Inter-Task Similarity for Lifelong Reinforcement Learning in Heterogeneous Tasks

Sergio Arredondo Serrano, Dr. José Martínez Carranza,
Dr. Luis Enrique Sucar Succar

Technical Report No. CCC-21-001
May, 2021

Luis Enrique Error 1,
Santa María Tonantzintla,
72840, Puebla, México.

Abstract

Reinforcement learning (RL) is a learning paradigm in which an agent interacts with the environment it inhabits to learn in a trial-and-error way. By letting the agent acquire knowledge from its own experience, RL has been successfully applied to complex domains such as robotics. However, for non-trivial problems, training an RL agent can take very long periods of time. Lifelong learning is a learning setting in which the agent learns to solve tasks sequentially, by leveraging knowledge accumulated from previously solved tasks to learn better/faster in a new one. Most lifelong learning works heavily rely on the assumption that tasks are similar to each other. However, this may not be true for some domains that could benefit from adopting a lifelong learning approach, *e.g.*, service robotics. Therefore, in this research proposal we address the problem of learning to solve a sequence of RL tasks that differ in their state-action space. We propose an inter-task similarity measure as the basis for a lifelong reinforcement learning agent. By employing a similarity measure as a heuristic, we expect the agent to effectively select knowledge from previous tasks to improve its performance in a new one, despite the mismatch between their state-action spaces. As the agent accumulates new knowledge, the similarity measure can be used to assess which pieces of knowledge can be stored together without harming each other. We present preliminary results on the development of an inter-task similarity measure. After evaluating on a set of RL tasks, we were able to establish correlation to some degree between the similarity scores and the benefit of transferring knowledge between the pairs of tasks. These results seem to suggest that a similarity measure approach is a feasible solution to transfer knowledge between tasks with different state-action spaces. Finally, the main expected contributions of this work are: i) an inter-task similarity measure, ii) a transfer learning algorithm and iii) a lifelong reinforcement learning algorithm, all three for tasks that do not share the state-action space.

Keywords: Lifelong Reinforcement Learning, Inter-task Similarity, Heterogeneous Tasks, Robotics.

Contents

Acronyms

BKT Backward Knowledge Transfer.

CF Catastrophic Forgetting.

FKT Forward Knowledge Transfer.

LML Lifelong Machine Learning.

MDP Markov Decision Process.

MTL Multi Task Learning.

RL Reinforcement Learning.

TL Transfer Learning.

TSM Task Similarity Measure.

1 Introduction

Reinforcement Learning (RL) is a subarea of machine learning concerned with sequential decision-making. RL encompasses a class of learning problems that differ from other learning paradigms (supervised and unsupervised learning) mainly due to how the learner acquires knowledge: through its own experience [1]. An RL agent learns as a result of interacting through trial-and-error with its environment. Hence, RL provides a powerful tool to address problems that are too difficult to model with precision. Instead, one can let the learning agent interact with the environment until it figures out how to properly behave.

Recent developments in the area of RL have led to decision-making agents that achieved outstanding results in tasks that, until a few years ago, were considered too complicated for a computer to learn how to solve them, *e.g.*, pixel-based games [2, 3, 4] and the game of Go [5]. Even more than video games, robotics applications require robust decision-making systems in order to solve complex tasks. However, considering that RL methods usually require long training periods, it is infeasible to let a real robot interact with its surroundings until it learns, because of the wear and tear this would cause on it (*curse of real-world samples* [6]). Thus, additional methods are required to reduce the training time in RL.

In recent years, Transfer Learning (TL) methods have shown a great capability to decrease the training time required in RL tasks [7, 8], which is possible by transferring useful knowledge between a pair of tasks. In Lifelong Machine Learning (LML), a special case of TL, the agent learns to solve tasks sequentially, leveraging the knowledge from previous tasks (see Fig. 4). Thus, the agent is able to undertake large sequences of tasks. However, there are some robotic domains (*e.g.*, service robotics) with a great task diversity [9] that limits the usage of TL methods, since inter-task similarity is necessary for them to work.

In this document, we present a research proposal with the aim of extending the capabilities of current lifelong reinforcement learning systems, to cope with scenarios in which tasks differ in their state-action spaces (*i.e.*, heterogeneous tasks [10]). Our research efforts will be focused on using inter-task similarity methods to estimate the

benefit of transferring knowledge between a set of previously learned tasks and a new one. Also, we believe that an inter-task similarity measure has the potential to aid a lifelong reinforcement learning agent to decide when a pair of tasks can be stored together, without harming each other. Thus, with a similarity-measure approach, we will develop a new lifelong reinforcement learning methodology to address tasks that do not share the state-action space.

2 Background

In this section, background knowledge that is relevant to our dissertation proposal is presented. First, basic definitions on RL and Markov decision processes are covered, followed by transfer learning, multi-task learning, lifelong learning and similarity functions concepts.

2.1 Reinforcement Learning

Reinforcement Learning (RL) a subarea of machine learning that addresses sequential decision-making problems. The main objective of an RL algorithm is to learn a *satisfactory* behavior, through a trial-and-error interaction of the learning agent with the environment (see Fig. 1). According to [1], RL is a machine learning paradigm on its own since it holds important differences with supervised learning and supervised learning. These differences are described below:

- Supervised Learning: the main difference lies in the source of knowledge from which the system learns. In supervised learning, an external entity (*e.g.*, a machine learning engineer) provides pairs of scenario examples and the correct action to perform in that situation (label). Conversely, an RL agent learns from its own experience, as it interacts with the environment.
- Unsupervised Learning: although unsupervised learning and RL have in common that labeled data is not required, the main difference between them is their objective. While in unsupervised learning the system strives to find structure in a collection of unlabeled data, an RL system will try to maximize a reward signal.

Additionally, the RL problem is formalized as finding the optimal control for a partially-known Markov Decision Process (see Section 2.2). In this formalization, the agent must be able to observe the state of the environment, take actions and have a goal (or a collection of goals). Thus, any technique that is capable of solving this class of problems is considered to be an RL method.

2.2 Markov Decision Processes

Markov Decision Processes (MDP) are a mathematical framework used to model sequential decision-making problems for dynamic systems, that is, systems in which the state changes over time [11]. A discrete MDP is formally specified by a tuple $\langle S, A, \Phi, R \rangle$, where

- S: finite set of states in which the agent can be found.
- A: finite set of actions the agent can perform.
- Φ: the transition function $\Phi : S \times A \times S \to [0, 1]$ specifies a probability distribution for every state-action pair (s, a), such that $\Phi(s, a, s')$ is the probability of the agent transiting to s' after executing action a from state s, where $s, s' \in S$ and $a \in A$.
- R: the reward function $R : S \times A \to \mathbb{R}$ specifies a scalar value for every state-action pair (s, a), such that $R(s, a)$ is the immediate reward signal the agent will perceive after executing action a in state s, where $s \in S$ and $a \in A$.

The purpose of specifying an MDP is to formally describe the setting in which an agent can interact with the environment. The transition function (Φ) represents the stochastic effects of the actions in the state of the system, whereas the reward function (R) represents (in an implicit form) the desired behavior for the agent. That is, in the reward function one should assign positive large values to pairs (s, a) such that executing a from s is desirable. On the other hand, one should penalize undesirable state-actions pairs by assigning large negative values.

2.2.1 Policy

In the context of discrete MDPs, a policy $\pi : S \rightarrow A$ is a function that returns an action a for every state s, where $s \in S$ and $a \in A$. Furthermore, in order to solve a decision-making problem, actions must be carefully selected, in such way that the criterion of what is considered a desirable behavior is fulfilled, as shown in Fig. 1.

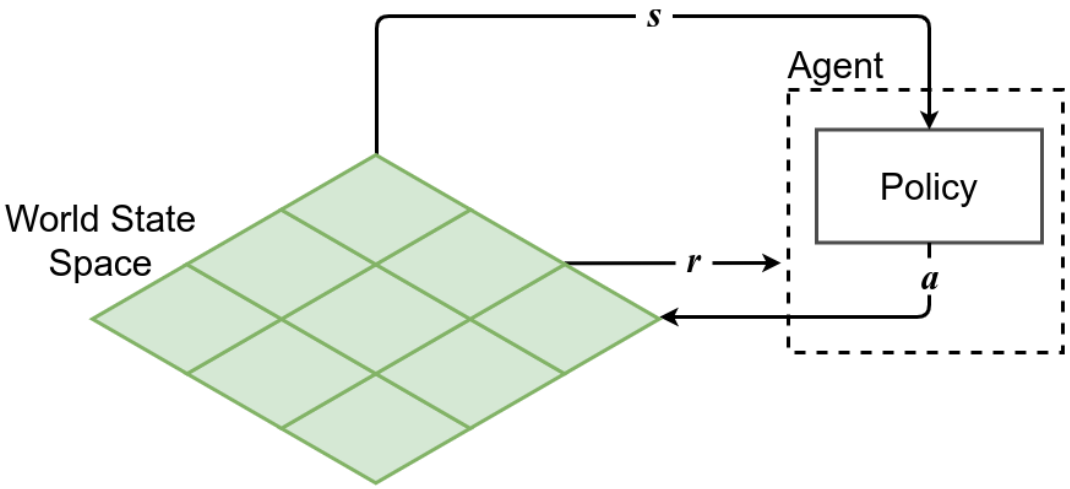


Figure 1: In an MDP setting, every time the agent executes an action (***a***) it will receive in return a reward signal (***r***) associated to the performed action and the current state (***s***) of the world (also called system or environment). Then, the agent consults its policy to perform the best action given the new state of the world.

Thus, an optimal policy π^* will select, for every state, the action that maximizes the expected reward, that is, the best possible action. For an infinite horizon MDP (those in which the number of steps that the task will last is unknown) with a discount factor γ (where $0 \leq \gamma < 1$), the optimal policy is defined by equation 2, which is obtained from equation 1 (Bellman's equation [12]).

$$V^{\pi}(s) = max_a \left\{ R(s,a) + \gamma \sum_{s \in S} \Phi(s,a,s') V^{\pi}(s') \right\} \tag{1}$$

$$\pi^*(s) = argmax_a \left\{ R(s,a) + \gamma \sum_{s \in S} \Phi(s,a,s') V^{\pi}(s') \right\} \tag{2}$$

$$V_t(s) = max_a \left\{ R(s,a) + \gamma \sum_{s \in S} \Phi(s,a,s') V_{t-1}(s') \right\} \quad (3)$$

In order to compute π^* for a fully specified MDP, an iterative algorithm is capable of finding such policy. For instance, value iteration is an algorithm that starts by assigning every state with a value of 0, and through an iterative process the value of every state is updated using equation 3. The algorithm stops once $|V_t(s) - V_{t-1}(s)| < \epsilon$ is satisfied for every state in S, given a margin of error ϵ [13].

2.2.2 Learning Approaches

The way an MDP policy can be computed greatly depends on how much information about the environment there is available. That is, the more information the agent has, the easier it will be to compute a good policy. According to [14], there are many ways in which a policy can be obtained, from which we present three of the most common approaches:

1. **Dynamic programming**: value iteration and policy iteration are examples of algorithms of this type of approach, in which the algorithm assumes that a fully specified MDP is available (*i.e.*, $\langle S, A, \Phi, R \rangle$). Therefore, as the system has a full model of the environment (which is also assumed to be correct), no interaction is required.

2. **Model-based**: these methods relax the assumption dynamic programming approaches make on the availability of the full MDP, as they interact with the environment in order to estimate an approximated model of it, *i.e.*, Φ and R. This approach is mostly used in environments where the model is stationary but unknown, hence, once the model is estimated it can be reused.

3. **Temporal difference (TD)**: methods that learn an action-value function $Q : S \times A \rightarrow \mathbb{R}$ by backing up all the rewards that have been perceived through time. For a pair (s, a), $Q(s, a)$ represents the expected return when a is executed from s. At any time, the best current policy is equivalent to select the highest valued action form the current state, *i.e.*, $\underset{a}{\text{argmax}} Q(s, a)$. Q-learning [15] and Sarsa [16] are examples of TD algorithms.

4. **Deep Reinforcement Learning**: according to [17], deep RL results from using a deep neural network (*i.e.*, a neural network with two or more layers) to approximate an RL component, such as a value function, a policy, a transition model or a reward model. The parameters that describe these components consist of the weight and bias values in the neural network. Deep Q-Network [3] and A3C [18] are some examples of deep RL algorithms.

2.3 Transfer Learning

The purpose of transfer learning (TL) is to improve the performance of machine learning methods by means of transferring knowledge between tasks [19]. Furthermore, according to [14], the main insight that motivates the use of TL methods is that generalization might occur across tasks, and not only within them. Although several taxonomies have been proposed for TL methods (*e.g.*, based on the availability of labeled data [20] or the goal in a multi-agent setting [21]), we present a taxonomy based on the availability of tasks and the order in which they must be learned (see Fig. 2). Additionally, the performance of a TL method applied to RL tasks can be measured in multiple ways (see Fig. 3).

- Jumpstart: The performance of a learning agent at the beginning of the training process. This metric can potentially be improved if knowledge is transferred from a source task before the training begins in the target task.
- Asymptotic Performance: The final performance of a learning agent.
- Total Reward: The total reward accumulated by the agent. Visually, the area under the learning curve represents this metric.
- Transfer Ratio: This metric is equivalent to the ratio of the total reward accumulated by the transfer learning method and the total reward accumulated by the non-transfer learner.
- Time to Threshold: The required by a learning agent to achieve a pre-specified performance level.

Hence, TL methods can help an RL agent to reduce the time to the threshold (*learn faster*) and to raise its total reward and asymptotic performance (*learn better*). In

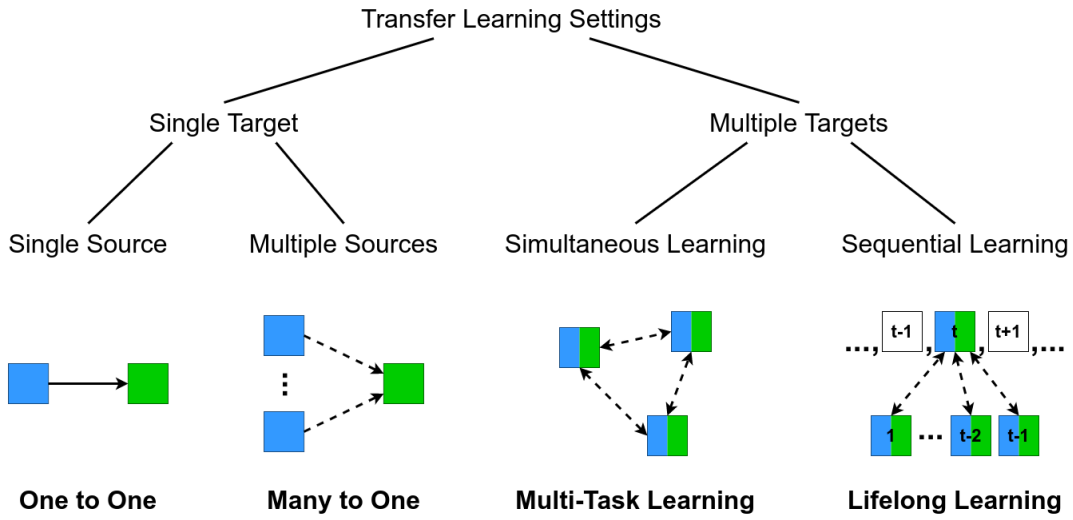


Figure 2: Classification of transfer learning settings based on the availability of source (blue) and target (green) tasks at the moment knowledge needs to be transferred, as well on the order in which tasks must be learned. Solid arrows indicate the direction in which knowledge is transferred, whereas dashed arrows indicate that it is up to the TL system to decide whether to transfer knowledge in that direction. In the multiple target setting, tasks can behave both as a source and target task.

RL, the learning time is known as sample complexity, which is the amount of data required by an algorithm to learn. Additionally, when a TL method harms the performance of a learning agent in a target task (*i.e.*, increases sample complexity or decreases the jumpstart, asymptotic performance, or total reward) is called negative transfer [14].

2.4 Multi-Task Reinforcement Learning

In Multi-Task Learning (MTL) there are multiple learning tasks, from which a subset of them is assumed to be related to each other in some way that is unknown. Then, the objective of an MTL system is to improve the generalization performance, since learning tasks that are related jointly can lead to a better performance [22]. MTL can be seen as a particular form of TL, in which each of the tasks that are being learned is both a source and a target task.

In the context of deep RL, MTL can be applied in different forms, *e.g.*, a single agent for multiple tasks or multiple agents for multiple tasks [23] (wherein the num-

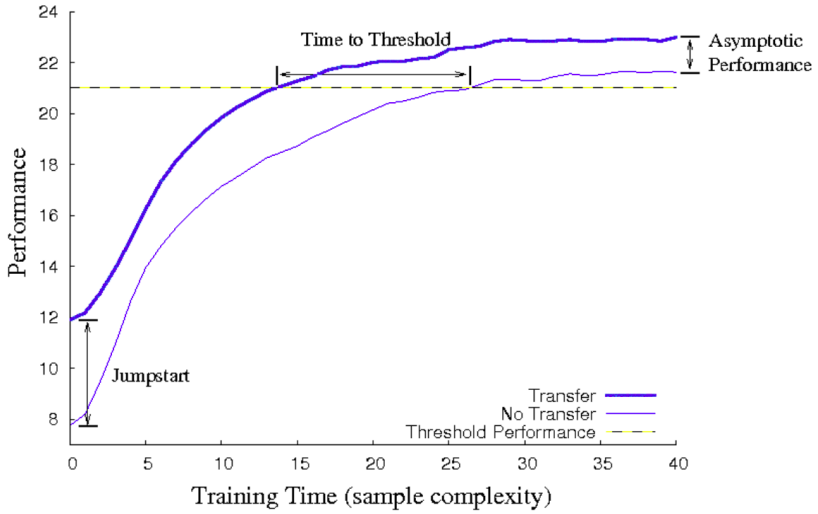


Figure 3: Example of improvement in the jumpstart, asymptotic performance, total reward and time to threshold as a result of transferring knowledge (figure taken from [14]).

ber of agents is not necessarily the same as the number of tasks). For instance, one could train a single network on multiple tasks by having multiple task-specific expert networks teaching it on their respective task [24]. Alternatively, one could train multiple networks to solve multiple tasks, while they share the first layers (usually the feature-extraction layers) [25].

2.5 Lifelong Reinforcement Learning

Lifelong Machine Learning (LML) is a continual learning process in which, at any time, the learning system has learned to perform a sequence of N tasks (tasks can be from different domains). When the $(N+1)$-th task is encountered, the learning system can exploit the past knowledge in its knowledge base to improve the learning process in the current task [26] (see Fig. 4).

In the context of RL, tasks are represented by MDPs, while the learning system (agent) can be modeled in a variety of ways. According to [27], lifelong RL systems can be classified in two categories, depending on how parameters are shared among the tasks learned:

- Single-model: In this approach, the learning system consists of only one model

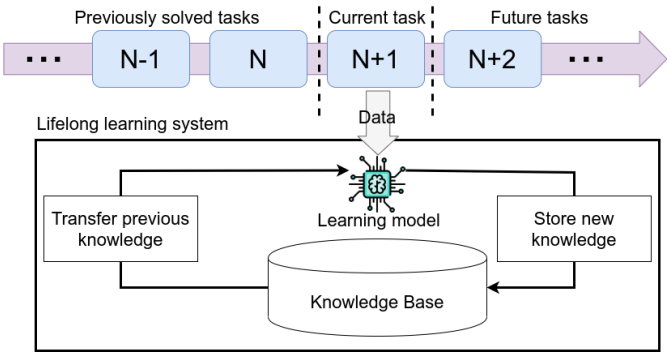


Figure 4: Lifelong learning system. The system learns to solve tasks sequentially. When it encounters with a new task, the system leverages knowledge from previous tasks that are similar to the current one. After the current task has been learned, the knowledge base is updated by storing the knowledge recently acquired.

to learn and store the knowledge of multiple tasks. However, it is necessary that one of the next two assumptions holds: all tasks must be very similar, or the model must be over-parameterized to learn the diversity among the tasks.

- Multi-model: The multi-model approach consists of a learning system constituted by a set of parameters that are shared among all tasks, and a collection of task-specific parameters. The set of shared parameters enable transferring knowledge across tasks, while the task-specific parameters are responsible for capturing the particularities tasks may have.

2.6 Distance and Similarity Functions

According to [28], the concepts of similarity and distance are important for artificial intelligence in general as they provide a way to organize, classify, and generalize over some class of objects. Despite distance and similarity functions have been widely explored for propositional representations (*i.e.*, feature vectors), they can also be employed to assess the similarity of objects from a graph-based representation (*e.g.*, an MDP).

Distance and similarity functions can be seen as a complement to each other. Distance functions assign larger values to pairs of objects that are *more dissimilar*, whereas similarity functions associate larger values to pairs of objects that are *more*

similar, or *closer*, with respect to some criterion. A formal definition of distance metric and similarity function [29] is provided below.

Definition 1 (distance metric) A *distance metric* d over objects in a set X is a function $d : X \times X \to [0, \infty)$ such that, for each $x, y, z \in X$ the following properties are satisfied:

- $d(x, y) \geq 0$ (Non-negativity)
- $d(x, y) = 0 \iff x = y$ (Identity)
- $d(x, y) = d(y, x)$ (Symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (Triangle inequality)

Definition 2 (similarity function) A *similarity function* s over objects in a set X is a function $s : X \times X \to [0, u]$, where u is an upper bound, and where for each $x, y \in X$ the following properties are satisfied:

- $s(x, y) \geq 0$ (Non-negativity)
- $s(x, y) \leq u$ (Boundedness)
- $s(x, y) = u \iff x = y$ (Identity)
- $s(x, y) = s(y, x)$ (Symmetry)

3 Related Work

This section presents a review of related work in the areas of measuring task similarity, MTL and LML in reinforcement learning tasks.

3.1 Task Similarity Measures

With the objective of compressing similar states to reduce the size of an MDP state space, in [30, 31] the concept of bisimulation is used as an equivalence class for

states, resulting in a partitioned state space where states that belong to the same partition are considered to be equivalent. Essentially, two states of a stochastic process (*e.g.*, an MDP) are deemed to be *bisimilar* if their transitions match and the transition results are also *bisimilar*. The authors propose two bisimulation-based semimetrics which differ in the metric employed to measure the similarity between probability distributions (*i.e.*, the transition distributions): i) Kantorovich metric [32] and ii) total variation distance (TVD) [33]. Despite the Kantorovich-based metric is computationally more expensive than TVD, the former is better suited to provide useful information about the similarity between the value function of two states. That is, the Kantorovich-based bisimulation is a tighter bound of the difference between the optimal value of two states than TVD, and does not require the exact bisimulation partition to be computed.

In [34] three task similarity measures are proposed, one of them is based on the immediate reward of state-action pairs (d_R), while the other two are based on what the agent has learned, that is, policy overlap in states (d_P) and mean square error between Q-values (d_Q). These measures are evaluated by computing their correlation to the speed up a target task experiences when knowledge is transferred from a source task. Results show that when d_Q and d_P are completely learned in the target task, they are capable of approximating the learning speedup a learned source task will provide, however, both of them fail to provide a good approximation when the target task is partially learned. Despite the aforementioned drawback, d_Q and d_P showed to be useful for task clustering, as they generated cluster trees of subtasks that are meaningful, semantically speaking.

Similar to the work presented in [31], where the notion of similarity is built upon an equivalence class, in [35] the concept of MDP homomorphism [36] is extended to a probabilistic variation, called soft homomorphism. Roughly speaking, an MDP homomorphism from M to M' is a surjection $h^{M \mapsto M'} : S \to X$, where S and X are the state spaces of M and M', respectively. The homomorphism $h^{M \mapsto M'}$ preserves an algebraic structure with respect to the immediate reward of $s \in S$ and $h(s) \in X$, as well as with respect to their transition functions (see Eq. 4, where T and T' are the transition functions of M and M', respectively). Since these restrictions are quite hard to suffice in stochastic processes, in [35] a soft version of them is introduced, by defining the soft homomorphism as $f^{M \mapsto M'} : S \times X \to [0, 1]$. The main advantage

of using $f^{M \mapsto M'}$ over using $h^{M \mapsto M'}$ is that $f^{M \mapsto M'}$ can be learned (approximately) from data within certain bounds, even in online learning settings, thus dismissing the need for the transition model of the target task.

$$T'(h(s), a, x) = \sum_{s' : h(s') = x', s' \in S} T(s, a, s') \tag{4}$$

In [37] a family of metrics to measure state similarity in MDPs whose state space has an infinite cardinality is proposed. These metrics extend the notion of bisimulation to continuous state spaces, and provide a method to compute the difference between the value function of two states in an approximated way, with guarantees of bounds for the error introduced by the approximation. These metrics have the potential to serve as a criterion for state abstraction in continuous state spaces, as well as for transfer learning.

In [38] a data-driven MDP similarity measure is proposed, in which the difference of the transition models of two tasks is the criterion upon which similarity is assessed. A Restricted Boltzmann Machine (RBM) [39] is trained to estimate the distribution of a data set D_A of state transitions sampled from a task T_A. Then, the trained RBM reconstructs each transition in the data set D_B sampled from task T_B, where the average reconstruction error over D_B is the similarity between T_A and T_B (the smaller the error, the more similar T_A and T_B are). Experimental results show that the proposed measure successfully separates tasks with parameters significantly different, and groups those that are similar. Furthermore, the measure shows a positive correlation to the *jumpstart* [14] a learning agent exhibits in the target task after knowledge is transferred from the source task.

In [40] the bisimulation-based metric for states in an MDP (introduced in [31]) is employed as the basis for two MDP distance metrics, which are based on the Kantorovich and Hausdorff metric [41], respectively. The Kantorovich and Hausdorff metrics are used to composite the distance of every pair (s_i, s_j) into an MDP distance, where $s_i \in S_1$ and $s_j \in S_2$ are the state sets of MDPs M_1 and M_2, respectively. To compute the Kantorovich distance between MDPs M_1 and M_2, the sets S_1 and S_2 are treated as two uniform distributions. This work also introduces the conditions that

a pair of MDPs (*homogeneous MDPs*) must suffice so that the proposed metrics can be used to compute the distance between the MDPs. The metrics are evaluated with respect on two dimensions: i) their correlation to the knowledge transfer performance between a pair of MDPs, and ii) the transfer performance achieved when they are used to select a source task from a pool of tasks. Both metrics showed to consistently select source tasks that bring a positive transfer in the target task, however, only the Kantorovich-based metric showed a positive correlation to the transfer performance of the pairs of tasks.

Alternatively to previous works that define task similarity based on bisimulation or homomorphisms, in [42] the Jensen-Shannon Distance (JSD) is proposed to measure the similarity between state-action distributions of two MDPs, such that the summation of all the distances $JSD(\cdot, \cdot)$ represents the distance between two MDPs (*i.e.*, tasks). After evaluating in a Taxi-like domain [43], the JSD showed correlation to the transfer performance of several source tasks with the same transfer algorithm.

In [44] a measure for structural similarity of states and actions within the same MDP is introduced. The authors propose a graph representation G_M, for an MDP M, that contains nodes for each state and each valid state-action pair (see Fig. 5). Since in a graph G_M state nodes only have action nodes for neighbors (and vice versa), the authors define similarity between states as a function of the similarity of their neighbor action nodes, and action similarity between action nodes based on the similarity of their neighbor state nodes. To compute the state and action similarities, an iterative algorithm is proposed (since the similarities are defined recursively), that guarantees convergence to a unique solution. Also, the measures provide boundaries on the differences of state value functions and action value functions. Experimental results show that the structural measure is better than bisimulation at capturing state similarity. Despite that the structural measure is restricted to operate within a single MDP, contrary to previous works, it does not rely on the labels and is able to identify similarity between different actions solely based on their reward-transition structure.

Table 1 summarizes the task similarity measure (TSM) works covered in this section, where some of these works were revised to provide context, since they are not

Table 1: Summary of task similarity measure works. Columns show (left to right): the reference to the work, if the similarity can be measured between different state and action spaces, whether the state and action spaces can be discrete (D), continuous (C) or both (D/C), if the similarity can be measured between different MDPs, if the measure provides bounds for the difference in pairs of value functions, temporal complexity and the knowledge required from the MDPs (where *full* means a complete MDP is required). In the complexity column, N.P. means that the temporal complexity of the method was not provided, nor sufficient details to infer it.

Work	**Diff. S**	**Diff. A**	**S**	**A**	**Diff. MDP**	**Bound**	**Complexity**	**MDP K.**
[31]	no	no	D	D	no	yes	$O(\|A\|\|S\|^4 log\|S\|\frac{ln\delta}{lnC_T})$	*full*
[34]	yes	yes	D/C	D/C	yes	no	$O_P(\|S_1\|\|A_1\| + \|S_2\|\|A_2\| + \|S_1\|\|S_2\|)$	S,A
							$O_Q(\|S\|\|A\|)$	S,A
							$O_R(\|S\|\|A\|)$	S,A,R
[35]	yes	no	D	D	yes	yes	N.P.	S,A
[37]	no	no	D/C	D	no	yes	N.P.	*full*
[38]	no	no	D/C	D/C	yes	no	N.P.	S,A
[40]	yes	no	D	D	yes	no	$O(\|S_1 + S_2\|\|S_1\| \|S_2\|log(S_1 + S_2))$	*full*
[42]	no	no	D	D	yes	no	$O(\|S\|^2\|A\|)$	S,A,T
[44]	no	no	D	D	no	yes	$O(N\|S\|^2\|A\|\frac{K^2_{max}}{\epsilon^2})$	*full*

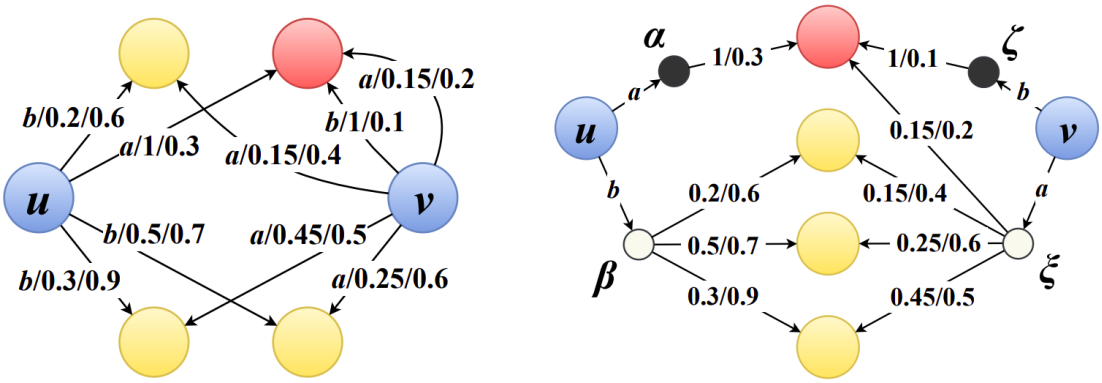


Figure 5: Example of an MDP (left) and its corresponding graph representation (right). Edge labels have the format "(action/)transition probability/reward". In this example, the graph representation exposes the similarity shared by actions b and a when they are performed in states u and v, respectively (figure taken from [44]).

applicable to the model-free learning setting (*e.g.*, [31, 37, 40, 44]). Considering that our research proposal is to measure inter-task similarity to transfer knowledge, we are interested in three dimensions regarding TSM works: i) the theoretical guarantees of the similarity measure, ii) the assumptions about characteristics tasks have in common and iii) their applicability to model-free settings. Only a subset of the reviewed works offers theoretical bounds for the difference between the value function of a pair of MDPs ([31, 35, 37, 44]). This feature is highly desirable, as it bounds how different the solutions for the MDPs might be. Therefore, they have the potential to serve as a heuristic for transferring pieces of a value function to a task that is being learned. However, only in [35] a TSM that does not need a complete description of the MDPs is presented.

On the other hand, in [34] the most flexible work is introduced since it is able to measure similarity between different MDPs, with different state and action sets, which may be discrete or continuous. Some of these works enable comparing MDPs with different state sets or action sets (*e.g.*, [34, 35, 40]), however, they require their states to come from the same representation and their sets of actions to hold a known one-to-one correspondence (which are requirements for what is defined to be *homogeneous MDPs* [40]).

Although there is an interest in being able to transfer knowledge across tasks that do not share state-action spaces, and whose possibly existing relations are unknown [8, 45], to the best of our knowledge there are no works focused in measuring inter-task similarity in this scenario. This is probably due to how difficult it might be to provide theoretical guarantees for tasks defined over different representations.

Therefore, in order to apply inter-task similarity concepts in tasks that do not share state-action spaces, it is necessary to develop methods that contextualize MDP components (*i.e.*, states, actions, transition and reward functions) so that tasks can be compared. Thus, a lifelong learning agent will be able to make informed decisions on how to transfer and consolidate knowledge across a sequence of tasks, in a way that is beneficial.

3.2 Multi-task Reinforcement Learning

In [46] the authors propose an MTL architecture that follows a teacher-student approach, based on the policy distillation technique [47]. In a setting where a collection of networks trained on specific tasks is assumed to be available, a multi-task network is trained by following the guidance of the expert networks. For each expert available, the loss function of the actor-mimic network (AMN) includes a loss term that penalizes the divergence between the policies of the AMN and the expert network. The AMN network is evaluated as an MTL agent and as a knowledge transfer method by initializing the parameters of a network. After evaluating the system in 20 Atari games from the Arcade Learning Environment [48], AMN showed a competitive final performance (in most of the games) in comparison to the expert network. Also, the weights of the AMN showed to be beneficial, as initial parameters, for learning a specific task. However, AMN had problems to learn on some games (such as seaquest which is known to be among the difficult ones), which might be a consequence of compressing the knowledge of multiple tasks into a single model (knowledge interference). Additionally, the authors provide convergence bound guarantees for the AMN learning agent with respect to the performance of the guiding expert, which is important since neural networks are known to be unstable learners, however, the potential negative transfer that an expert may induce is not assessed.

In [49] authors expand the actor-critic architecture to the MTL setting by proposing a network that contains an actor subnetwork for each task, and a critic subnetwork that is shared across all the actors. Since the authors assume that tasks share the state-action space, the architecture exploits this inter-task similarity to learn the critic subnet in conjunction for all tasks, so that knowledge is shared across actors. After evaluating the architecture in a mobile robot environment, where maneuvers are regarded as tasks, it showed to learn all the tasks almost as fast as the single-task networks. This shows the potential MTL networks have for knowledge compression when beforehand tasks are known to be similar. However, the network is unable to decide to which of the actors it should obey when the state is fed into its input layer, thus, requiring states to be task-labeled for it to operate.

In [50] an extension of the A3C architecture [18] to the MTL problem is proposed by running the thread works in several environments, rather than in several copies of the same environment (as in the original work). By evaluating in two pairs of games from the Arcade Learning Environments (ALE) [48], the authors test the potential of the A3C architecture in the MTL setting. Results show that A3C could be employed as MTL since it outperformed 3 out of 4 single-task learners, however, results on the last task suggest that A3C is prone to negative knowledge transfer and that additional mechanisms may be necessary to prevent this issue.

In [51] the MTL problem is addressed with a multi-model architecture called Distral, which models common and task-specific knowledge separately. Distral consists of a shared policy network π_0 and a collection of task-specific networks which are simultaneously trained as follows: the task-specific policies distill common behaviors into the shared policy and the shared policy guides the task-specific policies via regularization by adding the Kullback-Leibler (KL) divergence in their loss function. Furthermore, they also include a loss term that models the entropy in task-specific policies, which is employed to promote exploration. Experimental results suggest that the entropy loss term improves the final performance of the system, since the agent keeps exploring even if the optimal solution for one of the tasks has been found.

In [24] a teacher-student framework is proposed to train, via policy distillation [47], a student network when multiple task-specific expert networks are available. During the training phase of the student network, via the KL divergence between the student and a teacher policy, the student is encouraged to behave similar to the teacher. However, an important difference with respect to previous distillation-based works is that in [24] the KL divergence is dynamically weighted so that the influence a teacher has over the student can be adjusted while training. The authors employ Population Based Training (PBT) [52] to schedule the value for distillation weight throughout the training process. Results show that by using a larger distillation weight at the beginning of the training and decreasing towards the end, the student is able to exploit the knowledge from its teachers in early stages, and ignore them later on. In this way, the student is not restricted to replicate its teachers but it is capable of keep improving beyond their performance.

In [25] an MTL architecture is proposed (shown in Fig. 6). Constituted by a collection of n subnetworks to learn the parameters for m tasks, with an additional subnetwork (the attentive network) the goal is to learn to weight the outputs of the other subnetworks in a way that can be used to solve the m tasks. That is, instead of assigning a subnetwork for each task, during the training phase the attentive network learns to adjust the sub-network outputs to solve all m tasks. Thus, the attentive network decides which tasks are similar enough to share the parameters of a subnetwork and which require a specialized subnetwork. The attentive architecture was able to outperform both of its baseline competitors (Distral [51] and PNN [53]) while requiring fewer parameters with respect to the number of tasks.

In [54] authors address the challenge of simultaneously learning multiple tasks whose reward signals vary in sparsity and magnitude. That is, since the reward signal can be an arbitrarily large/small scalar, tasks with reward functions that differ in magnitude might hinder the learning of an MTL agent, due to the imbalance in the signals. Therefore, the authors propose a method to normalize the value function to deal with variance of rewards among a set of tasks. Experimental results show that their normalization step enables learning on a large set of tasks (57 games from the ALE [48]) and outperforms more simple normalization methods (*e.g.*, reward clipping).

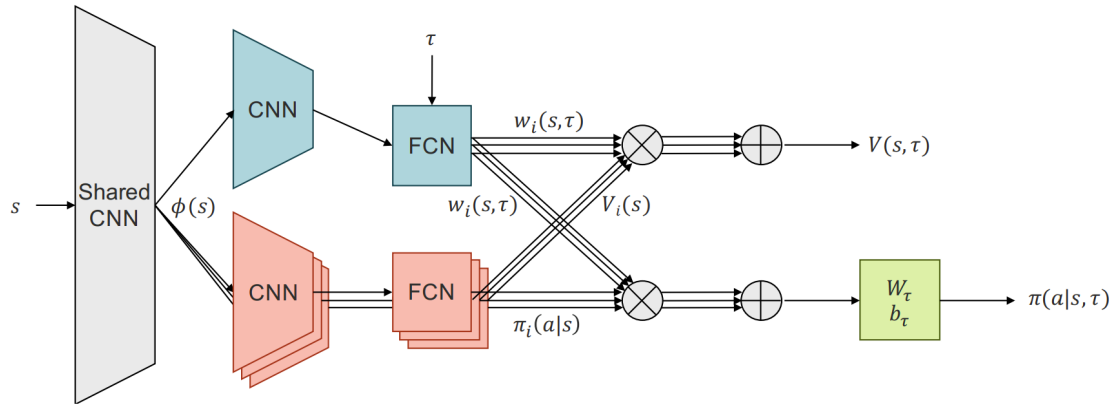


Figure 6: Architecture proposed by [25] (figure taken from [25]). The architecture is constituted by a shared (gray), the attentive network (blue), the subnetworks (red) and a collection of task-specific weight matrices and bias vectors (green).

Despite MTL and LML are two different problem settings, they share a fair amount of characteristics: having to i) train an agent to solve a set of tasks, ii) store knowledge from multiple tasks in way that does not harm the performance of the agent in any task and iii) exploit similarity (if it exists) across tasks to improve the overall performance. These common attributes present an opportunity to extrapolate ideas from MTL and adapt them to the lifelong learning setting.

Nevertheless, learning to solve multiple tasks with a single agent is still an open problem in both MTL and lifelong learning. Some of the works covered in this section [46, 50] showed to suffer from a low performance in certain tasks, probably due to negative knowledge transfer [23]. Conversely, other works (such as [25]) use a mechanism specifically to cope with negative knowledge transfer (*e.g.*, instantiating several subnetworks to learn multiple tasks (see Fig. 6) but do not provide a methodology to specify parameters that are critical for those mechanisms to work (*e.g.*, how many subnetworks are necessary to learn a particular set of tasks).

Therefore, by adapting an MTL system to the lifelong learning setting the learning agent must still face a set of open challenges: to decide which pieces of knowledge (parameters) can be reused among several tasks (*i.e.*, knowledge transfer) and how should the knowledge be stored, such that the agent is both memory efficient and effective in solving all the tasks it has seen (*i.e.*, scalability and catastrophic forgetting). In this context, an inter-task similarity has the potential to help solve these

challenges.

Similar to the work presented in [25], in which the attentive network learns during training to extract task-specific parameters from the set of shared subnetworks, a similarity measure could be used, interleaved with the training process, to evaluate which tasks are similar enough so that by sharing parameters they might improve their performance. On the other hand, identifying which tasks are too different and assigning them their own set of parameters would prevent harming the knowledge of the rest of the tasks.

3.3 Lifelong Reinforcement Learning

In [53] authors propose a framework for lifelong learning agents, called Progressive Neural Networks (PNN). Given a sequence of tasks $T_1, ..., T_{t-1}$ that PNN has learned to solve with the neural networks $N_1, ..., N_{t-1}$, respectively, and new task T_t, PNN instantiates a network N_t and connects the output of every hidden layer from each previous neural network (via lateral connections), then, it trains the parameters in N_t and the lateral connections by feeding the input to every network while freezing the parameters in the previous networks (as shown in Fig. 7). The PNN framework shows to effectively counter catastrophic forgetting, since it maintains a good performance on previously learned tasks, while it successfully transfers knowledge when it learns to solve the most recent task. However, since each network is connected to all previous models, the number of parameters in PNN grows quadratically with respect to the number of tasks.

In a similar fashion to PNNs, in [55] a transfer learning approach called PathNet that suits the lifelong learning setting is proposed. PathNet is a neural network whose layers are constituted by modules, which are also neural network themselves. During training, a genetic algorithm is used to evolution a population of genotypes, each representing a pathway that connects modules from different layers and determines where the gradient descent should be performed to update the weight and bias parameters. To transfer knowledge, after learning a pathway for task A, its parameters are frozen (like in a PNN) and used in the forwards passes while learning the pathway for task B. After evaluating in 18 different RL tasks (from Atari [48] and

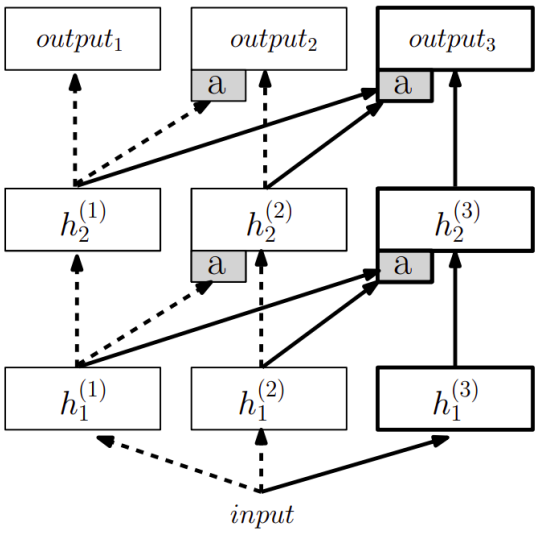


Figure 7: Example of a PNN architecture that is learning to solve a third task (figure taken from [53]). Each column represents a task-specific neural network, the dashed lines are connection parameters that remain fixed while the solid lines are parameters that will be optimized to solve the latest task. The gray boxes represent adapters that combine the outputs from layers of previous models.

Labyrinth [56] games), PathNet showed a consistent capability to transfer positive knowledge.

In [57] authors address the lifelong learning setting with a hierarchical architecture that uses previously learned skills in new tasks. Reusable knowledge is encapsulated in skills, that they call Deep Skill Networks (DSN), and train a policy that is capable of invoking both primitive and temporally extended actions (DSN). The architecture is evaluated on several tasks, from the Minecraft game, that require the agent to perform a sequence of subtasks (*e.g.*, going to a specific room, pickup an object, take it to another room, etc.) in a specific order. Results show that by invoking the DSNs, the hierarchical approach is able to obtain higher success rates than a non-hierarchical agent, such as the vanilla DQN [3].

In [58] authors focus in addressing the problem of catastrophic forgetting in neural networks that learn to solve a collection of tasks sequentially (like a lifelong learning agent). Contrary to the approach adopted in [53], instead of instantiating task-specific networks, the authors propose to stick to a single neural network and

slow down the modification of those wights that are more important. With their method, Elastic Weight Consolidation (EWC), the network learns to solve new tasks by optimizing the parameters that are less important (based on the Fisher information matrix, which is included as a loss term) for the previously learned tasks. After evaluating in supervised (MNIST [59]) and RL (ALE [48]) settings, by employing EWC the network showed to retain knowledge better than by using stochastic descent gradient and an L_2 regularization. Thus, EWC shows to effectively make the most out of a single neural network by retaining knowledge, while being scalable.

Similar to the work in [58], in [60] is proposed to counter catastrophic forgetting in a single neural network by replaying data from previous tasks based on a ranking function, thus, the network is periodically trained on previous experiences to avoid forgetting them. Besides the FIFO experience replay buffer neural networks usually incorporate to eliminate the correlation in data when RL settings are addressed, the authors propose an additional FIFO buffer that works as a long-term episodic memory. Four ranking functions (which sort the data in the additional buffer) were evaluated, from which two of them showed a capacity to effectively retain knowledge: coverage maximization (prefers data examples that have fewer data neighbors, *i.e.*, outliers) and distribution matching (prioritizes experiences randomly, by using a reservoir sampling strategy).

In [61] authors propose a lifelong learning framework that incorporates the concepts of EWC, policy distillation and lateral connections (like in PNN) to learn new tasks with one network (the active column), while retaining knowledge of previous tasks in a second one (the knowledge base), see Fig. 8. The Progress and Compress (P&C) process handles new tasks by following a two-step procedure: i) when a new task arrives, the weights in the active column are reset, the knowledge base is connected to it (via lateral connections) and trained to solve the task at hand, then, ii) the new knowledge is distilled into the knowledge base, using EWC so that previous knowledge is retained (see Fig. 8). In experimental results, P&C shows a capability to retain knowledge that is comparable to EWC, in addition to an ability to positively transfer knowledge when learning a new task that outperforms EWC.

In [62] is presented a lifelong learning model that extends on policy gradient methods for RL tasks, called Policy Gradient Efficient Lifelong Learning Algorithm (PG-ELLA). A factored representation $L \cdot s_i$ is proposed for the policy parameters of multiple tasks, where L is a matrix that encodes a latent basis for the set of tasks, and s_i is a task-specific vector. In this setting, the system does not know *a priori* the total number of tasks, nor has control over the order in which they are presented, however, it may encounter the same task in multiple occasions. Every time the i-th task T_i is encountered, L and s_i are updated with state-action-reward trajectories gathered from T_i. In case T_i has not been visited before, the system does not start from scratch since it has the knowledge from other tasks encoded in L, and which is employed to initialize the policy parameters in an informed way, rather than using random initial values. After evaluating in four domains, PG-ELLA successfully overcome the base PG learner, showing its capability to effectively employ knowledge across tasks.

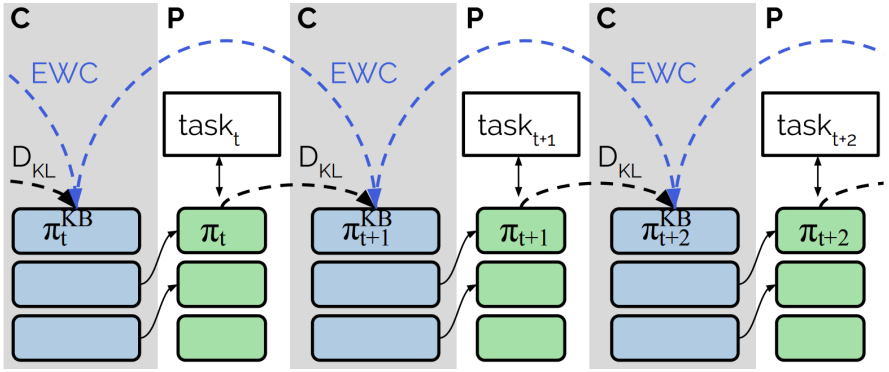


Figure 8: Progress and Compress learning process (figure taken from [61]). New tasks are learned by the active column network (green) and knowledge is transferred to it via lateral connections from the knowledge base (blue). Then, knowledge is distilled to the knowledge base through a Kullback-Leibler divergence term, while EWC helps to retain already stored knowledge.

In [63] authors introduce a teacher-student framework that can be seen as a general version of PNN [53]. In their proposal, rather than restricting all teacher layers to be connected to their respective student layer (like in PNN), any subset of teacher layers can be connected to any student layer through lateral connections. After evaluating in several games from ALE [48], and comparing to PNN [53] and PathNet [55], their architecture outperformed both baselines in most of the games.

In [27] the lifelong learning setting is addressed with a learning framework in which knowledge is represented in a multi-model approach, and extends the work in [62] to the cross-domain setting. That is, knowledge is shared across tasks through a matrix (L) of latent components, while task-specific vectors (s_i) are learned, such that $L \cdot s_i$ returns the parameters for the model that solves the i-th task (*e.g.*, the parameters of a policy network). Thus, knowledge is factorized in L and s_i. After evaluating their architecture in several continuous control tasks from the MuJoCo physics engine [64], it showed a capability to learn almost as fast as EWC in most tasks, while achieving the highest average performance across all tasks. Thus, this work showed promising results on retaining knowledge and learning tasks (like Progress and Compress), with the benefit that it is not restricted to neural networks, since the task-specific vectors s_i are model-agnostic.

In [65] a lifelong reinforcement learning algorithm is proposed based on the concept of task similarity. Considering that for a state-action pair (s, a), the optimal action value functions $Q^*_M(s, a)$ and $Q^*_{\overline{\mathrm{M}}}(s, a)$ (for MDPs M and $\overline{\mathrm{M}}$) are Lipschitz continuous in the MDP space, this work exploits this property to estimate, with a probably approximate correct (PAC) approach, provable upper bounds on the difference of the optimal Q-function of two MDPs. For every new MDP M, the algorithm estimates an upper on the difference between the Q-function of every MDP $\overline{\mathrm{M_i}}$ solved so far and M. As M is being explored, Q-values are transferred from the Q-function of the previously solved MDP with the lowest upper bound. The proposed algorithm outperforms another state-of-the-art transfer learning work [66] (that also computes upper bounds on tasks) by learning new tasks faster, as a consequence of estimating tighter bounds.

According to [61], a continual learning system (such as a lifelong learning agent) should ideally meet the following characteristics:

1. It should not forget how to solve previously learned tasks (catastrophic forgetting).
2. It should exploit knowledge from previously solved tasks to learn better/faster a new task (forward knowledge transfer).
3. It should be able to train on a large number of tasks (scalability).

Table 2: Comparative table of lifelong reinforcement learning works. The columns (from left to right) show the authors, model approach, main assumptions made by the work, whether the work addresses catastrophic forgetting (CF), scalability, and whether the work performs forward knowledge transfer (FKT) and backward knowledge transfer (BKT).

Work	Model approach	Assumptions	CF	Scalability	FKT	BKT
[53]	Multi	Same state space	✓	×	✓	×
[55]	Multi	Same state space	✓	×	✓	×
[57]	Single	Pre-training of reusable skills	✓	×	✓	×
[58]	Single	Same state-action space	✓	✓	✓	×
[60]	Single	Same state-action space	✓	✓	✓	×
[61]	Single	Same state-action space, agent can be revisit tasks	✓	✓	✓	✓
[62]	Single	Same state-action space, agent can be revisit tasks	✓	✓	✓	×
[63]	Multi	Same state-action space	✓	×	✓	×
[27]	Multi	State-action space may differ	✓	✓	✓	×
[65]	Multi	Same state-action space	✓	×	✓	×

4. It should strive to improve its performance on previously learned tasks after a learning new tasks that are similar (backward knowledge transfer).

5. It should be able to learn without task labels and clear task boundaries.

Despite the works reviewed in this section are aimed towards lifelong reinforcement learning (which are summarized in Table 2), they do not tackle the same set of challenges. None of them meets the fifth characteristic as tasks are assumed to be bounded, while all of them address catastrophic forgetting and perform forward knowledge transfer. A subset of these works is concerned with the scalability of their model ([27, 58, 60, 61, 62]), and only [61] performs backward knowledge transfer.

Considering that our research proposal attempts to address the problem of lifelong reinforcement learning with a similarity measure approach, [65] present the most similar approach to ours. On the other hand, [27] is the closest related work, since they do not make assumptions about tasks having the same state-action space, and meets the first three characteristics from the desiderata presented by [61] (which is the aim of our proposal). However, none of the works presented in this section

meets the three aspects of the desiderata list in heterogeneous tasks with a similarity measure approach.

3.4 Summary

In this section, we presented the main related works in the areas of TSM for MDPs, multi-task reinforcement learning and lifelong reinforcement learning. Currently, most of the work developed in these areas is focused in measuring the similarity and transferring knowledge between tasks that have a common state-action space. Alternatively, with our proposal we aim at relaxing this assumption in the lifelong reinforcement learning setting with an approach based on a TSM.

4 Research Proposal

In this section, a research proposal is presented to address the problem of lifelong reinforcement learning in heterogeneous tasks with a TSM-based approach.

4.1 Motivation

Reinforcement learning is a learning paradigm in which, contrary to supervised learning, the engineer/programmer does not need to know nor provide the correct answer on what the learning agent should do in certain situation. Conversely, the agent learns through trial-and-error, as it interacts with its environment. However, for non-trivial problems, learning in this way takes long periods of time to train. In applications such as robotics, long training periods are prohibitive because of the *curse of real-world samples* [6], which refers to the physical wear and tear suffered by robots as a consequence of interacting with the real world.

In certain robotics applications, such as service robotics, in which the robot is expected to face an indefinite long series of tasks, a sequential learning mechanism such as lifelong learning represents a natural approach to exploit previous experiences for the sake of learning faster in new tasks. However, considering that the inter-task diversity may manifest not only as differences in the dynamics (*i.e.*, transition and

reward functions) but also as differences in the representation (*i.e.*, state and action sets), it is critical to develop strategies to identify similar tasks and transfer knowledge between them despite the mismatch between their state-action spaces. Additionally, adaptability is essential for robots, as they may experience modifications to their hardware, both intentional (*e.g.*, changing a robotic arm [67]) and unintentional (*e.g.*, a leg failure in a walking robot [68]).

On the other hand, measuring inter-task similarity has been widely explored for transfer learning applications in reinforcement learning, as a heuristic to decide when will transferring knowledge between a pair of tasks improve the performance of the learning agent instead of harming it [40]. Therefore, employing an inter-task similarity measure in a lifelong reinforcement learning setting might lead to a robot with the ability to decide when and how it should transfer previous experiences to improve its performance in a new task, in spite of the overall inter-task diversity.

4.2 Justification

The following open problems reported in the literature, related to transfer learning and lifelong reinforcement learning, will be addressed in this research.

1. **Large sample complexity**: This problem refers to the large amount of data samples reinforcement learning agents usually require to learn to solve a task.
2. **Negative transfer**: This issue refers to the decrease in the performance of a learning agent in a target task, as a consequence of transferring knowledge from a source task.
3. **Catastrophic forgetting**: This problem refers to the situation in which a lifelong learning agent decreases its performance in old tasks as a consequence of learning to solve new tasks.
4. **Poor scalability**: This issue refers to how fast the number of parameters of a lifelong learning agent increases with respect to the number of tasks learned so far.

4.3 Problem Statement

To consecutively learn a finite sequence of heterogeneous RL tasks (of unknown length) with a lifelong learning approach is difficult for several reasons. The challenges a lifelong RL agent must face in this scenario are the following:

1. The agent must be able to compare tasks that do not share state-action spaces in order to identify in which of the previously solved tasks there may be pieces of reusable knowledge that can help to solve a new task faster/better.
2. Considering that knowledge of previous tasks must be retained to (possibly) reuse it in future tasks, the agent must consolidate knowledge in a way that is scalable with respect to the number of tasks seen so far.
3. Knowledge of previous tasks should not only be used to speed up the process of learning of new tasks, but also to solve the tasks from which it was learned. Thus, the agent should avoid forgetting how to solve previous tasks after new tasks are learned.

Formally: Let a task $T = \langle S, A, E, t \rangle$ be defined by a pair of (finite or infinite) sets S and A that contain the states and actions the agent can adopt and perform, a function $E : S \times A \rightarrow S \times \mathbb{R}$ that models the dynamics of the environment, and a threshold performance $t \in \mathbb{R}$ that indicates when a task has been learned. Also, let $(T_1, ..., T_i, ..., T_N)$ be a finite sequence of heterogeneous RL tasks (*i.e.*, if $i \neq j$ then $S_i \neq S_j$ and $A_i \neq A_j$), D_L be a lifelong RL agent, D_R be a regular RL agent, $K_L(\{T_1, ..., T_M\})$ and $K_R(T)$ be the size of the knowledge acquired by agents D_L and D_R after learning to solve the set of tasks $\{T_1, ..., T_M\}$ and task T, respectively, and $P_i(D_L, T_j)$ be the performance of agent D_L on task T_j after D_L learned to solve task T_i, where $j < i \leq N$. Hence, this research will address the problem of a lifelong RL agent D_L transferring knowledge between $\{T_1, ..., T_{i-1}\}$ and T_i, as well of consolidating the knowledge learned in each task, such that $\forall i \in [1, N]$:

- D_L learns task T_i at least as fast (requires less data queries of E_i to achieve a performance equal or higher than t_i) or as good (achieves a larger asymptotic performance) as D_R,
- $K_L(\{T_1, ..., T_i\}) \leq \sum_{j=1}^{i} K_R(T_j)$, and

- $\forall j \in [1, i-1]$, $P_i(D_L, T_j) \geq P_{i-1}(D_L, T_j)$.

4.4 Research Questions

The main questions that will guide this research are the following:

1. How can similarity between tasks that do not share state-action spaces be measured, in a way that helps determining if positive knowledge transfer can be performed between them?
2. What form of knowledge and how can it be transferred between tasks that do not share state-action spaces in a way that produces a positive knowledge transfer?
3. How can knowledge be consolidated in such way that helps prevent catastrophic forgetting and increases the scalability of a lifelong RL agent?

4.5 Hypotheses

Given a finite sequence of reinforcement learning tasks and a lifelong reinforcement learning agent, by measuring inter-task similarity between tasks that do not share state-action spaces it is possible to:

1. *Learn tasks faster than learning from scratch when previous related tasks have been learned,*
2. *Maintain a model smaller than having multiple single-task models, and*
3. *Consolidate knowledge and transfer knowledge between related tasks in a way that the agent does not forget to solve previous tasks.*

4.6 General Objective

To design and develop a transfer learning methodology, for a lifelong reinforcement learning agent, that is capable of performing positive knowledge transfer across tasks that do not share state-action spaces.

4.6.1 Specific Objectives

1. To design and develop an inter-task similarity measure that can be applied to heterogeneous RL tasks, such that selects source tasks that produce positive knowledge transfer and sets of parameters that can be shared across tasks without harming performance in those tasks.

2. To design and develop a transfer learning algorithm for heterogeneous RL tasks that learns *better* than a scratch learner: requires less data to achieve a threshold performance, or achieves a larger asymptotic performance.

3. To design and develop a knowledge consolidation algorithm for heterogeneous RL tasks, such that knowledge acquired from multiple tasks is stored in a *compact* way (requires equal or less memory space than multiple single-task models) and the learner does not *forget* how to solve any of these tasks (the performance of the learner is not harmed on any of the tasks).

4. To integrate the similarity measure and algorithms from the previous objectives to develop a lifelong reinforcement learning algorithm for heterogeneous RL tasks, such that the agent: learns *better* than a scratch learner, maintains a *compact* model of the knowledge being stored, and does not *forget* how to solve previous tasks.

4.7 Scope and Limitations

- Each task is modeled as a simulated environment from which the agent can sample as many data pieces as it needs.
- This research is concerned with control and robotic tasks.
- Tasks with discrete and continuous state-action spaces are considered.
- The state of the environment is assumed to be fully observable.
- RL tasks whose observations are images are out of the scope of this research.
- The agent does not have knowledge nor control over the order in which the sequence of tasks is presented to it.

4.8 Expected Contributions

1. An inter-task similarity measure for RL tasks that do not share their state-action spaces.
2. A transfer learning algorithm for RL tasks that do not share their state-action spaces.
3. A lifelong reinforcement learning algorithm for RL tasks that do not share their state-action spaces.

4.9 Methodology

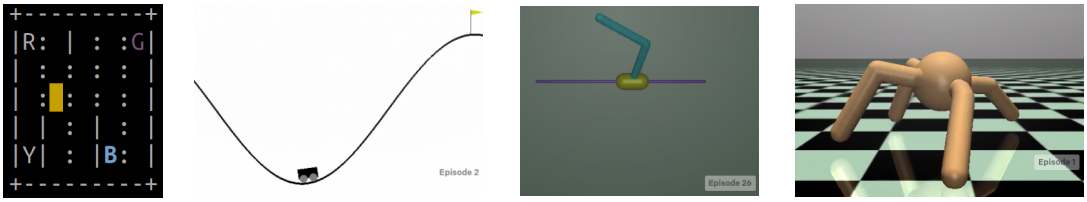
The methodology to follow along the present research project consists of five main stages, as described in detail below.

4.9.1 Analysis and selection of RL tasks

In order to validate the inter-task similarity measure and the algorithms that constitute the specific objects of this research (see Section 4.6.1), a collection of RL tasks is required. The task selection criteria are based on the scope of this research: control and robotic RL tasks whose state spaces are fully observable and whose state-action spaces may be discrete or continuous. Furthermore, considering that the motivation of our work is to enhance the robustness of decision-making agents in the face of task-diversity, the set of validation tasks should contain groups of tasks that are similar, and also tasks that significantly differ from each other. In this way, with a set of tasks that is rich in both diversity and similarity, we will be able to evaluate the proficiency of the lifelong RL agent to exploit similarity across tasks to improve the learning process, and identify the differences between them to avoid harming its performance.

Considering our requirements, in the Gym OpenAI library [69] one can find a wide variety of RL tasks that include discrete small-scale tasks from RL literature, classic control problems, control tasks involving 2D and 3D robots and classic Atari games (pixel-based tasks). For the evaluation of our algorithms, we are considering to use

discrete small tasks, classic control tasks and 2D/3D robot control tasks. Figure 9 shows an example of a task from each one of these categories.



(a) (b) (c) (d)

Figure 9: Examples of RL tasks that will be employed for evaluation purposes. From left to right: the taxi domain [43] (Fig. 9a), mountain car [70] (Fig. 9b), inverted double pendulum (Fig. 9c) and four-legged ant [71] (Fig. 9d).

4.9.2 Design and development of inter-task similarity measure

In this stage, the design and development of an inter-task similarity measure for RL tasks that do not share state-action spaces will take place. In order to develop the similarity measure, it is necessary to analyze how feasible it is to extend ideas in similarity measures that have been proposed in the revised literature for tasks that share the state-action space. The purpose of this analysis is to determine how the mismatch between the state-action spaces precludes the use of these works, and explore ways to evaluate similarity despite the differences in the state-action spaces. Since this research will address the model-free RL setting, the similarity measure developed in this stage must be able to compare tasks without having at its disposal the transition and reward function of the tasks, and should refine its estimation as new data is observed by the learning agent.

Considering that we need to compare RL tasks with different state-action spaces, it is not possible to measure similarity based on bisimulation [40] or MDP homomorphisms [35], since they rely on the assumption that the states that are being compared exist in the same space. Hence, a similarity measure based solely on behavioral properties (such as the transition and reward functions) fits better the restrictions imposed by our problem. Additionally, tasks with discrete and continuous spaces present particular challenges and opportunities.

Since RL tasks with small and discrete spaces can usually be solved with iterative methods with convergence guarantees (*e.g.*, Value iteration and Q-learning), one can expect reproducibility after training several agents in the same task under the same circumstances. This is convenient for evaluating initial ideas of the similarity measure, as iterations over the design and development process will take less time, in comparison to training agents with function approximation methods (*e.g.*, neural networks [2]). On the other hand, when tasks have spaces defined over continuous variables, the total ordering present in the values these variables can take provides additional information that does not exist in discrete tasks. For this reason, the design and development of the inter-task similarity measure will start considering discrete RL problems and subsequently in continuous domains.

Validation: the evaluation of the inter-task similarity measure will consist in computing the similarity between source-target pairs of tasks, transfer knowledge between them, and compute the difference between the performance of the agent in the target task with and without using knowledge from the source task. Two metrics will determine the success of the similarity measure:

1. The correlation between the similarity score of two tasks and the improvement of the agent in the target tasks when knowledge is transferred.
2. How many data is required by the similarity measure to provide a good estimate of how similar two tasks are.

That is, the greater the correlation is and the less data needed the better. In order to evaluate the correlation between the similarity measure with the improvement of performance in target tasks, a simple value-transfer approach will be employed as transfer learning technique (in both discrete and continuous space tasks).

4.9.3 Design and development of transfer learning algorithm

In this stage, the transfer learning algorithm for RL tasks that do not share state-action spaces will be designed and developed. Once the inter-task similarity measure from stage 2 has been developed, the transfer learning algorithm will be designed based on the characteristics we find relevant to establish similarity across different

tasks. The main focus of this stage will be to define a procedure to map knowledge between a pair of source and target tasks, such that the performance of an agent improves on the target task when knowledge is transferred from the source task, in comparison to learning without an additional source of knowledge. Besides evaluating ways to adapt knowledge across domains, we will explore at which rate should knowledge be transferred to exploit it as much as possible while learning the target task, *i.e.*, should knowledge be transferred as a single batch of data at some period of the learning process (perhaps at the beginning), or should it be gradually poured depending on the current stage of the learning process.

Since the focus of this stage is to explore ways in which knowledge can be transferred between tasks with different state-action spaces, similar to the second stage of the methodology with the use of a value-transfer approach (see Section 4.9.2), and considering that the third stage in the methodology is responsible for defining how knowledge should be represented, in this stage several forms of knowledge will be employed to evaluate the transfer learning algorithm. In [14], a classification of the types of knowledge that can be transferred between RL agents is presented, and it consists of two categories:

1. Low-level knowledge: Refers to knowledge that can be directly used by the learner in the target task. For example, $\langle s, a, r, s' \rangle$ instances, a policy, an action-value function (Q-function), a prior distribution or a full task model (transition and reward models).

2. High-level knowledge: The type of knowledge that cannot be employed by a learning algorithm as training data, but that can help guide the learning process in the target task. For instance, subsets of actions that should be prioritized in certain situations, partial policies or options [72], shaping rewards, important features, sub-task definitions or rules.

Validation: The evaluation of the transfer learning algorithm will consist in transferring knowledge between several pairs of tasks, in a single direction (*i.e.*, only form source to target task), and the success of the algorithm will be measured based on the five metrics described by [14] for transfer learning systems in reinforcement learning tasks: time to threshold, jump start, asymptotic performance, total reward

and transfer ratio. For comparison purposes, there are several works that can perform transfer learning between tasks that do not share state-action spaces, such as [8, 27, 45, 73]. Additionally, there are works that could be slightly modified to evaluate them over tasks with different state-action spaces such as [53, 61].

4.9.4 Design and development of knowledge consolidating algorithm

In this stage of the methodology, the knowledge consolidating algorithm will be designed and developed. Given that a lifelong RL agent must consecutively learn to solve a sequence of tasks, an algorithm for the consolidation of the knowledge learned in each task should employ a representation that is compact (since the agent does not know how long the sequence of tasks is), and adaptable so that a mismatch in the state-action space of a pair of tasks does not hinder sharing knowledge between them. In a similar fashion to stage 3, in this stage we will incorporate in this algorithm the features that we find relevant for the similarity between tasks. The main objective of the knowledge consolidating algorithm is to exploit the similarity across tasks to avoid redundancy in the knowledge that is shared, as well to identify pieces of knowledge that are significantly different, so that they are stored separately to prevent harming the overall performance of the agent as a consequence of the interference of knowledge.

According to [27], lifelong RL systems can be grouped in two broad classes based on how they share parameters (which store the knowledge learned in each task):

1. Single-model: This type of lifelong system assumes that there is one set of parameters (or model) that will be used to learn all tasks. For example, Elastic Weight Consolidation [58] (EWC) and Selective Experience Replay [74] (SER), in which a single neural network is submitted to solve a sequence of tasks.

2. Multi-model: This class of lifelong system assumes that there is a set of parameters that are shared across all tasks, as well that there is a set of task-specific parameters for each task. For instance, PG-ELLA [62], PNN [53] and Knowledge Flow [63].

In terms of scalability, single-model systems have the advantage over the multi-model approach due to their size remains constant throughout the entire sequence of tasks, whereas multi-model agents add certain number of parameters as new tasks are learned. On the other hand, in terms of overall performance, multi-model systems usually perform better than single-model agents, because with the sets of task-specific parameters the learning algorithm has more space to store separately pieces of knowledge that conflict with each other.

Regarding the knowledge consolidating algorithm from this stage, we consider that the inter-task similarity measure from stage 2 (see Section 4.9.2) could be used to develop an algorithm to consolidate knowledge in a multi-model approach. However, instead of adding a predefined number of task-specific parameters for each new task, the similarity measure could be employed to evaluate the similarity between the latest task and the previously learned ones. Thus, the agent could determine if it is necessary to instantiate new task-specific parameters (because storing it with any of the previous tasks would cause knowledge interference), or if the new task can be stored among the knowledge of some of the previous tasks (since they are similar). Hence, the lifelong agent would be able to make informed decisions about increasing its total amount of parameters, in a way that prevents harming its overall performance as a consequence of conflicting pieces of knowledge, and maintains an affordable scalability as the number of parameters will only increase if it is necessary.

Validation: To validate the knowledge consolidating algorithm, the inter-task similarity measure developed in stage 2 will be employed to define several sequences of tasks that will vary in length and the overall inter-task diversity. In each sequence of tasks, for each task a base RL agent (*e.g.*, [15] and [18] for discrete and continuous state-actions spaces, respectively) will be trained from scratch and the knowledge consolidating algorithm will store the new knowledge. Once the knowledge of every task in the sequence has been stored, an agent will be instantiated from the knowledge base and tested in each task. The success of the knowledge consolidating algorithm will be measured based on the overall performance achieved with the agent instantiated from the knowledge base (asymptotic performance and total reward), the size of the knowledge base, and the overall inter-task diversity of the sequence of tasks. Hence, it is desirable for the algorithm to achieve a high performance and a low knowledge base size, in relation to the length of the sequence of tasks and its

overall inter-task diversity. For comparison purposes, [27] is a lifelong RL system that supports tasks with different state-action space. Also, there are works that could be compared in a sequence of tasks that share state-action spaces, such as [53, 58, 60, 61, 75].

4.9.5 Design and development of lifelong learning algorithm

In this stage of the methodology, the lifelong reinforcement learning algorithm will be designed and developed. The design of this algorithm will consist in coupling the knowledge consolidating algorithm from stage 4 with the transfer learning algorithm from stage 3, and defining how the similarity measure will control the interactions between them. The objective is to analyze what needs to be modified in both algorithms, so they can work in a single system, as the transfer learning will depend on the representation employed by the knowledge consolidating algorithm, and on the knowledge it decides to maintain from each task.

In order to transfer knowledge, there are multiple approaches that can be adopted. To transfer knowledge forward (from old tasks to the latest task), ranking the old tasks based on their similarity to the most recent task might help to select a subset of useful source tasks. Alternatively, the system could employ the inter-task similarity measure from stage 2 (see Section 4.9.2) to evaluate which pieces of knowledge should be transferred from each previous task (rather than transferring tasks as a whole).

Validation: In order to validate the lifelong RL algorithm, similarly to the evaluation of the knowledge consolidating algorithm, several sequences of tasks will be defined so that they vary in length and in inter-task diversity (according to the inter-task similarity measure from stage 2). The success of the lifelong learning algorithm will be determined based on the following aspects:

1. The improvement of the agent in the target task, with respect to the five metrics described by [14], when knowledge is transferred from the tasks previously solved, in comparison to learning from scratch.
2. The total size of the agent (expressed in number of parameters or bits) after every task in a sequence has been learned.

3. Its capacity to not forget how to solve older tasks, despite the differences these might have with upcoming tasks.

For comparison purposes, we will contrast the algorithm to a single-task learner in each task (base learner), to [27] a lifelong RL system that works in sequences of tasks that do not share state-action space, and to the following lifelong RL works that assume tasks share the state-action space: [53, 58, 60, 61, 75].

4.10 Work Plan

In Fig. 10 is shown the Gantt chart for the activity schedule to carry out during this research project.

4.11 Publications Plan

The expected publications and their respective objectives are presented below.

1. Conference article.

   International Joint Conference on Artificial Intelligence. The objective is to publish an inter-task similarity measure that can be applied to tasks that do not share the state-actions space.

   Estimated submission date: October 2021.

2. Journal article.

   Machine Learning. The objective is to publish the results obtained from the integration of the transfer learning and knowledge consolidating algorithm.

   Impact factor: 2.672.

   Estimated submission date: May 2022.

3. Conference article.

   International Conference on Machine Learning. The objective is to publish a lifelong reinforcement learning algorithm that can be applied to tasks that do not share the state-actions space.

   Estimated submission date: January 2023.

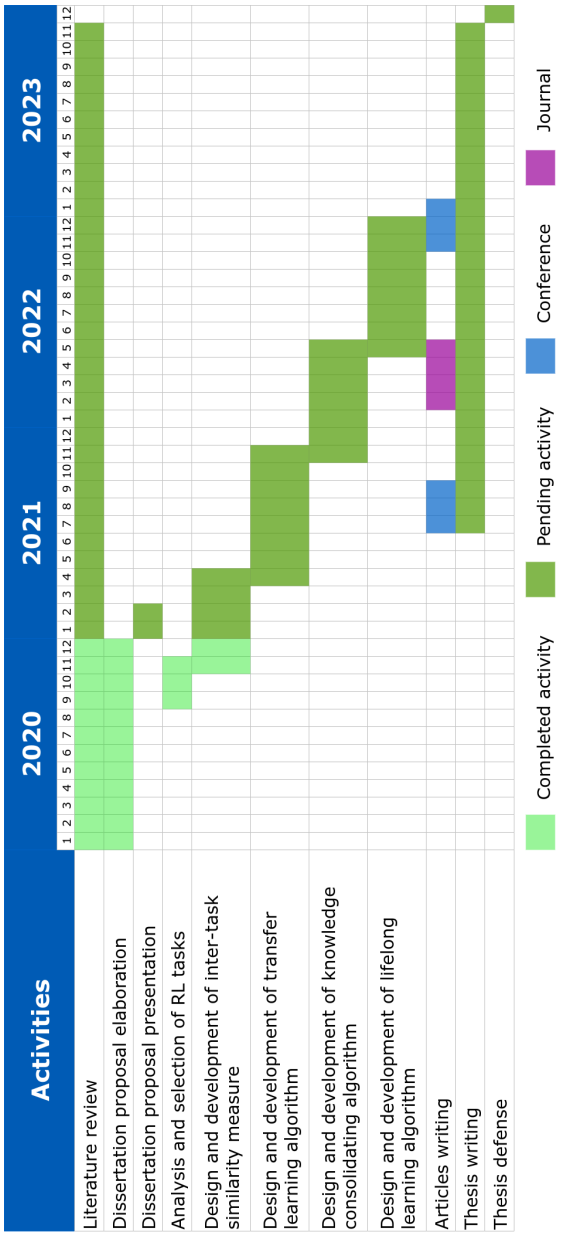


Figure 10: Activity schedule.

5 Preliminary Results

In this section, we present preliminary experimental results related to measuring similarity between tasks that differ in their state-action space.

5.1 Environments

In order to evaluate the inter-task similarity measure, a set of three discrete RL environments with different state-action spaces are employed. These environments are described below (see Fig. 11).

- Taxi domain: Originally introduced by [43], this task consists of a 5×5 grid in which there are four locations (labeled with a letter). The taxi (flat learning agent) must pick up a passenger from one of the locations and drop it in another one. The agent has six actions (up, down, left, right, pickup, drop) and there are 500 possible states.
- Frozen Lake: The environment consists of a 4×4 grid in which the agent can move between adjacent tiles in four directions (*i.e.*, up, down, left, right). At the beginning of the episode, the agent starts in the initial location (labeled with the letter S) and its objective is to reach a goal location (labeled with the letter G) without falling into the hole locations (labeled with the letter H).
- Frozen Lake 8×8: This environment very similar *Frozen Lake*, the only difference lies in the size of the grid, which is 8×8.

Figure 11: RL environments employed to evaluate inter-task similarity measures. From left to right: taxi domain, frozen lake and frozen lake 8×8.

5.2 Measuring inter-task similarity

The Q-table of an optimal policy can be seen as a map of where the agent would prefer to be within the state-action space. In an optimal Q-table, the state-action

pairs that take the system to a goal state will be among the highest q-value pairs. However, in tasks where the agent must travel through long sequences of state transitions, sub-goal states are likely to be present, as well as undesirable states that should be avoided.

Despite tasks having different state-action spaces, it is possible for them to have a similar structure in terms of undesirable, sub-goal, and goals states. Thus, if an LML agent could identify that the task it is currently exploring has a similar structure to one of the tasks it already solved, then it might be able to transfer some of the previous knowledge to the current task, such that the exploration phase could be reduced and, hopefully, learn faster. Hence, in this section an inter-task similarity measure based on the concept of sub-goals structure for heterogeneous tasks is presented. The goal of the will be to identify pairs of state-action pairs that perform similar roles (*e.g.*, as undesirable, sub-goal, or goal states) in their respective task, based on their q-value.

To measure the similarity between two tasks, the method presented in this section takes as input the Q-functions of two tasks, in the form of matrices Q^1 and Q^2 with dimensions $N_1 \times M_1$ and $N_2 \times M_2$, respectively. In matrices Q_1 and Q_2, rows correspond to states whereas columns correspond to actions. Thus, the following steps describe how to measure the similarity between Q^1 and Q^2.

First, the following steps are performed to $Q \in \{Q^1, Q^2\}$, where $N \times M$ are the dimensions of matrix Q, and K is the number of clusters to be generated.

1. Computation of row-cluster matrix $C^{row} \in \mathbb{N}^{N \times M}$. Let $Q_{i,*} = [Q_{i,0}, ..., Q_{i,M-1}]$ be the i-th row of matrix Q. K-means is applied to the elements of $Q_{i,*}$ to generate K clusters. The i-th row of matrix C^{row}, *i.e.*, $C^{row}_{i,*} = [C^{row}_{i,0}, ..., C^{row}_{i,M-1}]$, consists of the cluster labels assigned to each element of $Q_{i,*}$. Thus, $C^{row}_{i,j}$ holds the label assigned to $Q_{i,j}$ after generating K clusters with the elements of $Q_{i,*}$.

2. Computation of column-cluster matrix $C^{col} \in \mathbb{N}^{N \times M}$. Let $Q_{*,j} = [Q_{0,j}, ..., Q_{N-1,j}]^T$ be the j-th column of matrix Q. K-means is applied to the elements of $Q_{*,j}$ to generate K clusters. The j-th column of matrix C^{col}, *i.e.*, $C^{col}_{*,j} = [C^{col}_{0,j}, ..., C^{col}_{N-1,j}]^T$, consists of the cluster labels assigned to each element of

$Q_{*,j}$. Thus, $C^{col}_{i,j}$ holds the label assigned to $Q_{i,j}$ after generating K clusters with the elements of $Q_{*,j}$.

3. Computation of actions-frequency matrix $F^A \in \mathbb{R}^{M \times K}$. The j-th row of F^A, *i.e.*, $F^A_{j,*} = [F^A_{j,0}, ..., F^A_{j,K-1}]$, represents a normalized histogram of the label frequencies in the j-th column of matrix C^{row}. Thus, $F^A_{j,k}$ holds the proportion of times label k appears in $C^{row}_{*,j} = [C^{row}_{0,j}, ..., C^{row}_{N-1,j}]^T$.

4. Computation of states-frequency matrix $F^S \in \mathbb{R}^{N \times K}$. The i-th row of F^S, *i.e.*, $F^S_{i,*} = [F^S_{i,0}, ..., F^S_{i,K-1}]$, represents a normalized histogram of the label frequencies in the i-th row of matrix C^{col}. Thus, $F^S_{i,k}$ holds the proportion of times label k appears in $C^{col}_{i,*} = [C^{col}_{i,0}, ..., C^{col}_{i,M-1}]$.

After the frequency matrices F^{S_1}, F^{A_1} and F^{S_2}, F^{A_2} have been computed for Q_1 and Q_2, respectively, the intersection matrices I^A and I^S are computed.

- Computation of actions-intersection matrix $I^A \in \mathbb{R}^{M_1 \times M_2}$. The element $I^A_{i,j}$ holds the intersection value between the i-th row of matrix F^{A_1} (*i.e.*, $F^{A_1}_{i,*} = [F^{A_1}_{i,0}, ..., F^{A_1}_{i,K-1}]$) and the j-th row of matrix F^{A_2} (*i.e.*, $F^{A_2}_{j,*} = [F^{A_2}_{j,0}, ..., F^{A_2}_{j,K-1}]$). That is, $I^A_{i,j} = Intersection(F^{A_1}_{i,*}, F^{A_2}_{j,*})$ (see Eq. 5).

- Computation of states-intersection matrix $I^S \in \mathbb{R}^{N_1 \times N_2}$. The element $I^S_{i,j}$ holds the intersection value between the i-th row of matrix F^{S_1} (*i.e.*, $F^{S_1}_{i,*} = [F^{S_1}_{i,0}, ..., F^{S_1}_{i,K-1}]$) and the j-th row of matrix F^{S_2} (*i.e.*, $F^{S_2}_{j,*} = [F^{S_2}_{j,0}, ..., F^{S_2}_{j,K-1}]$). That is, $I^S_{i,j} = Intersection(F^{S_1}_{i,*}, F^{S_2}_{j,*})$ (see Eq. 5).

$$Intersection(u, v) = \sum_i min(u(i), v(i)) \tag{5}$$

Considering that the clusters generated for the cluster matrices are sorted (*i.e.*, label 0 is assigned to elements that belong to the cluster with the centroid of lowest value, whereas label $K-1$ to those in the cluster with highest valued centroid), the i-th row of C^{row} (row cluster) can be seen as a grouping of actions based on how preferred they are from the i-th state. Analogously, the j-th column of C^{col} (column cluster) groups states based on how preferred the j-th action is from each of them.

As Fig.12 shows, the label frequency histograms are computed transversely with respect to the row and column clusters. The j-th histogram computed across the row clusters in C^{row} represents the distribution of labels the j-th action received in the row clusters. Similarly, the i-th histogram computed across the column clusters in C^{col} represents the distribution of labels the i-th state received in the row clusters. Thus, histograms computed across row clusters represent the overall preference of actions from all states, whereas histograms computed across column clusters show the overall preference of states from all actions.

Hence, when the intersection value between the rows of two different F^A matrices is computed, the similarity of a pair of actions from different MDPs is being assessed. A high intersection value means that the pair of actions are preferred by the states from their respective in a similar way. For instance, in matrix I^A from Fig. 12 action $a_1 \in A_1$ is the most similar to action $a_3 \in A_2$, since a histogram $[1, 0]$ is closer to $[0.7, 0.3]$ than to $[0.3, 0.7]$.

Once the intersection matrices have been computed, their Frobenius norm $\| I^S \|_F$, $\| I^A \|_F$ and their mean $Mean(I^S)$, $Mean(I^A)$ are computed to obtain four similarity scores between Q^1 and Q^2.

5.3 Transferring Knowledge

After the intersection matrices I^A and I^S have been computed, in order to transfer knowledge from a source task to a target task, we define a matrix $T \in \mathbb{R}^{N_2 \times M_2}$, which will temporally hold q-values selected from the Q matrix of the source task (*i.e.*, Q^1). $T(i, j)$ will hold the q-value $Q^1_{k,l}$, where $k = \text{argmax}_{k'} I^S_{k',i}$ and $l = \text{argmax}_{l'} I^A_{l',i}$. Finally, after the previous step has been performed for every pair $(i, j) \in N_2 \times M_2$, matrix T is used to initialize the Q-function of the target task (*i.e.*, Q^2).

In other words, considering that the values in I^S and I^A represent the intersection between states and actions from two different tasks, we consider the k-th state from the source task to be the most similar to the i-th state from the target task. Similarly, the action from the source task that maximizes the intersection with the j-th action from the target task is considered to be the most similar one. Therefore,

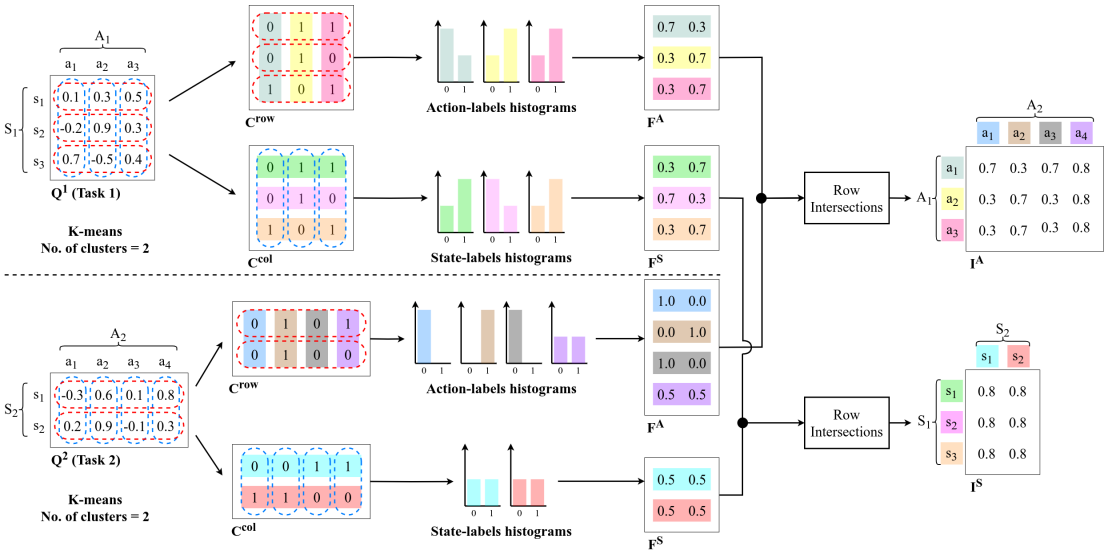


Figure 12: Computation of the intersection matrices I^S and I^A for a pair of Q-tables Q^1 and Q^2, whose rows and columns correspond to the states and actions in their respective task, using a number of two clusters. A row-cluster matrix (C^{row}) contains the labels assigned to the elements of the Q-table if they were clustered row-wise (red dotted ovals), similarly, a column-cluster matrix contains labels for elements clustered column-wise (blue dotted ovals). Then, each row in a states-frequency matrix (F_S) is a histogram for the label frequency of each row in C^{col}, whereas each row in an actions-frequency matrix counts the label frequency of each column in C^{row}. Finally, each element in the states intersection matrix (I^S) contains the intersection value between a pair states-labels histograms that belong to different state-frequency matrices. Analogously, the actions intersection matrix (I^A) is computed with a pair of actions-frequency matrices.

the q-value of the (k, l) state-action pair from the source task is transferred to the (i, j) state-action pair from the target task. For instance, considering the example in Fig. 12, to transfer a q-value for the pair $Q^2(s_2, a_3)$ in task 2 from the Q-table in task 1, we would assign it either $Q^1(s_1, a_1)$, $Q^1(s_2, a_1)$ or $Q^1(s_3, a_1)$, since a_1 (from task 1) has the highest intersection value with a_3 (from task 2), and every state from task 1 has the same intersection value with s_2 (from task 2).

5.4 Results

In order to evaluate the inter-task similarity measures presented in the section above, they were applied among a set of three tasks: Taxi, Frozen Lake, and Frozen Lake 8×8 (see Fig. 11). The similarity scores are summarized in Tables 3 and 4. The

similarity measures here presented describe how similar two tasks are based on the similarity between their states ($\| I^S \|_F$, $Mean(I^S)$) and the similarity between their actions ($\| I^A \|_F$, $Mean(I^A)$). Thus, the larger the similarity scores, the greater the similarity between the tasks.

The Q-table matrices employed to compute the similarity scores reported in Tables 3 and 4 were obtained after training an agent with Q-learning [15] during 300,000 episodes. The initial exploration probability and learning rate were set to 1, and were scheduled to reach the final values of 0.05 and 0.01 respectively, by decreasing them linearly with respect to the training episodes. Additionally, a fixed amount of 3 clusters were built to compute the inter-task similarity.

Table 3: Inter-task similarity Frobenius-norm scores among three tasks: Taxi domain (Taxi), Frozen Lake (FL) and Frozen Lake 8×8 (FL8). The larger a score is, the more similar a pair of tasks are to each other.

	Taxi		FL		FL8	
	$\| I^S \|_F$	$\| I^A \|_F$	$\| I^S \|_F$	$\| I^A \|_F$	$\| I^S \|_F$	$\| I^A \|_F$
Taxi	317.52	4.29	53.04	3.13	111.91	3.27
FL	-	-	9.86	3.19	19.29	3.07
FL8	-	-	-	-	43.41	3.43

Table 4: Inter-task similarity mean-based scores among three tasks: Taxi domain (Taxi), Frozen Lake (FL) and Frozen Lake 8×8 (FL8). The larger a score is, the more similar a pair of tasks are to each other.

	Taxi		FL		FL8	
	$Mean(I^S)$	$Mean(I^A)$	$Mean(I^S)$	$Mean(I^A)$	$Mean(I^S)$	$Mean(I^A)$
Taxi	0.5196	0.6222	0.4814	0.6215	0.4923	0.6468
FL	-	-	0.4961	0.7813	0.4634	0.7627
FL8	-	-	-	-	0.5125	0.8516

From Table 3 one can observe that the largest scores are located in the first row. According to the $\| I^S \|_F$ scores, Frozen Lake is more similar to Taxi than to itself. This is simply not true; however, these results show that computing the Frobenius norm to the intersection matrices may not be the best option. On the other hand, the highest $\| I^A \|_F$ scores are in the main diagonal, which means that each task is closer to itself than to any other task. Regarding the mean similarity scores, Table

4 shows that by averaging the state intersection values of two Q-tables, $Mean(I^S)$ is able to report a greater similarity for tasks with themselves than with any other task (such as $\| I^A \|_F$). On the other hand, $Mean(I^A)$ fails to do the same due to the scores from the first row.

Moreover, since the end goal of our proposal is to use the similarity measure as a heuristic to make better decisions about other problems, such as transferring and consolidating knowledge, the similarity scores were compared against the transfer ratios (see Table 5) obtained after transferring knowledge with the method described in Section 5.3. Similar to the agents that learned the Q-tables employed to compute their similarity, the agents that were trained from scratch and with knowledge transferred to them, were given 300,000 episodes to train. However, for the transfer learners, the initial exploration probability and learning rate were set to 0.25 and 0.5 (instead of 0.05 and 0.01). This is because, since the transfer learner is given an initial boost of knowledge, it should not ignore it (therefore a smaller exploration probability). On the other hand, the smaller learning rate is for preventing the agent to completely erase the initial knowledge with a few interactions with the environment.

Table 5: Transfer ratio scores among three tasks: Taxi domain (Taxi), Frozen Lake (FL) and Frozen Lake 8×8 (FL8). Rows correspond to the source task while columns correspond to the target task. Transfer ratio values above 1 indicate that the target task benefited from the knowledge that was transferred from the source task.

	Taxi	FL	FL8
Taxi	1	0.358	0.059
FL	0.994	1	1.276
FL8	0.992	1.167	1

Table 6: Mean square error (MSE) of the four similarity measures through the learning process in three tasks: Taxi domain (Taxi), Frozen Lake (FL) and Frozen Lake 8×8 (FL8). The MSE is computed between the similarity score for the final Q-table with itself and the similarity scores for the final Q-table with the Q-table that is being learned. Figure 14 illustrates the error of the four measures through the learning process.

	$\| I^S \|_F$ MSE	$\| I^A \|_F$ MSE	$Mean(I^S)$ MSE	$Mean(I^A)$ MSE
Taxi	161.8	0.0768	0.0014	0.0012
FL	0.1371	0.1139	0.0005	0.0073
FL8	32.9373	1.3871	0.0116	0.0942

From the set of evaluated tasks, a positive transfer of knowledge was exhibited only between the Frozen Lake and Frozen Lake 8×8 tasks (as shown in Table 5). This is understandable since they are two variants of the same problem that differ in the size of their state space. Alternatively, transferring knowledge from the Taxi domain task only harmed the performance of the other tasks (see Fig. 13). To assess the relation the similarity measures hold with transferring knowledge, the Pearson correlation coefficient was computed between the transfer ratios achieved by the pairs of tasks and their similarity scores. For the Frobenius-based scores $\| I^S \|_F$ and $\| I^A \|_F$, correlation values of -0.601 and -0.581 were obtained, whereas for the mean-based scores $Mean(I^S)$ and $Mean(I^A)$, the correlation values of -0.664 and 0.627 were observed. These correlation values show that, to some degree, the similarity measures are linearly related to the improvement in performance after transferring knowledge between tasks.

Additionally, in order to evaluate how much training data is required to compute a good estimate of the similarity between a task that is being learned and one that is already solved, the similarity measures were computed at different instants in the training process of the set of evaluated tasks. Figure 14 shows that the similarity measures $\| I^A \|_F$, $Mean(I^S)$ and $Mean(I^A)$ are able to stabilize at the final score significantly before the agent reaches its final performance, which leads to small MSE values (see Table 6). Regarding the lifelong learning setting, when the agent starts learning a new task, it is desirable to identify as early as possible which of the already solved tasks may help in the current training process. Thus, considering the correlation shown by the similarity measures here presented, and their convergence at an early stage in different training processes, we believe that employing an inter-task similarity measure in a lifelong reinforcement learning system is an approach worth exploring further.

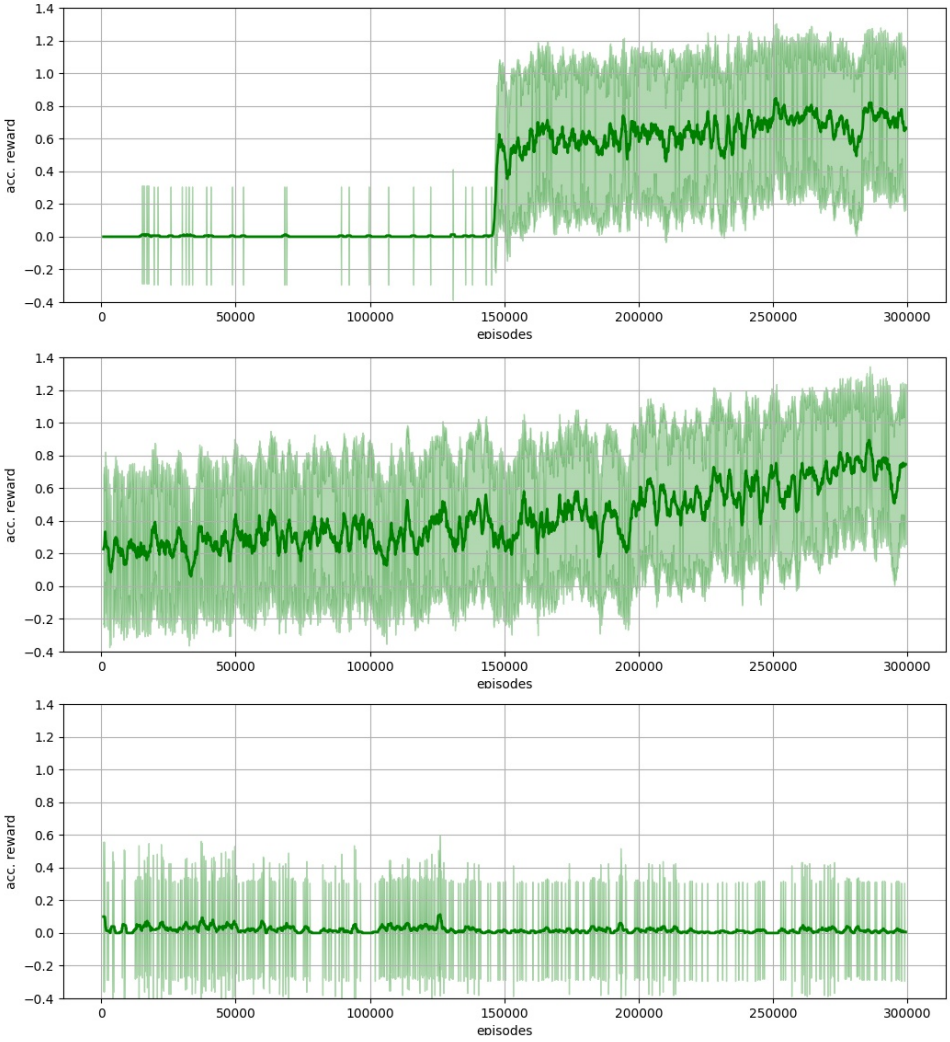


Figure 13: Average and standard deviation of accumulated evaluation reward of different agents learning in the Frozen Lake 8×8 environment. From top to the bottom: learning from scratch, learning after transferring from Frozen Lake, and learning after transferring from Taxi.

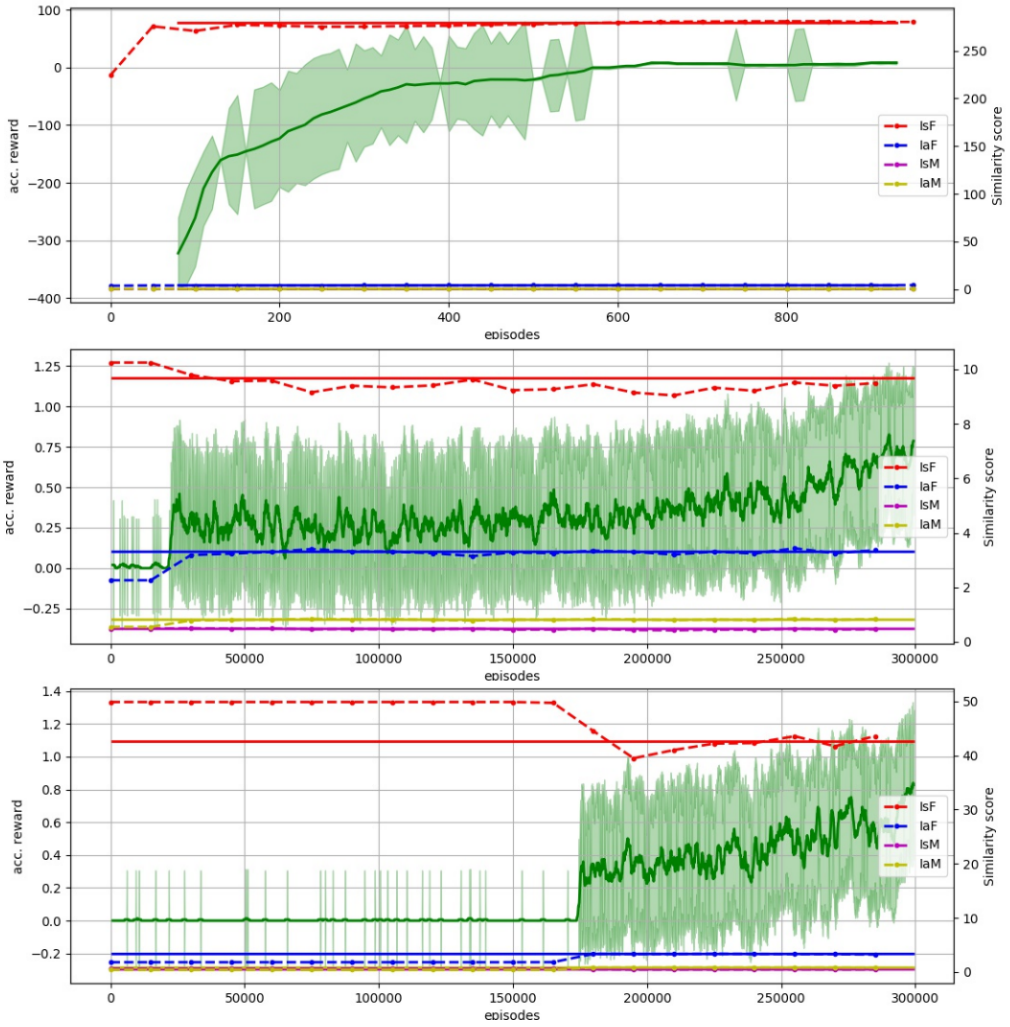


Figure 14: Similarity scores (vertical axis on the right) of the four similarity measures: $\| I^S \|_F$ (IsF), $\| I^A \|_F$ (IaF), $Mean(I^S)$ (IsM) and $Mean(I^A)$ (IaM). The dotted lines show the similarity score between the Q-table that is being learned and the final Q-table. The solid lines show the similarity score between the final Q-table with itself. The graphs show from top to the bottom: Taxi domain, Frozen Lake and Frozen Lake 8×8.

6 Final Remarks

Reinforcement learning is one the most powerful learning paradigms that there are, mainly because the learning agent is responsible for acquiring its own experience. In an effort to extend the capabilities of current RL systems, in this research proposal we presented a lifelong reinforcement learning approach based on the concept of task similarity. We believe that being able to compare tasks is a vital ingredient in any intelligent system. That is, as the system is able identify similarities and differences between different problems, it may use this information to decide if any of the knowledge accumulated so far can be reused. We have presented some preliminary results on measuring similarity between tasks with different state-action space. From the four measures presented, $\| I_A \|_F$ and $Mean(I_S)$ were able to correctly rank tasks to be the most similar one to themselves, which is necessary to effectively select source tasks in a lifelong learning setting. Additionally, three of the measures were able to provide an estimate with low error at early stages of three training processes. Based on the results obtained, we consider that measuring similarity between tasks with different state-action spaces is a suitable approach to transfer knowledge, and hopefully, it may also serve to consolidate knowledge. As future activities, according to the activity schedule, we will continue on developing an inter-task similarity measure that is useful to our purposes, by evaluating in control tasks with continuous spaces.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the Game of Go without Human Knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[7] M. E. Taylor and P. Stone, "Representation Transfer for Reinforcement Learning," in *AAAI Fall Symposium: Computational Approaches to Representation Change during Learning and Development*, pp. 78–85, 2007.

[8] M. E. Taylor, G. Kuhlmann, and P. Stone, "Autonomous Transfer for Reinforcement Learning," in *AAMAS (1)*, pp. 283–290, Citeseer, 2008.

[9] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artificial Intelligence*, vol. 247, pp. 10–44, 2017.

[10] K. Lin, S. Wang, and J. Zhou, "Collaborative Deep Reinforcement Learning," *arXiv preprint arXiv:1702.05796*, 2017.

[11] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[12] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[13] L. E. Sucar, "Probabilistic Graphical Models," *Advances in Computer Vision and Pattern Recognition. London: Springer London. doi*, vol. 10, pp. 978–1, 2015.

[14] M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domains: A Survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[15] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[16] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[17] Y. Li, "Deep Reinforcement Learning: An Overview," *ArXiv preprint arXiv:1701.07274*, 2017.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[19] A. Lazaric, "Transfer in Reinforcement Learning: A Framework and a Survey," in *Reinforcement Learning*, pp. 143–173, Springer, 2012.

[20] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[21] F. L. Da Silva and A. H. R. Costa, "A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems," *Journal of Artificial Intelligence Research*, vol. 64, pp. 645–703, 2019.

[22] Y. Zhang and Q. Yang, "A Survey on Multi-task Learning," *arXiv preprint arXiv:1707.08114*, 2017.

[23] N. Vithayathil Varghese and Q. H. Mahmoud, "A Survey of Multi-task Deep Reinforcement Learning," *Electronics*, vol. 9, no. 9, p. 1363, 2020.

[24] S. Schmitt, J. J. Hudson, A. Zidek, S. Osindero, C. Doersch, W. M. Czarnecki, J. Z. Leibo, H. Kuttler, A. Zisserman, K. Simonyan, *et al.*, "Kickstarting Deep Reinforcement Learning," *arXiv preprint arXiv:1803.03835*, 2018.

[25] T. Bräm, G. Brunner, O. Richter, and R. Wattenhofer, "Attentive Multi-Task Deep Reinforcement Learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 134–149, Springer, 2019.

[26] Z. Chen and B. Liu, "Lifelong Machine Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–207, 2018.

[27] J. A. Mendez and E. Eaton, "Lifelong Learning of Factored Policies via Policy Gradients," 2020.

[28] A. Tversky, "Features of Similarity," *Psychological review*, vol. 84, no. 4, p. 327, 1977.

[29] S. Ontañón, "An overview of distance and similarity functions for structured data," *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5309–5351, 2020.

[30] N. F. Ferns, *Metrics for Markov Decision Processes*. PhD thesis, McGill University, 2003.

[31] N. Ferns, P. Panangaden, and D. Precup, "Metrics for Finite Markov Decision Processes," in *UAI*, vol. 4, pp. 162–169, 2004.

[32] Y. Deng and W. Du, "The Kantorovich Metric in Computer Science: A Brief Survey," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 3, pp. 73–82, 2009.

[33] A. L. Gibbs and F. E. Su, "On Choosing and Bounding Probability Metrics," *International statistical review*, vol. 70, no. 3, pp. 419–435, 2002.

[34] J. L. Carroll and K. Seppi, "Task Similarity Measures for Transfer in Reinforcement Learning Task Libraries," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 803–808, IEEE, 2005.

[35] J. Sorg and S. Singh, "Transfer via Soft Homomorphisms," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 741–748, 2009.

[36] B. Ravindran and A. G. Barto, *An Algebraic Approach to Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts at Amherst, 2004.

[37] N. Ferns, P. Panangaden, and D. Precup, "Metrics for Markov Decision Processes with Infinite State Spaces," *arXiv preprint arXiv:1207.1386*, 2012.

[38] H. B. Ammar, E. Eaton, M. E. Taylor, D. C. Mocanu, K. Driessens, G. Weiss, and K. Tuyls, "An Automated Measure of MDP Similarity for Transfer in Reinforcement Learning," in *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[39] I. Sutskever, G. E. Hinton, and G. W. Taylor, "The Recurrent Temporal Restricted Boltzmann Machine," in *Advances in neural information processing systems*, pp. 1601–1608, 2009.

[40] J. Song, Y. Gao, H. Wang, and B. An, "Measuring the Distance Between Finite Markov Decision Processes," in *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, pp. 468–476, 2016.

[41] J. Henrikson, "Completeness and Total Boundedness of the Hausdorff Metric," *MIT Undergraduate Journal of Mathematics*, vol. 1, pp. 69–80, 1999.

[42] A. Narayan and T. Y. Leong, "Effects of Task Similarity on Policy Transfer with Selective Exploration in Reinforcement Learning," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2132–2134, International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[43] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[44] H. Wang, S. Dong, and L. Shao, "Measuring Structural Similarities in Finite MDPs," in *IJCAI*, pp. 3684–3690, 2019.

[45] M. Wan, T. Gangwani, and J. Peng, "Mutual information based knowledge transfer under state-action dimension mismatch," *arXiv preprint arXiv:2006.07041*, 2020.

[46] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning," *arXiv preprint arXiv:1511.06342*, 2015.

[47] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy Distillation," 2015.

[48] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[49] Z. Yang, K. E. Merrick, H. A. Abbass, and L. Jin, "Multi-Task Deep Reinforcement Learning for Continuous Action Control," in *IJCAI*, pp. 3301–3307, 2017.

[50] M. A. Birck, U. B. Correa, P. Ballester, V. O. Andersson, and R. M. Araujo, "Multi-Task Reinforcement Learning: An Hybrid A3C Domain Approach," 2017.

[51] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust Multitask Reinforcement Learning," in *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

[52] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, *et al.*, "Population Based Training of Neural Networks," *arXiv preprint arXiv:1711.09846*, 2017.

[53] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive Neural Networks," *arXiv preprint arXiv:1606.04671*, 2016.

[54] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, "Multi-Task Deep Reinforcement Learning with PopArt," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3796–3803, 2019.

[55] 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," *arXiv preprint arXiv:1701.08734*, 2017.

[56] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement Learning with Unsupervised Auxiliary Tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[57] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A Deep Hierarchical Approach to Lifelong Learning in Minecraft," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[58] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Over-

coming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[59] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[60] D. Isele and A. Cosgun, "Selective Experience Replay for Lifelong Learning," *arXiv preprint arXiv:1802.10269*, 2018.

[61] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & Compress: A Scalable Framework for Continual Learning," *arXiv preprint arXiv:1805.06370*, 2018.

[62] H. B. Ammar, E. Eaton, P. Ruvolo, and M. Taylor, "Online Multi-Task Learning for Policy Gradient Methods," in *International conference on machine learning*, pp. 1206–1214, 2014.

[63] I.-J. Liu, J. Peng, and A. G. Schwing, "Knowledge Flow: Improve Upon Your Teachers," *arXiv preprint arXiv:1904.05878*, 2019.

[64] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[65] E. Lecarpentier, D. Abel, K. Asadi, Y. Jinnai, E. Rachelson, and M. L. Littman, "Lipschitz Lifelong Reinforcement Learning," *arXiv preprint arXiv:2001.05411*, 2020.

[66] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman, "Policy and Value Transfer in Lifelong Reinforcement Learning," in *International Conference on Machine Learning*, pp. 20–29, 2018.

[67] B. Bocsi, L. Csató, and J. Peters, "Alignment-based Transfer Learning for Robot Models," in *The 2013 international joint conference on neural networks (IJCNN)*, pp. 1–7, IEEE, 2013.

[68] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning," *arXiv preprint arXiv:1803.11347*, 2018.

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

[70] A. W. Moore, *Efficient memory-based learning for robot control*. PhD thesis, University of Cambridge, 1991.

[71] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[72] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[73] H. B. Ammar, E. Eaton, J. M. Luna, and P. Ruvolo, "Autonomous Cross-Domain Knowledge Transfer in Lifelong Policy Gradient Reinforcement Learning," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[74] D. Isele, M. Rostami, and E. Eaton, "Using Task Features for Zero-Shot Knowledge Transfer in Lifelong Learning," in *IJCAI*, pp. 1620–1626, 2016.

[75] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv preprint arXiv:1511.05952*, 2015.