Deep Generative View on Continual Learning Report

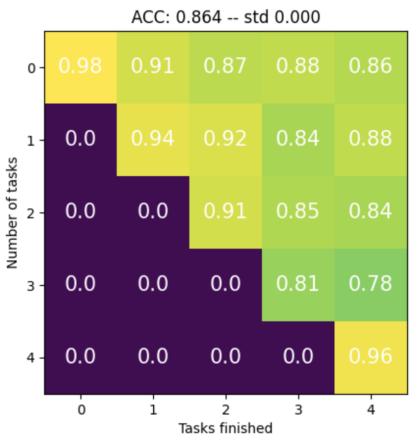
Andrea Giuseppe Di Francesco, Farooq Ahmad Wani

1 Methodology

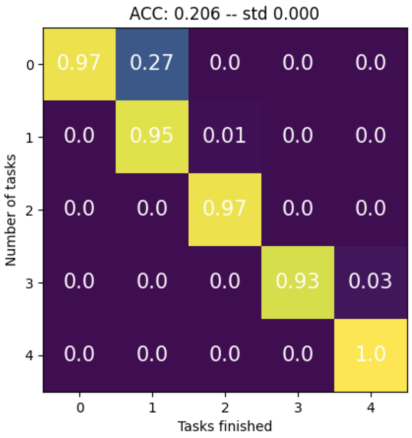
The method that we used to conduct the experiments is inspired from [1]. In our implementation, we trained a Generative Adversarial Network, our *generator*, to reconstruct MNIST images. Having at our disposal the generator we could then train our *solver* model, a simple multi-layer perceptron (MLP), both on new task images as well as replayed samples from the generator. This approach is really helpful, since we do not have to store past data, and it also aligns better with real world scenarios, where past data may be not available or is private.
We evaluate our method against a vanilla replay approach, where we past data is stored and the model is trained on it. The metrics that we evaluate are the average accuracy, forward and backward transfer, and are reported in Table 1. We also report the task confusion matrix from both the scenario in Figure 1. We can notice easily, how the vanilla approach is superior in terms of performance. However, we attribute these significant gap to the quality of the generated images. In fact, in our implementation we were not able to generate sufficiently representative samples, and we are not as successful as the Vanilla approach. On the other side, in Vanilla experiments, even though we have a total number of parameters that is order of magnitudes lower than the Generative experiments, we notice that the overall memory storage within the gpu is lower w.r.t. Vanilla. This is due to the storage of past samples. So in terms of efficiency, here we stress the advantage of our approach. In Figure 2, we show how our samples are generated, and what we refer to as low quality. Additional details on the hyperparameters and the training strategy can be found in the code attached to the assignment.

Baselines	Avg. Acc	FWT	BWT	n_params	GPU usage (MB)
Vanilla Replay	0.864	-0.101	-0.063	44,860	4863
Generative Replay	0.206	-0.13	-0.949	1,159,629	3534

Table 1: Evaluation metrics on Vanilla and Generative replay methods for continual learning.

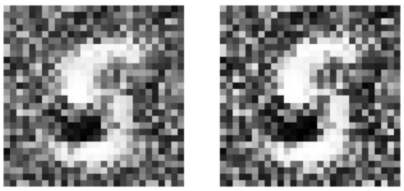


(a) Task confusion matrix in the case of vanilla replay.

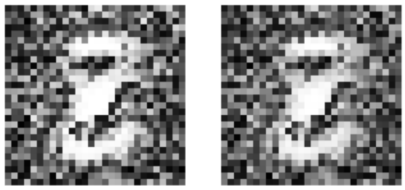


(b) Task confusion matrix in the case of generative replay

Figure 1: In these figures we note how generative replay is less effective than vanilla.



(a) Low quality generated samples for 5.



(b) Low quality generated samples of 8.

Figure 2: Low quality inputs generated through a Generative Adversarial Network.

2 Answers

1. Are the memory and computational requirements of the two methods comparable?
2. How would that solution scale with n tasks?
3. What are the downsides of your approach?
4. What are your ideas for making your solutions more memory and computationally efficient?

1. As already specified the number of parameters in the two settings is very different. Assumed that the solver model (e.g. the one that we want to continually train), retains the same number of parameters in both cases, choosing a

generative approach involves also a Generator and a Discriminator network to be trained. In this way, the overall number of parameters will be always higher w.r.t. vanilla. Details on these are in Table 1. However, generative methods have a clear advantage in terms of GPU memory, since we do not need to store data incrementally.

2. Since we are in a **class incremental learning** setting, we can assume that the last classifier head has $n_1 \times n$ parameters, where n is the number of parameters, and n_1 is a fixed constant. If we increase n, we would have a linear increase in the computational complexity related to to number of parameters. As concerns the Discriminator and the Generator, adding more tasks would not affect their complexity directly. So in both vanilla and generative approaches we would only experience a linear increase in the number of parameters. However, adding more tasks, could increase the overall complexity of the architectures under another perspective. In fact, it should be necessary designing more complex architectures because of the task complexity, requiring a more powerful solver, generator and discriminator. In this case the network's changes would not be limited to the last classifier's head.

3. The downsides of our approach are related to the quality of the generated images. In fact, this aspect, hinders the model from replaying reliable samples and avoiding catastrophic forgetting.

4. Generally, to improve this pipeline we would change our architecture from MLPs to Convolutional Neural Networks, being this more efficient in terms of parameters (thanks to the parameter sharing properties) and also in terms of learning. Given also the nature of the task, that is image classification, a Convolutional Neural Network is known to be better in handling these tasks.

3 Contributors

The authors of this work, have equally contributed to its realization.

References

[1] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," 2017.