# A Study of Continual Learning Methods for Q-Learning

1st Benedikt Bagus
*dept. of computer science*
*University of Applied Sciences Fulda*
Leipzigerstr. 123, 36093 Fulda, Germany
benedikt.bagus@cs.hs-fulda.de

2nd Alexander Gepperth
*dept. of computer science*
*University of Applied Sciences Fulda*
Leipzigerstr. 123, 36093 Fulda, Germany
alexander.gepperth@cs.hs-fulda.de

***Abstract*—We present an empirical study on the use of continual learning (CL) methods in a reinforcement learning (RL) scenario, which, to the best of our knowledge, has not been described before. CL is a very active recent research topic concerned with machine learning under non-stationary data distributions. Although this naturally applies to RL, the use of dedicated CL methods is still uncommon. This may be due to the fact that CL methods often assume a decomposition of CL problems into disjoint sub-tasks of stationary distribution, that the onset of these sub-tasks is known, and that sub-tasks are non-contradictory. In this study, we perform an empirical comparison of selected CL methods in a RL problem where a physically simulated robot must follow a racetrack by vision. In order to make CL methods applicable, we restrict the RL setting and introduce non-conflicting subtasks of known onset, which are however not disjoint and whose distribution, from the learner's point of view, is still non-stationary. Our results show that dedicated CL methods can significantly improve learning when compared to the baseline technique of "experience replay".**

***Index Terms*—continual reinforcement learning, Q-learning, replay methods**

## I. Introduction

This article is in the context of continual reinforcement learning (CRL) [1], [2], which describes the application of dedicated continual learning (CL) [3], [4] algorithms to reinforcement learning (RL) [5]. Both are concerned with learning from non-stationary data distributions. While assumptions in CL are quite varied, RL assumes a well-defined scenario. RL is founded on Markov decision processes (MDPs), which are generally formalized as 5-tuple $M = \langle S, A, P, R, \rho_0 \rangle$. $S$ and $A$ are the sets of all *valid* states/actions, $P$ is the probability function of the transition $S \times A \to \mathcal{P}(S)$, with $P(\vec{s}_{t+1} = \vec{s}\,'|\vec{s}_t = \vec{s}, \vec{a}_t = \vec{a})$ being the probability of transitioning into state $\vec{s}\,'$ if action $\vec{a}$ is taken in state $\vec{s}$. $R$ is called reward function, which maps $S \times A \times S \to \mathbb{R}$ and provides the return signal $r_t = R(\vec{s}_t, \vec{a}_t, \vec{s}_{t+1})$. Finally, $\rho_0$ is the distribution of the initial state. Here, an *agent* interacts with an *environment*, trying to maximize a *reward* $r_t \in \mathbb{R}$ signal by selecting the most appropriate *action* $\vec{a}_t \in A$ based on an *observation* $\vec{o}_t$, e.g., a camera signal, of the actual state $\vec{s}_t \in S$ following a policy $\pi$. This feedback loop inherently defines an incremental manner of learning and implies appropriate models.

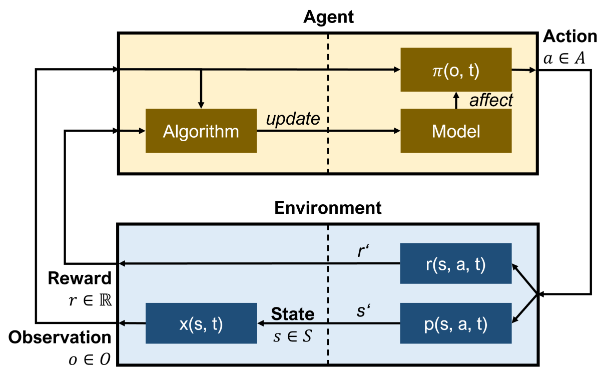

Fig. 1. Schematic representation of a RL control loop, which enables decision-making.

In this article we focus on *Q-learning* [6], an important flavor of RL, where the agent selects actions that maximize the expected future return $Q(s_t, a_t)$, whose dependency on state and action must be acquired through learning.

$$\begin{aligned} Q'(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \\ + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a)) \end{aligned} \tag{1}$$

Q-Learning represents an "off-policy" approach, where exploration is commonly ensured by a $\epsilon$-greedy strategy and policy updates are immediately performed after an iteration. The exploration ensures the discovery of diverse state-action combinations. This is crucial for models like Q-tables, since they implement a defined assignment (lookup) between states and actions.

### A. Catastrophic forgetting in RL

Conventional assumptions in machine learning are that data distributions are stationary and samples are iid. In RL, these assumptions can be systematically violated in several ways:

**Concept drift/shift** Even if the environment is stationary, the observations will not be, at least not on the short timescale due to the ongoing exploration of the state-action space. For example, when taking an action $\vec{a}\,'$ as a consequence of observation $\vec{o}\,'$, the obtained rewards may be in disagreement at first, since the environment is supposed to be stationary but

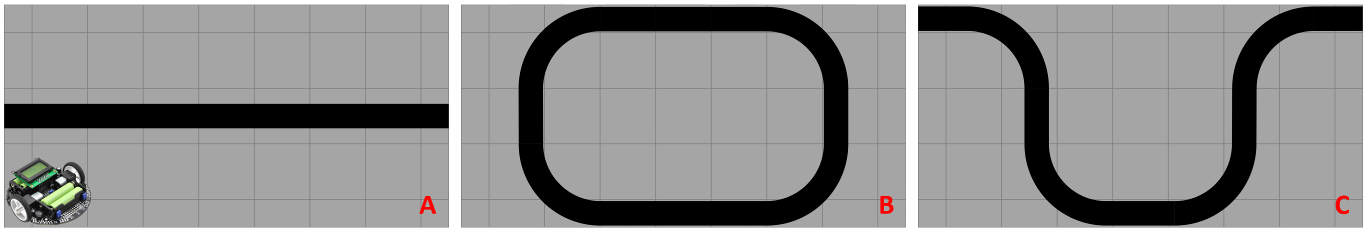

Fig. 2. Illustration of environment shifts considered in this article. A simulated robot is trained to follow the black line in a succession of environments: A) straight line B) straight line + left curve C) straight line + left/right curve. Please note that all of these scenarios include concept drift as well from the point of view of the learner, due to the exploration of the state-action space.

not deterministic. Similarly, the sampling of the state-action space is changing over time due to the decreasing influence of exploration, effectively creating another source of concept drift.

**Environment shift** In contrast to this, the environment itself may be non-stationary and subject to *environment drift/shift* (see, e.g., [7]), and thus automatically lead to non-stationary observations. This includes the case where the agent encounters novel situations within the environment or is transferred to a completely different one. Examples of environment shift are given in Fig. 2.

Deep neural networks (DNNs) are typical machine learning models employed in deep reinforcement learning (DRL). This raises the issue of catastrophic forgetting (CF) which especially DNNs are subject to (see, e.g., [8]) as a consequence of non-stationary data distributions. Common workarounds include the use of *experience replay* (ER), see [9]. In this approach, newly arriving samples are stored, e.g., by reservoir sampling, in a large buffer $\mathcal{M}$ with fixed size $M = |\mathcal{M}|$. Instead of training the model directly, random mini-batches are drawn from the buffer $\mathcal{B} \sim \mathcal{M}$, which simulates a stationary data distribution. This mitigates CF, but incurs a huge cost in memory. In addition, reaction to concept drift/shift is delayed because new data samples will take a while before they are significantly represented in the buffer. Simultaneously, if the sample selection is insufficient, the subset may not match the real distribution or underrepresented samples could be unbalanced even with buffer.

## B. Related work: CL approaches

The research field of continual learning (CL) investigates the problem of learning under non-stationary data distributions, see [3], [4] for reviews. Systematic comparisons between different approaches to avoid CF are performed in, e.g., [8], [10]. As discussed in [8], [11]–[13], many recently proposed methods demand specific experimental setups, which deviate significantly from applications. In contrast to learning from stationary data, CL scenarios are very diverse [1], [2], [12], [13], depending on what type of non-stationarity is assumed [14]. Theoretical analyses of CL are presented in [15]–[17], again underscoring the fact that a universally accepted definition of CL has not yet been reached.

A very common assumption in CL is the decomposition of a problem into several sub-tasks of *stationary* statistics [1], [18], which are locally iid. Likewise, the onset of each sub-task is assumed to be known [12], [19] which side-steps the issue of *detecting* the boundaries. Furthermore, sub-tasks are often assumed to be *disjoint* [11]: different sub-tasks contain different classes of the classification problem. This implies that no re- or un-learning takes place, where samples from known classes would be assigned a different label in later sub-tasks [14].

Among the proposed CL methods, three major directions may be distinguished according to [3]:

**Parameter Isolation** Parameter isolation methods aim at determining (or creating) a group of parameters that are mainly "responsible" for a certain sub-task. CF is then avoided by protecting these parameters or adding new ones when training on successive sub-tasks, see [20]–[22].

**Regularization** Regularization methods mostly propose modifying the loss function by including additional terms of criterions that protect knowledge acquired in previous sub-tasks, see [19], [23], [24].

**Replay** Replay methods keep small subsets of real samples or train generators to reconstruct an arbitrary number of pseudo-samples afterwards. CF can be circumvented by putting constraints on current sub-task training or by adding retained samples to the current sub-task, see [18], [25]–[29].

## C. Related work: Continual RL

As stated previously, CL is a broad topic and methods can be tailored to specific scenarios. Transferring them to a more general domain such as RL is therefore a non-trivial objective. First attempts to use CL methods in the domain of RL are performed in [1] and [2]. The work of [1] gives detailed insights into important aspects and describes RL as a natural fit to CL. In particular, the work of [2] elaborates extensively upon the application of CL in RL and describes essential criteria on an abstract level. Both works introduce generic frameworks for CRL and discuss concepts, without implementing them.

In general, several frameworks for benchmarking RL have been presented, e.g., [30]. However, only a few of them are formulated with CRL in mind [31]–[33]. Due to the divergent formalism, established metrics as accuracy or forgetting/transfer measures [18], [25] are inapplicable. In RL, samples are not classified but assigned to Q-values. However, new metrics for CRL are proposed in [1], [2] and [32], [33].

The works of [34], [35] combine CL and RL and suggest own approaches tailored to RL. For example, [34] employ regularization-based models with multi-head outputs and [35] try to mitigate CF within single tasks by context detection. However, established state-of-the-art CL methods have barely been reused and studied under generic conditions as provided by the RL domain.

### *D. Mapping CL to RL*

RL is founded on an incremental formalism of a decision problem and offers an "real-world" application for CL. However, when endeavoring to apply CL approaches in RL, there are restrictions that may be problematic due to deviating conditions:

**Decomposition into sub-tasks** One of the most wide-spread assumptions in CL concerns the decomposition of the learning problem into a sequence of *sub-tasks*, with stationary data statistics and that they are locally iid. In a stationary environment, the only entity that can be identified with a CL sub-task is an *episode*, that is, a sequence between two terminal states. It is unclear, though, whether statistics within an episode should be considered stationary. Epochs themselves may contain context changes and would therefore violate the common definition of CL sub-tasks.

**Overlapping and contradictory sub-tasks** Almost universally, sub-tasks in CL are supposed to be disjoint. As a consequence, many approaches try to identify parameters that are important for certain sub-tasks, or to directly dedicate parts of the ML model to certain sub-tasks. Typical representatives are regularization approaches like [19], [23], [24], but also parameter isolation methods like [20]–[22]. In RL, episodes do not contain consistent labels or separated classes because they are treated differently. Each "label" consist of non-one-hot always changing Q-Values, whereby each sample (state) can be assigned to one discrete action. This will lead to difficulties if there exist contradicting, common samples in different sub-tasks, a frequent situation in RL, especially when sub-tasks are defined by RL episodes. Even environment shifts would not alleviate this problem. Therefore, virtually all CL methods will encounter problems in such cases.

**Small number of sub-tasks** Most CL models tacitly assume that the number of sub-tasks is small, as they need to store significant amounts of data for each sub-task. As an example, EWC [23] needs to store a Fisher Information Matrix (FIM) plus all trainable parameters of a DNN for each sub-task, apart from the fact that the loss function picks up a new term for each additional sub-task. Parameter isolation methods face similar problems: either they need to track which parameter is important for which sub-task, or new structures must be added to the ML model for each sub-task, both of which are costly in terms of memory and computational effort. Exceptions are GEM and A-GEM [18], [25], since they just store a small percentage of samples per sub-task, and thus the number of sub-tasks can be large. Pure rehearsal-based methods such as NSR+ [27] pursue a similar strategy, with similar advantages and high computational efficiency (storing a few samples is cheap). Pseudo-rehearsal methods such as, e.g., [28], do not store samples but train a generator. This can be done for any number of sub-tasks, but incurs a huge *fixed* computational and memory cost, and is therefore slightly less suited.

**Sharp and known boundaries** In addition, CL sub-task-boundaries are usually assumed to be sharp, and their onset known. If common CL approaches are to be applied, transitions between sub-tasks must be detected. However, sub-task onsets in RL may be smooth to a point where even the concept of a sub-task becomes questionable.

### *E. Goals and contribution*

In order to apply CL methods to RL, we created an RL scenario where non-stationarities are mainly characterized by environment shifts, see Fig. 2. Sub-tasks are overlapping but non-contradictory (e.g., high rewards in situations that previously obtained low rewards are non-existing) and their onsets are assumed to be known. Although we use only a small number of tasks, we chose CL methods that could scale well even for a moderate amount of sub-tasks: Gradient Episodic Memory (GEM) [18] and Averaged Gradient Episodic Memory (A-GEM) [25] and Naive Sample Rehearsal Plus (NSR+) [27]. As a baseline for RL performance, a matrix-based model and the conventional ER approach are investigated.

We present several novel contributions:

- Review of existing CL methods w.r.t. suitability for reinforcement learning
- Adaptation of CL reference methods to the domain of RL
- Experimental comparison of various CL methods in a realistic RL scenario

## II. Methods

**Simulation environment** We use Robot Operating System 2 (ROS2) and Gazebo 11[1] to physically simulate and control a line-following robot. ROS2 is a common middleware framework for robotics, which allows decoupled communication between multiple heterogeneous entities (nodes). We employ it to communicate with the simulated robot by sending control signals and receiving sensor data. Gazebo 11 is a physics-based simulator for ROS2 and includes plugins for various predefined types of sensors as well as actuators. Our simulations consist of at least one simulated world (environment), each of which contains a particular racetrack, and the robot model (interacting agent), which can observe via its sensors and perform actions via its actuators. Control signals and sensor readings are transmitted at a defined frequency (5 Hz).

**Racetracks** All racetracks consist of separate lines on a plane, which can have an arbitrary course, but never crosses. Subsections of them can therefore be divided into three categories: straight, left turn or right turn. Context changes can

[1] docs.ros.org and gazebosim.org for further information

be reasonably interpreted as transitions between these distinct categories, see Fig. 2.

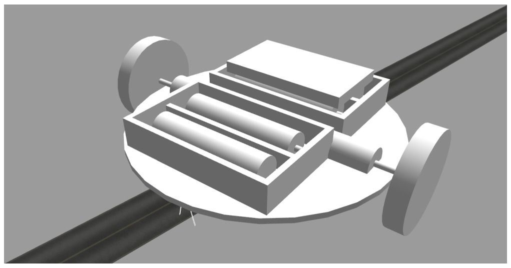

Fig. 3. The simulated robot, which closely models the popular $3\pi$ robot.

**Simulated robot** The simulated robot is modeled after the popular $3\pi$ robot from Pololu robotics, see Fig. 3. It is controlled by a differential drive, with two wheels (radius: $\approx 1.55\,cm$, separation: $\approx 9\,cm$) driven by independent motors. In addition, there is a passive caster wheel for balancing.

TABLE I
KEY DATA OF THE USED ROBOT, WHICH IS A REPLICA OF THE POLOLU $3\pi$.

| Height | Width | Length | Weight |
|---|---|---|---|
| $\approx 3\,cm$ | $\approx 9.5\,cm$ | $\approx 9.5\,cm$ | $\approx 135\,g$ |

**Action space** The action space consists of 9 discrete actions $a(t)$: the three basic actions (drive straight, turn left or turn right) executed at three different speeds. An action is a 2-tuple containing a value for the speed of each wheel (left and right). The individual wheel speeds for all actions are given in Tab. II.

TABLE II
DISCRETE ACTION SPACE, WHEEL SPEEDS ARE GIVEN IN METER PER SECOND ($\frac{m}{s}$).

| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Left speed | 0.00 | 0.05 | 0.10 | 0.00 | 0.00 | 0.05 | 0.05 | 0.10 | 0.10 |
| Right speed | 0.00 | 0.05 | 0.10 | 0.05 | 0.10 | 0.10 | 0.00 | 0.00 | 0.05 |
| Type | | Straight | | | Left | | | Right | |

**Reward** The reward signal is calculated based on the deviation $d$ (in pixels) of the left edge of the line to the center of the image, which is assumed to have width $W$:

$$r(t) = 0.5 - \left| \frac{d - \frac{W}{2}}{\frac{W}{2}} \right|, \text{ with } d \in [0, W] \quad (2)$$

The deviation is computed by image processing, assuming that the line is significant darker than the remaining ground plane. This measure provides a "dense" reward, which is computable immediately for each iteration in our feedback loop. The default range of the signal is within $[-0.5, +0.5]$, but terminal states and neutral actions (no movement) are penalized by a value of $-1.0$. A terminal state always occurs when image processing is unable to detect the left edge of the line in the image. The reward function does not reward or penalize different speeds, except for the neutral action (no movement).

**State Space** The agent observes its environment by a downward directed camera ($5 \times 100 \times 3$ pixels). Matrix controllers are preprocessing such data to produce a scalar state $s(t)$, which is realized by the deviation $d$. All other controllers stack the last $n = 5$ received images to obtain a state representation $\vec{s}(t)$ of dimension $5n \times 100 \times 1$.

**Episode definition** The investigated scenario describes a control problem with a theoretically infinite length of episode. A single episode can therefore be of arbitrarily length and trajectories consist of varying numbers of samples. An episode ends only if a terminal (*invalid*) state is reached, and the agent must be reset in order to start over again.

**Sub-task definition** Each new sub-task consists of segments in which novel skills must be acquired, but also consist of sections from previous sub-tasks, which require the adoption of already learned ones. In this work, for the time being, only a deliberate change of the racetrack (referred as environment shift) will be considered as a sub-task. This relaxation permits the application of existing state-of-the-art methods.

## A. CL approaches

We investigate three different CL approaches. All of them are replay-based methods and should replace the original ER.

**GEM and A-GEM** Gradient Episodic Memory (GEM) [18] and Averaged Gradient Episodic Memory (A-GEM) [25] are both constraint-based replay approaches. They project the current loss gradient onto loss gradients computed from samples in the buffer, if deviations are judged to be excessive. GEM and A-GEM share a common principle, but differ in the way of computing loss gradients from buffered samples. The primary parameter of (A-)GEM is the size $M$ of buffer $\mathcal{M}$. GEM and A-GEM do not define a specific sample selection strategy and propose to store randomly $m = \frac{M}{t}$ samples of the current sub-task. However, the selection requires information about sub-task boundaries in order to assign distinct partitions per sub-task.

**NSR+** Naive Sample Rehearsal Plus (NSR+) [27] is another replay approach, which can be executed with the information about task boundaries, but unlike the (A-)GEM the sample-selection does not depend on it. NSR+ has two parameters: the size $M$ of buffer $\mathcal{M}$ as well as the ratio $r$ of replayed samples. The selection strategy keeps the $M$ worst-performing samples in a buffer, for this the buffer is evaluated as a separate mini-batch and compared with the losses of the current mini-batch.

## B. Models

Q-learning is performed using matrix-type and DQN-type models. The matrix-type model is trained either by the original update rule [6] or by the update rule of speedy Q-learning [36]. DQN-type models are trained either by the standard DQN update rule [37] or by double Q-learning [38].

**Q-tables** First of all, matrix-type models (Q-tables) are used to obtain reinforcement learning in the original manner and to compare with the other models. A Q-table consists of $(|S| \times |A|)$ entries and is able to map discrete states onto discrete actions. Each entry (state-action tuple) represents the expected return as Q-value $Q(s(t), a(t))$ if this action would be chosen in the given state. Therefore, high values are to be preferred, and the maximum value is to be determined via the respective state. Like all Q-learning models, Q-tables are updated after each single time step. However, only the value of the last state-action tuple will be updated, thus all other values remain unaffected.

**DQN** Instead of matrices, DNNs can be used as back-end to obtain deep reinforcement learning (DRL) by deep Q-networks (DQNs). This has multiple advantages: the state space can be continuous, the state can be in context, the state can be consisted of multi-modal data and the model can generalize between samples. The type of network is freely selectable and independent of the chosen algorithms. Some networks are more or less suitable for different scenarios, for simplicity we use conventional convolutional neural networks (CNNs). Like Q-tables, DQNs perform an update after every time step but contrary this involves modifications to all trainable parameters ($\theta$). Each update is w.r.t. the current sample or mini-batch, and can therefore amplify the effect of catastrophic forgetting (CF) if gradients are opposite.

### C. RL Sub-tasks

TABLE III
SUB-TASK DEFINITIONS FOR CONTINUAL RL AS EMPLOYED IN THIS ARTICLE. EACH SUB-TASK CONSISTS OF A TOTAL OF 50,000 TIME STEPS AND ADDS NEW CAPABILITIES.

| **ID** | **1** | **2** | **3** |
|---|---|---|---|
| **Type** | straight | zero | slalom |
| **Start it.** | 0 | $50,000$ | $100,000$ |

CL problems are typically divided into individual *sub-tasks*, following the terminology from [8]. Frequently, class labels are used as a basis to group samples into disjoint sub-tasks, since most work on CL is in the context of supervised learning. Since class labels do not exist in RL, this definition of sub-tasks cannot be used for RL. Instead, we focus on environment shifts for defining RL sub-tasks, which we create by manipulating the track definitions (Fig. 2) at defined intervals, see Tab. III. In contrast to the main body of CL, sub-tasks as understood here are not disjoint. More precisely, each new sub-task requires additional skills to obtain high rewards, but also re-uses skills acquired in previous sub-tasks. For example, an agent solving the second sub-task can re-use the skill to drive on a straight line (acquired in the first sub-task), but needs to acquire the skill to execute curves to the left in addition to that.

By implementing sub-tasks as synthetic environmental shifts, see Fig. 2, we control when they occur and make this information available to the RL process. Concretely, we reset the parameter $\epsilon$ which controls exploration behavior to a value of $\frac{1}{t}$ for each sub-task, after which the normal decaying-$\epsilon$-greedy is performed as usual. Moreover, the utilized Cl methods can use this information to, e.g., generate new partitions. In a fully realistic RL application, the agent would have to discover such sub-task boundaries by itself.

### D. Benchmarking

While in supervised learning the terms benchmarks and datasets are synonymous, in reinforcement learning a distinction must be made. Henceforth, the term "benchmark" describes a possibility to evaluate the current policy ($\pi$) of an agent against certain metrics to capture its performance, but without predetermined samples to serve as a "dataset". The fundamental issue here is that the reward is not only a function of the current observation, but also of a previous action. In the same way, the current observation depends on previous actions. In order to perform an offline evaluation of a learned policy, this policy would be required to take a predetermined sequence of actions, which in turn would prevent a thorough evaluation of the policy whose goal, after all, is to *select* appropriate actions.

Any evaluation therefore has to be performed *within* an environment using a fixed policy (i.e., by setting the learning rate to 0 and suppressing exploration), in contrast to the offline evaluation of classifiers that is common in CL. We choose to perform benchmarking directly after training on a sub-task is completed.

### E. Metrics

We are using the history of measured rewards to evaluate learned policies. An elementary quantity of our evaluation is the sum over all rewards $\Sigma_e$ received during a single RL episode $e$. We chose summation over averaging since rewards are bounded, and the sum thus reflects not only the total obtained reward, but also the length of the episode. By default, we plot the $\vec{\Sigma}_t = \Sigma_e, \forall e \in t$ over the whole sub-task $t$ to obtain a visualization with intuitive meaning. We can condense the information contained in $\vec{\Sigma}_t$ further by plotting an exponentially smoothed version, or even averaging them to obtain a scalar quality measure $\Sigma_t$ (which is strictly of a "the higher the better" nature).

As already described in Sec. II-D, each action during evaluation is chosen w.r.t. the current *static* policy, without choosing some of them randomly, e.g., by a $\epsilon$-greedy strategy.

## III. EXPERIMENTS

Different agents have to perform our benchmark in real-time. Updates are executed in a strictly online fashion during the entire simulation. After each sub-task, the respective policy of the agent is stored to compare experiments successively. Finally, these agents should at least beat the baselines.

### A. Baselines

In RL, baselines are closely linked to the respective benchmark and are non-transferable. Furthermore, the variance between isolated runs can vary significantly, especially if the number of iterations is limited. For comparison, all methods should be applied under identical conditions. We chose state-of-the-art methods like Q-tables and DQNs with ER as possible lower bounds. Both are commonly used in the domain of

(D)RL, whereas we apply them here in the context of continual RL (CRL). Both baselines (Tab. IV) achieve predominantly

TABLE IV
PERFORMANCE OF THE BEST HYPER-PARAMETER SETUP FOR ALL BASELINES, REPRESENTED BY $\Sigma_t$ FOR EACH SUB-TASK.

| Baseline | Type | Sub-task 1 $\Sigma_1$ | Sub-task 2 $\Sigma_2$ | Sub-task 3 $\Sigma_3$ | Overall |
|---|---|---|---|---|---|
| **Q-tables** | history | $-911.833$ | $-5973.567$ | $-383.633$ | $-7269.033$ |
| | last policy | $-164.333$ | $-622.000$ | $17.600$ | $-768.733$ |
| **DQNs + ER** | history | $-1744.880$ | $-2777.807$ | $-996.160$ | $-5518.847$ |
| | last policy | $-57.053$ | $-256.953$ | $-55.227$ | $-369.233$ |

negative scores, while the second sub-task is performing the worst. Negative scores indicate, that an agent is not able to keep the left edge of the line centered. This does not have to result in a loss of the line from its field of view, but rather in an inconvenient driving style. The benchmark itself seems already challenging, as the negative first sub-task indicates, regardless of environmental shifts being performed. To rule out the possibility of having too few iterations to acquire needed skills, we additionally run these experiments 10 times longer for $500,000$ iterations per sub-task.
The runs differ because they are influenced by numerous *random* factors, e.g., content-related aspects as exploration or initial decisions and also technical aspects as transmission time or calculation time. However, the presented values are averaged over multiple runs (min 3) to be as representative as possible.

### *B. Results*

The results of the investigated CL methods are shown in Tab. V. Compared with the baselines of Tab. IV, each individual method results in a significant higher score. But here, too, the second sub-task performs the worst. Our results

TABLE V
PERFORMANCE OF THE BEST HYPER-PARAMETER SETUP FOR ALL CL METHOD, REPRESENTED BY $\Sigma_t$ FOR EACH SUB-TASK.

| Baseline | Type | Sub-task 1 $\Sigma_1$ | Sub-task 2 $\Sigma_2$ | Sub-task 3 $\Sigma_3$ | Overall |
|---|---|---|---|---|---|
| **GEM** | history | $3143.780$ | $-1899.703$ | $2303.273$ | $3547.350$ |
| | last policy | $450.300$ | $-109.193$ | $273.873$ | $614.980$ |
| **A-GEM** | history | $5568.810$ | $-4470.237$ | $2663.120$ | $3761.693$ |
| | last policy | $431.017$ | $-386.953$ | $316.243$ | $360.307$ |
| **NSR+** | history | $4513.033$ | $-4049.577$ | $2746.470$ | $3209.927$ |
| | last policy | $664.837$ | $-352.727$ | $304.730$ | $616.840$ |

demonstrate that RL methods as Q-learning converges faster in combination with CL methods, since the first sub-task already has a high positive value. New skills after sub-task changes (realized by environment shifts) are also adapted faster than with conventional ER approaches. While ER trains over mini-batches, Q-Tables and the CL methods use single samples per update.

Fig. 4. All $\Sigma_e$s of an outstanding A-GEM run. Data points are scaled by episode length and colored by the average. Sub-task boundaries are marked as vertical lines.

If an experiment has too few iterations (time steps), the results depends strongly on the learning process initiation, which leads to divergent or almost inconsistent results. In addition to already mentioned factors, early good decisions in particular favor this, since wrong decisions merely rule them out but do not offer a selectable option. The run visualized in Fig. 4 is nevertheless a representative trajectory of A-GEM experiments, as they are generally able to adapt new sub-tasks quickly. Only the second sub-task performed in this case significantly better than average.

## IV. DISCUSSION

The experiments presented in Sec. III-B allow to draw the following conclusions:

**CL methods can be transferred to the RL domain** Although we consider a simplified RL scenario here, where concept drift occurs only at sub-task boundaries, we nevertheless showed that the assumptions made by several dedicated CL methods can be made compatible with RL. A significant percentage of CL methods were, however, excluded from our considerations due to memory or scalability issues, see Sec. V for a justification.

**CL methods can improve RL performance** W.r.t. to the baseline of experience replay, we observe that CL methods achieve faster convergence on single sub-tasks. This shows that more knowledge is retained by CL methods. Experience replay converges, but struggles with environment shifts, since it takes some time until the replay buffer is sufficiently populated with new sub-task samples.

**Sample selection is essentially** Both investigated CL methods perform replay, i.e., they re-use stored samples from past sub-tasks. Our investigation indicates that sample selection is an essential ingredient, especially in RL. To illustrate this, we recall that sub-tasks in RL (whatever their definition) are never disjoint. Even very rudimentary sample selection strategies, like selecting the worst-performing samples as we did for NSR+, increases the chances of storing only samples characteristic to a certain sub-task.

## V. CONCLUSION AND OUTLOOK

In this article, we consider a RL problem with known, hard sub-task boundaries. Sub-tasks are not disjoint but overlapping, and no contradictions occur between sub-tasks, meaning that for the same sensory state, the Q-values do no change between sub-tasks. In this somewhat restricted RL setting, we could show that two common CL methods, (A-)GEM and NSR+ can outperform the common RL baseline of experience replay. In this setting, we expect other CL methods to be applicable as well, e.g., generative replay and GMR. However, several open issues remain and will need to be addressed in future work on CRL.

**Known sub-task boundaries** First of all, virtually all existing CL methods depend on the knowledge of sub-task boundaries, but are themselves incapable of detecting them. Thus, CL algorithms with outlier detection capacity should be beneficial in this context, or else an additional mechanism performing the detection of sub-tasks. However, the case of hard sub-task boundaries, as we introduced it here by the concept of environment shifts, is not a common one in RL. Rather, data statistics change gradually, at least from the learner's point of view. The absence of hard sub-tasks may exclude many CL approaches (e.g., constraint-based methods), or else force a major redesign.

**Contradictions** Any RL agent is likely to encounter conflicting data statistics, where the same state-action tuples will get assigned very different rewards as a consequence of, e.g., concept drift. Most CL approaches implicitly assume that knowledge from previous sub-tasks should be preserved, and not modified. This is no longer possible if systematic contradictions occur. Identifying or designing a CL method that can work with conflicting data statistics will be a challenge to solve.

**Scalability** The scaling behavior of many CL approaches w.r.t. time and memory is unfavorable. Some approaches require the storage of a complete model for each new sub-task, whereas replay methods must generate an ever-increasing amount of previous data for each new sub-task. Since RL is often conducted over long time scales, with a succession of many sub-tasks, employing a scalable CL approach will be vital.

## Supplementary Material

### A. *DQN back-end*

Table VI presents detail about the back-end, we employed for all experiments. The model is an intentionally small CNN and consists of two convolutional layers as well as two fully connected layers and the output layer. For activation, ReLU is applied and after each convolution a maxpooling is performed. Compared to the state-action space of our Q-tables ($33 \times 9 = 297$), this model ($48,613$) seems already overparameterized.

TABLE VI
Summary of the CNN back-end architecture. The number of trainable parameters is $48,613$ in total.

| **layer** | **type** | **properties** | **parameters** |
|---|---|---|---|
| input | identity | $5 \times 100 \times 3$ | |
| conv 1 | convolutional | filters: 4 | 368 |
| | | kernel size: (3, 5) | |
| | | strides: (1, 1) | |
| maxp 1 | pooling | pool size: (1, 2) | |
| relu 1 | activation | ReLU | |
| conv 2 | convolutional | filters: 8 | $1,936$ |
| | | kernel size: (3, 5) | |
| | | strides: (1, 1) | |
| maxp 2 | pooling | pool size: (1, 2) | |
| relu 2 | activation | ReLU | |
| *flatten* | *reshape* | *352* | |
| fc 1 | dense | 100 | $35,300$ |
| relu 3 | activation | ReLU | |
| fc 2 | dense | 100 | $10,100$ |
| relu 4 | activation | ReLU | |
| output | dense | 9 | 909 |
| linear | activation | linear | |

### B. *Hyper-parameters*

The list of all hyper-parameters we used in our specific RL setup is given by Table VII. For better clarity, these have been grouped by subject. The majority is needed to perform Q-learning or RL in general, only a few of them belong to CL methods. We chose the sizes of CL buffers to be $\frac{1}{10}$-th of the ER buffer sizes.

TABLE VII
List of all hyper-parameters and their corresponding values, we consider in this article.

| **subject** | **name** | **values** |
|---|---|---|
| generic | transmission frequency | 5 Hz |
| generic | state space complexity | 33 tuples (Q-tables) |
| | | 500 pixels (DQNs) |
| generic | action space complexity | 9 tuples |
| generic | sequence size | 1 sample (Q-tables) |
| | | 3 samples (DQNs) |
| task | iterations sub-tasks (train) | 50,000 |
| task | iterations sub-tasks (eval) | 5,000 |
| algorithm | type | original, speedy (Q-tables) |
| | | original, double (DQNs) |
| algorithm | learning rate | 0.5, 0.1 (Q-tables) |
| | | 1e-2, 1e-3 (DQNs) |
| algorithm | discount factor | 0.75 |
| algorithm | repeat action | 1 |
| model | update frequency | 1 |
| model | mini-batch size | 1 (non ER) |
| | | 8, 16 (ER) |
| model | repeat update | 1 (non CL) |
| | | 3 (CL) |
| model | loss function | Huber |
| model | optimizer | SGD |
| $\epsilon$-greedy | strategy | non-linear decay |
| $\epsilon$-greedy | start | $\frac{1}{t}$ |
| $\epsilon$-greedy | stop | 0.005 |
| $\epsilon$-greedy | step | 0.001 |
| ER buffer | sample selection | reservoir sampling |
| ER buffer | buffer size | 10,000, 30,000 |
| (A-)GEM | averaged | yes, no |
| (A-)GEM | buffer size | 1,000, 3,000 |
| (A-)GEM | memory strength | 0.5 |
| NSR+ | buffer size | 1,000, 3,000 |
| NSR+ | replay ratio | 1.0 |
| NSR+ | sample selection | worst performing |

### C. *Policy evaluation*

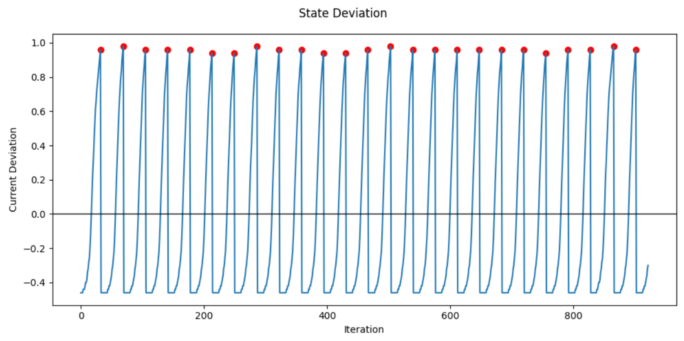

Fig. 5. The agent's deviation from the line, evaluated before training.

Figures 5 to 7 visualize the evaluation of line deviation within the observed images for $1,000$ iterations. If the agent

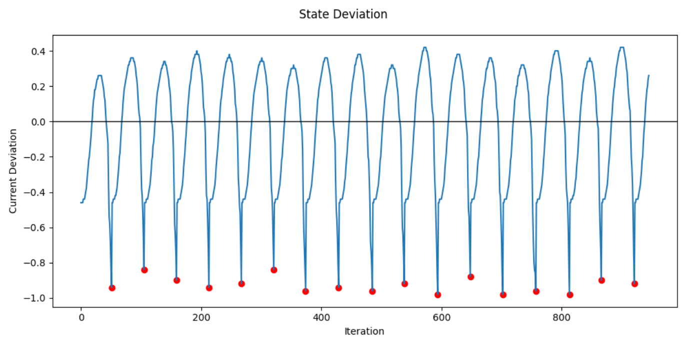

Fig. 6. The agent's deviation from the line, evaluated early after training.

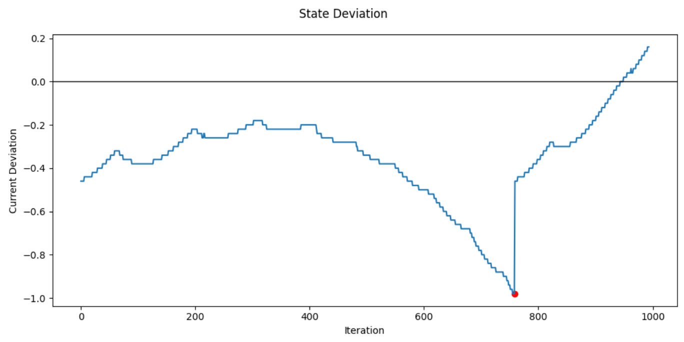

Fig. 7. The agent's deviation from the line, evaluated later after training.

drives perfectly centered over the left edge of the line, the deviation will be $0$. If the agent deviates to the left, the edge is moving to the right in the received image accordingly and vice versa. Finally, the boundary of $-1$ encodes the edge in the leftmost position within the image, and $+1$ thus the rightmost position.
However, Fig. 5 shows an evaluation of an untrained agent, which always drives to the left and loses the track immediately. The evaluation of the same agent after some train steps is given in Fig. 6. Now, if the agent deviates too much to the left, he has learned to countersteer to the right, but still loses the track quickly. Nevertheless, this represents the first acquired skill of the investigated agent. Finally, Fig. 7 visualizes this agent at an even later stage. It can be seen that actions are now much more finely motorized than before, but sometimes the agent still loses the track.

Fig. 8. Frequency of actions, which the agent takes if its policy is evaluated on the first racetrack (straight only), after learning the first sub-task.

Fig. 9. Frequency of actions, which the agent takes if its policy is evaluated on the second racetrack (straight and left curve), after learning the first and second sub-task.

Frequencies of chosen actions are shown in Figures 8 and 9, where Fig. 8 evaluates the first sub-task and Fig. 9 the second one. In both figures, all actions that have the same wheel speed (driving straight) are colored gray. Actions which have a higher wheel speed on the right than on the left (perform a left curve) are shown in orange, and actions which have a higher wheel speed on the left than on the right (perform a right curve) are shown in blue.
As expected, w.r.t. the first sub-task the agent takes straight actions the majority of time, so they are most likely for this track. If the agent deviates from the line, other steering impulses are necessary, but this case is less common. However, the number of diverse actions is still large, as $7/9$ are used. During driving the second sub-task, frequencies are changing, because the racetrack has additionally sections of left curves. Correspondingly, orange actions (performing a left curve) are more frequently chosen, than before. To counteract still a possible overdrive, the frequency of blue actions (performing a right curve) is also increasing. At the same time, it can be noted that the number of diverse actions decreases, by a kind of self-quantization of the action space.

## References

[1] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, 2020. 1, 2, 3

[2] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, "Towards continual reinforcement learning: A review and perspectives," 2020. 1, 2, 3

[3] M. D. Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks." *IEEE transactions on pattern analysis and machine intelligence*, 2021. 1, 2

[4] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019. 1, 2

[5] R. S. Sutton and A. G. Barto, "Introduction to reinforcement learning," 1998. 1

[6] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, 1992. 1, 4

[7] A. R. T. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *ESANN European Symposium on Artificial Neural Networks*, 2016. 2

[8] B. Pfülb and A. Gepperth, "A comprehensive, application-oriented study of catastrophic forgetting in dnns," in *ICLR International Conference on Learning Representations*, 2019. 2, 5

[9] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *NIPS Conference on Neural Information Processing Systems*, 2019. 2

[10] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *AAAI Conference on Artificial Intelligence*, 2018. 2

[11] A. Prabhu, P. Torr, and P. Dokania, "Gdumb: A simple approach that questions our progress in continual learning," in *ECCV European Conference on Computer Vision*, 2020. 2

[12] C. Zeno, I. Golan, E. Hoffer, and D. Soudry, "Task agnostic continual learning using online variational bayes," 2018. 2

[13] F. Normandin, F. Golemo, O. Ostapenko, P. Rodriguez, M. D. Riemer, J. Hurtado, K. Khetarpal, D. Zhao, R. Lindeborg, T. Lesort, L. Charlin, I. Rish, and M. Caccia, "Sequoia: A software framework to unify continual learning research," 2021. 2

[14] T. Lesort, M. Caccia, and I. Rish, "Understanding continual learning settings with data distribution drift analysis," 2021. 2

[15] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *ECCV European Conference on Computer Vision*, 2018. 2

[16] J. Knoblauch, H. Husain, and T. Diethe, "Optimal continual learning has perfect memory and is np-hard," in *ICML International Conference on Machine Learning*, 2020. 2

[17] T. Doan, M. A. Bennani, B. Mazoure, G. Rabusseau, and P. Alquier, "A theoretical analysis of catastrophic forgetting through the ntk overlap matrix," in *AISTATS International Conference on Artificial Intelligence and Statistics*, 2021. 2

[18] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *NIPS Conference on Neural Information Processing Systems*, 2017. 2, 3, 4

[19] R. Aljundi, M. Rohrbach, and T. Tuytelaars, "Selfless sequential learning," in *ICLR International Conference on Learning Representations*, 2019. 2, 3

[20] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "PathNet: Evolution channels gradient descent in super neural networks," 2017. 2, 3

[21] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," in *CVPR Computer Vision and Pattern Recognition*, 2018. 2, 3

[22] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016. 2, 3

[23] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, 2017. 2, 3

[24] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *NIPS Conference on Neural Information Processing Systems*, 2017. 2, 3

[25] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with A-GEM," in *ICLR International Conference on Learning Representations*, 2018. 2, 3, 4

[26] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. S. Torr, and M. Ranzato, "On tiny episodic memories in continual learning," 2019. 2

[27] B. Bagus and A. Gepperth, "An investigation of replay-based approaches for continual learning," in *IJCNN International Joint Conference on Neural Networks*, 2021. 2, 3, 4

[28] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *NIPS Conference on Neural Information Processing Systems*, 2017. 2, 3

[29] A. Gepperth and B. Pfülb, "Gradient-based training of gaussian mixture models for high-dimensional streaming data," *Neural Processing Letters*, 2021. 2

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016. 2

[31] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning*, 2020. 2

[32] M. Wołczyk, M. Zajac, R. Pascanu, L. Kucinski, and P. Miłoś, "Continual world: A robotic benchmark for continual reinforcement learning," in *NIPS Conference on Neural Information Processing Systems*, 2021. 2, 3

[33] S. Powers, E. Xing, E. Kolve, R. Mottaghi, and A. Gupta, "CORA: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents," 2021. 2, 3

[34] S. Kessler, J. Parker-Holder, P. J. Ball, S. Zohren, and S. J. Roberts, "Same state, different task: Continual reinforcement learning without interference," 2021. 3

[35] T. Zhang, X. Wang, B. Liang, and B. Yuan, "Catastrophic interference in reinforcement learning: A solution based on context division and knowledge distillation," 2021. 3

[36] M. G. Azar, R. Munos, M. Ghavamzadeh, and H. J. Kappen, "Speedy Q-learning," in *NIPS Conference on Neural Information Processing Systems*, 2011. 4

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. 4

[38] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *AAAI Conference on Artificial Intelligence*, 2016. 4