Reinforcement Learning:
A survey of *Transfer Learning*

Cañas Tarrasón, Eric

Master in Artificial Intelligence

14th June 2020

Contents

Abstract

Transfer Learning is a technique that has been gaining popularity along diverse areas of the *Machine Learning* field. It aims to generate models for solving a task, not from scratch but taking as baseline the prior *Knowledge* of models which are already able to solve a similar tasks. In *Reinforcement Learning* field, where the costs of exploration can be extremely high in some applications, *Transfer Learning* proposes a solution for fastening the learning process and even improve the final performance of the model. In this work, we review and classify the most relevant methods that have been gaining popularity in the field during last years.

1 Introduction

The International Encyclopedia of Education (1992) defines Transfer of Learning as: *"When learning in one context or with one set of materials impacts on performance in another context or with other related materials"*[1]. When this idea is applied on the *Machine Learning* field, we study how to design methods able to analyze the *knowledge* obtained in one or various *source tasks*, in order to set a good starting point for learning to solve a new related *target task*. For example, if we want to train a wolf classifier -using supervised learning- we can do it not only faster but with less samples re-training a dog detector model than training from scratch. Moreover, if the *Transfer Learning* method is able to detect correctly the similarities between both tasks, probably, we will end up with a better model. From another point of view, this ability of storing *knowledge* from different task, not only mimics the natural way in which thinking beings learn, but also helps to obtain a more general and abstract representations of the *knowledge*. General representations have been widely studied by the *Representation Learning* theory [2], being in short, one of the key-points for obtaining good *Machine Learning* algorithms.

In *Reinforcement Learning*, where usually the amount of samples which must be generated is extremely high, and where learning to solve single task can took an incredible amount of time, the ability of transferring *knowledge* between similar tasks acquire capital relevance. Especially when the cost of exploring the *state* space can be extremely high, as for instance in robotics[3].

2 The problem: *Reinforcement Learning* from Scratch

Maybe the most important key-point for understanding which is the greatest problem of *Reinforcement Learning* is the concept of *Sample Efficiency* [4]. In summary, *Sample Efficiency* measures how much an algorithm is able to squeeze the information within each sample, in order to rapidly improve its policy with each single experience that it takes from the environment. For exemplifying this concept we could compare three algorithms: First, is the most basic and primitive *Q-learning* algorithm [5], Second is a *Deep Q-Network* (*DQN*) model, that implements a *CNN* and experience replay[6] and third is a *Rainbow* model[7], in which we implement also *Hindsight Experience Replay*[8] in the replay buffer for dealing with sparse and binary rewards or even *ranking* methods[9] for *ranking* these experiences. If we benchmark these methods with the same problem (For example, the Arcade Learning Environment (ALE)[10]), it is clear that each one will be able to obtain better performance than previous, using orders of magnitude less samples. Nevertheless, none of these methods can, by itself, be compared with humans in terms of *Sample Efficiency*, since we need approximately five orders of magnitude less steps for solving similar tasks.

The biggest gap between human and *Reinforcement Learning Sample Efficiency* does not come then by the methods used, but from our *prior knowledge* [11] and even our innate abilities for understanding flexible environments [12]. Humans, do not start to learn from scratch, since from our childhood we have been learning how to see, predict trajectories and even solving tons of base tasks from which infer solutions for a lot of new ones. This ability of reusing the *prior knowledge* for speeding up the process of learning to solve new tasks, is where *Transfer Learning* focuses its efforts.

3 The solution: *Transfer Learning* in *Reinforcement Learning*

Transfer Learning is neither a specific topic of *Reinforcement Learning* nor a new field of research [13]. The concept of reusing the *knowledge* learned from a task A for solving a similar task B, has been widely applied in heterogeneous areas of the *Machine Learning* such as *Clustering*, *Classification* or *Regression*, and is by itself the main concept of the *Experiential Learning* theory[14]. However, in this document we will focus on providing a big picture of the different *Transfer Learning* approaches that have been developed for the *Reinforcement Learning* field, reviewing the most relevant methods that have contributed to build the current *State of the Art*.

3.1 General *Reinforcement Learning* Framework

In order to compare different methods in sections below from a shared point of view, it is important to define which is the common framework where *Reinforcement Learning* works.

3.1.1 Understanding tasks through *Markov Decision Processes*

For the purposes of *Reinforcement Learning* it is extremely usual to consider that tasks which with we have to deal, can be represented as *Markov Decision Processes*[16]. These *Markov Decision Processes* can be defined as follows:

- Time is discrete $(t, t+1, t+2)$.
- The *MDP* is defined by 4 parameters:
  1. **A set of *states*** S. Being an *state* the partial observation of the world that we experiment at time t.
  2. **A set of *actions*** A**.** Being A_s the subset of *actions* executable from the *state* s.
  3. **A set of probabilities** $P_a(s, s')$**.** Being $P_a(s, s')$ the probability of make a transition from the *state* s to the *state* s' when we execute the *action* a $(P(s_{t+1} = s' | s_t = s, a_t = a))$.
  4. **A set of expected *rewards*** $R_a(s, s')$**.** Being $R_a(s, s')$ the expected immediate *reward* that we obtain when passing from the *state* s to the *state* s', by executing the *action* a.
- The *Markov Property* is met[17]. So, as the process do not have any memory, S_{t+1} only depends on S_t.

Figure 2 shows a graphical representation of this defined *MDP*.

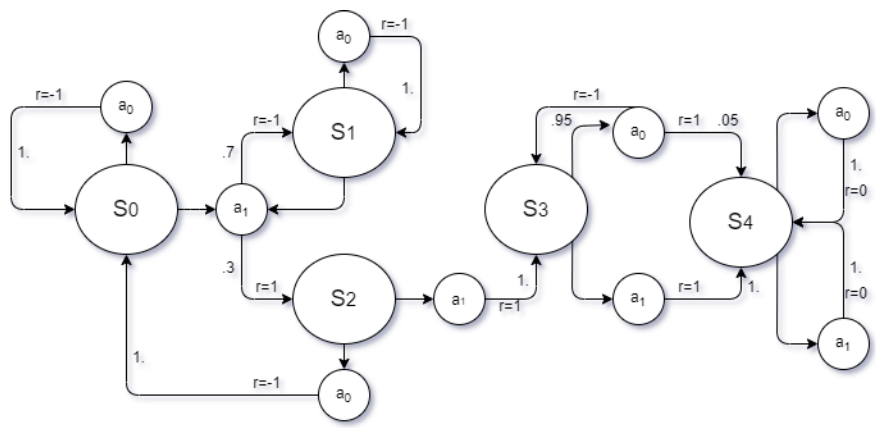

Figure 1: Example of a *MDP* with 5 *states* $(\{S_0, S_1, S_2, S_3, S_4\})$ and 2 possible *actions* $(\{a_0, a_1\})$. Each *action-state* pair (a, s) is associated to transition probability $(P_a(s, s'))$, and an immediate *expected reward* $(R_a(s, s'))$.

Using this definition, we can define tasks as the process of walking through this *MDP*, with a concrete purpose, which usually, in *Reinforcement Learning*, will be to maximize the total *reward* obtained at the end of the walk. Depending on how this walk is, we could classify two kind of tasks:

- **Episodic Tasks:** The agent starts into an *initial state* S_{start}, and take *actions* until it arrives to a *final state* S_{final} (The completion of an *episode*). The purpose of the agent will be to maximize the average *reward* obtained in each *episode*. For example, playing an *Atari 2600* game like *Pong*, would be an *episodic task*.
- **Non-Episodic Tasks:** There is not a defined *final State* S_{final}, so the agent will be walking through the *MDP* trying to maximize the total reward. In these tasks is common to find *negative rewards* for those transitions which leads to undesired *states*. For example, walk as much meters as possible (for a robot) would be a *non-episodic task*.

3.1.2 How *Reinforcement Learning* solve tasks. Finding good policies

In section above, the concept of *MDP* has been defined (figure 2) and it has been mentioned what is the purpose of the agent when dealing with each task: To maximize the average episode *reward*, or the total *expected reward* in non-episodic tasks. However, for understanding how *Transfer Learning* methods work, it is mandatory to define how *Reinforcement Learning* defines the solution of a task.

If we look at figure 2, it is clear that in almost each *state* are *actions* that are more desirable than others. For example: If the agent is located at the *state* s_0, it is clear that take *action* a_1 is better than taking a_0, since there is no way to obtain any *reward* higher than -1 if we still looping at s_0. Same occurs if we look at s_1, s_2 or s_3 (s_4 could define the final *state* S_{final} in episodic task where we start at S_0 , or an *state* where the agent is completely trapped in non-episodic tasks (the robot have fallen into a hole)). Then, if we would like to define an optimal *policy* π for this agent, it would look like: $\pi = [s_0 \rightarrow a_1, s_1 \rightarrow a_1, s_2 \rightarrow a_1, s_3 \rightarrow a_1, s_4 \rightarrow a_0]$. This *policy* represents the solution found by the agent, and is part of the most relevant *knowledge* that it owns. Although this intuitive definition can be easy to understand, lets define a formal framework around it, in order to being precise about how the problem is solved.

Lets define which is the *Value of a state* in a given *policy* π as:

$$V^{\pi}(s) = E(R|s, \pi) \tag{1}$$

Where $E(...)$ is the statistical *expectation* operator and R is the total *reward* obtained by following the the *policy* π when starting from the *state* s. Then, given this *Value* of a *policy* for each *state*, we could define that the optimal *Value* V^* for a given task is:

$$V^*(s) = \max_{\pi \in \Pi} |V^{\pi}(s)| \tag{2}$$

Where Π is the set of all possible *policies* that could be defined for the task at hand. Using this definition, then, since we assume that the *Markov Property* is met, searching the optimal *policy* $\pi \in \Pi$ can be seen as the optimization problem of finding the π which maximizes the *expected reward* at each *state* ($\sum_{s \in S} V^{\pi}(s)$).

With these definitions it is clear then, that a concrete *policy* (a solution) can be defined by its *Values* for each *state* ($\{V^{\pi}(s_1)$, $V^{\pi}(s_2)$, ... $V^{\pi}(s_n)\}$). However, it is important to denote that it is by itself a poor *Knowledge Representation* for dealing with *Transfer Learning* problems, since we loss the information about any sub-optimal *action* which is not in the *policy*. For this reason, it is extremely useful to keep a complementary *Knowledge Representation* which gives more information about the problem. This richer *Knowledge Representation* is the complete matrix which includes the *Values* that the agent can obtain when taking any *action* a from any *state* s (Supposing that it follows then the policy π): The table with the *Q-values*.

3.1.3 Representing richer solutions through the *Action-Value function* Q

The best way for finding which is the best *policy* π within a set of *policies* Π is to measure which is the *expected reward* of taking any possible *action* a in any *state* s. This is, unlike the single *Value* vector $\{V^{\pi}(s_1),\ V^{\pi}(s_2),\ ...\ V^{\pi}(s_n)\}$ a richer *Knowledge Representation* which would allow easier modifications, in the case that the current *MDP* which represent the given task would be modified to a similar *MDP'* representing a similar task. This function that gives the *Value* for a combination of a *state* and an *action* is known as the *Action-Value function Q*, and is defined as:

$$Q^{\pi}(s,a) = E(R|s,a,\pi) \tag{3}$$

This definition is identical to the *Value* function $V^{\pi}(s)$ (equation 2), but taking the *action* as new conditioning variable. So, it will still containing all the information from $V^{\pi}(S)$, since:

$$V^{\pi}(s) = \max_{a \in A_s} |Q^{\pi}(s,a)| \tag{4}$$

Then, at this point, given a *policy* π we can define all the *knowledge* of this solution as its *Q-Values*, which could be represented in the form of a table, but also through more complex functions like *Neural Networks* when the dimensions of the *state space* is prohibitive.

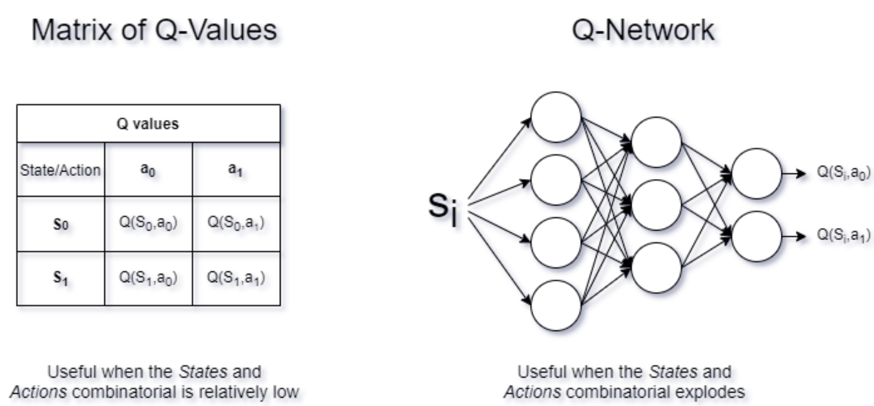

Figure 2: Representation of the *Action-Value function Q*. In cases where the *states* and *actions* combinatory is relatively low, the complete matrix of *Q-values* can be a precise *knowledge representation*. However, in cases where this combinatory explodes (For example, when the *state space* is an image or a large set of features), it is necessary to define more abstract representations of the *knowledge*, through for example *Deep Neural Networks* like *Deep Q-Networks*[6].

These richer *knowledge representations* will be the ones from which most of *Transfer Learning* algorithms that will be seen in future sections will take take profit.

3.2 Main classification of the *Transfer Learning* methods

Once we have defined a formal framework for how the *Reinforcement Learning* methods represents its *knowledge*, we have a common field for comparing how different *Transfer Learning* methods behave. Therefore, in this section we will present the main characteristics that differentiate them.

Transfer Learning methods could be classified in three different ways, according to which details of the methods we focus on[15]. This is due that every method takes its own *knowledge representation* (experiences, weights of a function, parameters...) and aims one or various objectives (to accelerate the process, to improve the performance...). Moreover, each one of them supposes a different relationship between source and target task/s. On this section we will analyze how *Transfer Learning* algorithms can be classified according to these three points of view.

3.2.1 Depending on the relation between source and target task/s

We could classify the *Transfer Learning* methods among three main classes, depending on which is the relation between the source and the target task/s.

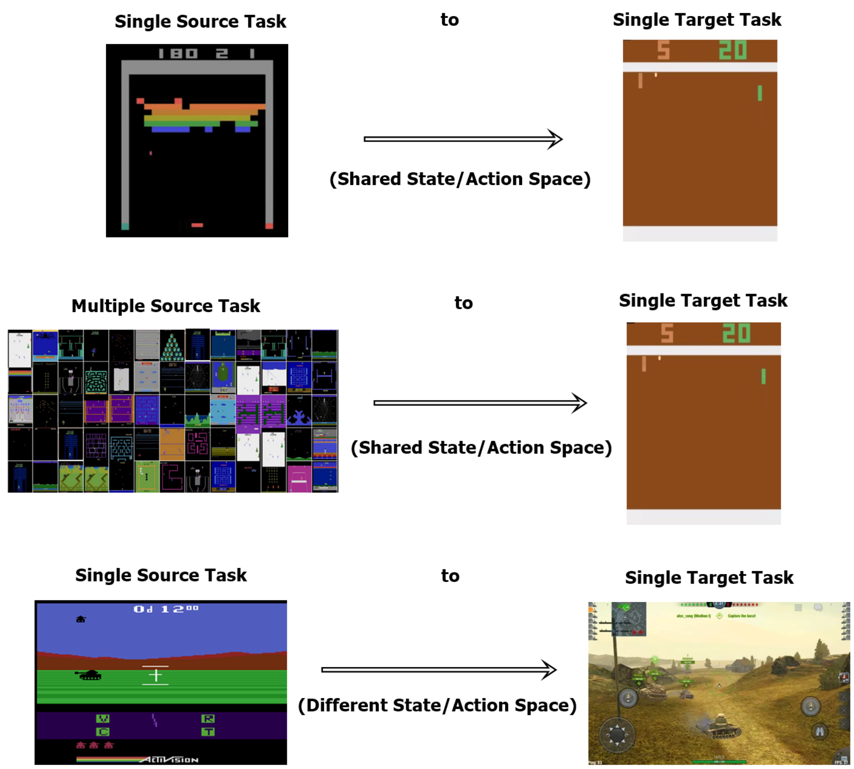

Figure 3: General Classification of the different *Transfer Learning* methods depending on which is the relation between the source and target task/s. Exemplified through the *Atari 2600 games* environment. Note that two *Atari 2600* games can be similar, but do not share in strict terms the same *state space*, since possible renderizable images will differ from one game to another. However, for the sake simplicity, we will assume that both spaces are shared (like most methods do). An example of an strictly shared state spaces can be seen in figure 4.

As figure 3 exemplifies, these three main classes -ordered by the complexity of the problem- are:

1. **Single source to single target with shared domains**: This is the easiest case, where the both tasks are not only similar but also share the *state* and *action spaces*. These methods could be applied, for example, for transferring the *prior knowledge* of an agent that is an expert *Breakout* player to a new one, that wants to learn how to play *Pong*. Since both tasks shares the objective of following and hitting a ball using a controlled bar, the amount of exploration which the new model will need for solving the task will decrease. These methods could be sub-classified among those which do a direct blind transference and those which focuses on extracting from the source model the abstract characteristics which should be relevant for solving the target task.

2. **Multiple sources to a single target with shared domains:** This problems adds up the additional problematic of merging the *knowledge* of a heterogeneous set of experts in specific tasks, that can have different similarities with the target task. If we imagine the previous example as transferring the *knowledge* from an expert in playing *Breakout*, we could imagine this one as merging and transferring the *knowledge* of a complete team of expert *Atari* gamers.

3. **Single source to single task, with non-shared domains:** This last case could be interpreted as the most complex, since these methods, despite of working with similar source and target tasks at conceptual level, works with task that do not share the same *state* and/or *action spaces*. These methods could be applied on, for example, transferring the *knowledge* from an expert *Robot Tank* player (A 2D *Atari 2600* game with 5 controls) to another agent which aims to learn how to play *World of Tanks* (A 3D *multi-platform* game with several controls). While both games have the same objective (defeating enemy tanks), the *action space* of both and even the representation of the world are totally different. Using these methods we could even imagine cross-domain transfers of *knowledge* (for example, from *Supervised* to *Reinforcement Learning*) or even transference between the virtual and the real world, extremely useful in robotics. Most of these approaches focuses on finding the *mapping functions* between the source and the target domain, since once this *mapping function* is found, this problem is analogue to the one seen on point 1.

This classification point of view is the one that can divide in a better way the different *Transfer Learning* methods, since beside the point of view which is presented in section below will completely condition the nature of the method.

3.2.2 Depending on the knowledge representation

If we observe how a *Transfer Learning* method represents the *knowledge* to be transferred, we could define three main categories [18]:

1. ***Knowledge* is represented by instances:** These techniques consider that those experiences that are useful for a source agent can be directly transferred to the target agent. These methods are extremely useful when the *state* space of the source tasks can be represented by a relatively small set of experiences. We could think those methods as the perfect scenario for when the *policy* is based on a *Lazy Learning* algorithm[19] like *K-Nearest Neighbor*[20]. The principal advantage of methods using this *knowledge representation* is that can be combined with a wide range of other *Transfer Learning* methods, since it does not interfere with model parameters.

2. **Transferring the abstract representation of the model:** In the current *State of the Art*, most of the methods are based on *Deep Reinforcement Learning*, including abstract representations of the *knowledge* through *Neural Networks*[21]. These methods aims to transfer this *abstract knowledge*, that is commonly represented by the weights of the *Neural Network* or equivalent model.

3. **Transferring the parameterization of the method:** In these cases the method is initialized with a set of convenient parameters which are able to solve a source task. These parameters could be simple like *learning rates* or γ update parameters but also more complex like the complete *Q-table* found for solving the source task.

3.2.3 Depending on the objective

There are several measures that can be taken into account for classifying the performance of a *Reinforcement Learning* algorithm. Depending on which performance measure the algorithm wants to improve we could be categorize them in three main groups [22]:

1. **Reducing the amount of experiences needed for solving the task:** One of the key-points which have had the *Transfer Learning* to gain popularity among a wide range of fields has been it capability for fastening the learning processes. From this point of view, the *Transfer Learning* methods are evaluated in how much they are able to reduce the amount of episodes and experiences needed for solving a target task (in comparison with an equivalent approach starting from scratch). This objective is the most required in those environments (especially from the real world) where the cost of exploration is extremely high in terms of time. This is also the objective which better improvements provokes in terms of *Sample Efficiency*.

2. **Improving the performance of first episodes:** These methods aims to use the optimal *policies* of similar task, for getting to the agents a good initial hypothesis for the new source task. The supposition that the optimal *policy* of a task could be a nice initial *policy* for another one, have proven to be true up to a determinable boundary (as will be seen in equation 9). For instance, we could think about the relation between two games of *Atari 2600*: *Breakout* and *Pong* (figure 3 (a)) -lets make the supposition that controls right-up and left-down and are not re-associated-. Even when both have really similar natures, the movement of *Breakout* is horizontal, while the movement of *Pong* is vertical. Setting the *Breakout* policy as starting policy for learning to play *Pong* could be an awful idea, since the actions where *Breakout* focuses (*Left* and *Right*) have no sense in *Pong*. However, if we transpose the *state* and *action spaces* of one of the games, then our agent will know from the very first episode how to follow and hit balls, improving tremendously its performance at this time.

3. **Improving the asymptotic performance of the task:** Each *Reinforcement Learning* method tends to have an associated *asymptotic performance*, that is the maximum performance that it could reach in a concrete task if would train during an infinite amount of time. This performance, is usually given by how well the method is able to represent the domain of the target problem. Methods with this objective takes profit of including in the learning process *knowledge* of different experiences, that could not be obtained by focusing only in the target task. In this way they are able to build a better representation of the problem space, and therefore, to improve the maximum performance that the algorithm can achieve in the desired task.

3.3 Review of most relevant *Transfer Learning* methods

In this section we will review which are the principal methods which have build the current *State of the Art*, according with where they are placed on the previous classification. It is important to denote that each method could aim one or more *objectives* (section 3.2.3) -for example, improving the *asymptotic performance* at same time that speed ups the learning process- and even transfer different *knowledge representations* at same time (section 3.2.2) -For instance, transferring the weights of the model as well as its parameterization-. For this reason, the current analysis will focus on the aspect that can distinguish each method more precisely, which is the relation between the source and target task/s (section 3.2.1).

3.3.1 Methods transferring from a single source to a single target task, that shares the same domain

These methods define a very concrete field where source and target task shares equivalent *MDP*s, that is, the same set of *states* S and set of *actions* A. But they differ on the *transition probabilities* $P_a(s, s')$ and the immediate expected *reward* of these transitions $R_a(s, s')$ (enumerate 3.1.1 on section 3.1.1).

In a case like shown in figure 4 we will assume that *MDP* and *MDP'* are representing really similar tasks, and therefore, we could make some assumptions like that the optimal *policy* π found for solving the first *MDP* should also be really similar to the optimal *policy* π' which would solve *MDP'*, or that they share sub-parts of the graph that could be solved in the exactly same way.

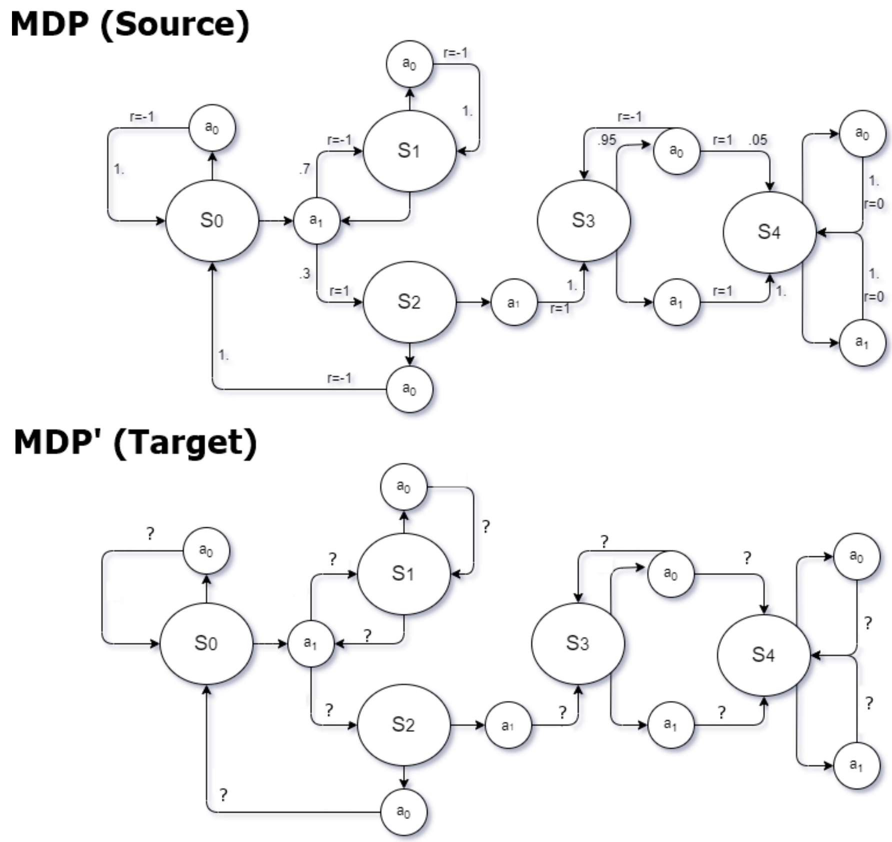

Figure 4: Representation of the relation between the *MDP* of the source task and the *MDP'* of a desired target task. This target task shares the State Space S and *action Space* A. However, the *transition probabilities* $P_a(s, s')$ and the *immediate rewards* $R_a(s, s')$ may be different and they should be explored.

So lets review which are the best current *Transfer Learning* methods for obtaining this new policy π' taking profit of the *knowledge* that we have about the source *MDP* and its solution.

3.3.1.1 Approaches based on *Hierarchical Reinforcement Learning.*

Hierarchical Reinforcement Learning [23], is the generalization of one of the most relevant frameworks proposed for including *temporal abstraction* in *Reinforcement Learning* [24]. This relevant framework in which is based, proposed a transition from *Markov Decision Processes* to *Semi-Markov Decision Processes*[25]. Extending the concept of *actions* to *options*, and transforming the concept of time to something not constant, but variable. It is variable since an *option*, is composed by a group of *actions* (that are also *options* by itself) and thus each one could need a different quantity of time steps for being executed. For example, the *option* "*Shoot to the basket*" could be composed by the temporal subset of actions: "*Take the ball*", "*Look to the basket*"... And those actions are, precisely, a highly valuated *abstract knowledge* from which *Transfer Learning* can take profit.

Hierarchical Reinforcement Learning, decompose the problem of *Reinforcement Learning* in a hierarchy of sub-tasks, where each sub-task can be either learned (usually by *Option Q-Learning* algorithms[25]) or provided (by different *Transfer Learning* methods). So, in other words, it define a sub-task as a set of simpler modules that could be reused by other sub-tasks, in different problems or even in different parts of the same general problem (see figure 5)

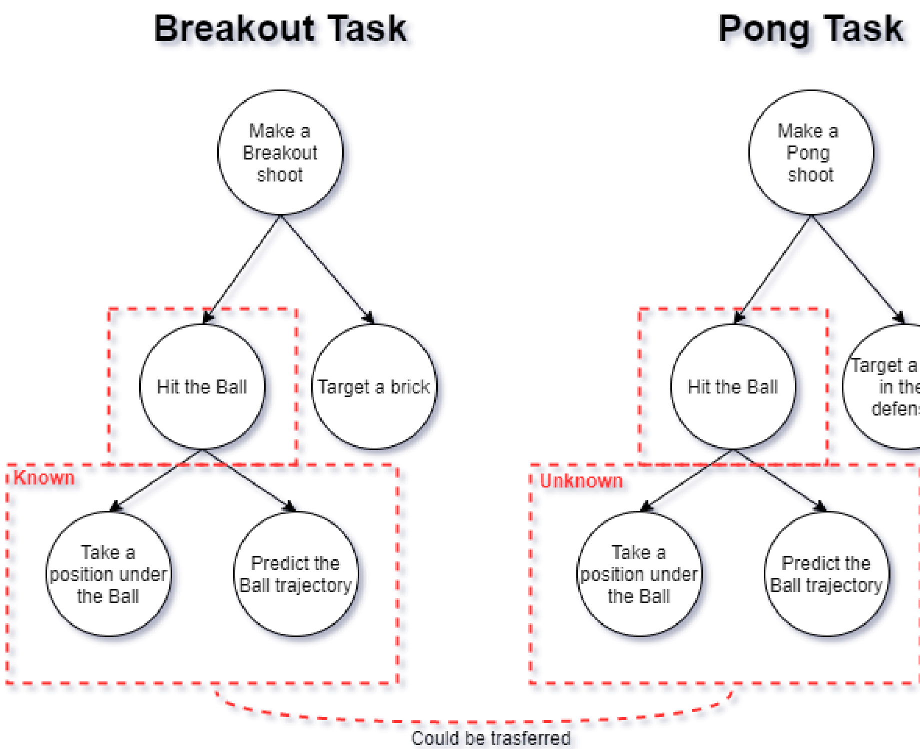

Figure 5: Example about how the *Breakout* and *Pong* tasks would be interpreted as a *Hierarchical Reinforcement Learning* problem. Appearing in the process shared sub-tasks that could be transferred and reused.

Most *Transfer Learning* methods in this area focuses on study the *SMDP*s for identifying which sub-tasks compose the source task and which of them are shared with the target task, in order to fastening the learning process by initializing target *options* with these shared sub-solution [26].

Transfer Learning methods based on *Hierarchical Reinforcement Learning* usually detect common sub-tasks between a single source task and a target task and transfer this sub-solution *knowledge* between them. However, it is important to remark that posterior works evolved these ideas, in order to generalize them for the *Multiple Sources to Single Target problem*[27].

For ending with this approach, it is important to analyze the principal *benefits* and *drawbacks* of these methods:

- **Benefits:** Defining tasks as trees of *hierarchical options* produce a kind of *knowledge* that can be easy understood and analyzed by humans. Moreover, these methods transform the solutions in something modular, making the process of *Transfer Learning* easier.
- **Drawbacks:** These methods transform the models in *Semi-MDP*s that, in spite of maintaining all the information of the correspondent *MDP*s makes more difficult the process of combining them with other kind of *Transfer Learning* methods. We will see in future sections how some methods could be combined with another ones if the *Knowledge Representations* that they use are compatible. In addition, for this method, we rely in the supposition that both tasks have sub-tasks in common, that can or can not be true.

3.3.1.2 Restrictions of the *action Space*

This is a different kind of *Transfer Learning* method, which focus on reducing the search space (and thus fastening the learning process) not by the side of the *states* but from the side of the *actions*[28]. This method proceeds as follow:

1. From the solved source task, a new set of random derived tasks is generated.
2. Using the source task solution, all derived tasks are also solved.
3. We reduce the initial set of *actions* A erasing those *actions* which did not appeared in any sub-task solution ($A \to A' \subseteq A$).
4. We use this reduced subset of *actions* A' instead the original set A for learning to solve the target task.

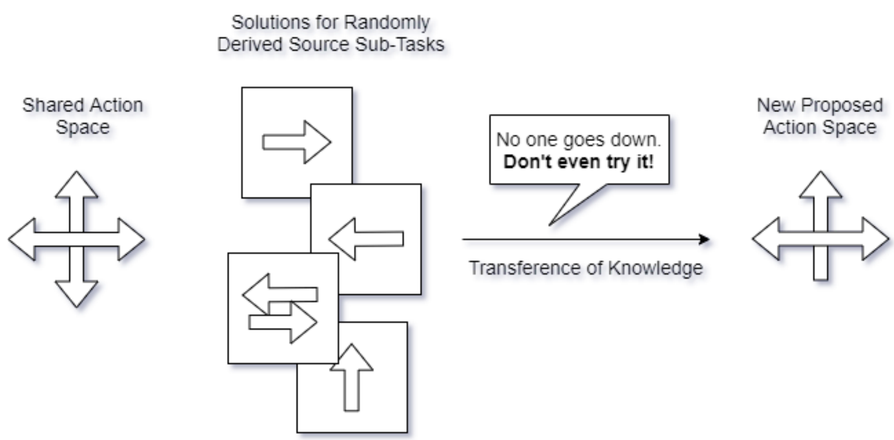

Figure 6: Exemplification of how the action space is reduced after study the solutions of the sub-tasks generated.

This method would be quite useful, for example, for accelerating the learning of an agent that must deal with a problem with an enormous *action space*, but also for reduced action spaces like in the *Atari 2600* games problems. If we would apply it over two similar lateral movement games, we could deduce from the source that it is not necessary to explore two of the directions and maybe even the fire button. However, it is important to denote that what we are really doing here is to generate an heuristic about which zones of the target action space have a lower probability of containing part of the optimal *policy* π' and therefore should not be explored.

- **Benefits:** Reduce the *action space* can accelerate the learning process. Moreover, since this method do not interfere with any explicit *knowledge representation*, it can be combined with a wide range of other *Transfer Learning* methods.
- **Drawbacks:** The optimality of the *policy* found is lost, since erased actions could be needed.

3.3.1.3 *Proto-Transfer Learning* approaches

Proto-Transfer Learning methods [29] are based on finding what they call *Proto-Value Functions* (*PVF*s). These *PVF*s include the most essential *knowledge* of the source *MDP* graph, since they are extracted by executing *random walks* through it. Therefore, the *Transfer Learning* method is composed by two main steps: The generation of the *PVF*s (which will represent the *knowledge* of the source task) in the source *MDP* and the adapted transference of this *knowledge* to the target *MDP'*. There are several methods using this idea [30], [31], however all of them are adaptations that evolves the idea of the following base algorithm[29]:

1. **Representation Learning:** Perform, on the source *MDP*, J *random walks* of N steps. And store in a dataset D_S each *state* s visited.
   (a) Generate from D_S a weighted undirected graph G. This graph is constructed by connecting to each vertex $s \in D_S$ its *K-Nearest Neighbors* (with an initial weight of 1). This graph will be a representation of the most relevant dynamics of the source task.
   (b) For each pair of nodes u and v within G compute the *normalized laplacian* $\mathcal{L}(u, v)$ as:
   $$\mathcal{L}(u,v) \begin{cases} u = v \; and \; d_v \neq 0 & 1 - \frac{1}{d_v} \\ u \; is \; adjacent \; to \; v & -\frac{1}{\sqrt{d_v d_u}} \\ otherwise & 0 \end{cases}$$
   (c) Compute the K smoothest eigenvectors of $\mathcal{L}$ and use each one of them as the columns of a function basis matrix Φ which will have as rows as *states* in D_s.
   (d) Once we have this matrix, we will have an analogue to the *Q State-Value Function*, but extracted from this simplified graph G. It will be used for defining the *Proto-Value function* (*PVF*) as: $\phi(s, a) = e_a \otimes \phi(s)$ (where e_a is the unit vector of the *action* a and $\phi(s)$ the row of Φ corresponding to the *state* s).
2. **Control Learning:** Perform the same N *random walks* than in step 1, but this time over the target *MDP'*, for generating a new Dataset D_t.
   (a) Set i as 0 and w^i as a random vector of size k.
   (b) **Repeat:**
      i. $i \leftarrow i + 1$
      ii. For each *state transition* in D_t (represented as $(s_t, a_t, s'_t, a'_t, r_t)$) compute the low rank approximations of the matrices $\bar{A}$ and $\bar{b}$ by:
      $$\bar{A}^{t+1} = \bar{A}^t + \phi(s_t, s_a)(\phi(s_t, s_a) - \gamma\phi(s'_t, s'_a))^T \tag{5}$$
      $$\bar{b}^{t+1} = \bar{b}^t + \phi(s_t, s_a) r_t \tag{6}$$
      For the cases where the target state was not present in the source dataset ($s_t \notin D_s$) we would approximate the function $\phi(s_t, a_t)$ by:
      $$\phi_i(x) = \frac{1}{1 - \lambda_i} \sum y \; x \frac{w(x, y)}{\sqrt{d(x)d(y)}} \phi_i(y) \tag{7}$$
      Note: Here is important to remark that using this approximation, method could be generalized for transferring *knowledge* between different domains.
      iii. Transfer the *Proto-Values Functions* (*PVF*s) of the source task to this new target domain, finding the new w^i by solving the system $\bar{A}w^i = \bar{b}$
   (c) **While** $||w^{i-1} - w^i||^2 > \epsilon$ (loop until convergence)
3. Return the approximation of the optimal *Value Function* for the target task as $Q^\pi = \sum_i w^i \Phi$

As seen, the *knowledge representation* that is transferred in this case comes from a generated Φ matrix, that could be understood as a set of features, since it is, in short, extracted from an stochastic process of random walking through the *MDP*.
It is important to analyze with special attention the principal benefits and drawbacks of this method:

- **Benefits:** Since it works through *feature extraction* and uses stochastic processes such as *random walks*, it can characterize better the principal dynamics within the *MDP*s. In the same way, since it makes the transference of learning through an iterative process of optimization, it is also the one that which can find more accurate similitude between the source and target task. In addition, by adapting the function $\phi(s, a)$ to an approximation (equation 7) we could even adapt the method for cases where the domain is not shared between the source and the target task.

- **Drawbacks:** This methods needs from *MDP*s that should be in some way easily "*random walkables*". In some fields like robotics this could be impossible, since the process of taking a concrete starting *state* S_s could be impossible without human interaction (which would have a prohibitive cost) or even the source *MDP* could have evolved over time. Additionally, the graph G which it uses to characterize the most relevant dynamics of the source task is composed by the dataset D_s of experiences taken from it, so the performance of the algorithm will depend on how representative these experiences are.

3.3.1.4 Direct *Policy Transfer*

Maybe, the easiest way of performing *Transfer Learning* is to make a direct transference of parameters, that is, to directly apply the *policy* of the source task π_s to the target task, using the source *Value Function* V^π as starting point for the learning process. If the *MDP*s of both tasks are similar enough (when they are finite, we can measure similarities between *MDP*s through different methods[32]), we could expect also a nice starting performance. In fact, it has been proven[33] that when the distance between two states is:

$$d(s) = \max_{a \in A}(|R_s(s,a) - R_t(s,a)| + \gamma \mathcal{T}(d)(T_s(\cdot|s,a), T_t(\cdot|s,a))) \tag{8}$$

Where $\mathcal{T}(d)$ is the *Wasserstein* distance [34] between the *transitions probabilities* $T_s(\cdot|s,a)$ and $T_t(\cdot|s,a)$, the maximum loss of performance that we can obtain for transferring π_s to the target MDP_t is:

$$||V_t^{\pi_s} - V_t^{\pi_t^*}|| \leq \frac{2}{1-\gamma} \max_{s \in S} d^*(s) + \frac{1+\gamma}{1-\gamma} ||V_s^{\pi_s} - V_s^{\pi_s^*}|| \tag{9}$$

So, it is important to denote that, when $\pi_s = \pi_s^*$ (we transfer the optimal *policy* of the source task) the second factor of the sum will be 0., so the maximum loss will be given only by the maximum *state* distance: $\max_{s \in S} d^*(s)$.

From this prove, can be deduced that when the distance between both *MDP*s is low, that is, when the tasks are really similar, the direct *policy* transference can be not only the easiest but also a really nice method for ensuring a good initial performance. In fact, this is usually the most used *Transfer Learning* method when *Deep Neural Networks* are implied[35], not only in *Deep Reinforcement Learning* but also in most of *Supervised Learning* fields.

- **Benefits:** The direct transference of *policies* is the easiest and most natural way of performing a transference of *knowledge*, since we can expect that the solutions that worked well for a source task will also work well for a similar target task. In fact we dispose of a measure (equation 9) about the minimal starting performance that can we expect. Moreover, in cases where *Neural Networks* are implied, these processes of *re-usage* and *re-training* are widely studied and can be especially exploited for obtaining better properties. For example, we could freeze those layers which focuses on feature extraction (like convolutionals) and using them as embedding, for extracting features of the target task in the language of the source task.

- **Drawbacks:** Maintaining only the source *policy* and losing the rest of information (for example, the *action-Value Function* Q), could lead to re-explore unnecessary transitions that were already well known by the source task.

3.3.2 Methods to transfer from multiple sources to a single target with same domain

These methods try to maximize the efficiency of the *knowledge transference* when we have not only a single source task, but a set of them, sharing the same *action* and *state spaces* (figure 3 (b)). Therefore, these methods have to deal with two main problems:

- **Merging the *Knowledge*:** With many sources, we do not have a single solution which could be directly transferred. It is necessary to generate a new representation mixing all solutions.
- **Avoiding *Negative Learning*:** The set of sources can be highly heterogeneous in terms of similarities with the target task. For this reason it is necessary to detect those solutions that, by its dissimilarity, far from helping to solve the new task could damage the learning process.

Although we will not focus on it again, the main idea of these methods is highly compatible with the concepts of *options* and *Hierarchical Reinforcement Learning* proposed on section 3.3.1.1, since that conception of tasks as independent blocks allows high modularity for building new solutions[36].

3.3.2.1 Transferring Experiences

These approaches proposes that experiences which are useful for the source tasks, can be also relevant for solving the target task. The most useful method of this kind [37], proposes to train the target task with a mixed dataset of experiences, including few experiences taken from the new task and a lot of experiences from the source tasks ($D = D_t + \sum_{s \in S} D_s$, where $|D_S| \gg |D_t|$).Moreover, for determining how each each source task should contribute to the dataset ($|D_s|$), it presents a measure of similarity between tasks:

$$\Lambda S_t = \frac{1}{N_t} \sum_{n=1}^{N_t} P(s_n, a_n, s'_n, r_n | \bar{M}_{s_t}) \tag{10}$$

Which defines which is the similarity of the target dataset D_t (with N_t experiences collected) to each source task model, estimated by $\bar{M}_{s_t}$. In this way, the amount of experiences taken from each source task can be proportional to this ΛS_t similarity, avoiding the aforementioned *negative learning*.

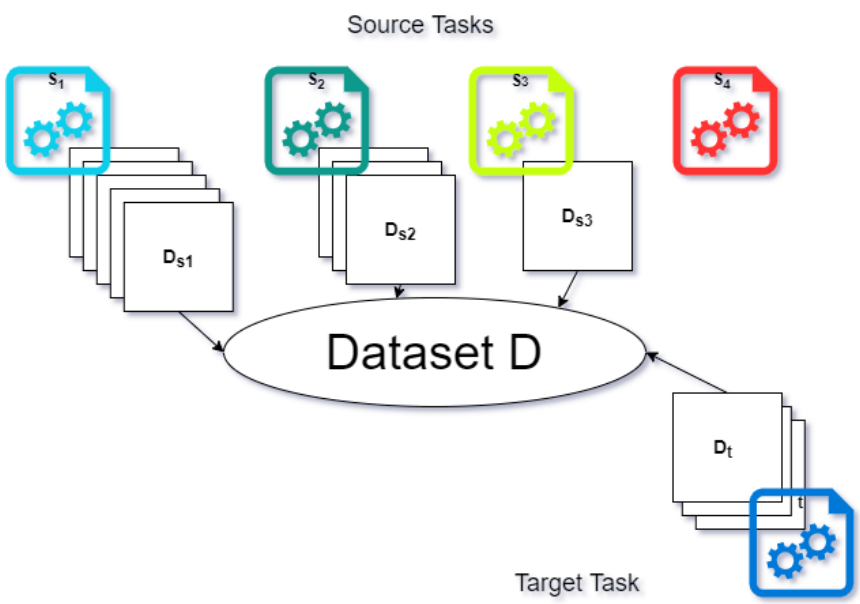

Figure 7: Exemplification of how each task would contribute with its experiences to the dataset D. Colors of each task codifies its similarity with the target task ΛS_t. Task more dissimilar contribute with less experiences or even without them for avoiding *negative learning*.

- **Benefits:** As the method only uses instances as *knowledge*, it can be complemented with other *Transfer Learning* methods based on models and parameters. For example, we could also take the *policy* π_s or the *action Value-Function* Q of the most similar task ($\text{argmax}_{S_t \in S} |\Lambda S_t|$) and initialize the target task with it, for fastening the process and obtaining a good starting performance since first episodes.
- **Drawbacks:** As instance based method, it needs to have access to the original environments of the source tasks for generating the D_s dataset. This access could be impossible in real world cases where the environment is temporal and experiment changes over time. Moreover, the method relies on the estimation of each source task model $\bar{M}_{s_t}$. This estimation is needs to obtain representative set source experiences, and could be imprecise in some cases.

3.3.2.2 *Actor-Mimic* for *Deep Reinforcement Learning*

The *Actor-Mimic* method[38], is the unique method presented which focuses *Deep Reinforcement Learning*, so it assumes that *knowledge* is represented through *Neural Networks* like *DQN*[6]. It aims to, generate an intermediate *multitask network* which could be able to execute any source task in a good enough level. The procedure that this methods defines can be summarized as follows:

1. Each *DQN* that solved a source task $\{S_1, S_2, ..., S_n\}$ is considered an expert $\{E_1, E_2, ..., E_n\}$.
2. In order to make experts to work in the same range, a softmax activation layer is append at the output of each source *Neural Network*.
3. Each expert *Q-Network* is transformed to a *policy network* by:

$$\pi_{E_i}(a|s) = \frac{e^{\mathcal{T}^{-1}} Q_{E_i}(s,a)}{\sum\limits_{a' \in A_{E_i}} e^{\mathcal{T}^{-1}} Q_{E_i}(s,a')} \tag{11}$$

   Where $\mathcal{T}$ is a given *temperature* parameter (Remember that the *action space* A_{E_i} is shared between all tasks).

4. We define an objective about the *policy* (for the multitask network) as:

$$\mathcal{L}^i_{policy}(\theta) = \sum_{a' \in A_{E_i}} \pi_{E_i}(a|s) \log(\pi_{AMN(a,s|\theta)}) \tag{12}$$

   being $\pi_{AMN}(a, s|\theta)$ the *policy* of the multitask *Actor-Mimic Network* using the parameter θ (The network which will be in training).

5. We also define an additional objective for a parallel *feature regression network*. Using the outputs of the last *hidden layer* (The last ones that can still be considered features), as:

$$\mathcal{L}^i_{FeatureRegression}(\theta, \theta_{f_i}) = ||f_i(h_AMN(s;\theta);\theta_{f_i}) - h_{E_i}(s)||^2_2 \tag{13}$$

   Where f_i represents the architecture of this *regression network*, that will try to predict, from the features of the last *hidden layer* of the *Actor-Mimic Network* ($h_{AMN}(s)$) the features of the the last *hidden layer* of the i^{th} expert network ($h_{E_i}(s)$). This dissociate architecture using a secondary network for predicting *features*, allows the *policy network* to have different feature dimensions than the experts. It allows the method to deal with heterogeneous experts that could use even different architectures.

6. Train the complete architecture combining both objectives:

$$\mathcal{L}^i_{Actor-Mimic}(\theta;\theta_{f_i}) = \mathcal{L}^i_{policy}(\theta) + \beta\mathcal{L}^i_{FeatureRegression}(\theta, \theta_{f_i}) \tag{14}$$

7. Once this *multitask network* is trained, use it as *pre-training* for any new target task.

As seen, the process presents a kind of *teacher-student* method where multiple *teachers* are taken into account. Learning from all these teachers, the method generates a *multitask network* that will contain the compressed *knowledge representation* of all source tasks. In best cases, when the set of *Experts* is large enough we could end up with a network which have a quite well *abstraction* of the concept that they share. For example, when we have a large set of *experts* in playing different *Atari 2600* games, we could end up with a *multitask* network that has a well representation of the concept of "*Play to Atari 2600*", and therefore, that could play to a wide range of new unseen games.

- **Benefits:** The method produces a *multitask network* that could be used in a wide range of cases as *pre-training* for starting the training from a good initial performance. Moreover, it does not need to have total access to the source *MDP*s, which can be quite convenient in some cases, and accepts set of *experts* with heterogeneous architectures. Finally, it could take especial profit of being combined with other methods, like the *instance based* method proposed in previous section.

- **Drawbacks:** The *Actor-Mimic* method focuses on architectures based on *Neural Networks*, and can not be applied to any other kind of *Reinforcement Learning* algorithm. In addition, it does not take into account the similarities between each source task and the target task, not dealing with the *negative learning* problem.

3.3.2.3 Transferring only the statistics of the Q-Table

The simplest approach for avoiding to deal with estimations of the source tasks (like, for example, the used in equation 10) is to only estimate their statistics[39]. It is to use the *mean* and *variance* of the *action-value functions* Q of each source task for setting the initial *Q-Table* of the target task. In this way each values vector that we have to generate can be taken from this distribution. Even it is possible to generate some distributions by *clustering* methods, in order to take different vectors from different distributions. In this simple way, we can include a good initialization of the *Q-Table* that is in between of all the source tasks.

As evolution proposal, if we would use more information, including model estimations on the method, we could combine it with the *MDP*s similarity defined on section 3.3.2.1 (equation 10). Using this similarity measure we could substitute the *mean* and the *variance* by the *weighted mean* and the *weighted variance*, in order to make the initial *Q-Table* closer to the *Q-Tables* of those task which are most similar (and thus avoiding the *negative learning*)

- **Benefits:** It is an extremely simple and fast method that can be executed without any model estimation so it is perfect when we have no access to the source *MDP*s. Sets a good starting point for the learning process and can be combined with any *Instance* based method.

- **Drawbacks:** It does not take into account that some source tasks could have different ranges of value functions, giving the same weight to all of them. Therefore, it could easily lead to *negative learning*.

3.3.3 Methods transferring from a single source to a single target task, that does not share the domain

Previous seen cases based their hypothesis in the assumption that all implied tasks shared the same *state* and *action spaces*. In this case this assumption is wrong, and therefore, the knowledge can not be directly transferred, since the *MDP*s of both tasks do not match. The most important problem with which deal in these cases will be to define *mapping functions*, able to transform *knowledge* represented in the source domain into *knowledge* represented in the target domain. For example, to use source tasks defined in the 2-D space, for helping to solve target tasks defined in a 3-D space (figure 3 (c)) would be a common problem in this field.

3.3.3.1 Use hand-crafted mapping functions for transferring experiences

Methods using this approach [40] propose to hand-craft two *mapping functions* able to transform *states* or *actions* of the source domain to *states* or *actions* of the target domain. For example, lets suppose that we have the source *action space* $\{a_{s1}, a_{s2}, a_{s3}, a_{s4}\}$ and the target *action space* $\{a_{t1}, a_{t2}\}$. In this case we could define the following mapping function: $\mathcal{X}_a(a_s, a_t) = \{a_{s1} \rightarrow a_{t1}, a_{s2} \rightarrow a_{t1}, a_{s3} \rightarrow a_{t2}, a_{s4} \rightarrow a_{t2}\}$. Also the same we could do for the *state space*. These two hand-crafted mapping functions are called $\mathcal{X}_a$ (for mapping *actions*) and $\mathcal{X}_s$ (for mapping *states*).

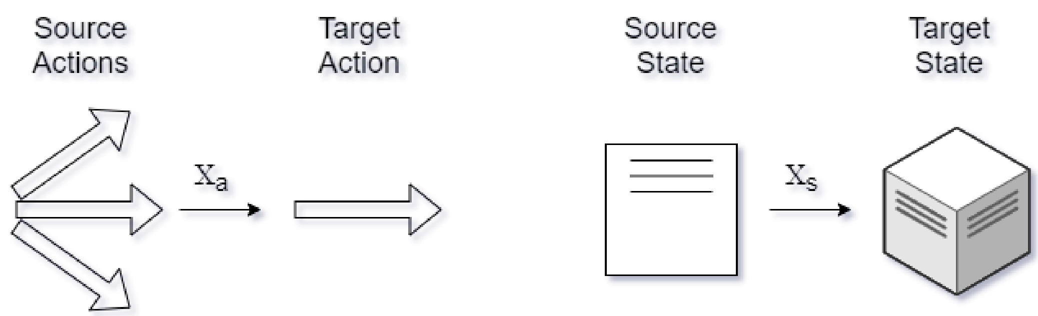

Figure 8: Visual example of how the hand-crafted mapping functions $\mathcal{X}_a$ and $\mathcal{X}_s$ would work

Once we have defined those *mapping functions*, the proposal is quite similar to the one seen on section 3.3.2.1 but only with one source task (since no *MDP*s similarities can be calculated (equation 10)). Generate a dataset $D_s = \sum_{n=1}^{N_t}(\mathcal{X}_s(s_n), \mathcal{X}_a(a_n), \mathcal{X}_s(s'_n), r_n)$ with the transformed relevant *states* and *actions* of the source task, and mix these transformed experiences with the experiences in the target task for learning.

- **Benefits:** It is an *instance based* method, therefore, it can be combined with several other methods to enrich even more the *knowledge transference*. In fact, if we observe the method, it could be even possible to introduce information coming from a set of different task, if we assume the cost of hand-crafting a *mapping function* for each task and we determine manually which is the similarity ΛS_t of each task with the source task.
- **Drawbacks:** The method requires to hand-craft specialized mapping functions for each pair of task. The cost of this hand-crafting can be extremely high, and usually these mapping functions could not be reusable by other tasks.

3.3.3.2 Transforming the *Hierarchical Reinforcement Learning Options* to its abstract representation

Using the same principles of *options* analyzed on section 3.3.1.1, these methods [41] build an abstract *MDP* where the representation of the convenient *options* do not depends on any specific domain of reference. The *options* of the source task are mapped to this *abstract MDP*, and from it, are mapped to the target *MDP*. For example, we could build an abstract *MDP* for representing the problems of moving inside a room. In this abstract *MDP* we would have what are called *Relativized Options*, like *translations* or *rotations*. Then, when we take a source task like a robot moving in a 2-D space, we will map its learned *options* to the terms of the abstract *MDP* (For example, $\{left, right\} \rightarrow \{Translate(Front), Rotate(180\,\mathrm{deg}), Translate(Front)\}$), and these *Relativized Options* will be transferred then to the concrete *MDP* of the target task (figure 9).

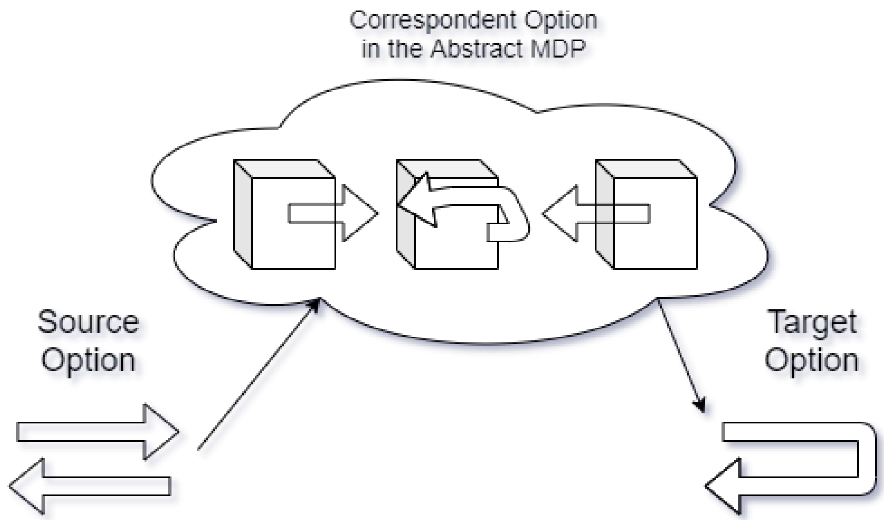

Figure 9: Visual example of how the source *options* can be interpreted by the abstract *MDP* and this interpretation be transferred to its interpretation in the target *MDP*.

The work of the *transfer phase* will be then to find these *mapping functions*. The commonly used methods for finding them suggest two different options:

- **Building a set of hand-crafted mapping functions** for any abstract *MDP* designed and determine through *Bayesian parameter estimation*[42] which of those *mapping functions* fit better for the concrete target task [43].

- **Re-defining the options** in terms of the agent space (non-Markovian). In that way, the representation will depend on the characteristics of the agent, and not of the environment. So, the mapping would be direct[44].

So let's enumerate the benefits and drawbacks of these proposed methods.

- **Benefits:** Proposing an intermediate *abstract MDP* produce mappings that are more interpretable and rich, since we determine the properties that the main problem has. Moreover, we could introduce implicit restrictions that could condition the *knowledge transference* positively. For example, if we are interested in restrict the movements of the agent to the four basic directions, we can choose to not represent half rotations in the *abstract MDP*, so there will be no way to transfer this *knowledge* to the target task.

- **Drawbacks:** In spite of including more *re-usability*, the proposal require more and more complex hand-crafting tasks than previous method. In real environments, solutions can be quite specific for the problems at hand, and it is hard to generalize them.

3.3.3.3 Mapped value function or policy transference

Most of these method are analogue to the ones already explained on section 3.3.1.4, but in this case an intermediate step must be included for mapping from the source to the target domain. Principal problem here is that, one more time, they must rely hand-crafted *mapping functions* [45], and therefore, solutions are quite specific for the given pair of tasks. The benefit here is that, at least, when hand-crafted mappings intercede there is a wide range of function possibilities for being explored. For this reason, not only value functions V^π or policies π are mapped, but also more flexible mappings are possible like transference between different architectures or learning algorithms.

- **Benefits:** The principal benefit of these methods is the high flexibility that they allow, on allowing to transfer learning between architectures that can be quite different in their definition. Moreover, the codification of specific *mapping functions* facilitates the process of including convenient restrictions or biases in the transference of *knowledge*.

- **Drawbacks:** One more time, hand-crafting the *mapping functions* implies a tedious work with low *re-usability*. By this reason, in spite of being a possible solution, does not solve the real problem of general *Transfer Learning* when the task domains are different.

3.3.3.4 Auto-generated mapping functions

In all this field governed by hand-crafted *mapping functions*, there is still a place for a concrete algorithm that aims to generate these tedious mapping functions between the source and the target domains in an automatic way. This algorithm is *MASTER*[46] and proceeds as follows:

1. Take as reference a large set of source task experiences D_s and a small set of target task experiences D_t ($|D_s| \gg |D_t|$).

2. Using D_t, build an estimation of the target task dynamics $\bar{T}_t$.

3. Using both datasets (D_s and D_t) find which are all the possible *mappings* $\mathcal{X}_s$ between experiences on both.

4. Using the target task dynamics estimation $\bar{T}_t$, each mapping $\mathcal{X}_s \in \mathcal{X}_S$, each *state* s_s in D_s and each *action* a in D_t, predict a supposed *state* s'_t in the target domain by: $s'_t = \bar{T}_t(\mathcal{X}_s(s_s), a)$.

5. Determine which *mapping function* $\mathcal{X}_s$ has generated results more consistent with the *knowledge* in D_t.

6. Use this $\mathcal{X}_s$ as general *mapping function*.

Although being a procedure that could produce quite easily a combinatorial explosion, it proposes one of the unique ways to produce automatic *mapping functions* in the non-shared domains problems. By this reason this last algorithm is the one which more benefits presents.

- **Benefits:** By first time, *MASTER*, proposes a way for discovering the *mapping functions* between the source and target domains in an automatic way. By this reason, it is the unique one among the presented methods that can be applied to general problems in a common way.

- **Drawbacks:** It is so difficult to find a good trade-off with the size of the target dataset D_t. A large D_t can produce a combinatorial explosion, with several *mapping function* candidates $\mathcal{X}_s$. A short D_t can produce poor target task dynamics estimation $\mathcal{T}_t$ making the whole process inconsistent.

4 Conclusions

In this work, we have defined which is the problematic of *Reinforcement Learning* when learning from scratch. That problematic could be summarized as the necessity of the agent of learning some basic abilities, that in fact, could be already known by another agent which solved a similar task. For dealing with this problematic we have presented the *Transfer Learning* approach, which is on the base of the natural learning processes that all thinking beings apply. This approach proposes to extract the *knowledge* of one or various expert agents (in different source tasks), for teaching to the new agent those common abilities that could be useful for learning its new task (the target task). For this reason, we first have identified the main classification of *Transfer Learning* problems -Depending on different points of view-, and then we have presented which are the most relevant approaches that are used in each type of problem.

We have analyzed principally three kind of problems:

- **Single source to single target with shared domains**: Here, we have analyzed why is this the easiest problem, and therefore, the one which best solutions and theoretical demonstrations allows. We have analyzed some solutions based on *Hierarchical Reinforcement Learning*, analysis of the *MDP*s for extracting *knowledge* and even in generating heuristics for reducing the *action space*. Moreover, we have seen here demonstrations about the minimal performance that could be achieved by applying only a *direct policy transference*.
- **Multiple sources to a single target with shared domains:** Here we have analyzed which is the main problematic when we want to exploit a heterogeneous set of source tasks instead only one. Aiming to deal with this problematic, we have presented since simple methods based on merging the statistics of the *Q-Tables* to *Deep Reinforcement Learning* based methods like *Actor-Mimic*, which exploit the benefits of using *Neural Networks* as *knowledge representations*. Moreover we have presented *instance based* methods which proposes a parallel framework that can be combined with rest of the algorithms, dealing at the same time with the detection of dissimilar tasks and thus avoiding the *negative learning*.
- **Single source to single task with non-shared domains:** Here we have analyzed the most difficult problem with which *Transfer Learning* have to deal in *Reinforcement Learning*, that is, to deal with heterogeneous domains. In these cases, as the *knowledge* of both are interpreted in different terms, they can be not mixed and it is mandatory to define *mapping functions*. By this reason most of the methods seen here, presented tedious works of hand-coding these functions. However, we have also seen that, by lucky, there are proposals that, at the cost of expensive researches, are able to find by itself which are these optimal *mapping functions*.

It is important to denote that, although *Deep Reinforcement Learning* have acquired an enormous relevance on the *Reinforcement Learning* field, this work have been focused on analyzing, especially, *Transfer Learning* approaches for classical *Reinforcement Learning* algorithms. It has been intentional, since most of the approaches thought for classical algorithms can be easily reinterpreted for being applied with *Neural Networks*, but this affirmation is not true backwards. For example, *instance based* algorithms can be applied directly with *Neural Networks*, and direct parameter transference implies only to transfer the whole *Network* instead the *Q-Table*. However, methods like *Actor-Mimic* would never apply for classic methods.

References

[1] D N, Perkins & G Salomon. *Transfer of Learning.* International encyclopedia of education, 1992

[2] Y Bengio, A Courville & P Vincent. *Representation Learning: A Review and New Perspectives*, 2014.

[3] S Barrett, M E. Taylor & P Stone. *Transfer Learning for Reinforcement Learning on a Physical Robot.* Proceedings of the AAMAS Workshop on Adaptive and Learning Agents, 2010.

[4] Yu, Y. Towards Sample Efficient Reinforcement Learning - Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. IJCAI. 2018

[5] Watkins, C.J.C.H. Learning from Delayed Rewards. Ph.D. thesis, Cambridge University. 1989.

[6] Mnih V., Kavukcuoglu, K., Silver, D. Rusu, A. A., Veness, J., Bellemare, M G., Graves, A., Riedmiller, M., Fidjeland A, K., Ostrovski, G., Petersen S., Beattie, C., Sadik A., Antonoglou I., King, H, Kumaran, D., Wierstra, D., Legg S. & Hassabis, D. Human-level control through deep reinforcement learning. Nature 518, 529–533. 2015

[7] Hessel, M., Modayil, J., Hasselt, H,V. Schaul, V., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M & Silver, D. Rainbow: Combining Improvements in Deep Reinforcement Learning. AAAI. 2018.

[8] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R,. Welinder, P., McGrew, B., Tobin, J., Abbeel, P. & Zaremba, W. Hindsight Experience Replay. NIPS. 2017

[9] Nguyen, H. & Hung L, M. Hindsight Experience Replay With Experience Ranking. Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob). 2019.

[10] Bellemare, M. G., Naddaf Y., Veness, J. & Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents Journal of Artificial Intelligence Research 47 (2013) 253–279. 2013.

[11] Dubey, R., Agrawal, P., Pathak, D., Griffiths, T. L. & Efros A. A. Investigating Human Priors for Playing Video Games. ICML. 2018.

[12] Marcus G. Innateness, alphazero, and artificial intelligence. 2018.

[13] Weiss, K., Khoshgoftaar, T.M. & Wang, D. A survey of transfer learning. J Big Data 3, 9. 2016.

[14] Serrano E., Molina M., Molina, M. Manrique, D. & Baumela, L. Experiential Learning in Data Science: From the Dataset Repository to the Platform of Experiences. Conference: 13th International Conference on Intelligent Environments - IE. 2017.

[15] Alessandro Lazaric. Transfer in Reinforcement Learning: a Framework and a Survey. Marco Wiering, Martijn van Otterlo. Reinforcement Learning - State of the art, 12, Springer, pp.143-173. 2012.

[16] Puterman, M. L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. 1994.

[17] Markov, A. A. Teoriya algorifmov. number OTS 60-51085. 1954.

[18] Lazaric A. Knowledge transfer in reinforcement learning. PhD thesis, Poltecnico di Milano. 2008.

[19] Aha D. W. Lazy Learning. Artificial Intelligence Review Vol 11, Nos, 1-5. 1997

[20] Fix, E., Hodges, J.L. Discriminatory analysis, nonparametric discrimination: Consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[21] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmille M. Playing Atari with Deep Reinforcement Learning. NIPS Deep Learning Workshop. 2013.

[22] Langley P. Transfer of knowledge in cognitive systems. Twenty-Third International Conference on Machine Learning. 2006.

[23] Hengst B. Hierarchical Reinforcement Learning. Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. 2011.

[24] Sutton, R. S. Precup, D. & Singh, S. P. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence, 112(1-2):181– 211. 1999.

[25] Puterman, M. L. Markov Decision Problems. Chapter 8, 1994

[26] McGovern, A. & Barto, A. G. Automatic discovery of subgoals in reinforcement learning using diverse density. Proceedings of the Eighteenth International Conference on Machine Learning (ICML). 2001.

[27] Mehta, N., Natarajan, S., Tadepalli, P. & Fern, A. Transfer in variable-reward hierarchical reinforcement learning. Machine Learning 73, 289. 2008.

[28] Sherstov, A. A„ Stone, P. Improving action selection in MDP's via knowledge transfer. Proceedings of the Twentieth National Conference on Artificial Intelligence. 2005.

[29] Ferguson, K. & Mahadevan, S. Proto-transfer learning in markov decision processes using spectral methods. In: Workshop on Structural Knowledge Transfer for Machine Learning at the Twenty-Third International Conference on Machine Learning. 2006.

[30] Mahadevan, S. & Maggioni, M. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. Journal of Machine Learning Research 38:2169–2231. 2007.

[31] Ferrante, E., Lazaric, A. & Restelli, M. Transfer of task representation in reinforcement learning using policy-based proto-value functions. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems, pp 1329–1332. 2008

[32] Ferns, N., Panangaden, P. & Precup, D. Metrics for finite markov decision processes. In: Proceedings of the 20th conference on Uncertainty in Artificial Intelligence, pp 162–169. 2004.

[33] Phillips, C. Knowledge transfer in markov decision processes. 2006.

[34] Rüschendorf, L. Wasserstein metric, Encyclopedia of Mathematics. 2001.

[35] Asawa, C., Elamri, C. & Pan, D. Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance. 2017.

[36] Bernstein, D. S. Reusing old policies to accelerate learning on new mdps. 1999.

[37] Lazaric, A., Restelli, M. & Bonarini, A. Transfer of samples in batch reinforcement learning. Proceedings of the Twenty-Fifth Annual International Conference on Machine Learning, pp 544–551. 2008.

[38] Parisotto, E., Lei Ba, J. & Salakhutdinov R. Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning. ICLR. 2016.

[39] Tanaka, F., Yamamura, M. Multitask reinforcement learning on the distribution of mdps. IEEE International Symposium on Computational Intelligence in Robotics and Automation, vol 3, pp 1108–1113. 2003.

[40] Taylor, M. E., Jong, N. K. & Stone P. Transferring instances for model-based reinforcement learning. Proceedings of the European Conference on Machine Learning, pp 488–505. 2008.

[41] Ravindran, B. & Barto, A. G. Relativized options: Choosing the right transformation. Proceedings of the Twentieth International Conference on Machine Learning, pp 608–615. 2003.

[42] Jaakkola, T.S. & Jordan, M.I. Bayesian parameter estimation via variational methods. Statistics and Computing 10, 25–37. 2000.

[43] Konidaris, G., Barto & A. G. Building portable options: Skill transfer in reinforcement learning. Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp 895–900. 2007.

[44] Torrey, L., Shavlik, J. W., Walker, T. & Maclin, R. Skill acquisition via transfer learning and advice taking. Proceedings of the European Conference on Machine Learning, pp 425–436. 2006.

[45] Taylor, M. E., Stone, P. & Liu, Y. Value functions for RL-based behavior transfer: A comparative study. Proceedings of the Twentieth National Conference on Artificial Intelligence. 2005.

[46] Taylor, M. E., Kuhlmann, G. & Stone, P. Autonomous transfer for reinforcement learning. Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems, pp 283–290. 2008