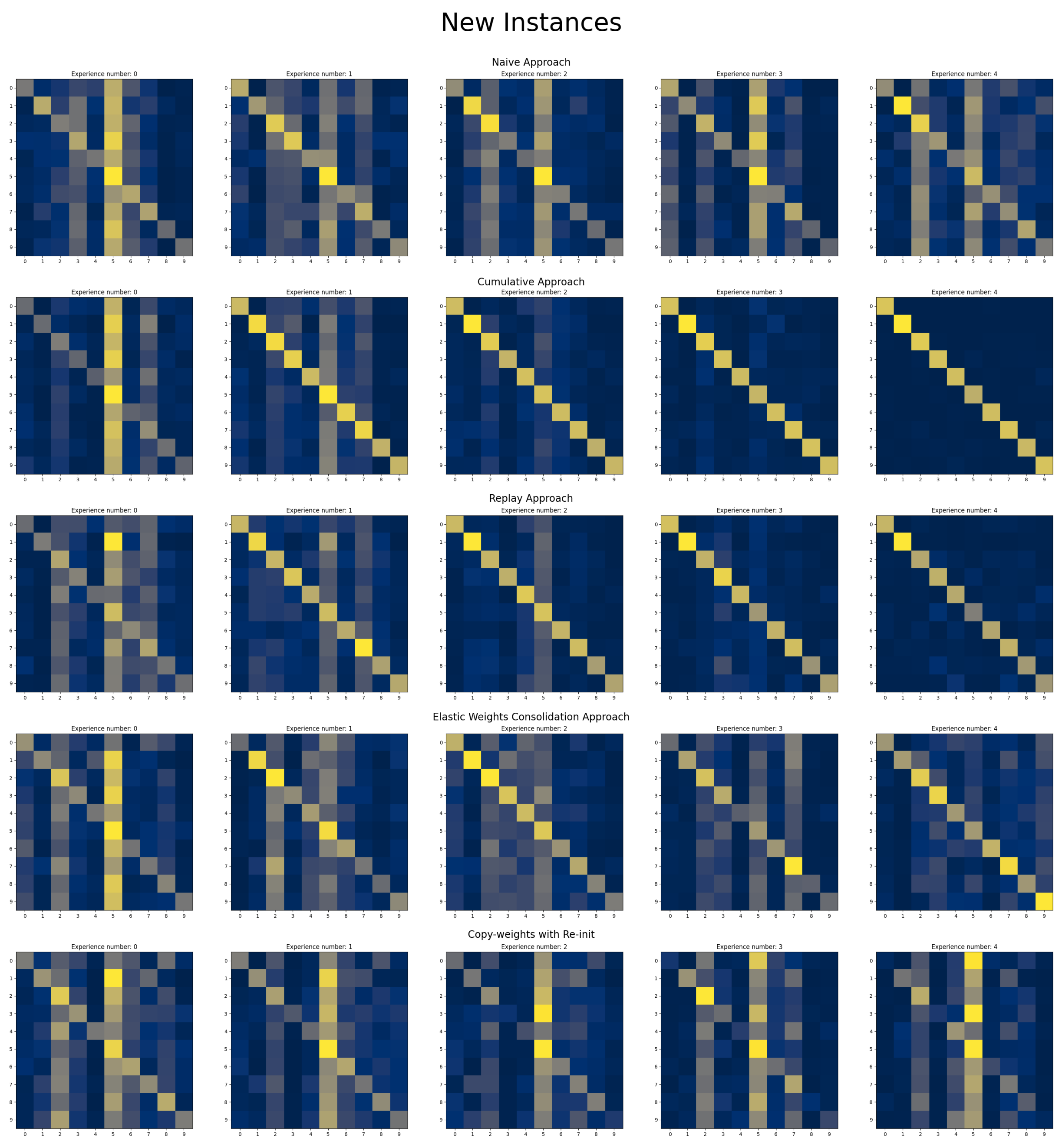

Figure 4: Confusion matrix across experiences

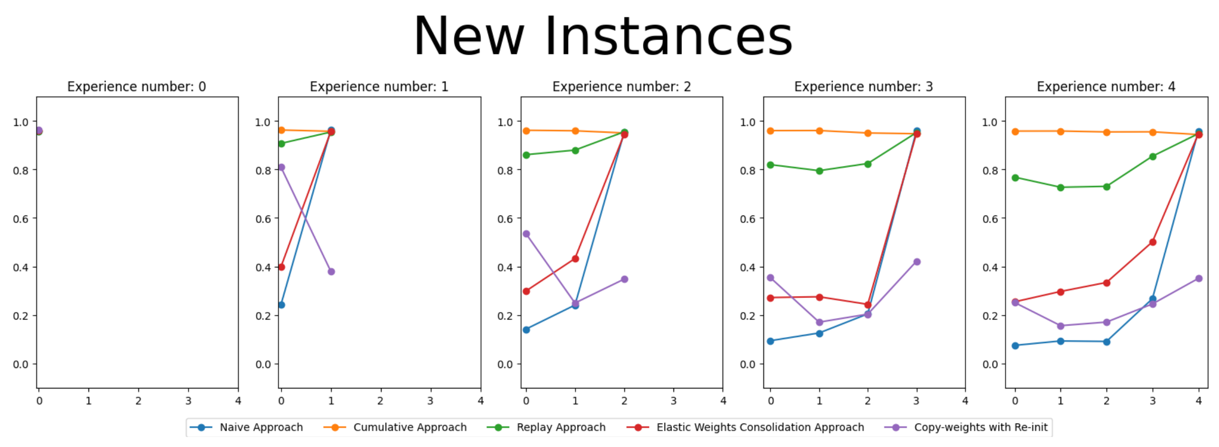

Figure 5: Accuracy across experiences

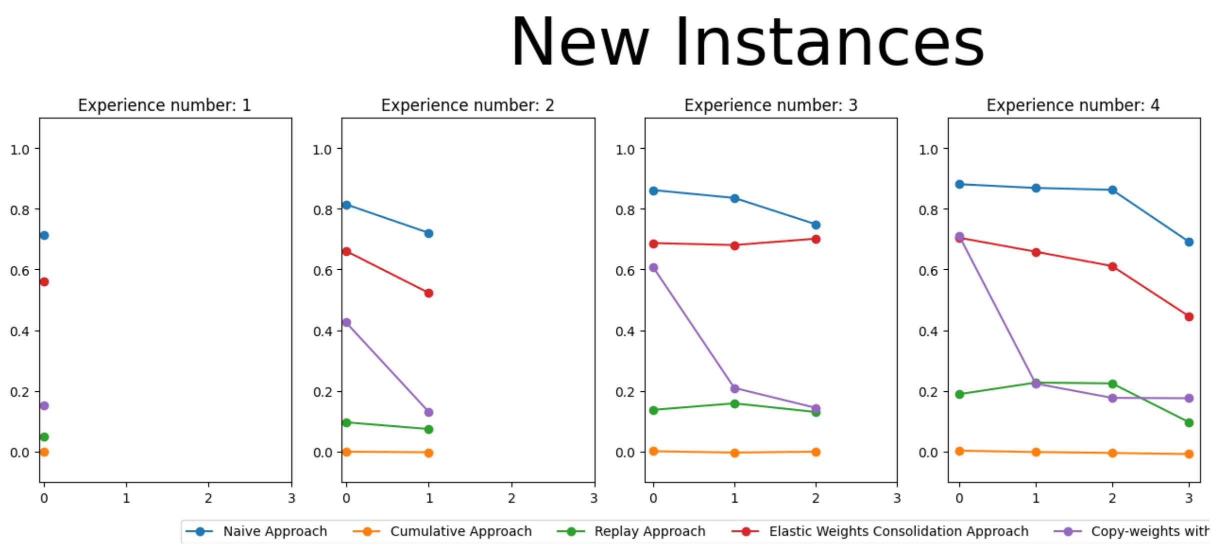

Figure 6: Forgetting metric across experiences