PhD in Data Science Assignment

Deep Generative View on Continual Learning

Data Science Doctoral Program **XXXVIII cycle** - Student: **Angela Pomaro**

Assignment Report: Class-Incremental Learning with GANs

Link to GoogleColab:
https://colab.research.google.com/drive/1rvuGUCV3NYDozhpkg2li56oMwD_qY3hQ?usp=sharing
Link to GitHub repository:
https://github.com/angelapomaro/CL-course_AP.git

The assignment aims to create a generative replay method for continual learning. I have implemented the project by modifying the code developed during our lab sessions. I used Generative Adversarial Networks (GANs) [1] to implement a class-incremental learning (CIL) framework on the MNIST [3] dataset. As requested, I evaluated the method in the class-incremental scenario with five splits of two classes each. This dataset consists of a number of classes equal to 10, consisting in 60.000 training and 10.000 testing grayscale images of handwritten digits from 0 to 9, and it offers a common benchmark for evaluating generative and incremental learning models.
The MNIST dataset was split into five parts, each containing two classes, to set a class-incremental learning scenario. This approach allows for incremental training where the model is updated sequentially with new classes while retaining knowledge of previously learned classes without revisiting the original training data using GANs.

The generative model setup consists on a GAN implementation to generate synthetic data for each split, then train the GAN on the first split and sequentially update it with the following splits. The class-incremental learning (CIL) scenario uses the GAN to generate data for previous classes while training on new data. The purpose is to update the model with new class data while retaining knowledge of earlier classes without re-accessing the entire dataset.
The model is iteratively trained and evaluated, measuring key metrics such as average accuracy, forward transfer, and backward transfer to assess its performance over time.

This report further explores the memory and computational requirements of generative replay and vanilla replay approaches with raw samples, analyzes their scalability, and discusses potential downsides and improvements to enhance the solution's efficiency.

The adopted Generative Model Setup comprises of the following GAN Architecture:

a) Generator, characterized by:
   - Input: Latent vector of size 100.
   - Output: Image vector of size 784 (28x28).
   - Layers: 4 fully connected layers with ReLU activations, ending with a Tanh activation.

b) Discriminator, characterized by:
   - Input: Image vector of size 784.
   - Output: Single scalar representing real or fake.
   - Layers: 3 fully connected layers with LeakyReLU activations and dropout for regularization.

The Classifier Architecture consists of:

- Input: Image vector of size 784.
- Layers: 2 fully connected layers with ReLU activation.
- Output: Number of classes learned so far.

The chosen Training Process Hyperparameters are:

- **Learning Rate**: 0.0002. For GAN training a lower learning rate (0.0002) was chosen to ensure stable training and convergence, , especially with multiple tasks to prevent catastrophic forgetting.
- **Batch Size**: 100. This size was chosen to balance between computational efficiency and stable gradient updates.
- **Training Epochs**: 10 for each task. This number of epochs provides sufficient time for convergence on the incremental tasks without overfitting.
- **Number of class-incremental tasks**: 5
- **Generated Samples per Task**: 1000 samples per class to balance training data for previous tasks, while ensuring adequate representation of each class for training the classifier with adequate memory requirements and use.

The adopted evaluation metrics are described herewith.

1) **Average Accuracy (ACC)**: Mean classification accuracy across all classes learned. This metric is calculated by evaluating the performance of the model on each task after training on subsequent tasks, offering a measure of the method's capability to maintain knowledge across different tasks.

The proposed code results are listed below:

```
Training on task 1/5...
Accuracy on task 1: 99.91%

Training on task 2/5...
Accuracy on task 1: 25.77%
Accuracy on task 2: 99.46%

Training on task 3/5...
Accuracy on task 1: 10.78%
Accuracy on task 2: 33.79%
Accuracy on task 3: 99.79%

Training on task 4/5...
Accuracy on task 1: 9.65%
Accuracy on task 2: 21.65%
Accuracy on task 3: 7.52%
Accuracy on task 4: 99.80%

Training on task 5/5...
Accuracy on task 1: 0.05%
Accuracy on task 2: 6.61%
Accuracy on task 3: 0.75%
Accuracy on task 4: 1.96%
Accuracy on task 5: 99.39%

Final Average Accuracy per Task:
Task 1: 99.91%
Task 2: 62.61%
Task 3: 48.12%
```

Task 4: 34.65% Task 5: 21.75% Forward Transfer: -78.23% Backward Transfer: -39.57%
Tab. 1 – Results of code final version

For each new task, the model achieves high accuracy on the current task, indicating that the model is learning the new tasks effectively.
Nonetheless the model exhibits significant forgetting of previously learned tasks. See for instance how after learning on Task 3, accuracy on Task 2 drastically drops to 24,88%.
After Task 5 the accuracy on the first tasks is almost entirely lost.
The results were obtained by introducing a list to store the synthetic dataset for each previous task, adjusting the combination of real and synthetic data and fine-tuning the classifier on a combined dataset from all previous tasks after training on new task.

Before this improvements, the model's average accuracy results performed as follows:

```
Training on task 1/5...
Accuracy on task 1: 99.95%

Training on task 2/5...
Accuracy on task 1: 0.00%
Accuracy on task 2: 99.27%

Training on task 3/5...
Accuracy on task 1: 0.00%
Accuracy on task 2: 0.00%
Accuracy on task 3: 99.68%

Training on task 4/5...
Accuracy on task 1: 0.00%
Accuracy on task 2: 0.00%
Accuracy on task 3: 0.00%
Accuracy on task 4: 99.80%

Training on task 5/5...
Accuracy on task 1: 0.00%
Accuracy on task 2: 0.00%
Accuracy on task 3: 0.00%
Accuracy on task 4: 0.00%
Accuracy on task 5: 99.04%

Final Average Accuracy per Task:
Task 1: 99.95%
Task 2: 49.63%
Task 3: 33.23%
Task 4: 24.95%
Task 5: 19.81%

Forward Transfer: -80.18%
```

Backward Transfer: -32.13%
Tab. 2 – Results of code old version

This metric was computed across all tasks. In the old version code, the accuracy for tasks significantly drops after their initial training, indicating that the model's performance degrades progressively with the addition of new tasks. This trend is typical in scenarios where catastrophic forgetting occurs.

2) **Forward Transfer (FWT)**: This metric offers a measure of how learning a new task affects the learning of future tasks.
   In the proposed code, the result for the forward transfer metric is highly negative (-78,23%), which means that learning new tasks negatively affects the performance on the tasks that are introduced afterwards. This large negative value indicates that each new task learning erases the knowledge of previous tasks, preventing positive knowledge transfer.
3) **Backward Transfer (BWT)**: This metric offers a measure of the impact of learning a new task on the performance of previously learned tasks.
   In the proposed code, the result for the backward transfer metric is also negative (-39,57%), though smaller that FWT, which means that learning new tasks significantly deteriorates the model's performance on prior tasks. Ideally, the value should be positive or closer to zero to indicate good retention of past knowledge.

In general, the model is still suffering from catastrophic forgetting, with reference to the older version, despite the introduced changes. This may suggest that the current generative replay method alone is insufficient to retain knowledge of previous tasks effectively. In other terms, the synthetic data generated by GANs might not be sufficiently representative of the original tasks, with consequent poor retention of earlier tasks. A possible further improvement may consider adopting Elastic Weight Consolidation (EWC) in the training loop to prevent high weights for previous tasks from changing or improving generative replay by introducing conditional GANs, which may contribute to improve the quality and relevance of synthetic data.

In terms of memory requirements, the Generative Replay with GANs:

- requires storing the GAN model parameters and generating synthetic data on-the-fly;
- is proportional to the model size, independently of the number of tasks;
- shows a reduced memory footprint by using synthetic data, which are generated only when needed, instead of storing all past data, which means it does not require significant memory storage. Only temporary storage is needed during the generation process and training phases.

while the Vanilla Replay with raw samples increases linearly with the number of tasks because it requires storing raw samples for all learned classes.
Therefore, about memory usage, Generative Replay with GANs is more efficient than Vanilla Replay with raw samples, which scales linearly in terms of data storage but has higher absolute memory usage due to the need to store raw samples. Memory efficiency grows with the number of tasks.

In terms of computational requirements, the Generative Replay with GANs' computational costs include training the GAN, which can be computationally expensive due to the adversarial nature and multiple epochs required for convergence and, as mentioned above, generating synthetic data, which is less expensive than training and can be done relatively quickly once the GAN is trained. The required resources are linear dependent on the number of tasks due to sequential GAN training and generation steps. For Vanilla Replay, computational costs involve data handling and training time on progressively increasing amounts of stored data. It is then analogously linear dependent on the number of tasks. Therefore, it may suffer from memory limitations as the dataset grows.

In terms of computational requirements, Generative Reply pays a higher computational cost in the GAN training phase, lower for data storage, while Vanilla Replay, while computationally simpler per task, is subjected to increasing computational costs with the dataset size growth.
In terms of scalability, both models scale linearly with the number of tasks in terms of computational complexity, but Generative Replay can be more computationally intensive due to the need for both training the model and data generation.

How Would That Solution Scale with n Tasks?
For Generative Replay with GANs, memory scaling is nearly constant at it scales linearly, with a low slope, being limited to storing model and temporary data, while computational requirements scale linearly with the number of tasks, as each task requires separate GAN training and data generation.
In the case of Vanilla Replay with raw samples, instead, both memory and computation requirements scale linearly, as it shows a higher memory increase per task, with reference to GANs.

The downsides of the GANs approach resides mainly on the following aspects:

- **Training complexity and instalbility**: Training GANs is challenging as it can be unstable and requires careful tuning of hyperparameters to achieve convergence.
- **Synthetic data quality and diversity**: The quality of generated data may vary, potentially leading to poor representation of past classes.
- **Computational cost**: GANs require considerable computational resources for training, which can be a limit for large datasets or high number of tasks.
- **Scalability challenges**: The linear scaling of computational and memory requirements may be impractical for large number of tasks, especially in environments with limited resources.

Possible improvements may include actions to tackle memory and computational efficiency, such as using data compression techniques to store more efficiently synthetic data or incrementally update GANs to save computational time in the training phase. Also, parallel and distributed computing techniques may be introduced to distribute both GAN training and data generation across multiple GPUs or nodes. Another possibility is to optimize the data generation process by generating only the samples needed, avoiding redundancy or by generating a small number of high-quality samples representative of the data distribution of each class, also by means of data augmentation techniques.
By addressing these actions, the generative replay approach can be made more scalable, efficient and suitable for real-world applications.

Using GANs for generative replay in class-incremental learning provides a memory-efficient solution compared to storing raw samples. The method scales linearly in terms of memory and computational requirements with the number of tasks, making it suitable for scenarios where memory is the limiting boundary. However, the approach requires careful handling of GAN training complexity and computational overhead. Future work could focus on enhancing the stability and efficiency of GAN training to further optimize this approach.