An empirical comparison of generators in replay-based continual learning

Nadzeya Dzemidovich[1] and Alexander Gepperth[1]

1- University of Applied Sciences Fulda - Applied Computer Science
Leipzigerstr. 123, 36037 Fulda - Germany

Abstract. This study is in the context of continual learning (CL) with DNNs. It compares several types of generators when performing replay, i.e., the generation of previously seen samples, to avoid catastrophic forgetting. Principal generators are generative adversarial networks (GANs) and variational autoencoders (VAEs). We evaluate these generators in various flavors (conditional, Wasserstein etc.) w.r.t. CL performance on a variety of CL tasks generated from the MNIST benchmark. Concerning generators, we find that VAEs are generally more compatible with CL than GANs. More generally, we find that replay-based CL faces counter-intuitive issues for seemingly simple problems: first, that performance degrades more strongly as less information is added, and, furthermore, that performance degrades even when only known information is added.

1 Introduction

This article is concerned with continual learning (CL), a branch of machine learning that investigates learning from non-stationary data distributions. A particular type of CL problem is supervised learning, where disjoint groups of classes (termed *sub-tasks*) are presented to the model in a sequential manner. Within each sub-task, a stationary data distribution is assumed. This setting is summarized in fig. 1. A large number of proposals have been suggested in recent years to address CL in this (or related) scenarios, see [1–3] for comprehensive overviews. A very promising line of research seems to be represented by so-called *generative replay* methods, see [4]. A scholar consists of a solver (classifier) and a generator, which has the task of producing samples that are similar to samples seen in past sub-tasks $T < T_0$ prior to training on sub-task T_0. These generated

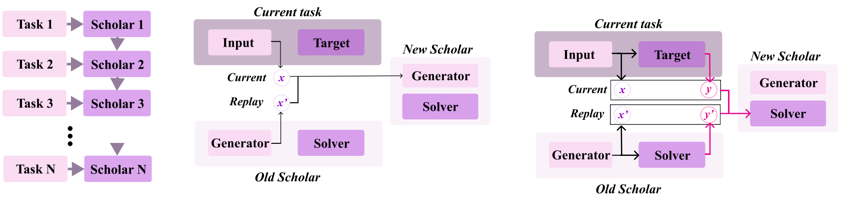

Fig. 1: Generative replay ina na nutshell. For each new *sub-task*, a new *scholar* (consisting of a DNN generator and a DNN solver) is trained based on data from the new sub-task, merged with data produced by the generator.

samples are then merged with samples from sub-task T_0 to form the training set for both generator and solver. At the end of sub-task T_0, a generator and solver together are capable of classifying and generating samples coming from all sub-tasks $T \leq T_0$, including T_0. Generative replay models can use several types of generators, such as variational autoencoders (VAEs) or generative adversarial networks (GANs) in all their flavors. Even the use of Gaussian Mixture Models (GMMs) has been described in [5]. In this article, we perform a systematic analysis of replay-based CL using Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs) and Wasserstein GANS.

1.1 Motivation and findings

This investigation is motivated by the enormous freedom that exists in the choice of the generators, and the general complexity of replay-based learning. More to the point, we would like to derive clear guidelines as to which generators are generally preferable, or preferable in certain well-defined situations. More generally, we would like to determine whether replay-based CL has any generic weak points that need to be taken care of for real-world applications. Our principal findings and contributions to the CL community, in the simplified case of two sub-tasks only, can be summarized as follows: first of all, without prior access to the data to tune GAN-based generators, VAEs are a better choice as they avoid mode collapse under resource constraints. This problem is alleviated somewhat by Wasserstein-based GANs, but at enormous computational cost. Furthermore, CL performance degrades even if only known classes are added in the second sub-task. Finally, we find that CL performance degrades as the second sub-task becomes smaller. This is counter-intuitive since the problem is supposed to be simpler.

1.2 Related work

Generative replay (or: pseudo-rehearsal) for CL has been the subject of numerous articles, see, e.g., [1, 6–8] for reviews or [9] for the original article. Benchmarks of replay approaches are performed in, e.g., [10] or [5]. However, very few articles actually deal with the question of generators for replay-based CL. In [11], several generators are compared, but more w.r.t. their ability to generate samples in a conditional fashion, and procedures for replay with conditional generators are proposed. The quality evaluation of generators is a difficult subject in general, see [12]. Gaussian Mixture Models have been used as generators in replay-based learning and compared to more conventional ones in [5]. Issues of mode collapse in GANs have long been observed, e.g., in [13]. On the whole, we believe that the issue of which generators are best suited for replay-based CL has not yet been exhaustively covered by related work.

EXP	1	2	3	4	5	6	7	8
T1	0-4	0-8	0	4	3 4	0-9	0-9	0-9
T2	5-9	9	1	9	8 9	1	1	1
T3	-	-	-	-	-	-	1	1
T4	-	-	-	-	-	-	-	1

Table 1: Conducted experiments with sub-task partitions . All experiments can be conducted with generators based on VAE, GAN or Wasserstein-GAN.

2 Methods and implementation

2.1 Dataset

For our experiments, we use the MNIST dataset, a common benchmark for visual classification problems. It consists of 70,000 greyscale images of handwritten digits (from zero to nine) of size 28x28, containing 55,000 training and 10,000 test samples that are approximately equally distributed over ten classes. While MNIST may be considered too simple as an outright classification problem, it was recently proven [3] that many CL approaches fail on simple two-task sequential learning tasks constructed from MNIST, so MNIST does constitute an adequate benchmark in a CL context.

2.2 Evaluation

We evaluate CL experiments by classification accuracy and by histogram plots of the generated class distributions. Classification accuracies are computed after all sub-tasks have been processed, and are evaluated on data from individual sub-tasks (e.g., $T1$, $T2$) or on the whole dataset (termed $TAll$). All evaluations are averaged over several experimental runs (repetitions with same hyper-parameters to exclude spurious results).

2.3 Experimental procedure

For every method and experiment listed in table 1, we perform 10 runs, produce the appropriate metrics (see section 2.2) and average them over all runs. Prior to a run, each DNN is independently and randomly initialized. Hyper-parameters of the training procedure are set as follows: solvers are trained for 10 (sub-task) epochs, whereas generators are trained for 50 epochs. The batch size is universally set to 100, the learning rate for solvers, GANs and VAEs to $\epsilon_S = 10^{-4}$, $\epsilon_{VAE} = \epsilon_{GAN} = 10^{-3}$. We use a VAE latent dimension of 25 and a disentangling factor $\beta = 1.$, whereas a noise dimension of 75 is used for GANs. Class-conditional generation for VAE is disabled unless specifically stated. The ratio between replayed and new data is set according to the ratio of samples from previous and current sub-task classes (balanced for EXP1, EXP3, EXP4, EXP5 and 9:1 for EXP2). The structure of all used DNNs is given in table 2.

Table 2: DNN structure for solvers and generators.

Method/DNN	**Structure**
VAE: Generator	Conv2D(32,5,2)-ReLU-Conv2D(64,5,2)-ReLU-Dense(100)-ReLU-Dense(25)-ReLU-Dense(50)
VAE: Solver	Dense(100)-ReLU-Dense(3136)-ReLU-Conv2DTranspose(32,5,2)-RelU-Conv2DTranspose(1,5,2)-Sigmoid
GAN: Generator	Dense(1024)-BatchNorm-LeakyReLU(0.2)-Dense(6272)-BatchNorm-LeakyReLU(0.2)-Conv2DTranspose(64,4,2)-BatchNorm-LeakyReLU(0.2)-Conv2DTranspose(1,4,2)-Sigmoid
GAN: Solver	Conv2D(64,4,2)-BatchNorm-ReLU-LeakyReLU(0.2)-Dropout(0.3)-Conv2D(128,3,2)-BatchNorm-ReLU-LeakyReLU(0.2)-Dropout(0.3)-Flatten-Dense(1024)-BatchNorm-ReLU-LeakyReLU(0.2)-Dropout(0.3)-Dense(1)-Sigmoid
WGAN: Generator	Dense(4096)-BatchNorm-LeakyReLU(0.2)-UpSampling2D(2)-Conv2D(128,3,1)-BatchNorm-LeakyReLU(0.2)-UpSampling2D(2)-Conv2D(64,3,1)-BatchNorm-LeakyReLU(0.2)-UpSampling2D(2)-Conv2D(1,3,1)-BatchNorm-Tanh-Cropping2D(2)
WGAN: Solver	ZeroPadding2D-Conv2D(64,5,2)-LeakyReLU(0.2)-Conv2D(128,5,2)-LeakyReLU(0.2)-Dropout(0.3)-Conv2D(256,5,2)-LeakyReLU(0.2)-Dropout(0.3)-Conv2D(512,5,2)-LeakyReLU(0.2)-Flatten-Dropout(0.2)-Dense(1)

ID	T1	T2	TAll	T1	T2	TAll	T1	T2	TAll
	VAE			GAN			W-GAN		
1	93.5	97.3	95.3	36.7	97.9	66.5	89.8	97.9	93.7
2	92.8	97.7	93.3	0	100	10.1	65.7	99.7	69.2
3	100.0	99.9	100.0	99.6	100	99.1	100	100	100
4	96.2	98.7	97.2	69.6	69.6	84.7	43.1	100	72.7
5	90.9	98.3	94.5	42.6	99.3	70.9	92.4	98.4	95.4

Table 3: CL accuracies averaged over 10 runs for EXP1-5 computed when using VAEs, GANs and Wasserstein-GANs as generators. Accuracies on sub-tasks are computed after the last sub-task is completed.

3 Experiments

Default values of the parameters are presented in section 2.3. All experiments are run on identical hardware. For all experiments, accuracy measurements are obtained after the execution of the last sub-task.

3.1 Two-subtask experiments

Initially, we focus on EXP1-EXP5, see table 1. Table 3 shows average accuracy values over ten runs, evaluated as detailed in section 2.2. As can be observed, VAE shows best performance for almost every experiment, whereas GAN shows the weakest results. We hypothesize that GAN results can be explained by mode collapse, i.e., the structure of the GAN is insufficient to represent the full probability distribution. The Wasserstein modification strongly improves GAN results. For EXP3, we hypothesize that it can be solved by virtually all methods since the involved classes are very dissimilar. Finally, EXP4 showed the worst result across all generators, which is curious but presumably due to the fact that the two classes involved are very similar. During evaluation of EXP1-EXP5, one can observe that some generators do not provide balanced class distributions between classes from the first sub-task, see fig. 2. For GANs, we interpret this as the occurrence of mode collapse, which is remedied by using the Wasserstein approach. Nevertheless, during some experiments using the Wasserstein approach, samples are still unbalanced, and for such cases, accuracy

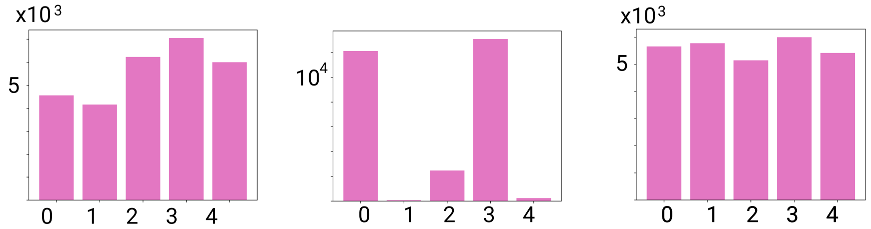

Fig. 2: Class distributions: VAE(left), GAN (middle) and W-GAN (right).

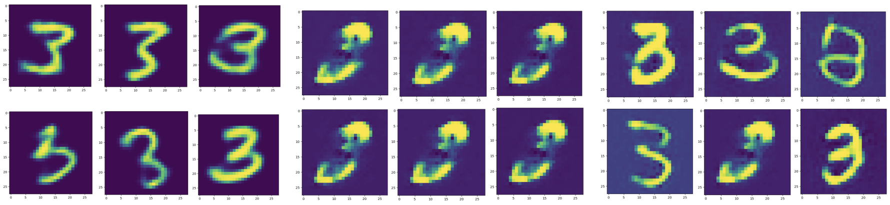

Fig. 3: Samples of class ”3”: VAE(left), GAN(middle) and W-GAN(right).

suffers, see EXP2 in table 3. Thus it can be stated that Wasserstein-GANs do not guarantee a perfectly balanced distribution of the generated samples, while increasing training time by a factor of at least 5. Typical samples produced by different generators during EXP1 are shown in fig. 3, showing a clear edge for VAEs. As is usual with VAEs, samples look less ”sharp” than GAN-generated ones, but this does not seem to impair CL performance.

3.2 Multiple sub-tasks and generic aspects of CL

Experiments EXP6-EXP8 investigate whether the (possibly repeated) addition of an already known class has an effect on replay-based CL performance. As generators, we use Wasserstein-GANs, conditional VAEs (CVAEs) and vanilla VAEs which showed promising performance in the previous set of experiments. Table 4 shows results for EXP6-8. We observe that, in general, T1 performance drops more strongly as the number of sub-tasks increases, although only known information is added.

4 Conclusions

Since the key findings were stated in section 1.1, we focus on an interpretation of the obtained results here. We expect these findings to hold whenever we cannot tune generators to a a specific CL problem by cross-validation on the whole dataset (e.g., MNIST), as this would require knowledge of data from sub-tasks that lie in the future. If these were generally available, the need for dedicated CL methods would literally vanish. We modeled this case here by choosing a fixed

Sub-Task	VAE			CVAE			W-GAN		
EXP	6	7	8	6	7	8	6	7	8
T1	94.6	85.6	80.8	94.6	85.6	80.8	93.1	87.9	77.1
T2	99.5	99.3	99.5	99.5	99.3	99.5	99.6	99.6	99.9
T3	-	99.4	99.5	-	99.4	99.5	-	99.6	99.9
T4	-	-	99.5	-	-	99.5	-	-	99.9
TAll	94.6	85.6	80.8	93.6	86.6	83.8	93.1	87.9	77.1

Table 4: CL accuracies averaged over 10 runs for EXP6-8 computed when using VAEs, CVAEs and Wasserstein-GANs as generators. Accuracies on sub-tasks are computed after the last sub-task is completed.

architecture for the generators, which is however powerful enough to generate MNIST data when trained on the whole dataset.

From the result that performance degrades even if new sub-tasks consist exclusively of known samples, we conclude that generative replay needs to be complemented by generative outlier detection method that can detect known (an unknown) data automatically. In this way, training could be conducted selectively on unknown data only, thus minimizing the damage to existing knowledge by re-training.

References

[1] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[2] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[3] Benedikt Pfülb and Alexander Gepperth. A comprehensive, application-oriented study of catastrophic forgetting in DNNs. In *International Conference on Learning Representations (ICLR)*, 2019.

[4] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[5] B Pfülb and A Gepperth. Overcoming catastrophic forgetting with Gaussian mixture replay. In *International Joint Conference on Neural Networks(IJCNN)*, 2021.

[6] Yifan Chang, Wenbo Li, Jian Peng, Bo Tang, Yu Kang, Yinjie Lei, Yuanmiao Gui, Qing Zhu, Yu Liu, and Haifeng Li. Reviewing continual learning from the perspective of human-level intelligence. *arXiv preprint arXiv:2111.11964*, 2021.

[7] Khadija Shaheen, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. *arXiv preprint arXiv:p2105.12374*.

[8] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2021.

[9] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, 2017.

[10] Benedikt Bagus and Alexander Gepperth. An investigation of replay-based approaches for continual learning. In *International Joint Conference on Neural Networks (IJCNN)*, 2021.

[11] Timothée Lesort, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Andrei Stoian, and David Filliat. Generative models from the perspective of continual learning, 2019.

[12] Timothée Lesort, Andrei Stoian, Jean François Goudou, and David Filliat. Training discriminative models to evaluate generative ones. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11729 LNCS:604–619, 2019.

[13] David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Hendrik Strobelt, Bolei Zhou, and Antonio Torralba. Seeing what a GAN cannot generate. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4502–4511, 2019.