Continual Learning

Baselines and Replay

Antonio Carta

antonio.carta@unipi.it

Outline

CL Strategies

- categorization
- Components and design choices
- Simple baselines
- Rehearsal methods

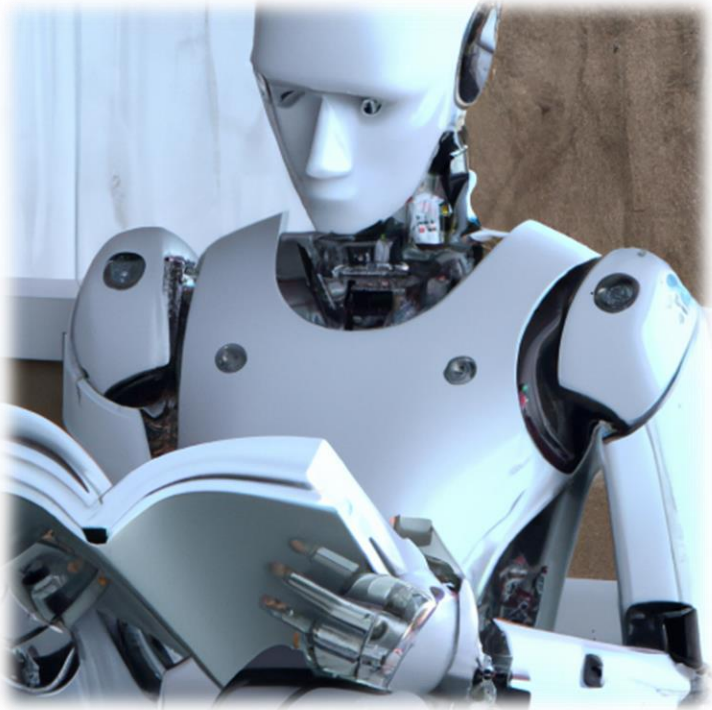


Image from Dall-e

CL Strategy

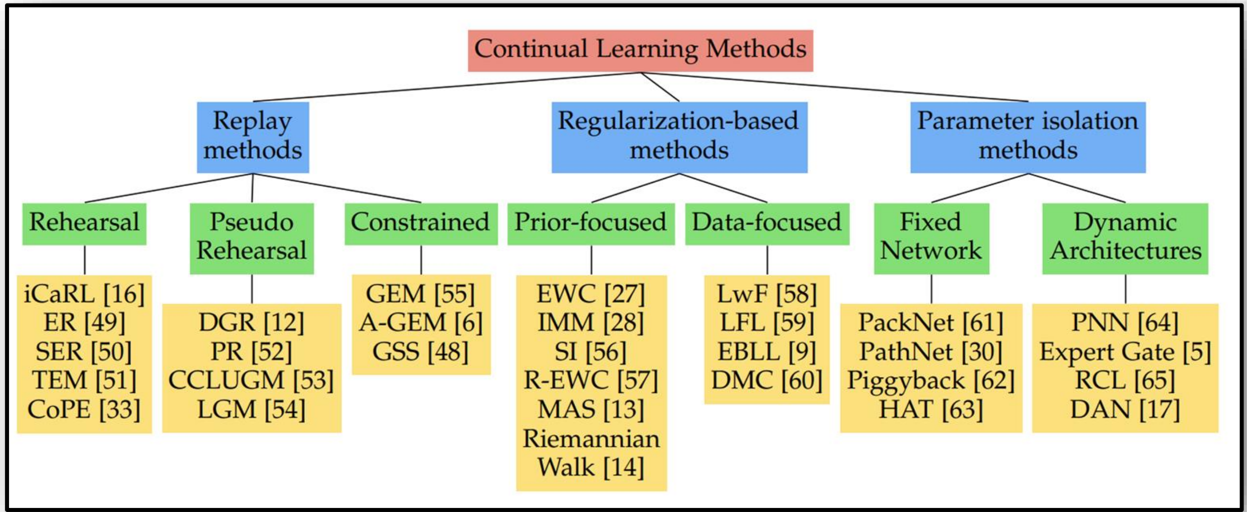
- A learning method designed for Continual Learning
- Typically a combination of naive finetuning plus some CL specific component
- Formal Definition:

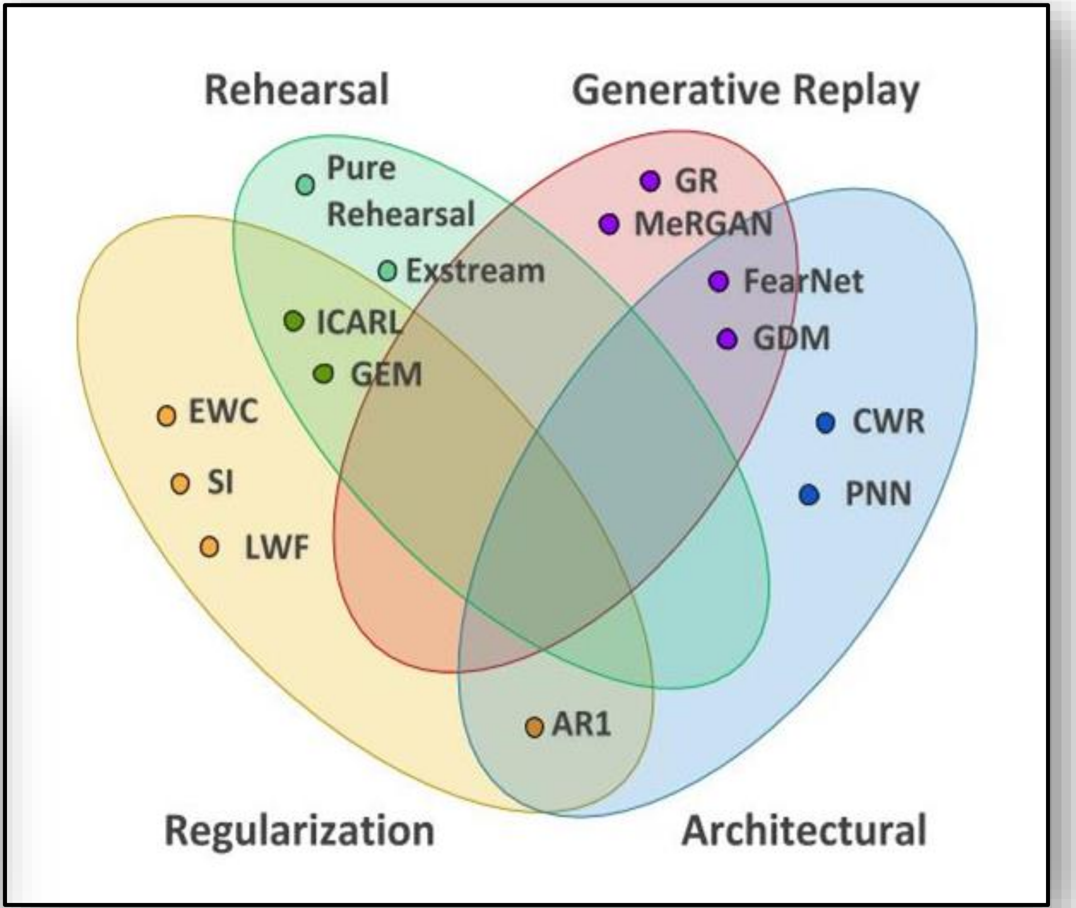
$$\mathcal{A}^{CL}: \langle \underbrace{f_{i-1}^{CL}}_{\text{previous model}}, \underbrace{\mathcal{D}_{train}^{i}}_{\text{new data}}, \underbrace{\mathcal{M}_{i-1}}_{\text{exemplars buffer}}, \underbrace{t_i}_{\text{(optional) task label}} \rangle \rightarrow \langle \underbrace{f_i^{CL}}_{\text{new model}}, \underbrace{\mathcal{M}_i}_{\text{new buffer}} \rangle$$

Carta, Antonio, et al. "Ex-Model: Continual Learning from a Stream of Trained Models." CLVISION Workshop @ CVPR2022.

CL Strategy Categorization

- **Replay**: store sample and revisit them.
- **Regularization**: penalize forgetting.
- **Parameter-Isolation/Architectural**: separate task-specific parameters



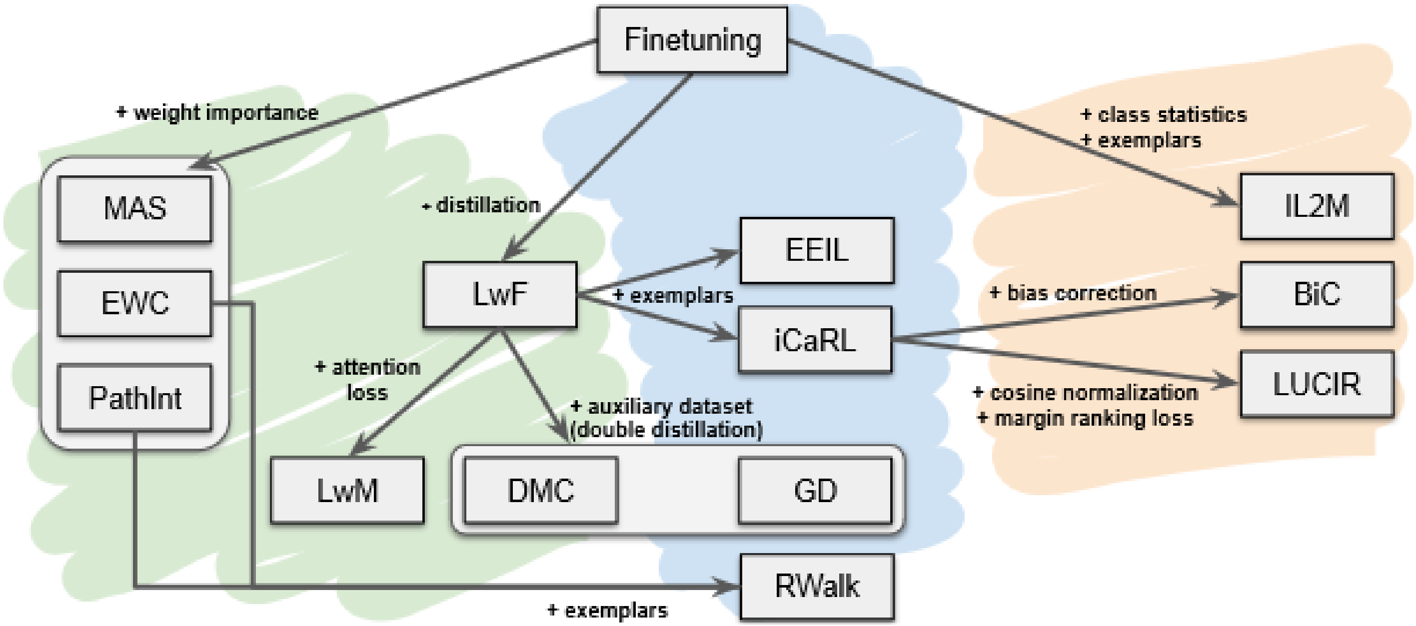


Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges, Lesort et al. Information Fusion, 2020.
A continual learning survey: Defying forgetting in classification tasks. De Lange et al, TPAMI 2021.

CL Strategy Components

CL methods can be combined together

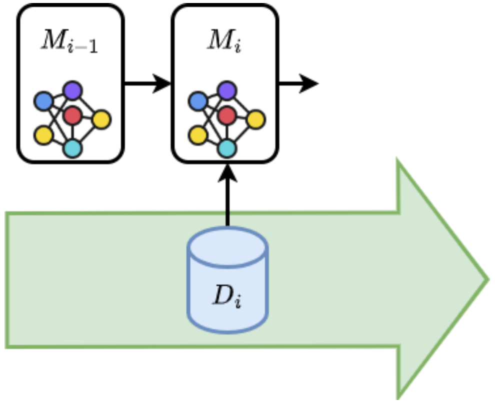
- Regularization +
- Replay +
- Architectural +
- Bias correction: methods for output layer



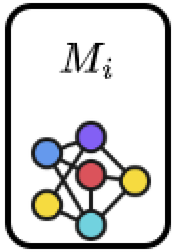
Masana, Marc, et al. "Class-incremental learning: survey and performance evaluation on image classification." TPAMI

Baselines: Finetuning

Train: sequential SGD, each time using only the current data.



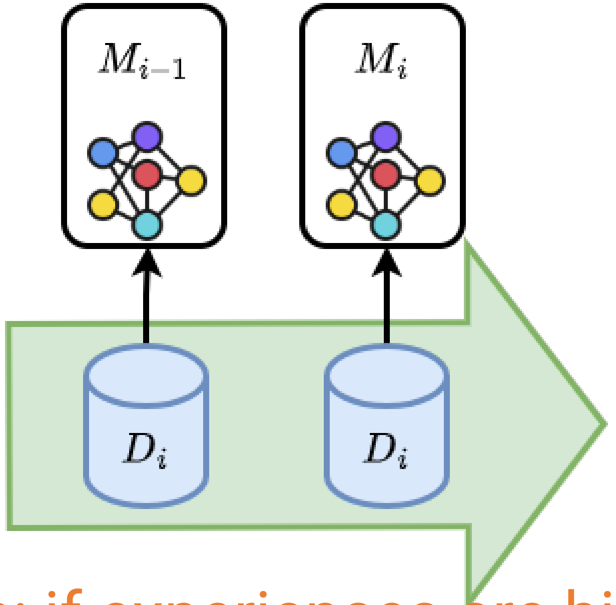
Inference: use last model (M_i)



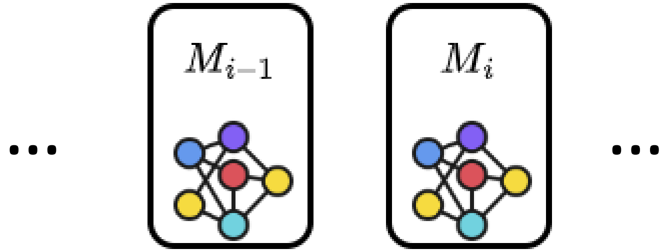
Note: Naive finetuning often results in catastrophic forgetting. CL methods should always beat the Naive baseline

Baselines: Ensemble

Training: Train one model for each experience. Each model is completely independent



Inference: Compute output using the correct model (assume oracle if task labels are not available).

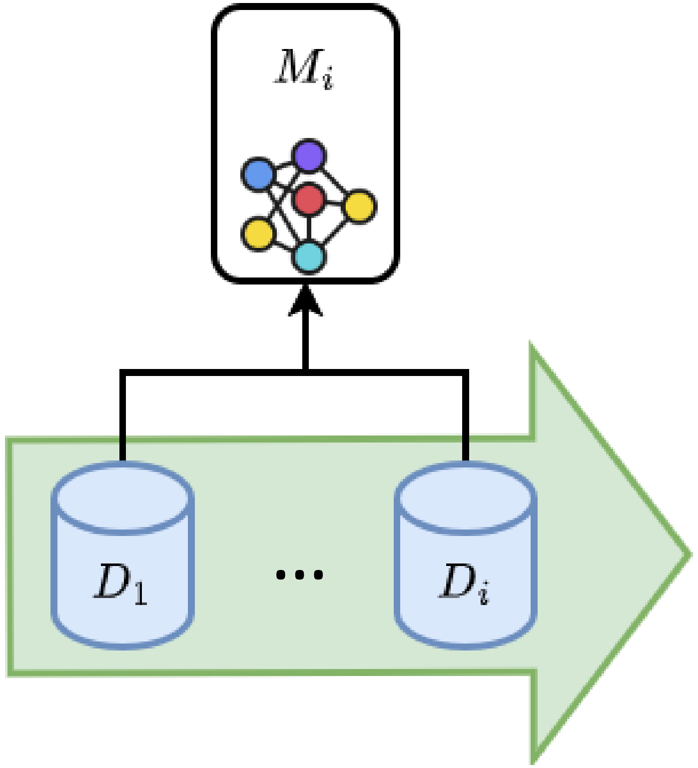


Note: if experiences are big enough, it may be hard to beat (especially without task labels).

Baselines: Joint Training

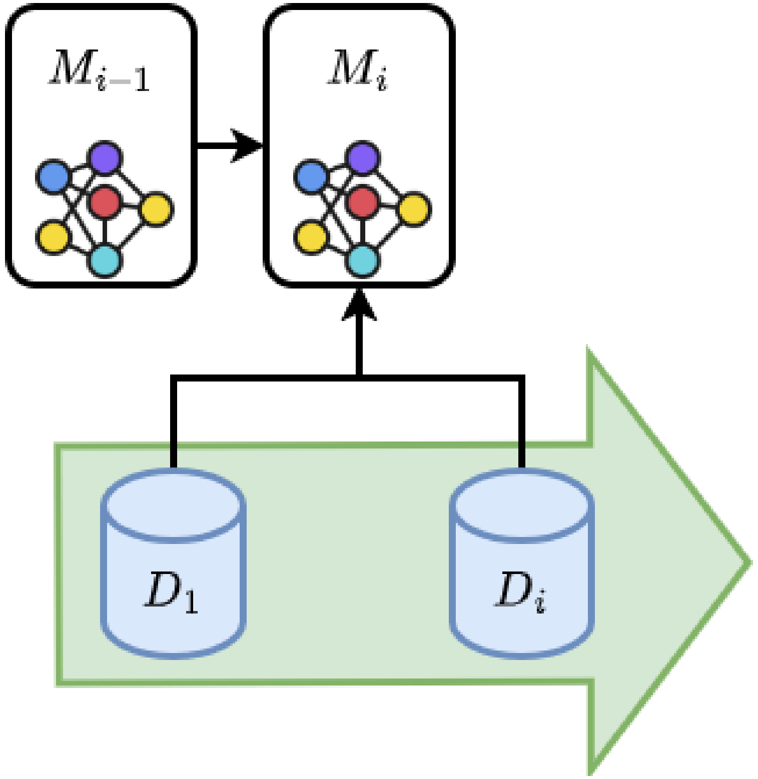
JointTraining / Offline: Concatenate all the data (keeping task labels) and train *starting from a random initialization*.

Sometimes referred as the **upper bound** (incorrect).



Baselines: Cumulative

Training: for every experience, accumulate all data available up to now ($\bigcup_{k=0}^{i} D_k$) and re-train *starting from the previous model*.



General rules

- If we have enough data, starting from scratch achieves a slightly higher performance than starting from the previous model.
- Starting from the previous model achieves faster convergence than training from scratch.

Baselines

We will use as lower bound

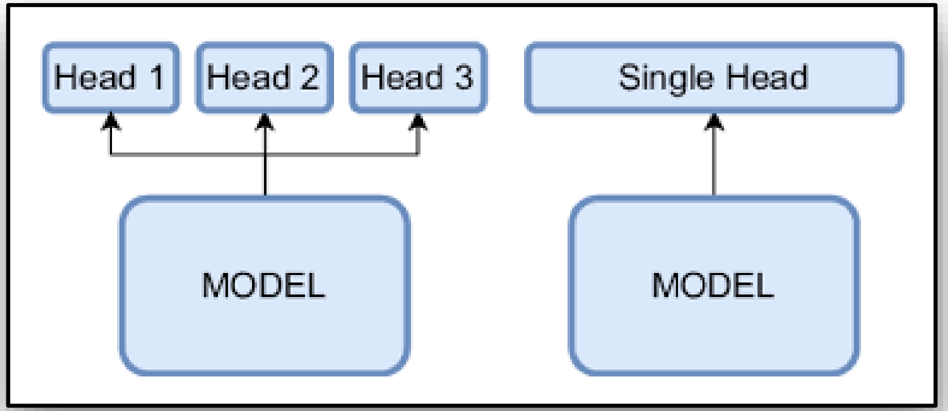
- Naive Finetuning

We will use as upper bound

- Ensemble
- Joint
- Cumulative

Basic Design Choices

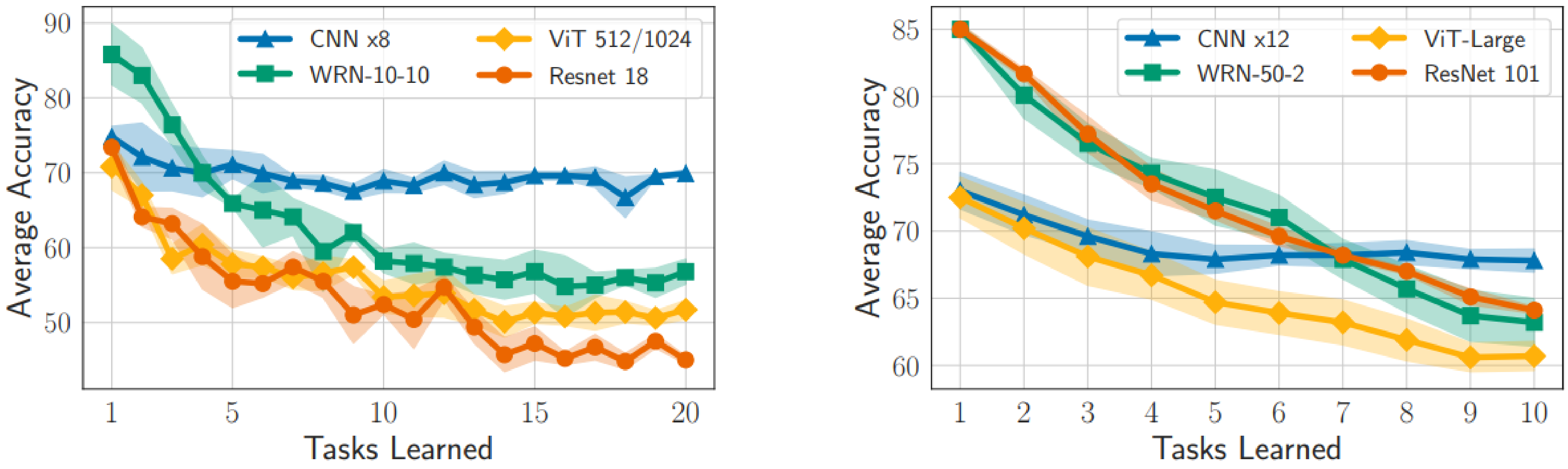
- **Pretraining**:
  - Always helpful if available
  - The literature suggests that early phases of training are critical. If the model only sees a small set of highly correlated samples in the first epochs, it may not be able to recover the performance later.
- **Model Architecture**:
  - CNN vs Transformers
  - Batch Normalization
  - Regularization: Dropout?
  - Wide vs Deep networks



Multi-Head vs Single-Head

Continual Learning for Recurrent Neural Networks: an Empirical Evaluation. Cossu et al, 2021.

Model Architecture in CL



(a) Split CIFAR-100 (b) Split ImageNet-1K

Figure 2: Evolution of average accuracy for various architectures on (a) Split CIFAR-100: CNNs have smaller forgetting than other architectures while WideResNets have the highest learning accuracy, and (b) Split ImageNet-1K WideResNets and ResNets have higher learning accuracy than CNNs and ViTs. However, the latter has smaller forgetting.

Mirzadeh, Seyed Iman, et al. "Architecture matters in continual learning." arXiv preprint arXiv:2202.00275 (2022).

Width and Depth

Benchmark	Model	Depth	Params (M)	Average Accuracy	Average Forgetting	Learning Accuracy
Rot MNIST	MLP-128	2	0.1	70.8 $\pm$0.68	31.5 $\pm$0.92	96.0 $\pm$0.90
Rot MNIST	MLP-128	8	0.2	68.9 $\pm$1.07	35.4 $\pm$1.34	97.3 $\pm$0.76
Rot MNIST	MLP-256	2	0.3	71.1 $\pm$0.43	31.4 $\pm$0.48	96.1 $\pm$0.82
Rot MNIST	MLP-256	8	0.7	70.4 $\pm$0.61	32.1 $\pm$0.75	96.3 $\pm$0.77
Rot MNIST	MLP-512	2	0.7	72.6 $\pm$0.27	29.6 $\pm$0.36	96.4 $\pm$0.73
CIFAR-100	CNN x4	3	2.3	68.1 $\pm$0.5	8.7 $\pm$0.21	76.4 $\pm$6.92
CIFAR-100	CNN x4	6	5.4	62.9 $\pm$0.86	12.4 $\pm$1.62	77.7 $\pm$5.49
CIFAR-100	CNN x8	3	7.5	69.9 $\pm$0.62	8.0 $\pm$0.71	77.5 $\pm$6.78
CIFAR-100	CNN x8	6	19.9	66.5 $\pm$ 1.01	10.7 $\pm$1.19	76.6 $\pm$4.78
CIFAR-100	ViT 512/1024	2	4.6	56.4 $\pm$1.14	15.9 $\pm$0.95	68.1 $\pm$7.15
CIFAR-100	ViT 512/1024	4	8.8	51.7 $\pm$1.4	21.9 $\pm$1.3	71.4 $\pm$5.52

Tab. 3 shows that across all architectures, over-parametrization through increasing width is helpful in improving the continual learning performance as evidenced by lower forgetting and higher average accuracy numbers. For MLP, when the width is increased from 128 to 512, the performance in all measures improves. However, for both MLP-128 and MLP-256 when the depth is increased from 2 to 8 the average accuracy is reduced, and the average forgetting is increased with a marginal gain in learning accuracy. Finally, note that MLP-256 with 8 layers has roughly the same number of parameters as the MLP-512 with 2 layers. However, the wider one of these two networks has a better continual learning performance.

Mirzadeh, Seyed Iman, et al. "Architecture matters in continual learning." arXiv preprint arXiv:2202.00275 (2022).

Global Pooling

Table 5: Role of Global Average Pooling (GAP) for Split CIFAR-100: related to our arguments in Sec. 3.1, adding GAP to CNNs significantly increases the forgetting. Later, we show that removing GAP from ResNets can also significantly reduce forgetting as well.

Model	Params (M)	Pre-Classification Width	Average Accuracy	Average Forgetting	Learning Accuracy	Joint Accuracy
CNN x4	2.3	**8192**	68.1 ±0.5	8.7 ±0.21	76.4 ±6.92	73.4 ±0.89
CNN x4 + GAP	1.5	**512**	60.1 ±0.43	14.3 ±0.8	66.1 ±7.76	76.9 ±0.81
CNN x4 (16x) + GAP	32.3	**8192**	73.6 ±0.39	5.2 ±0.66	75.6 ±4.77	77.9 ±0.37
CNN x8	7.5	**16384**	69.9 ±0.62	8.0 ±0.71	77.5 ±6.78	74.1 ±0.83
CNN x8 + GAP	6.1	**1024**	63.1 ±2.0	14.7 ±1.68	70.1 ±7.18	78.3 ±0.97
CNN x16	26.9	**32768**	76.8 ±0.76	4.7 ±0.84	81.0 ±6.97	74.6 ±0.86
CNN x16 + GAP	23.8	**2048**	66.3 ±0.82	12.2 ±0.65	72.3 ±6.02	78.9 ±0.27

NOTE: Global average pooling (GAP) is typically used in the final layer to reduce the number of features.

Mirzadeh, Seyed Iman, et al. "Architecture matters in continual learning." arXiv preprint arXiv:2202.00275 (2022).

Replay

Class-Incremental Learning with Replay

Setting: virtual drift with new classes at each experience. No repetitions in the stream.

***Problem**: How do we remember the old classes?*

We will see that in general this is a very hard problem if we never revisit the previous data.

Replay stores a limited set of samples from the previous experiences and use them for rehearsal.

$$\mathcal{A}^{CL} : \langle f_{i-1}^{CL}, \mathcal{D}_{train}^{i}, \boxed{\mathcal{M}_{i-1}}, t_i \rangle \rightarrow \langle f_i^{CL}, \boxed{\mathcal{M}_i} \rangle$$

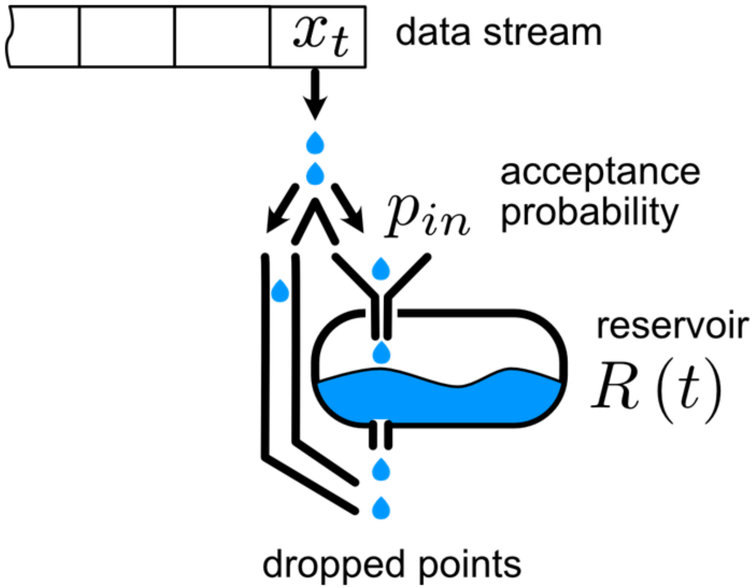


Image from https://towardsdatascience.com/reservoir-sampling-for-efficient-stream-processing-97f47f85c11b

Replay

Good News: Replay is a simple, general and effective strategy for CL.

- Approximates an i.i.d distribution
- Approximate cumulative training
- Relatively cheap in terms of computations

Bad News:

- Memory limitations or privacy constraints
- Scaling: for long streams we may need to store a large buffer. Memory increases over time

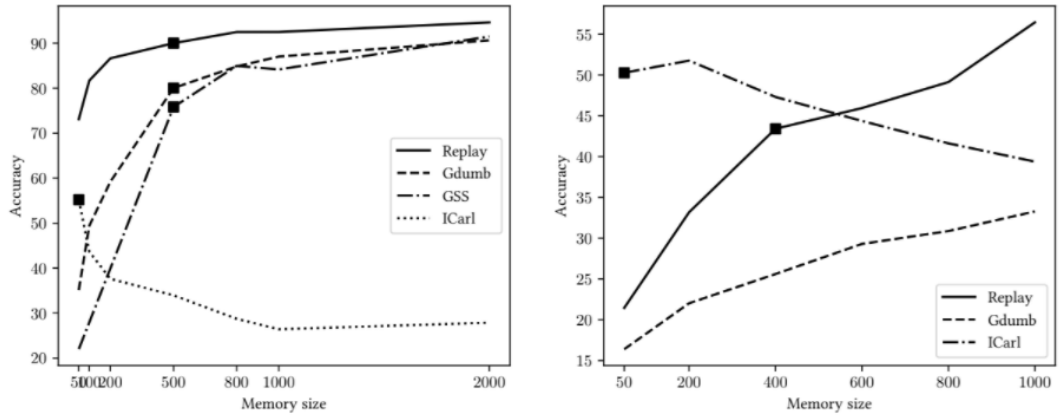


Figure 5.2: Split-MNIST memory-accuracy curve

Figure 5.3: Split-CIFAR-10 memory-accuracy curve

Replay Algorithm

Parameters: memory size

During training (finetuning + rehearsal):

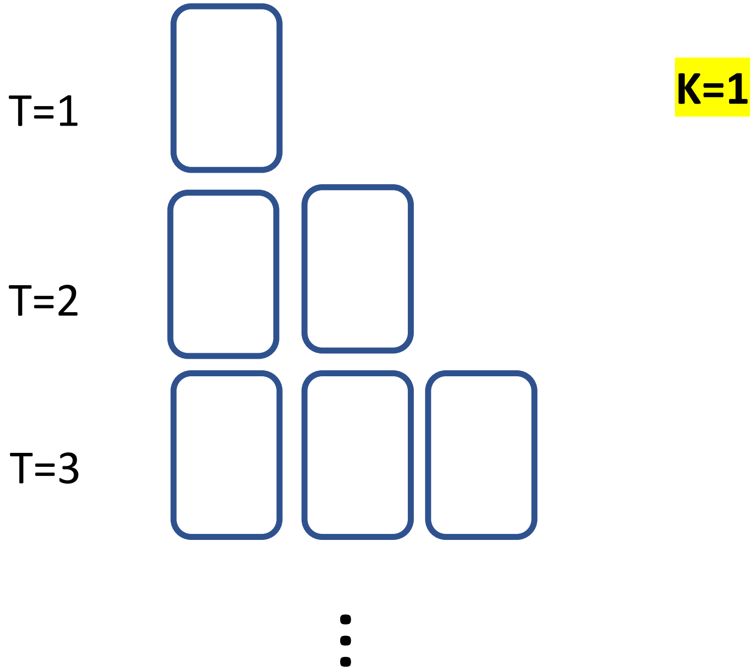
- Sample from the current data
- Sample from the buffer using sampling policy
- Do SGD step on the concatenated mini-batch

After each experience (buffer update):

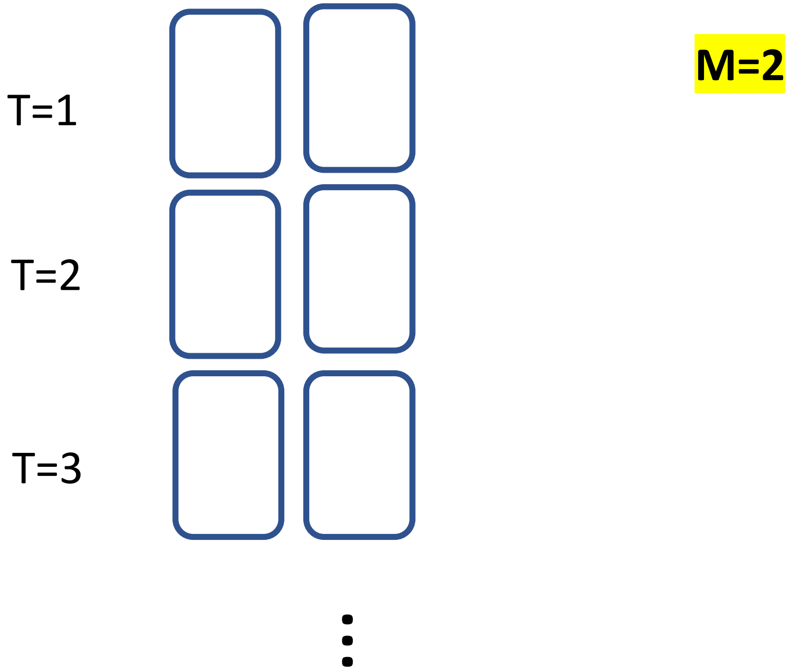
- Use insertion policy to choose data from the current experience
- Add example to the buffer
- Use removal policy if the buffer is too big

Growing vs Fixed Memory

Growing Memory: each experience adds k examples. Unbounded growth.



Fixed Memory: Maximum memory size M. Requires a removal policy.



Insert / Remove / Sample Policies

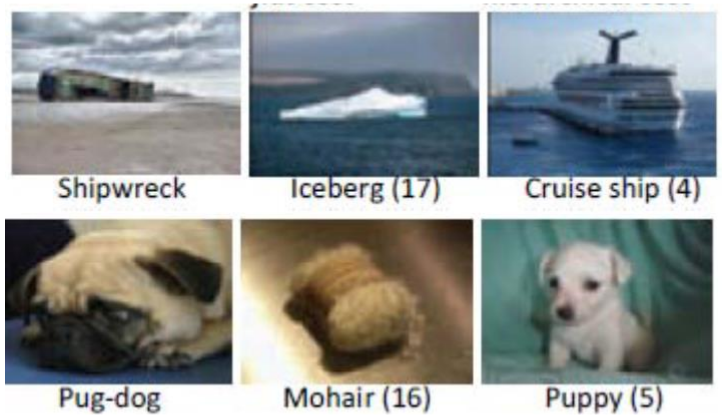
- **Insertion**: find the best examples to store in the buffer
- **Remove**: find the least useful example to remove them from the buffer
- **Sample**: find the best examples to use for the current training iteration

Data Balancing: Ideally, data should be balanced (approximated i.i.d.). Class/task/experience balancing are common choices.

Real vs Synthetic Samples

Real examples: taken from the original data.

Synthetic examples: sampled from a generative model. Neuroplausible but unfortunately CL of generative models is quite difficult and still largely unsolved.



Carta, Antonio, et al. "Ex-Model: Continual Learning from a Stream of Trained Models." CLVISION Workshop @ CVPR2022.
DeepInversion: https://github.com/NVlabs/DeepInversion

Input vs Latent Replay

Input Replay: store input images

Latent Replay: store latent representations at intermediate layers

We will see an example at the end of the lecture.

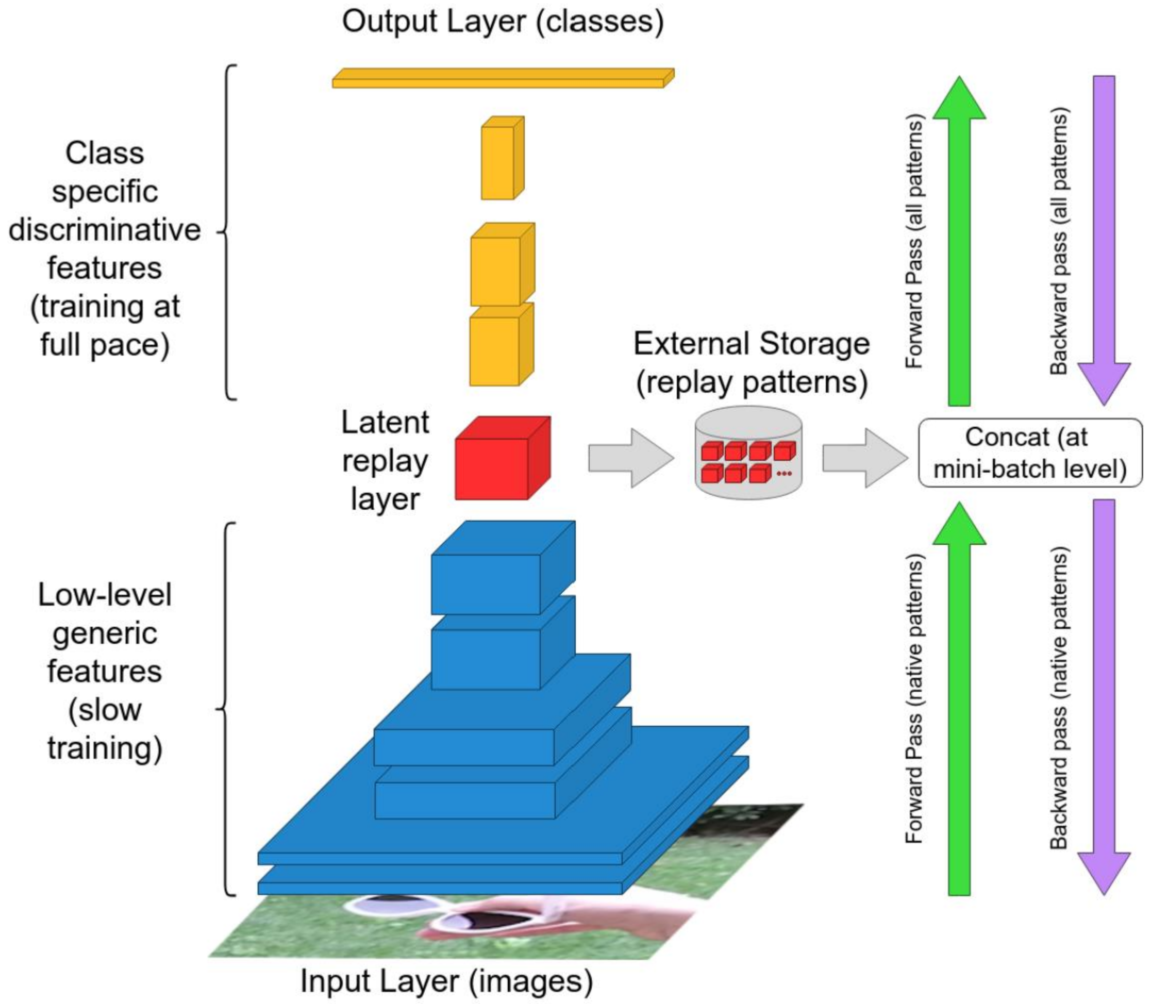


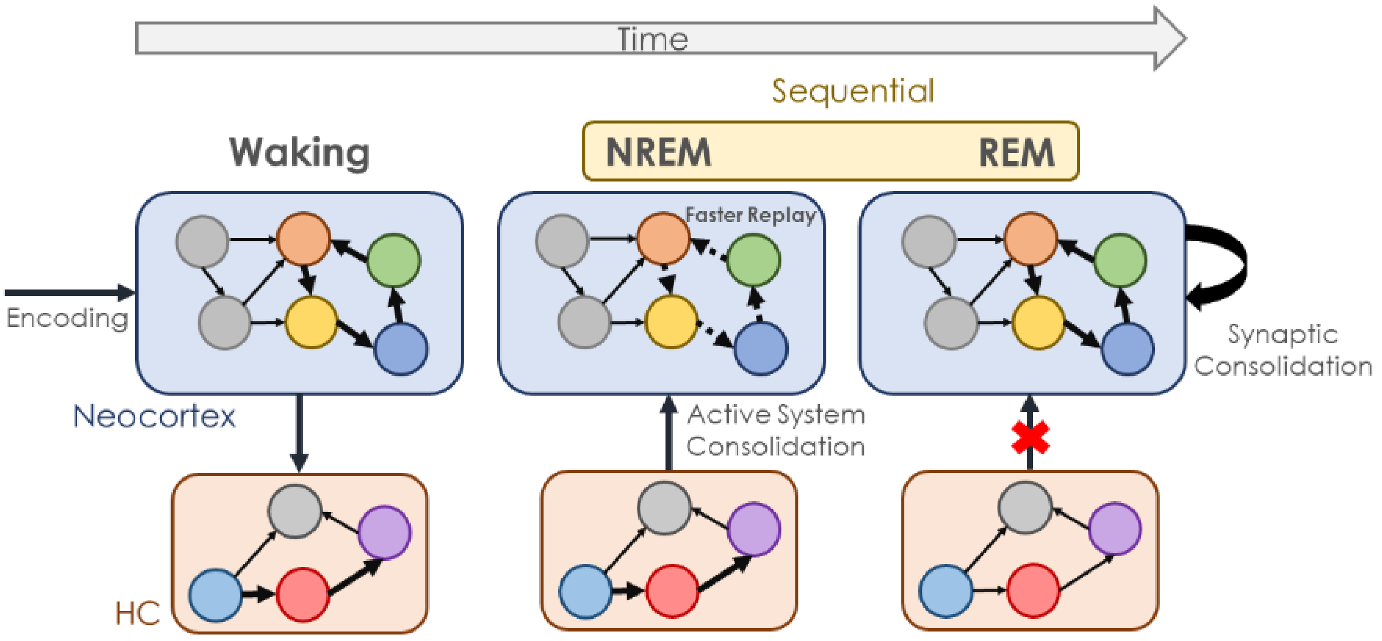
Figure 1: Architectural diagram of Latent Replay.

Latent Replay for Real-Time Continual Learning. Pellegrini et al. IROS, 2019.

Neuroplausibility of Replay

There are some (very high level) parallels with the brain:

- Memory formation is a key part of learning in the human brain
- Memories are noisy
- Memories are latent. The brain doesn't store the raw input
- Sleep/Wake cycles are a fundamental component of memory formation



(a) Depiction of the contributions of replay to memory formation and consolidation during waking, NREM, and REM stages.

T. L. Hayes et al. 2021. "Replay in Deep Learning: Current Approaches and Missing Biological Elements."

Random Replay

The most basic form of replay: random insertion, deletion, and sampling.

Parameters: memory size

During training

- Sample from the concatenated data
- Do SGD step

After each experience:

- Sample randomly from the current experience data
- Fill your fixed Random Memory (RM)
- Number of examples inserted/removed is inversely proportional to the stream length

Algorithm 1 Pseudocode explaining how the external memory RM is populated across the training batches. Note that the amount h of patterns to add progressively decreases to maintain a nearly balanced contribution from the different training batches, but no constraints are enforced to achieve a class-balancing.

$RM = \varnothing$
$RM_{size} =$ number of patterns to be stored in RM
for each training batch B_i:
  train the model on shuffled $B_i \cup RM$
  $h = \dfrac{RM_{size}}{i}$
  $R_{add} =$ random sampling h patterns from B_i
  $R_{replace} = \begin{cases} \varnothing & \text{if } i == 1 \\ \text{random sample } h \text{ patterns from } RM & \text{otherwise} \end{cases}$
  $RM = (RM - R_{replace}) \cup R_{add}$

Latent Replay for Real-Time Continual Learning. Pellegrini et al. IROS, 2019.

Random Replay – Improved Sampling

- In general, the experience data and the buffer may have very different sizes
- Instead of concatenating the data we should sample both separately

Improved Sampling:

- Sample from the current data randomly
- Sample from the buffer randomly
- Do SGD step using the concatenated mini-batches

Algorithm 1 Pseudocode explaining how the external memory RM is populated across the training batches. Note that the amount h of patterns to add progressively decreases to maintain a nearly balanced contribution from the different training batches, but no constraints are enforced to achieve a class-balancing.

$RM = \varnothing$
RM_{size} = number of patterns to be stored in RM
for each training batch B_i:
    train the model on shuffled $B_i \cup RM$
    $h = \dfrac{RM_{size}}{i}$
    R_{add} = random sampling h patterns from B_i
    $R_{replace} = \begin{cases} \varnothing & \text{if } i == 1 \\ \text{random sample } h \text{ patterns from } RM & \text{otherwise} \end{cases}$
    $RM = (RM - R_{replace}) \cup R_{add}$

Results

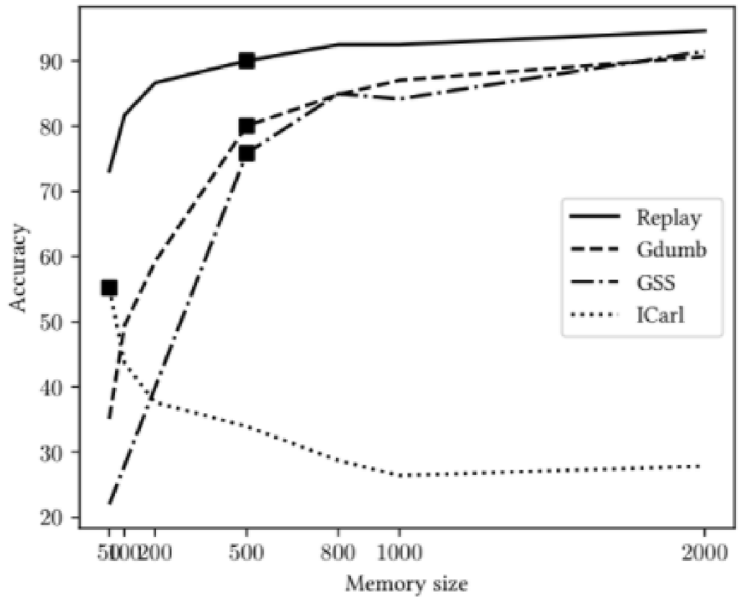


Figure 5.2: Split-MNIST memory-accuracy curve

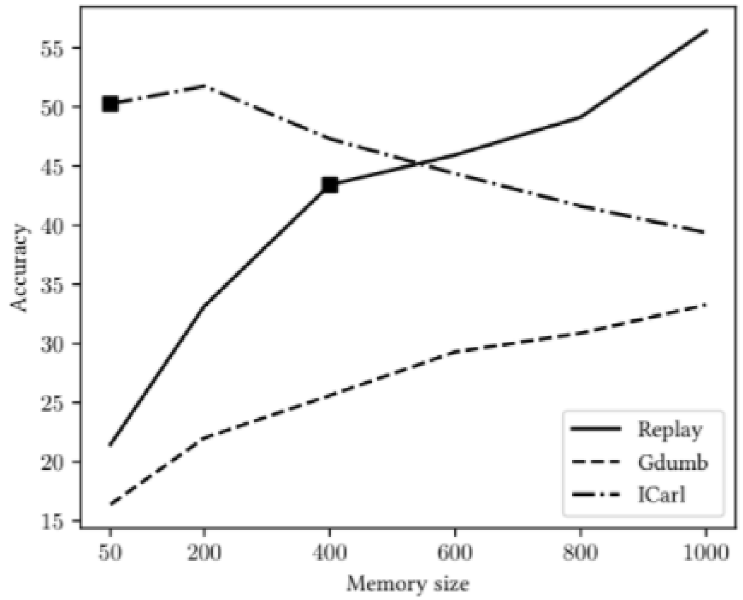


Figure 5.3: Split-CIFAR-10 memory-accuracy curve

Random Replay

- Very simple to implement
- Good performance in simple settings

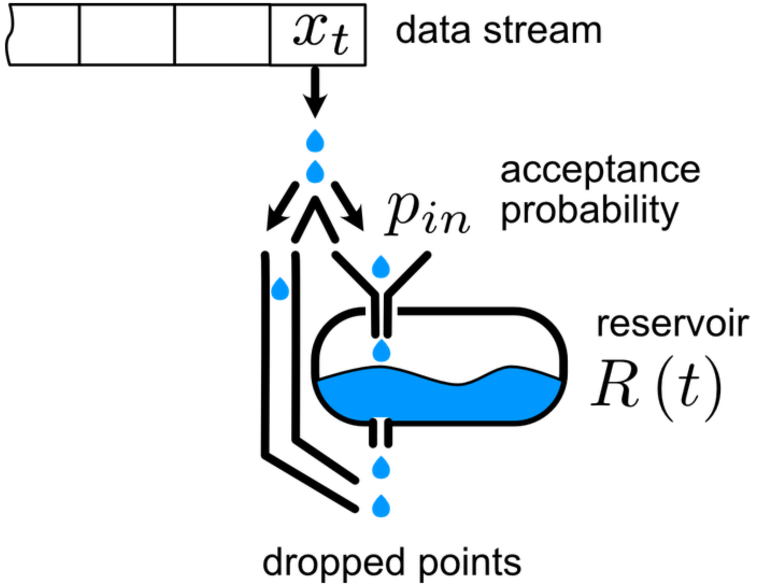
Problems:

- **How much data**: it doesn't work in OCL because we select a discrete number of samples to add.
  - It doesn't work if num_experiences >> mem_size
- **Type of shifts**: ignores imbalance in the stream. If a class is over-represented in the stream, it will also dominate the buffer
- **Task boundaries**: it doesn't need them (but it still not good in OCL)
- **Task labels**: unused. Simple improvement: split the buffer by task in a balanced way

Reservoir Sampling

Reservoir sampling: uniform random sampling, without replacement, of K items from an infinite stream S.

At time N, we want $p(x_t \in M) = \frac{K}{N}, \forall t \leq N$




```
(* S has items to sample, R will contain the result *)
ReservoirSample(S[1..n], R[1..k])
  // fill the reservoir array
  for i := 1 to k
      R[i] := S[i]

  // replace elements with gradually decreasing probability
  for i := k+1 to n
    (* randomInteger(a, b) generates a uniform integer
       from the inclusive range {a, ..., b} *)
    j := randomInteger(1, i)
    if j <= k
        R[j] := S[i]
```

RS – Probabilities

- **parameters**: N current step, K memory size, M_t memory after t elements
- **Goal**: At time N, we want $p(x_i \in M_t) = \frac{K}{N}, \forall t, i$
- If $t \leq K$ everything fits in memory and $p(x_t \in M_t)$
- If $t > K$
  - j uniformly random integer in $[1, N]$
  - New element: $\mathrm{p}(\mathrm{x_t} \in M_t) = p(j \leq K) = \frac{K}{N}$
  - Old element $(\mathrm{i} < t)$: $p(x_i \in M) = p(x_i \in M_{t-1}\ and\ x_i\ is\ not\ removed) = \frac{K}{N-1}\frac{N-1}{N} = \frac{K}{N}$

```
(* S has items to sample, R will contain the result *)
ReservoirSample(S[1..n], R[1..k])
  // fill the reservoir array
  for i := 1 to k
      R[i] := S[i]

  // replace elements with gradually decreasing
probability
  for i := k+1 to n
    (* randomInteger(a, b) generates a uniform integer
       from the inclusive range {a, ..., b} *)
    j := randomInteger(1, i)
    if j <= k
        R[j] := S[i]
```

Reservoir Sampling

- Reservoir sampling fixes some of the issues of the Random Replay (RR)
- **How much data**: works perfectly in OCL.
- **type of shifts**: ignores imbalance in the stream. If one class is much more frequent, it will be over-represented in the buffer
  - Easy to fix by balancing the buffer capacity by class (i.e. keep one reservoir for each class)
  - If the imbalance is at a different level (e.g. domain) we would need task labels to recognize them (which we probably don’t have)
- **Task boundaries**: it doesn’t need them. It’s a strong OCL baseline

Online Continual Learning with Replay

Notice that we apply different augmentations at each step!

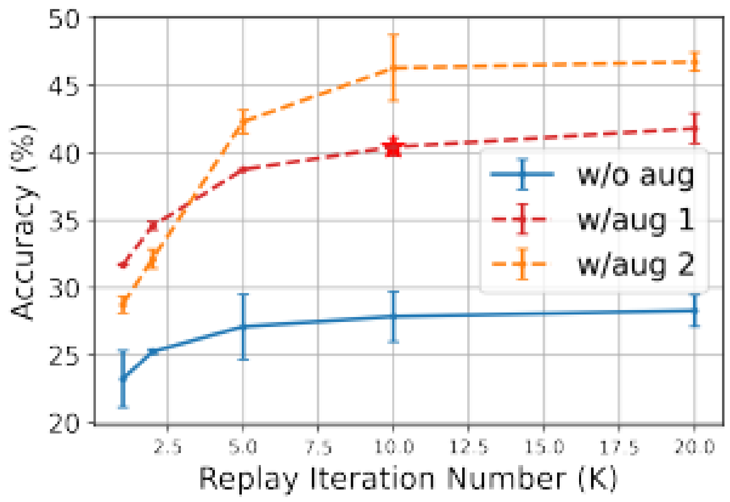
Online Continual Learning

```
# processing one example at a time
for (x_new, y_new) in train_stream:

    # performing k passes on the same data
    for k in train_passes:
        # optionally augment data
        x_new, y_new = augment(x_new, y_new)
        # extracting examples from the external memory
        x_mem, y_mem = augment(sample(memory))
        # perform an optimization step
        compute_loss_and_backprop(x_new, y_new, x_mem, y_mem)
        weights_udpate()

    # update the external memory
    update(memory, x_new, y_new)
    # eventually evaluate, inference only
    evaluation()
```

Soutif-Cormerais, Albin, et al. "A comprehensive empirical evaluation on online continual learning." *ICCV Workshops*. 2023.

A Note on Augmentations

- The buffer needs to provide a good coverage of the past data distribution
- Diversity is clearly fundamental to mitigate forgetting
- Diversity also helps overfitting the buffer
- Augmentation are a key implementation detail
- Especially important in online settings with multiple passes or with limited buffer sizes

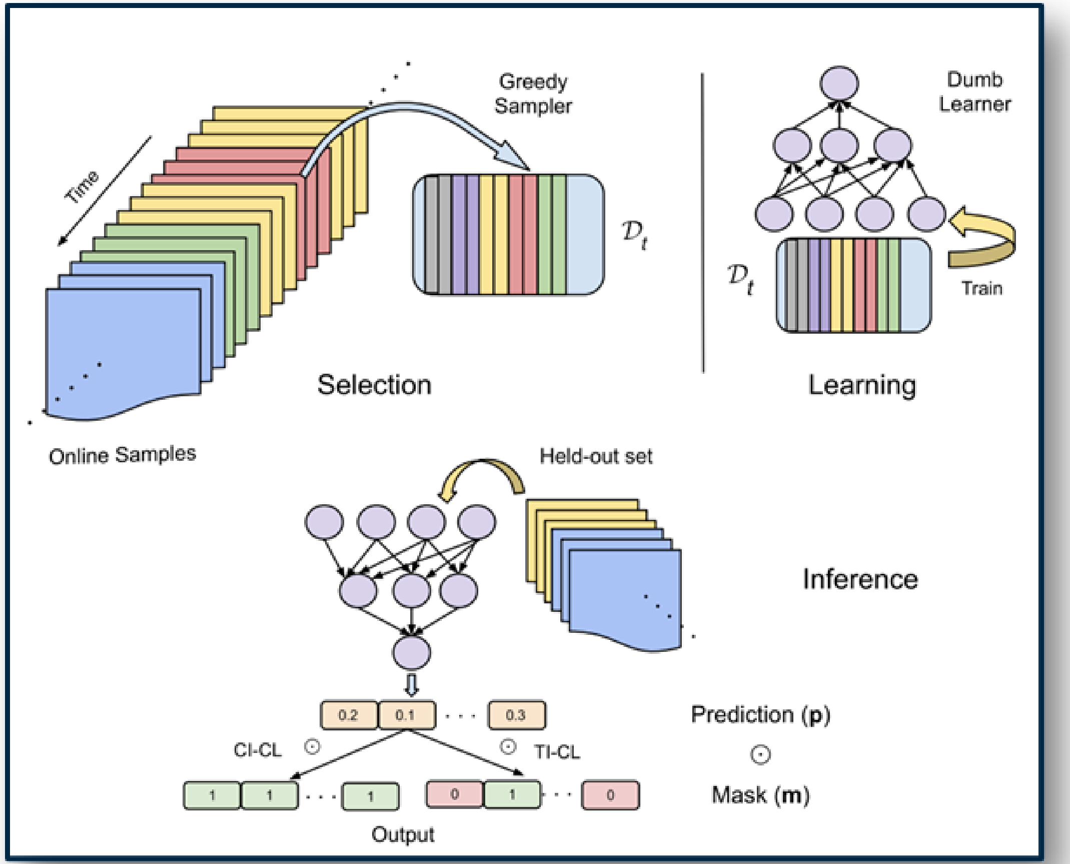


Zhang, Yaqian, et al. "A simple but strong baseline for online continual learning: Repeated augmented rehearsal." NeurIPS 2022

GDumb

A “Dumb” but popular replay baseline

- **Greedy Sampler**: The sampler greedily stores samples while balancing the classes.
- **Dumb Learner**: Before inference, the learner trains a network from scratch on memory D_t provided by the sampler.
- **Masking**: If a mask m is given at inference, GDumb classifies on the subset of labels provided by the mask.
    - **class-incremental**: mask only unseen classes.
    - **task-incremental**: mask classes outside of current task.



GDumb: A Simple Approach that Questions Our Progress in Continual Learning. Prabhu et al. ECCV, 2020.

Greedy Sampler

Algorithm 1. Greedy Balancing Sampler

Init: counter C_0={}, $\mathcal{D}_0$={} with capacity k. Online samples arrive from t=1

function SAMPLE(x_t, y_t, $\mathcal{D}_{t-1}, \mathcal{Y}_{t-1}$) ▷ Input: New sample and past state
    $k_c = \frac{k}{|\mathcal{Y}_{t-1}|}$
    if $y_t \notin \mathcal{Y}_{t-1}$ or $C_{t-1}[y_t] < k_c$ **then**
        if $\sum_i C_i >= k$ **then** ▷ If memory is full, replace
            $y_r = argmax(C_{t-1})$ ▷ Select largest class, break ties randomly
            $(x_i, y_i) = \mathcal{D}_{t-1}.\text{random}(y_r)$ ▷ Select random sample from class y_r
            $\mathcal{D}_t = (\mathcal{D}_{t-1} - (x_i, y_i)) \cup (x_t, y_t)$
            $C_t[y_r] = C_{t-1}[y_r] - 1$
        else ▷ If memory has space, add
            $\mathcal{D}_t = \mathcal{D}_{t-1} \cup (x_t, y_t)$
        end if
        $\mathcal{Y}_t = \mathcal{Y}_{t-1} \cup y_t$
        $C_t[y_t] = C_{t-1}[y_t] + 1$
    end if
    return $\mathcal{D}_t$
end function

GDumb: A Simple Approach that Questions Our Progress in Continual Learning. Prabhu et al. ECCV, 2020.

Is Gdumb Fair?

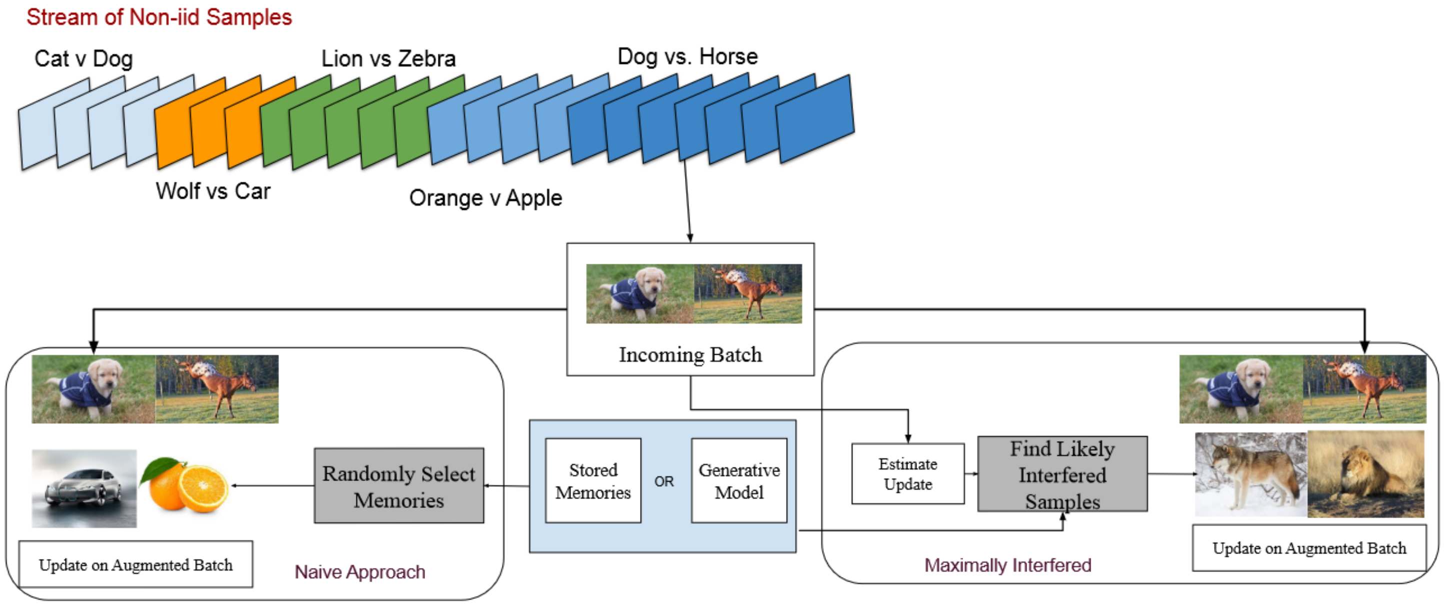
- **Gdumb** trains the model from scratch at each step
- **Zero knowledge transfer** because the network is always trained from scratch on the buffer data (quite dumb indeed!).
- Despite its simplicity, it is a **very competitive** method in the class-incremental setting.
- Q: What is Gdumb biggest limitation?
  - Suggestion: think about the constraints we had for OML methods.

Gdumb limitations

- Zero Knowledge Transfer
- **Limited use of the data**: the model is trained only on a number of samples equivalent to the buffer size
- **Latency**: before inference we need to train a DNN from scratch, which is very expensive if we need continuous evaluations
- Useful as a simple baseline for replay methods

Sample Selection Strategy

- How do we choose the best examples from the buffer?
- Often it’s a random (balanced) selection
- Can we do something better?



Online Continual Learning with Maximally Interfered Retrieval. Aljundi et al. 2019.

MIR – Maximally Interfered Retrieval

IDEA: Select examples that are more negatively impacted by the weights update.

High-Level Algorithm:

- Sample from the current data
- Estimate weight update
- Estimate loss of buffer samples with the new weights
- Select samples with the largest drop (s_{MI-1})
- Do SGD step on concatenated minibatch

New weights (new samples only)

$$\theta^v = \theta - \alpha \nabla \mathcal{L}(f_\theta(\boldsymbol{X}_t), \boldsymbol{Y}_t)$$

Maximal interference criterion

Loss at the old weights

$$s_{MI-1}(x) = l\left(f_{\theta^v}(x), y\right) - l\left(f_\theta(x), y\right)$$

Loss at the new weights

MIR – Maximally Interfered Retrieval

LIMITATION: Computationally expensive w.r.t. the actual accuracy gain over random selection

- It needs an additional forward pass on a large subset of the buffer examples to find the maximally interfered ones

New weights (new samples only)

$$\theta^v = \theta - \alpha \nabla \mathcal{L}(f_\theta(\boldsymbol{X}_t), \boldsymbol{Y}_t)$$

Maximal interference criterion

Loss at the old weights

$$s_{MI-1}(x) = l\left(f_{\theta^v}(x), y\right) - l\left(f_\theta(x), y\right)$$

Loss at the new weights

Online Continual Learning with Maximally Interfered Retrieval. Aljundi et al. 2019.

Example of MIR

Benchmark: OCL version of Split MNIST

(b) Most interfered samples while learning the last task (8 vs 9). Top row is the incoming batch. Rows 2 and 3 show the most interfered samples for the classifier, Row 4 and 5 for the VAE. We observe retrieved samples look similar but belong to different category.

Online Continual Learning with Maximally Interfered Retrieval. Aljundi et al. 2019.

Latent Replay

- **PROBLEM**: Replay in the input space is inefficient and biologically implausible
- **SOLUTION**: replay latent activations
  - Good Accuracy-Memory-Computation trade-offs.
  - There is no obvious choice for the layer. Middle layers can be very wide.
  - If we allow lossy storage activations can be compressed a lot
- **Algorithm**:
  - Store latent representation
  - Forward new samples up to latent replay layer
  - Concatenate new and stored representations
  - Forward to the output layer

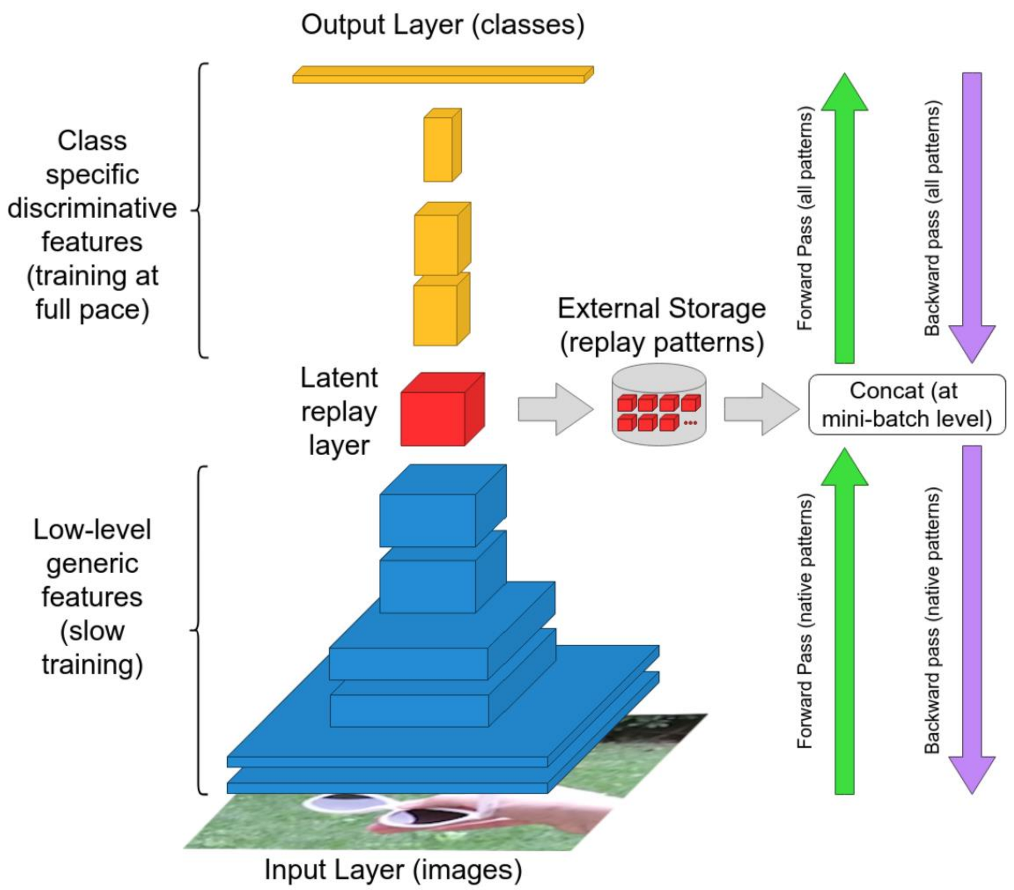


Figure 1: Architectural diagram of Latent Replay.

Latent Replay for Real-Time Continual Learning. Pellegrini et al. IROS, 2019.

Freezing + Latent Replay

- Low layers are trained early during training. They don't change much afterward.
- We can freeze them at some point.
- Improves latent replay. If we don't freeze the latent representation in the buffer will become outdated over time

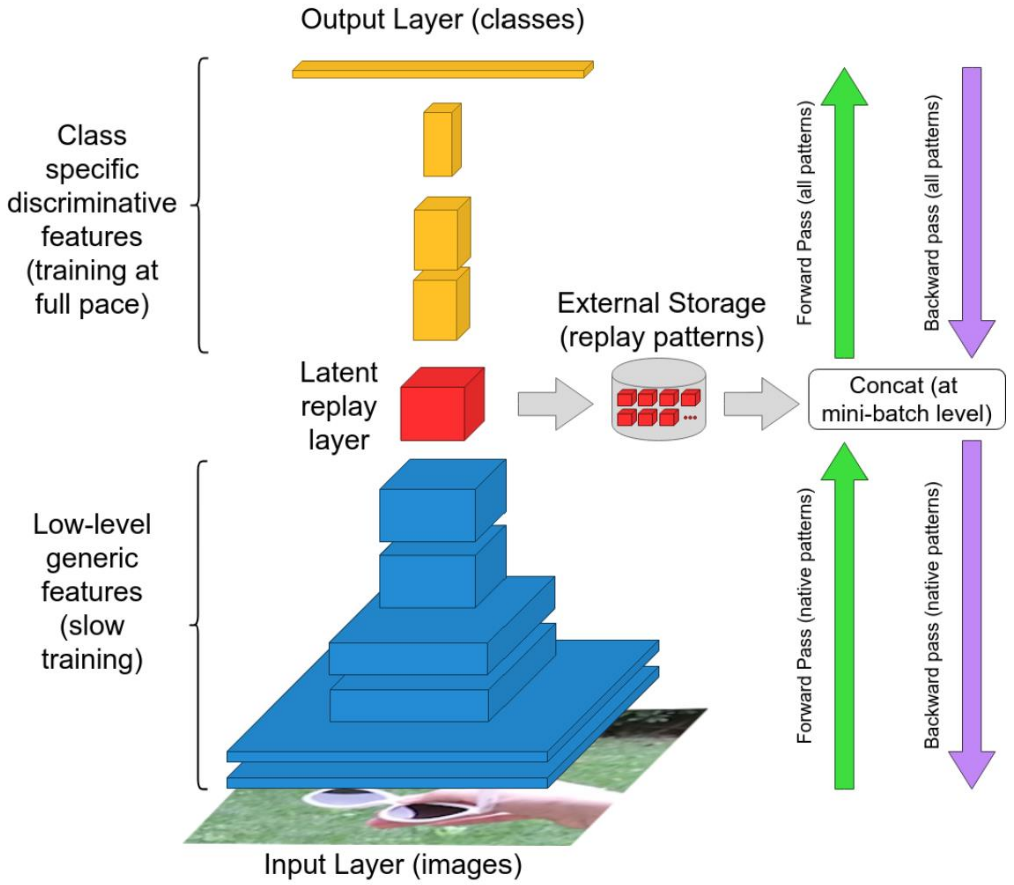
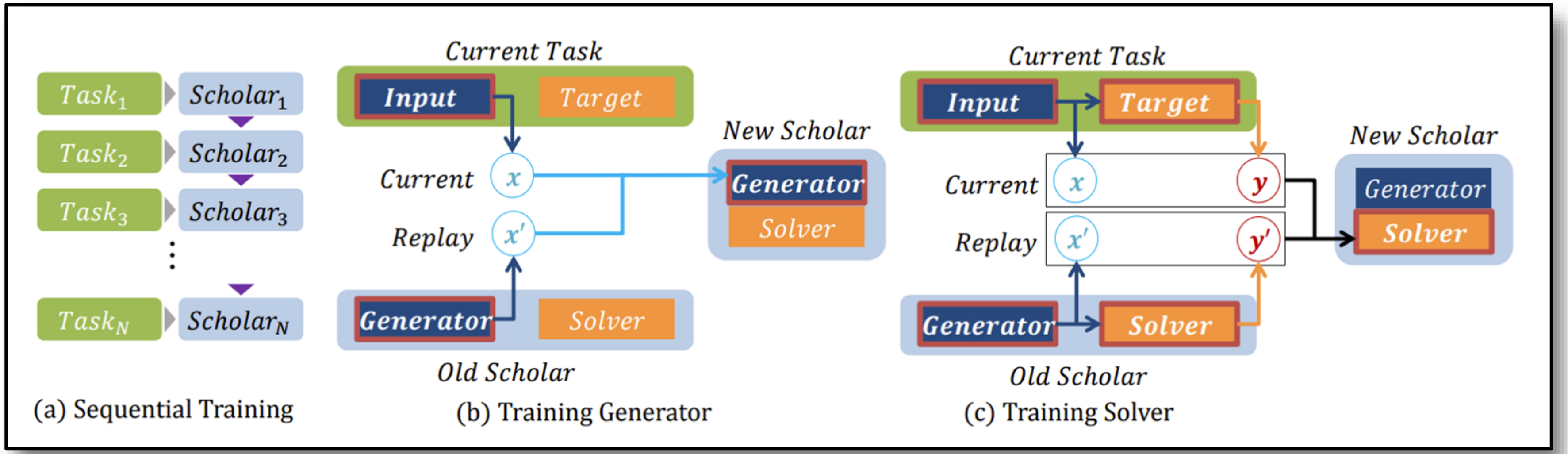


Figure 1: Architectural diagram of Latent Replay.

Generative Replay

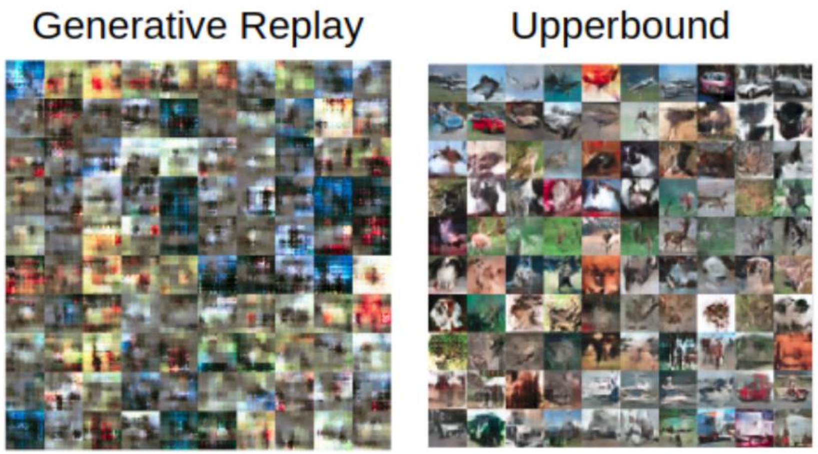
- Buffer size is limited by memory constraints.
- Using generative models we store a finite number of parameters but can sample as many examples as we want.
- Biologically plausible.



Continual Learning with Deep Generative Replay, Shi et al, 2017.

Generative Replay in Practice

- Currently there is no effective algorithm to train generative models continually.
- We can train one model per task/experience with a linear scaling of the memory cost.
- **Alternative**: Knowledge Distillation with generative samples
  - If we use classification losses examples should resemble the target class.
  - if we use methods such as knowledge distillation we can potentially use very distorted samples



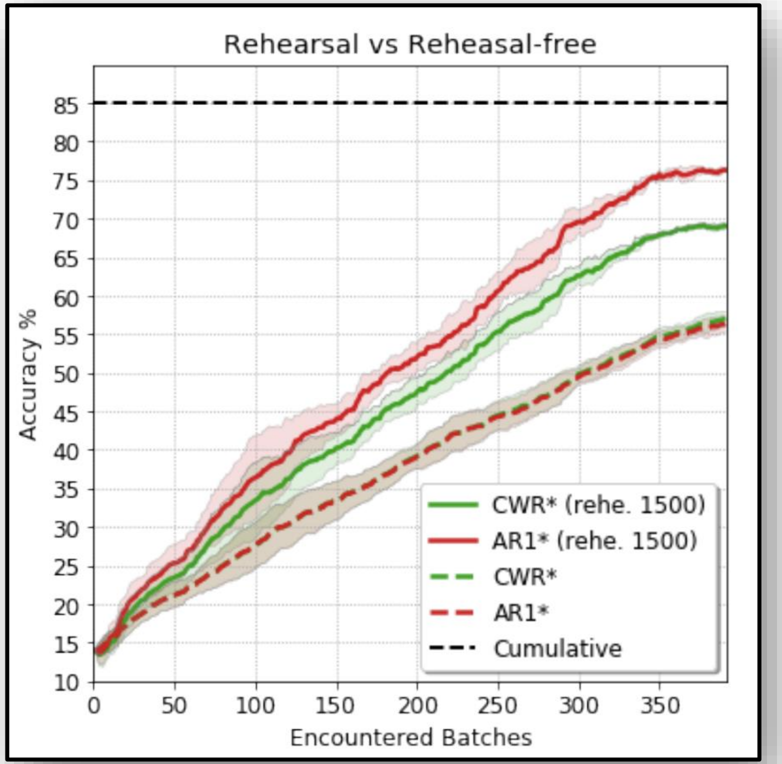
CIFAR10 samples from generative models

Lesort, T et al. *Generative models from the perspective of continual learning. In IJCNN 2019.*

Did we just solve continual learning?

Not really…

- The gap with offline training is still big.
- The accuracy improvements with respect to the memory size is often logarithmic.
  - Huge buffer sizes (approximating a cumulative strategy) are expensive.
  - Example: ImageNet 50 imgs per class means about 7 GB memory
- Additional forward and backward passes over the same examples
  - If you want to balance over tasks, this can easily become a linear cost over time



Latent Replay for Real-Time Continual Learning. Pellegrini et al. IROS, 2019.

How Many Samples do We Need?

- You may need a lot of exampes to recover the joint training performance
- For reference, CIFAR100 has 500 samples per class
- The plot shows a gap even when more than half (320) of the samples are stored for rehearsal
  - Early phases of training are critical. Learning from a small experience may hurt performance
  - On the other hand, CL avoids retraining from scratch and saves memory

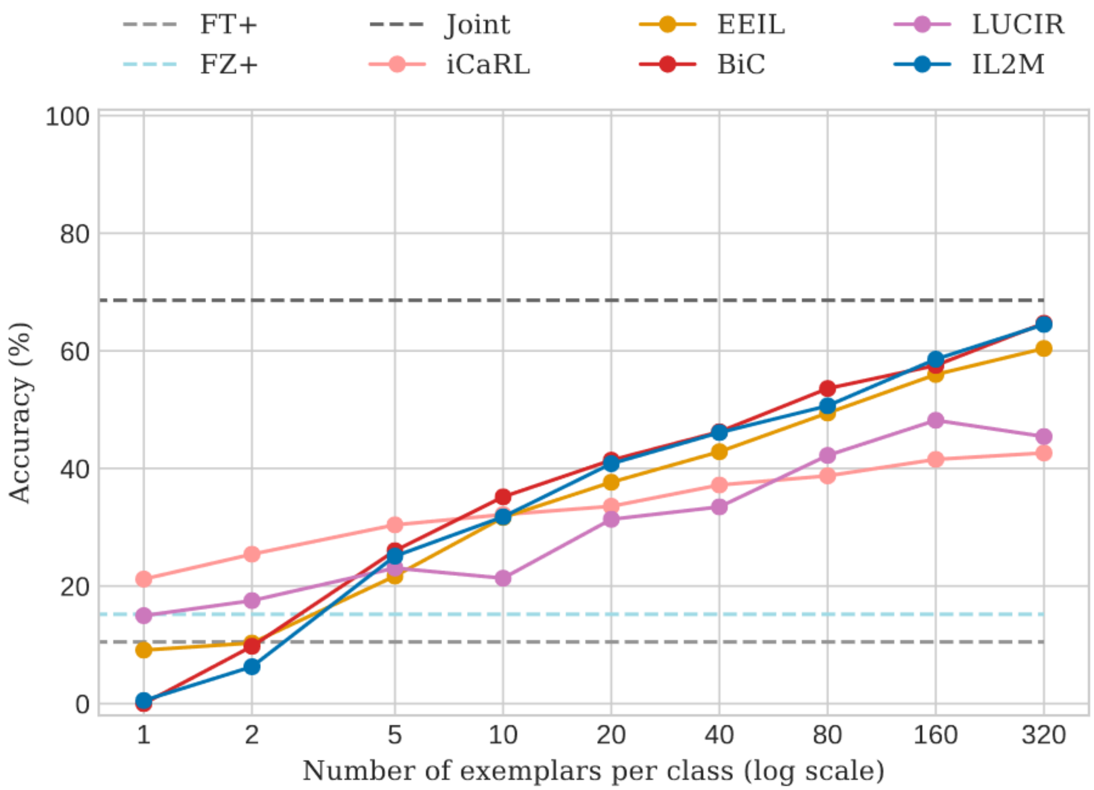


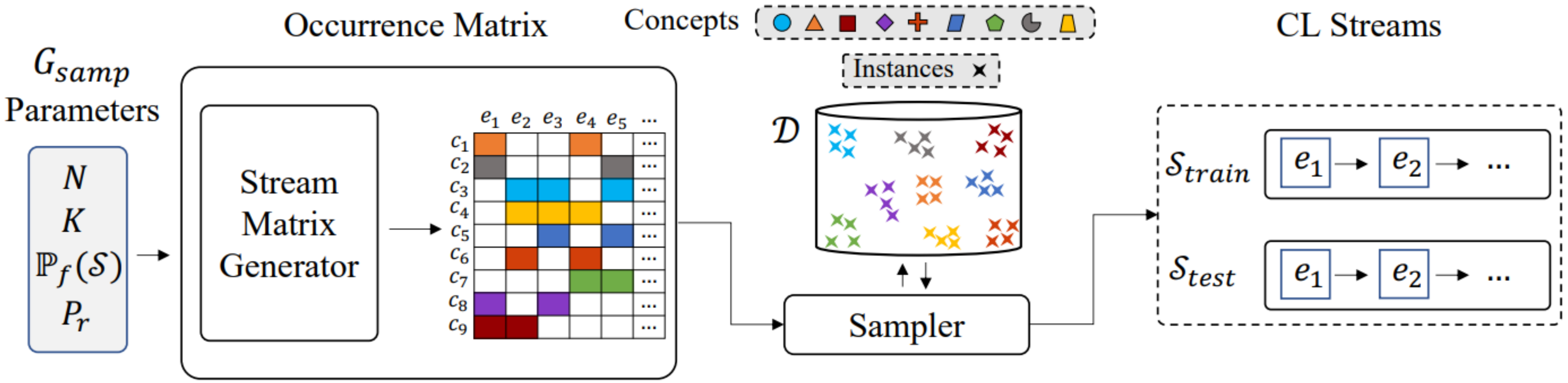
Fig. 7: Results for CIFAR-100 (10/10) on ResNet-32 trained from scratch with different exemplar memory sizes.

M. Masana et al. 2022. *"Class-Incremental Learning: Survey and Performance Evaluation on Image Classification." IEEE TPAMI*

Natural Repetitions

In real-world applications, **class repetitions may happen in the stream naturally**.

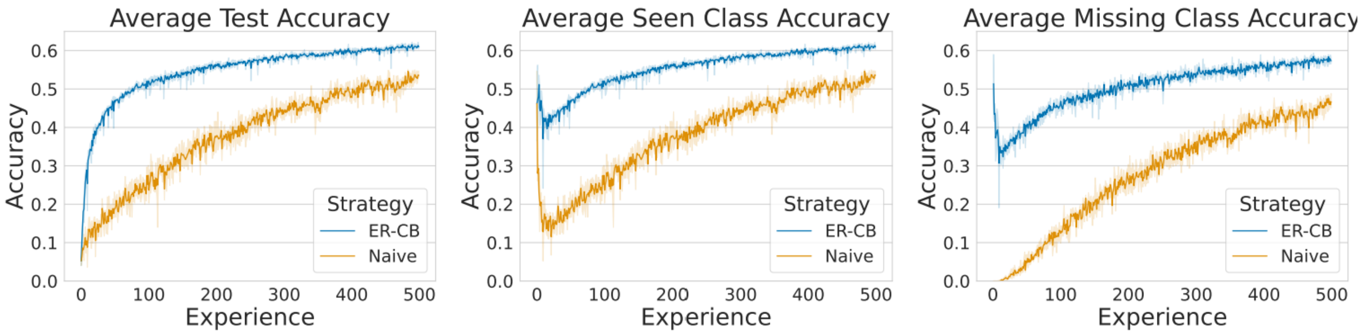
- Rehearsal naturally happens even without any replay buffer.
- May be a less effective form of rehearsal (missing class, unbalanced, biased, …)
- We can approximate it by generating a more natural stream with repetitions



H. Hemati et al. "Class-Incremental Learning with Repetition." CoLLAs '23

Effect of Natural Repetitions

Naive finetuning approaches replay for long streams with repetitions



Missing class accuracy improves over time, even for naive finetuning

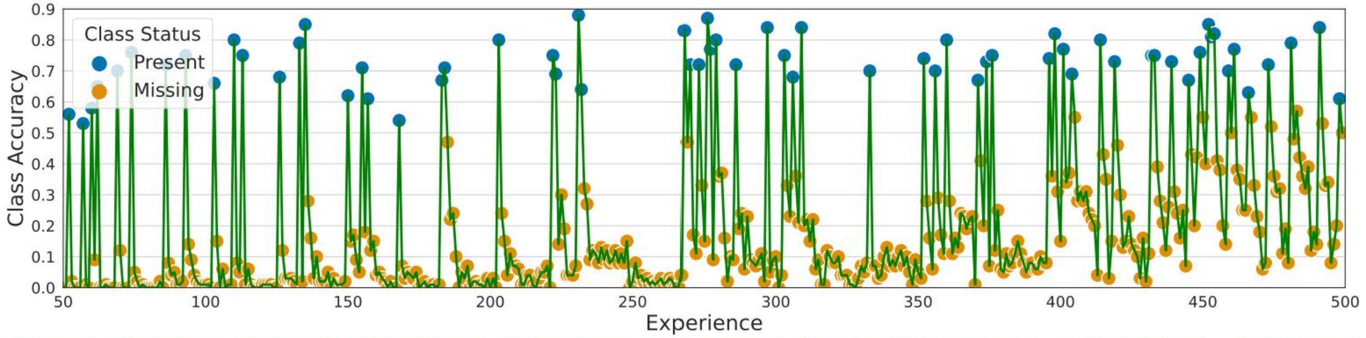


Figure 6: Accuracy of a particular class over the stream. The target class is either present or absent in the experiences indicated by the blue and orange points, respectively.

In unbalanced streams, class-balanced buffers and reservoir sampling are not effective

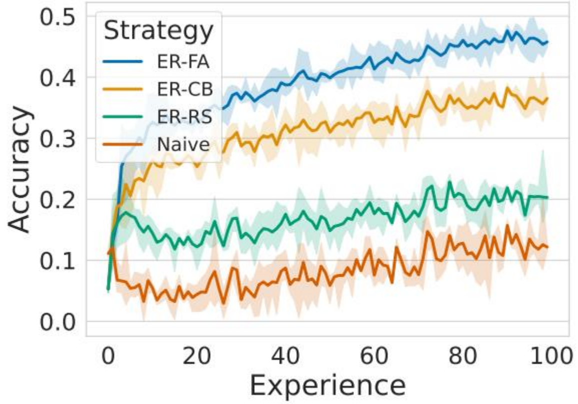


Figure 10: Accuracy of Infrequent Classes.

H. Hemati et al. "Class-Incremental Learning with Repetition." CoLLAs '23

Take-Home Messages

- Replay is almost always beneficial if you can afford it. You should use as much replay as you can.
- Easy to implement, even on low-powered edge devices.
- There are many improvements over the basic reservoir sampling, but the gain are often marginal for medium (or bigger) buffers.
- There is a lot of CL research that tries to limit the need for replay for practical reasons (memory, privacy) and because of biological plausibility.
  - latent replay is quite effective in this settings
  - Generative is promising but limited by the CL capability of generative models

Next Lecture

Regularization Methods

- Approximating the past task loss
  - With an approximation of the bayesian posterior
  - With an approximation of the curvature of the loss