Università degli Studi di Milano Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Online Experience Sharing in Deep Reinforcement Learning

Relatore: Prof. Giuseppe Vizzari

Co-relatore: Prof. Ivana Dusparic

Tesi di Laurea Magistrale di:
Federico Bottoni
Matricola 806944

Anno Accademico 2019-2020

A dissertation submitted to Università degli Studi di Milano-Bicocca,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

2021

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university.

Signed: 

Federico Bottoni

Dublin, 12 February 2021

Acknowledgments

First of all I would like to express my gratitude to my wonderful family, starting from my mother, father, brother and my dog but also my more distant family, everyone who has always supported me a lot both during my experience abroad and in general in my whole life, I know I will always give 100% if they are by my side. I would like to thank Erica for all the joy and energy she has shared with me in all these years and thank you to have encouraged me every day to follow my desires and my happiness.

Thanks to Prof. Giuseppe Vizzari and to Prof. Ivana Dusparic to have taught me Reinforcement Learning, and to feel always curious towards the world. It is a very important virtue, a kind of fuel that keeps people alive.

I would like also to thank Trinity College Dublin to have given me the opportunity to be its guest and specifically to Alberto who listened and discussed my ideas and insights for the research and the dissertation.

Thanks to my friends and to all the flatmates I lived with in these months in Dublin, especially to Nassim who has shared with me this beautiful experience.

Federico Bottoni

Università degli studi di Milano-Bicocca

12 February 2021

Abstract

One of the most widely used learning-based approach to estimate the best behaviour or policy suitable in an operating environment is Reinforcement Learning (RL). Frequent is the usage of artificial neural networks as function approximators in order to handle continuous state-spaces and larger-size problems: this approach is called Deep Reinforcement Learning (DRL). The task of the approach is to use reward signals to learn a successful agent function. A common issue of this method is the slowness that characterizes it in reaching the convergence on the desired policy, since before many state trajectories have to be experienced and the effects of the actions have to be evaluated from those states.

Transfer Learning (TL) is an accelerating class of methods that involves generalizations of knowledge across multiple distributions or domains. Usually it takes place among a source and a target, the transferred object can concern knowledge, experience, domain or other exchangeable information, with the purpose of enhancing agent's performance and speed. Among the several algorithms present in literature, TL has been experienced encouraging the knowledge transfer represented by q-values or sharing part of their rewards with the neighbours, usually in groups of agents with the same task or sometimes with heterogeneous tasks. Transfers usually take place offline, from skilled agents to enhance another agent's learning process, sometimes it can be partially online where the agents share a portion of the learning process and exchange information, but rarely it can be completely online where the agents learn and share at the same time, from the beginning to the end.

The main contribution of this thesis is Online Experience Sharing (OES), an algorithm that aims to study how a learning agent can utilize another agent's experience through TL (concretely sharing sets of transitions) if the agents run concurrently (online). Transferring information is not always a useful process so, in order to maximize the efficiency, OES includes the introduction of methods to handle the transfer frequency and the amount of transferred data. One of the most important challenges of TL consists in the determination of the selection criteria of data to transfer and to use

to learn: the thesis proposes multiple solutions, and compares them, based on a state or state-action confidence system implemented in different ways with different selection methods.

OES is evaluated in two different benchmark RL scenarios, Mountain Car and Cart Pole. The evaluation is based on different metrics, from jumpstart and speed-up to the possible asymptotic improvement and to a brief judgement on convergence stability. OES can improve the learning speed encouraging the agent to reach convergence quicker than an independent agent, it can push the agent to find quickly better trajectories in terms of cumulative reward bringing an improvement of maximum reward and total accumulated reward. OES can also improve the stability of the learning process especially in agent's convergence period. Every good performance depends directly on the environment's task as well as the used TL methods, the confidence implementation and the selection methods, generally every task requires a specific configuration to show improvements. OES introduces a set of hyperparameters that affect the agent's performances so they have been analyzed and tuned through Bayesian Optimization in order to find the direct correlations with the simulation performance in terms of total reward.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human brain is powerful, a world chess champion Mikhail Tal, once declared how his chess game was inspired from watching ice hockey. He could notice some habits that helped the hockey players to score when an opportunity came, in some way he used discovered similarities between the two games to improve his own performance. From the hockey game, he decided to try transferring the idea to his chess game adapting his strategy inspired by the hockey players [9]. This particular episode demonstrates how human brain is capable of finding similarities and performing adaptations between different worlds to find the best manner to solve a problem, especially in strategic games like chess everything can inspire the player to build a specific strategy. This "transferring" concept works in human brain, it can allow anyone to find a better solution and sometimes more quickly. It would be possible to generalise the concept and "transfer" it to machines in a similar way, it's necessary to find the right way to trigger the transfers, the proper abstraction of data that should be shared between the entities and finally analyse the result: multiple transfer-scenarios are feasible to implement, one possible could involve situations with a software trying to learn a determine task's solution and, in order to speed it up, some information may be exchanged among the learning processes. Another possible scenario could consist in different agents with different tasks but connected someway, they might share some information to try to adapt the acquired knowledge to the other tasks. These kind of situations are implementable as learning agents in reinforcement learning processes interacting with

an environment trying to solve tasks.

1.1 Research Motivation

This section gives a brief introduction and defines the gap in the research area that this dissertation is aiming to address.

1.1.1 Reinforcement Learning

The origins of Reinforcement Learning (RL) can be traced back in psychology to animal learning behaviour and was instigated by Edward Thorndike in his "trial-and-error" learning experiment in 1898 [32]. Thorndike described the effect of reinforcing events on the tendency to select actions in its "Law of Effect" referring to animal's reaction to responses accompanied by satisfaction and discomfort. He stated that "the greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond" [40].
In the early 1948, Alan Turing described a design for a "pleasure-pain system" that worked along the lines of the Law of Effect. He stated:

> When a configuration is reached for which the action is undetermined, a random choice for the missing data is made and the appropriate entry is made in the description, tentatively, and is applied. When a pain stimulus occurs all tentative entries are cancelled, and when a pleasure stimulus occurs they are all made permanent.
>
> *- A. Turing, 1948*

A stimulus can be interpreted as reinforcement where the pain stimulus depicts a negative reward and a pleasure stimulus depicts a positive reward [40]. These premises determined the birth of this artificial intelligence approach, RL.
RL is a powerful approach capable of modelling a behaviour or policy to react to the interactions with an environment at its best, exploiting the trial-and-error learning

system which consist in the exploration of multiple trajectories and strategies to solve the task and at the end to identify the best one, building a function that allows to calculate the proper action given a certain state.
RL algorithms are based on the exploration-exploitation dilemma, in which learning agent need to start the learning process from a exploration phase that tends to not exploit the built knowledge to select the actions since it is too young and not accurate. Gradually it switches to a exploitation phase where the agent tends to trust its knowledge assuming to have the necessary information to solve the task and minimising the possibility of visiting new states. An important RL's significant weakness is based on this inner necessity to explore in the initial phase, in particular the algorithms usually take much time to converge to a good policy due to the previous exploration phase takes much time. In addition they do not converge always to the best policy but sometimes stop to a good enough one that allows to solve the task, some other algorithms encourage the achievement of the best policy but can take much more time.
These weaknesses, convergence speed and asymptotic performance, are general issues that can be improved with multiple techniques.

1.1.2 Transfer Learning

One of the most widely used technique to achieve an improvement in speed and best policy found in reinforcement learning processes is in Transfer Learning (TL). This term includes many methodologies applicable in RL, starting from transferring an agent's knowledge to another one with different tasks, transferring knowledge among agents to enable some kind of collaboration which should improve the effectiveness of the learning agents or again modifying agent's learning processes to build dependencies and encourage the collaboration. The transfer of information can accelerate the exploring phase pushing the learnt hints to be exchanged among the agents to reach a global convergence of policy more quickly. In addition agents may spread advice on the most known states and on the best policy they reached to check if other agents have already reached the same outcome or a different one.

1.2 Challenges

The transfer mechanisms included in TL hide multiple challenges and questions to solve. To define a proper transfer the first point consists in setting the *involved actors*, sending and receiving agents and their relationship, different configurations may lead to different results so the goal should suggest the configuration to adopt.
Secondly a set of issues compose one of the main challenge about *transfer frequencies and sizes* with the aim of defining how and how long data are stored and used before being discarded and replaced. The set includes the definition of transfer frequency, which is important since it determines the number of communications needed among the agents and the size of the possible new learnt knowledge that can be acquired before the establishment of the next exchange; the definition of the amount of data to transfer, which is connected to the motivations related to the previous point, and consequently the definition of the storing structure for transferred data and its capacity.
Other important details about the transfer concern the *object to transfer* which can be literally any kind of information, from the knowledge representation, to the experience, segments of knowledge and so on. Tied to this aspect is the *data selection criteria*, because exchanging every information can be pointless as well as expansive in terms of computation cost, data should to be selected exploiting some logic related to the kind of information the agent is interested in exchanging.

1.3 Contribution

The main objective of this dissertation is to extend an already existing TL technique to a neural networks-based agent scenario adapting it to a continuous state-space too. The transfer algorithm to extend is Parallel Transfer Learning (PTL) which got interesting results using Q-learning technique in transferring knowledge represented by q-values, among set of learning agents. The usage of artificial neural networks as approximator function implies multiple changes brought as contribution including the following aspects:

- The transfer process cannot include a q-values merge mechanism since the q-table is not available. With neural networks approximator, it is possible to exchange transitions to perform deep q-network's weights updates;
- Transfer's frequency and dimension are redesigned to let them adapt to the different learning context, encouraging the transferred data usage in particular in the first part of the learning process;
- Selection of data to transfer are designed differently keeping PTL's basis to transfer most reliable data to replace the least confident but adapting it to continuous state-space system. Multiple methods are designed and compared to observe which settings are more suitable to every task/environment;
- PTL's confidence system works with discrete state space since it is based on counting the number of visits of each state. Confidence system has been redesigned in order to use properly it to evaluate state's confidences.

1.4 Evaluation

Being the techniques introduced by this novel transfer learning approach, new and different by the already existing methods, the designed solutions are evaluated with benchmark environments in single-agent systems: mountain car and cart pole are two of the most simple scenarios characterised by specific tasks to learn. Each experiment has been evaluated in its capabilities of converging quickly, stably, at the highest level of reachable cumulative reward. Graphs are analysed in terms of learning speed, learning stability and asymptotic improvement, with a brief judgement of the acquired total reward.

1.5 Roadmap

This document has been structured as following:

- **Chapter 2** provides background information on Single-Agent Systems, the state-of-the-art in RL and Deep Q-Learning with a specific focus on collaborative agents models and a couple of strategies;
- **Chapter 3** brings the main contribution of this thesis, Online Experience Sharing (OES). It introduces a transfer learning algorithm in Deep Q-Learning context to speed up the reinforcement learning processes by on-line sharing transitions among the agents;
- **Chapter 4** provides details about the software and the simulation environments;
- **Chapter 5** describes the applications, experimental set-up, metrics and outcomes obtained from the simulations;
- **Chapter 6** concludes the dissertation summarising the proposal and briefly discussing the open issues and the future works.

Chapter 2

Background and Related Work

This chapter begins with a formal introduction of agent and machine learning in general, it continues with reinforcement learning and the related elements along with a brief description of RL methods, successively an introduction of Q-learning and Deep Q-learning. It finishes presenting the main aspects of transfer learning and some concrete TL frameworks relevant in literature.

2.1 Agent Systems

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors. A software agent has encoded bit strings as its percepts and actions." [34]. A basic example of agent as shown in figure 2.1, agent acts accordingly only on the sequence of observations that the it has already observed. It can be defined as *rational* if it does the right thing [34]: to define if the action is the right choice it is necessary to introduce one or more evaluation measures.
The "behaviour" of an agent can be represented by a sequence of visited states and chosen actions, if the sequence is a desired one, the agent can be considered well performing and the behaviour correct. On the other hand if a sequence is different and

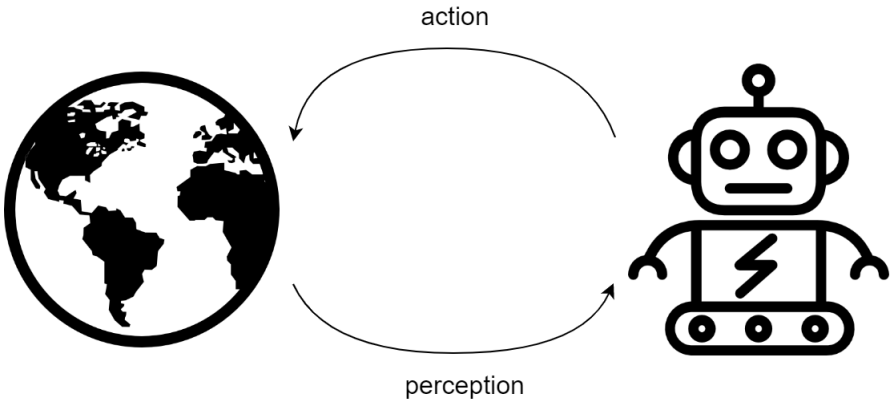


Figure 2.1: Representation of an agent interacting with the environment.

far from the goal, the agent is bad performing and of course, the evaluation negative. The introduction of evaluation measures is necessary to have a reliable feedback, they usually depend strictly on the task so everyone needs to be studied separately. For instance on an image classifier task, evaluation measure could be several: classifier accuracy, time to recognise a given image or others, it depends on the context and on the personal goal.
It is possible to identify several kind of agents and in [34] they are grouped in these categories:

- **Simple reflex agents** select actions on the basis of the current perception, ignoring the past ones.
- **Model-based reflex agents** save an internal state that depends on the observed world so far.
- **Goal-based agents** act based on a goal, thereby make decisions based on the best way it can achieve the goal.
- **Utility-based agents** are not just evaluating the goal they evaluate the efficiency as well. Usually there are different actions that lead the agent to the same goal, but some can be faster then others, some safer and so on.

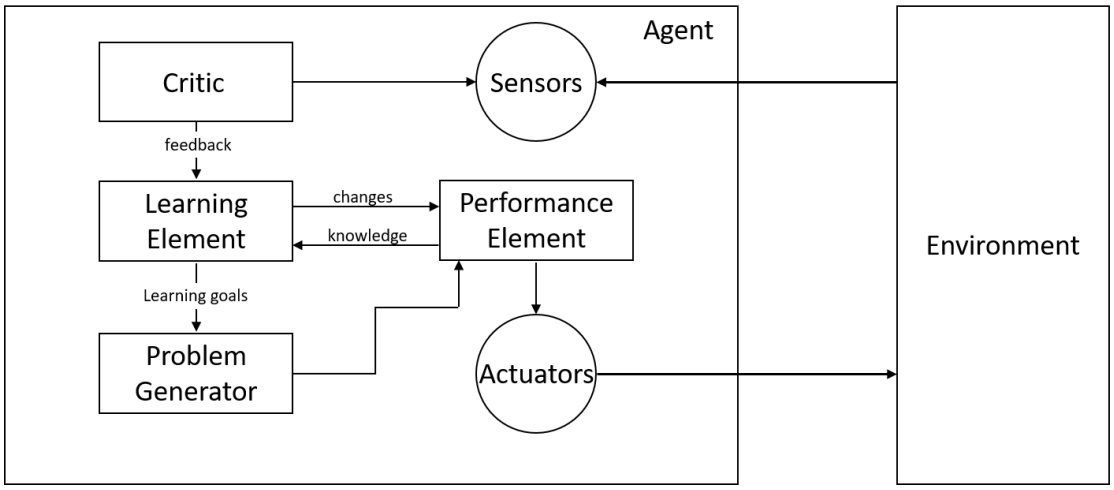


Figure 2.2: Scheme of a learning agent [34].

- **Learning agents** have the ability to learn the behaviour required to achieve their goals.

2.1.1 Learning Agents

Agents can improve their performances discovering new behaviours and possibilities. They start without knowledge and slowly, during the learning process, they explore more and more possibilities through the state-space and have the chance find their best behaviour. The structure is shown in figure 2.2

Russel and Norvig [34] gave a specific definition on when an agent is considered "learning agent":

> An agent is learning if it improves its performance on future tasks after making observations about the world.

Intelligent agents have the ability to learn the behaviour required to achieve their goals. As part of this learning process, agents can learn from other agents to optimise their local behaviour [1].

A learning agent can be divided into four conceptual components:

- **Critic** acts as a judge which establishes how well the agent is doing with respect to the fixed performance standard.

- **Learning Element** has the role of making improvements using the received feedback from the critic and control the performance element changing it.
- **Performance Element** is responsible for selecting actions, from sensors signals.
- **Problem Generator** is responsible for suggesting actions that will to new and informative experiences.

Learning in intelligent agents can be summarised as a modification process to bring the components into closer agreement with the available feedback information, thereby improving the overall performance of the agent.
The environment in which the agents have to perform in most of the cases is not well known and often is a dynamic context, moreover in most of the real-world situations the complexity grows exponentially, for example a face-recognition and labelling system. Agents should learn their tasks, in order to let them to explore the environment through the available actions and finds the optimal behaviour.

2.1.2 Environment

The environment is the space, either abstract or concrete, where the agent is located. For instance from the perspective of a robot agent it could be its working area, for autonomous vehicles it could be a city map representation such as in [10] or again for an agent player of Atari games the environment is the game scene as in [30].
Environments can be categorised through some proprieties described ins [34]:

- **Accessibility or Observability**

  Environment is defined as fully observable if the agent that acts in it can obtain through its sensors complete and up to date information.

  Fully observable environment are convenient for the agent, since it can act exploiting just the current perceived environment.

- **Determinism**

  In case the actions from every states lead to other states and the transitions are

completely determined by the starting state, the environment is called *deterministic*. Otherwise it is *non-deterministic* or *stochastic*.

- **Sequentiality**

  In an *episodic* environment, the agent's experience is divided into atomic episodes, which are independent, thus means that they do not affect the other episodes. Otherwise the environment is called *sequential*.

- **Dynamic**

  Reality is often a though kind of environment since it continuously change, or in general many events could change it making it slightly different, causing important consequences on the agent's learning process. For a rover robot, the weather or just the humidity are important conditions that make this kind of environments *dynamic*. Sometimes the environments are *static*, especially when the agent lies in a simple environment. In those cases it never changes. A *semi-dynamic* environment is when it is static but the dynamism is given by agent's performance score.

- **Discrete vs Continuous**

  The state-space can be *discrete* or *continuous*, it affects the complexity of the learning process since a discrete space implies a finite and fixed number of states and possibilities.

2.2 Machine Learning

Machine Learning (ML) is a category of techniques that use prior experience or examples to improve performance on a task. It can be considered into three under-categories based on the feedback provided. The classes are not strictly defined and there could be overlap:

- **Supervised Learning**. The basic question is "how do you automatically build a function that maps some input into some output when given a set of example

pairs?" [26]. The most common examples of this learning problems include text and image classification, object location, sentiment analysis, regression problems and so on. These problems share the same basis: given many examples and desired output, the goal is to generate outputs for other inputs. It is considered supervised since the learning is based on a *supervisor* which grantees the correct association of inputs and outputs. In the aforementioned cases the supervisor is contained in dataset;

- **Unsupervised Learning**. The main objective is to learn some hidden structure of the dataset. One of the most popular implementation of this kind of learning programs is data clustering, where the model tries to learn some relationship which ties subsets. Another one is *Generative Adversarial Networks (GAN)* where a neural network generates fake data to try to fool a second one which tries to discriminate the real samples from the artificially made. In these cases none can determine the correctness of the generated associations, so the supervisor is absent;

- **Reinforcement Learning** lies between the two partitions since despite the absence of a true supervisor there are some basic tools, including reward signal and value function, that provide the agent some hints about the effectiveness of the model's choices.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning approach that enables intelligent agents to learn the optimal behaviour in a unknown environment via trail-and-error [40].

In RL, learning task consists in picking the right choice to act according on the situation. Goal of this task for an agent is to explore the environment through its actions in order to choose the set of actions that maximise the *reward* (or *reinforcement*) given

by the environment [15].

In each RL technique is possible to identify two main components

- **Value function**

  Defined as functions of states that estimate how "good" it is, for the agent, to be in a given state (or how good it is to perform a given action from a given state). The notion of "how good" here is defined in terms of future reward. The *reward* consists in a simple number, $R_t \in \mathbb{R}$: informally, the agent's goal is to maximise the total amount of reward it receives, this means not maximising the immediate, but cumulative reward in the long run.

  The interest in cumulative reward, rather than the immediate reward, exists since a given action may be the best choice according to the current reward, but in the future it could turn out as a mistake because it leads to a non optimal solution. In order to avoid the greedy choice, two different concepts of "reinforcement" are introduced:

  - **Immediate rewards** is the feedback given by the environment after the execution of a single action by the agent.
  - **Return**, also known as *discounted cumulative reward*, is the feedback given by a sequence of actions that lead to an end state.

  In many cases the agent faces the possibility that every action leads to the same immediate reward, so returns describe rewards of different state-action sequences providing the agent a better information to choose the best action.

- **Policy**

  Policy defines the learning agent's behaviour at a given step and represent the goal of the algorithm. In term of a Markov Decision Process, defined below, policy π is a probability distribution over actions given states.

RL problem can be modelled using ideas from dynamical systems theory, specifically from Markov Decision Processes (MDPs). MDPs is a discrete time stochastic control

process that provides a mathematical framework for modelling decision making situations. It is defined as a 5-tuple (S, A, P_a, R_a, γ), where

- $\mathbf{S} = \{s_t, s_{t+1}, \ldots, s_n\}$ is a finite set of states.
- $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$ is a finite set of actions.
- $\mathbf{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the probability that an action a in state s at time t_i will lead to a state s' at time t_{i+1}.
- $\mathbf{R}(\mathbf{s}, \mathbf{a})$ is the immediate reward received after have performed action a in state s.
- $\boldsymbol{\gamma} \in [0, 1]$ is the discount factor which determine the weights of future rewards in relation to the immediate reward.

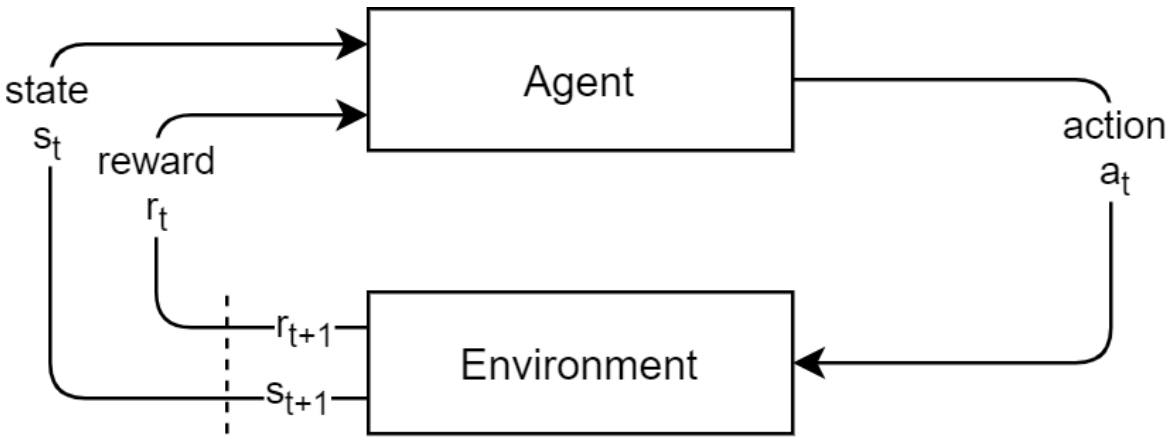


Figure 2.3: Reinforcement Learning process [40]: at time t the agent take an action a_t, environment returns a reward r_{t+1} and the next state s_{t+1}

The RL process in figure 2.3 is a Markov models, so it assumes the Markovian property [29], as follow:

$$P(s_{t+1}|a_t, s_{0:t}) \; = \; P(s_{t+1}|a_t, s_t) \tag{2.1}$$

Given a time t, state s_{t+1} is only determined by previous state s_t and action a_t, this means that the next state is independent from states and actions happened before the current time.

A fundamental propriety of value functions is that they satisfy particular recursive relationships, it is developed with all the concepts introduced so far and holds between the value of s and the value of its possible successor states. This consistency condition is expressed by Bellman equation which is present in its brief version in equation 2.2

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \Big[G_t | S_t = s \Big] \\ &\dots \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \big[r + \gamma v_\pi(s') \big] \end{aligned} \tag{2.2}$$

It expresses a relationship between the value of a state and the values of its successor states [40].

RL problem's peculiarity is that the agent can not access to every element of the MDPs, neither the immediate reward (R) nor how the environment will react due to its actions (P). The agent is precluded from planning a specific solution, but it can still find an optimal policy. It is possible to identify two classes of algorithms:

- **Model-based**

  RL algorithms which use models since they are available. There is the basic approach of *pure planning* to select actions like *Model Predictive Control (MPC)*. There are others like *I2A* and *MBVE* which exploit different strategies to learn the model.

- **Model-free**

  They are algorithms where the models are not available, it can be thought as an "explicit" trial-and-error algorithm [40]. They can be divided again in *Policy Optimisation* where the methods represent a policy explicitly as $\pi_\theta(a|s)$ and the optimisation happens on the parameters θ and Q-Learning methods where the methods learn an approximator $Q_\theta(s, a)$ for the optimal action-value function, $Q^*(s, a)$. The first collection of algorithms includes *Policy Gradient*, *Proxy Policy Gradient (PPO)*, *Trust Region Policy Optimisation (TRPO)* and *Actor-Critic (A2C / A3C)*; the second includes *Deep Q-Network (DQN)* and *Categorical 51 (C51)*.

2.3.1 Q-Learning

Q-learning is form of model-free RL and is widely used in literature [50]. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains [49].

In Q-learning, an agent tries an action in a state and evaluates the reaction in terms of reward (positive) or penalty (negative) that it receives. By repeatedly trying actions in all the explored states, the agent learns which trajectory (sequence of state-actions) leads to the optimal solution, which is judged by long-term discounted reward. To achieve the maximum feedback, an agent has to choose the best action possible for each visited state.

This technique's learning is based on the *q-update* function present in equation 2.3. This function updates the q-values starting from the previous ones and merging them adding a weighted update which depends on the current values, the earned reward in the current step and the discounted value of the best estimated state-action of the next step.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right) \tag{2.3}$$

where:

- $\mathbf{s_t}$ is the state at time t and s_{t+1} represents the next possible state;
- $\mathbf{a_t}$ is the current action;
- $\mathbf{r_t}$ is the earned reward for an agent taking a given action a_t in state s_t;
- $\boldsymbol{\alpha} \in [0, 1]$ is the learning rate, it balances the importance of the new experiences in the q-value updating phase. Higher is the rate, more changes will affect the model with the update. If it is 0 the agent is unable to learn, while 1 leads the agent to replace its previous knowledge with the new acquired information;
- $\boldsymbol{\gamma} \in [0, 1]$ represents the discount factor, it determines how the expected future reward influences the model then the immediate reward.

A discount factor equals 0 leads the agent to evaluate only the current reward (greedy behaviour), on the other hand a value closer to 1 leads the agent to take in consideration the long-term discounted reward with the same importance as much as it does for current reward;

- $\max_{a} \boldsymbol{Q(s_{t+1}, a)}$ is the maximum achievable value performing the optimal action a from the next state represented by s_{t+1}.

Q-learning algorithms use look-up tables to store $Q(s_i, a_j)$ values, called *q-table*. The method stores each pair $Q(s_i, a_j)$ in this table, once time and then updates its value without storing a new one. q-table is built dynamically while agent explores environment through a behaviour policy and it suits perfectly with dynamic environments.

2.3.2 Action selection

Agent's behaviour is represented by the selection of an action from an action set in function of the given state. It consists in a greedy choice applied to q-table entries: fixing the current state, the pair state-action with highest q-value represents the best action.

The greedy method allows the agent to exploit the acquired knowledge to behave in the best possible way, on the other hand the agent needs to explore the state-action space in order to find reward and values that else would never be considered. There are several ways to perform the exploration of the state-action space, the most common is usually effective: it consists in a periodic completely random sample of the action from a state.

Balancing the *exploration* and *exploitation* strategies is really important for a learning algorithm of this kind. The most common method is *ϵ-greedy* but many others exist, usually more complex, for example *Upper Confidence Bound (UCB)* [3], *Softmax* [35] *Thomson Sampling* [46]

In ϵ-greedy the agent has a fixed $\epsilon \in [0, 1]$ probability to take a random action, instead

performing the greedy choice exploiting the q-value.

$$\pi(s_t) = \begin{cases} \text{random(a} \in A) & \text{with probability } \epsilon \\ \underset{a}{\text{argmax}}\, Q(s_{t+1}, a) & \text{with probability 1 - } \epsilon \end{cases} \tag{2.4}$$

The technique grantees a more effective exploration but can lead to a state with high penalty and affect negatively all the pathway taken already.

Softmax improves ϵ-*greedy*'s issue using action-selection strategies which do not assign the same uniform probability to each actions, but a probability distribution according to their q-values for the current state. The probability distribution estimation of the actions with respect to states is realised through *Boltzmann distribution*.

$$\pi(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_b e^{\frac{Q(s,b)}{\tau}}} \tag{2.5}$$

where $\tau \in \mathbb{R}^+$ is a parameter called *temperature*, it balances exploration and exploitation. High temperatures promote exploration, since all actions have almost the same probability, while low τ promotes greedy action selection. For further detail such as advantages and disadvantages of both methods, please refer to [40].

Some derivatives from ϵ-*greedy* decrease the probability ϵ, to select random actions, as time step is increasing. At the beginning algorithm concedes more exploration then exploitation since the learning agent does not have any knowledge base to exploit, then gradually the trend is inverted, reaching asymptotically a minimum probability of take a random action (which is always available).

A more sophisticated technique that tries to overcome the drawbacks of the previous methods is ϵ-*greedy VDBE-Boltzmann* [47], the underlying idea is to automate the state-dependent exploration probability in dependence of the value-function. It is possible to achieve an higher level of exploration when it is uncertain, such as at the beginning of the learning process or when a new state has been visited. On the other hand, the exploration rate decreases when agent has enough knowledge on the situation.

Efficiency of Q-learning has been proved by several researcher, indeed this method always converges with to an optimal policy [25, 48, 49] thanks to its look-up table, but

unfortunately the number of (s, a) pairs stored may cause issues.
Real world scenarios are often complex and, given the dynamic nature of the environment, the reachable states are innumerable, it becomes unrealistic to manage these tables, either in terms of memory space used to save the state-action pairs either in term of time required to achieve a satisfactory exploration among the states.
Problems such as chess or backgammon can approximately have 10^{20} and 10^{40} states [34], and for each of these states are associated several actions, it is easy to see that the problem cannot be managed easily though this table. To deal with more complex problems, a *function approximation* should be used.

2.3.3 Approximate Q-Learning

Approximate Q-learning is a different version in which the q-table is replaced by a value function approximation, in that way it is possible to apply the algorithm to larger problems, even when the state space is continuous.
With the use of approximation function the value returned might not be the same as the one returned by the q-table. Function approximations allow to deal with very large spaces but the principal benefit is the compression achieved by a function approximator. This compression allows the learning agent to generalise from state it has visited to states it has not visited [34].
Is possible to distinguish two kind of function approximations, **linear** and **non-linear** [21]:

- Linear function approximation is composed by two mainly component,
  - **feature-extraction function** which maps a state in a feature array;
  - **linear function** which is a parametric function defined over the feature array.

  Linear function approximators are easier to implement and are more simpler than a non-linear. Unluckily these linear methods have a main drawback: the request of domain knowledge.

- Non-linear function approximators have the same feature arrays of the linear functions as input but they may approximate an unknown function with more accuracy than a linear method. In some lucky cases it is possible to avoid the use of feature-extraction function using directly the state variables as inputs. The main drawback is that it guarantees less convergence then the linear methods.

There are many function approximators, both linear and non-linear, such as linear combinations of features, nearest neighbour, decision trees, Fourier and Wavelet bases and so on.

In approximate Q-learning with linear function approximation, we define a set of features vector where the state-action pairs are mapped over it. There is a feature function Φ defined as:

$$\Phi : (s, a) \rightarrow F_{(s,a)} =< f_1(s, a), f_2(s, a), \ldots, f_n(s, a) > \tag{2.6}$$

Approximation function takes in input feature vector $F_{(s,a)}$ to calculate the approximated Q-value that will be returned as output. Each feature is weighed through a weight w_i as is shown in the simple linear q-value approximation function 2.7.

$$\hat{Q}(s, a) = \sum_{i=1}^{n} w_i f_i(s, a) \tag{2.7}$$

As was done in Q-learning with the q-values, either something, w_i referring to equation 2.7, have to been updated during exploration: at the beginning the weights are set up to a value, usually 0, and then at each step a correction Δ is computed through equation 2.8 and the value replaced by the following equation:

$$\begin{aligned} \Delta &= r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \\ w_i &= w_i + \alpha \cdot \Delta \cdot f_i(s_t, a_t) \end{aligned} \tag{2.8}$$

Main advantages of this methods could be summarised as strong generalisation of the state which leads to a huge reduction of memory usage compared to tabular form of Q-learning and it gives to the algorithm the capacity to handle continuous spaces. On the other side, approximate Q-learning with linear function inherits disadvantages of this family methods, such as the request of domain knowledge to perform the feature

selection, moreover in some cases is known to diverge [7, 19, 23]. Some works in literature [18, 33] have demonstrated the convergence of approximate Q-learning to regions instead of singular points.

2.4 Deep Reinforcement Learning

Similar to what was done in 2.3.3, aim of this section is to discuss non-linear approximation functions for more complex scenarios, in particular using Artificial Neural Networks (ANN) as function approximators.
Deep Reinforcement Learning (DRL) is the use of ANN as function approximators in RL.
Deep is a keyword, it derived from Deep Neural Networks (DNN), substantially DNN are neural networks with several layer between input and output, those layer are called *hidden layer* [12].
Artificial neural networks have been widely used as function approximators as shown in [5, 22, 39, 43, 45], nice empirical result have been achieved in several fields, for example [43, 44, 45] by Tesauro demonstrated ANNs as function approximators in Backgammon, or in robotics from Coulom with his PhD thesis [11]. Despite this, ANNs became very popular in these late years, when a research team of *DeepMind* demonstrated excellent results in their article *Playing Atari with Deep Reinforcement Learning* [31]. The team has challenged the world champion of the board game *Go* through their project *AlphaGo* and in 2017 they manage to defeat him. Other important achievements were the great performances of the team's project *AlphaStar* in one of the most complex strategic video game of the history, Starcraft II. From these projects they published many articles including [37] and [36].

2.4.1 Deep Q-Learning

Among the DRL techniques there is one which approximates the presented method, Q-Learning, using ANNs. It is a nonlinear representation that maps both state and

action onto a value, its training process is generalised briefly by [26] in these main steps:

1. Initialise $Q(s, a)$ with some initial approximation.
2. Obtain the transition tuple (s, a, r, s') interacting with the environment;
3. Calculate loss using loss functions like *Mean Square Error (MSE)* or *Huber loss*;
4. Update $Q(s, a)$ using an optimiser such as *Stochastic Gradient Descent (SGD)* or *Adam* minimising the loss with respect to the model parameters;
5. Repeat from step 2 until converged.

The model optimisation produces new q-values which are used to update the network weights, in this way the output will be a trained DNN (called DQN) capable of estimating a value to each action providing the agent's state as input, the highest value represents the best action. Other versions of deep q-learning algorithm exist which exploit some strategies to improve the effectiveness brought in an array of steps.
The list is "generalised" since it does not consider important issues of a good RL-agent training like the exploration-exploitation balance strategy introduced in section 2.3.2 or how to exploit agent's recent knowledge and experience to speed up the learning process that will be introduced in 2.4.2.

2.4.2 Experience Replay Buffer

In DQN, we rarely deal with immediate experience samples, because of their temporal correlation, which leads to instability in the learning process. Experience Replay Buffer is a technique for generating uncorrelated data for online training of deep reinforcement learning systems [51]. It has proven to be very effective for convergence of such systems [2, 28, 31]. The system first stores all the data relative to its experiences e_t in a large table rather than executing Q-learning algorithm on (s, a) pairs.
Experience is defined as a tuple of 4 elements,

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

These tuples are then used during training that consists on taking random samples from this table instead of the most recent transition. These experiences are randomly spread in batches which are finally fed into the network. This procedure prevents the reaching of the network to a local minimum, because each input sample is not correlated with the one before. Articles like [51] underline how the replay buffer's size can lead

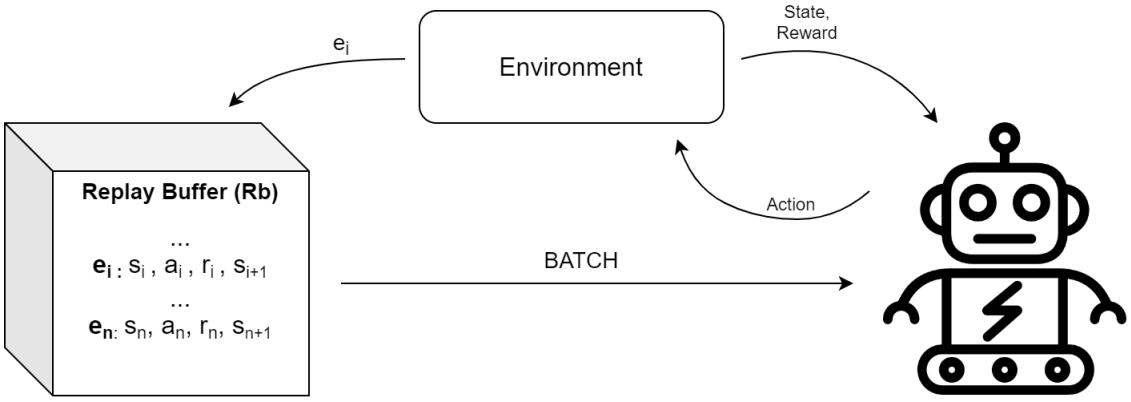


Figure 2.4: Experience replay model.

to convergence issues and proposes an alternative solution that overcomes encountered problems. This solution can remedy the negative influence of a large replay buffer with $\mathcal{O}(1)$ computation, it is named *Combined Experience Replay (CER)* and it suggests to add the *latest transition* to the random samples fed into the agent.

2.5 Collaborative Agents

Human beings encounter streams of learning tasks in their lives. Humans does not just learn concepts or motor skills, they learn how to generalise as well to learn to solve tasks. Humans own the intelligence to generalise efficiently often using a very few examples and therefore learn a new task better and faster by utilising the knowledge retained from solving similar tasks.

In multi-agent systems the present agents could work to reach the same objective, for instance in *Predator-Prey* environment (which is considered often as a benchmark environment because of its ease and scalability), used in [41], there are agents in a

gridworld (predators) that try to catch the preys and these run away from the firsts. In this case the predators can try to collaborate or cooperate to make the preys' task harder rather than working alone to achieve the objective. Another example is *Robot Soccer* in [38], where a soccer match is played by two robotic teams. In this case the collaboration is necessary since the team members have a common objective, to score goals.

In the years, researchers have realised new representations of multi-agent Markov Decision Processes with collaborative agents: in [20] each agent selects an individual action in a particular state. Based on the resulting joint action the system transitions to a new state and the agents receive an individual reward, then the global reward is made by the sum of the all the individuals, while in [4] and in [6] all agents observe the global reward.

In a collaborative MDP, agent's goal is still to optimise the global reward, but the individually received rewards allow for solution techniques that take advantage of the problem structure.

There are many contemporary approaches to speed up "vanilla" RL methods. Transfer learning (TL) is one such approach.

2.5.1 Transfer Learning

Transfer learning is an *umbrella* term used when knowledge is generalised not just across a single distribution of input data but when knowledge can be generalised across data drawn from multiple distributions or even different domains, potentially reducing the amount of data required to learn a new task. There are many cases in which a classifier learns the classification of a certain set of objects and then, thanks to fine-tuning techniques, it is trained to classifier other classes of objects, it is also called **domain adaptation**.

In TL we assume that many of the factors that explain the variations in a distribution P_1 are relevant to the variations that need to be captured for learning the distribution P_2. [17]. This variations are not always easy to recognise, when they are detected, are exploitable in these terms.

Transfers happen in human scenarios as well: according to [9], world chess champion Mikhail Tal, once told how his chess game was inspired from watching ice hockey. He noticed some habits that helped the hockey players to score when an opportunity came. Following the hockey game, he decided to try transferring the idea to his chess game changing his strategy inspired by the hockey players.

As shown by the figure 2.5, in the standard learning process (above), the learning algorithm gets as input some form of knowledge about the task and returns a solution. In the transfer setting (below), a transfer phase first takes as input the knowledge retained from a set of source tasks and returns a new knowledge which is used as input for the learning algorithm. The dashed line represents the possibility to define a continual process where the experience obtained from solving a task is then reused in solving new tasks [27].

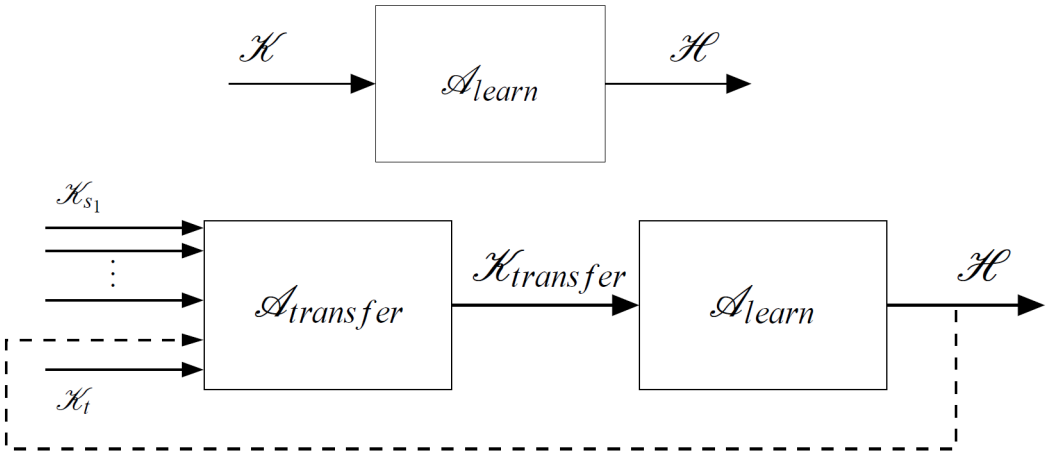


Figure 2.5: Comparison among learning in a non-transfer scenario and learning in a transfer scenario [27]

The same paper, [27], has introduced a distinction of three different categories of transfer settings, they depend strictly on the tasks and domains:

- **Transfer from source task to target task with fixed domain**

  Domain of a task is determined by its state-action space, while the specific structure and goal of the task are defined by the dynamics and reward. The transfer algorithm might or might not have access to the target task at transfer time. If no target knowledge is available, some of the transfer algorithms perform a

shallow transfer of the knowledge collected in the source task (e.g., the policy) and directly use it in the target task. On the other hand, when some target knowledge is available at transfer time, it is used to adapt the source knowledge to the target task.

- **Transfer across tasks with fixed domain**

  In this case, tasks share the same domain and the transfer algorithm takes as input the knowledge collected from a set of source tasks and use it to improve the performance in the target task. The objective is usually to generalise over the tasks according to the related probability distribution.

- **Transfer across tasks with different domains**

  Tasks have a different domain, that is they might have different state-action variables, in terms of numbers and ranges. Most of the transfer approaches focus on how to define a mapping between the source state-action variables and the target variables so as to obtain an effective transfer of knowledge.

Object to Transfer

TL is based on the notion that if the tasks are drawn from the same distribution then an algorithm which achieves a good performance on a set number of source tasks will be able to generalise across target tasks from the same distribution. The TL algorithm is the result of a transfer of knowledge and a learning phase and can be defined as a mapping.

$$A_{learn} : K_{transfer} \times K_t \rightarrow H \tag{2.9}$$

In the above equation, $K_{transfer}$ represents the final knowledge transferred to the target task and is knowledge gained from one or more source task(s) while K_t is the knowledge gained (if any available) from the target task itself. H is the space of hypotheses that are returned by the algorithm [27].

The definition of transferred knowledge and the specific transfer process are the main aspects characterising a transfer learning algorithm. According to [27] it contains the *instances* collected from the environment, the *representation* of the solution and the

parameters of the algorithm itself. Form the definition it is possible to classify the categories of possible knowledge transfer approaches:

- **Instance transfer** consists in collecting samples gained from source tasks and using them in learning the target task. They are collected from the direct interaction with the MDP.
- **Representation transfer** involves the transfer of the task or solution representation such as neural networks, state aggregation or a set of basis function which are used to approximate the value function.
- **Parameter transfer** consists in approaches which change and adapt the parameters used in an algorithm according to the source tasks. These parameters can be a set which characterises the used RL algorithm. For instance in Q-Learning the q-table is initialised with arbitrary values and it is updated using a gradient-descent rule with a learning rate α. The initial values and the learning rate define the set of input parameters used by the algorithm.

2.5.2 Transfer Evaluation Metrics

In ML problems such as classification and regression, the performance of the transfer algorithm can be evaluated in terms of prediction error. But in RL many possible measures can be used to judge the effectiveness of the transfer algorithm [27]. Transfer learning algorithms in RL are hence evaluated based on the different goals of transfer and the situations where the transfer of knowledge will be beneficial [42]. In other words, the metrics chosen to judge the performance of an algorithm depend on how the learning performance is measured.
The metrics introduced to help to ascertain the success of a transfer learning algorithm by [42] and [27] are shown in Figure 2.6. They consist in:

- **Jumpstart** is the improvement of the initial performance of a target task due to knowledge transfer from a source task. The learning process usually starts from either a random or an arbitrary hypothesis: the improvement of the initial

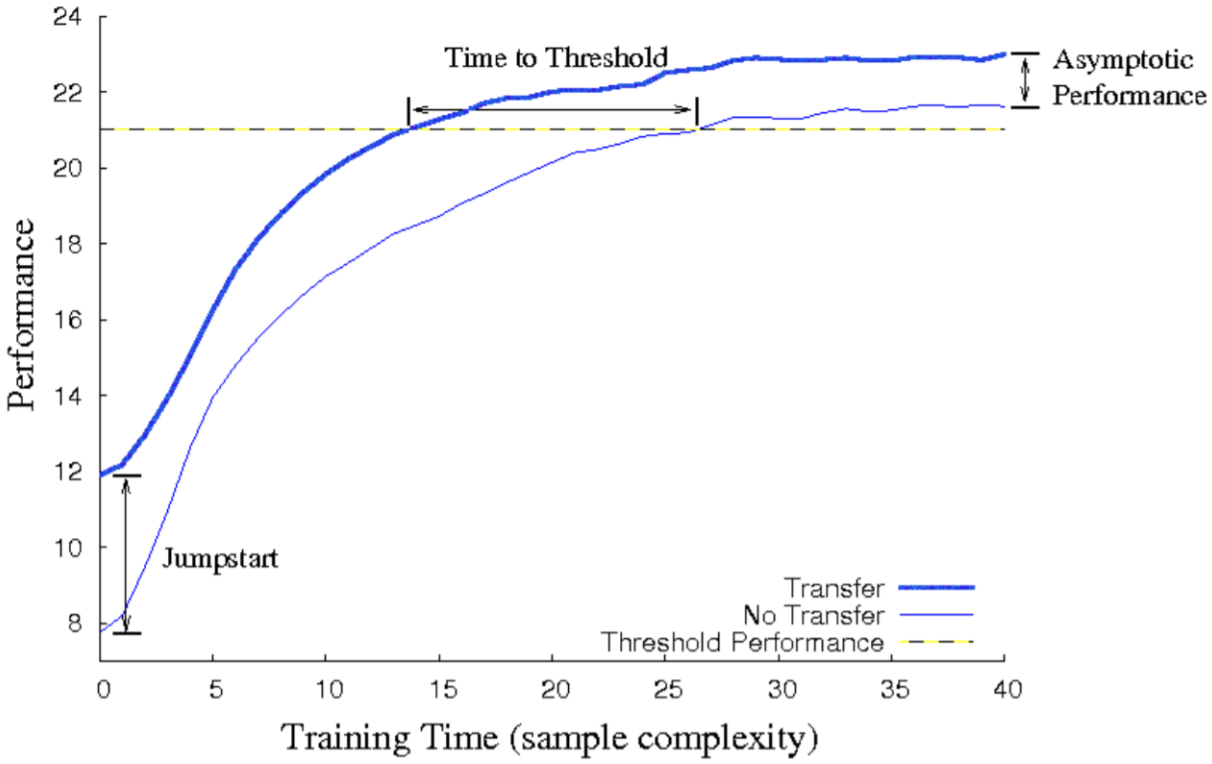


Figure 2.6: TL evaluation metrics [42]

performance can be obtained by initialising the learning algorithm to the optimal policy of the source task, but this may lead to worsen the learning speed [27].

- **Asymptotic Performance** is the final improved performance of an agent in a target task after learning. It compares the final performance of the learning agent with and without transfer.

- An objective is about the reduction in the amount of the experience needed to learn the solution of the task at hand. For this reason the metric **Time to Threshold** consists in the time taken by the agent to reach a fixed threshold performance and can represents one major speed improvement.

- **Total Reward** is basically the final total reward accumulated by the agent after the learning process is completed. Improving initial performance and achieving a faster learning rate will help agents accumulate more on-line reward. This metric is most appropriate for tasks that have a well-defined duration since if

enough samples are provided to agents, a learning method which achieves a high performance relatively quickly will have less total reward than a learning method which learns very slowly but eventually plateaus at a slightly higher performance level [42]

- **Transfer Ratio** is defined as following:

$$r = \frac{area_T - area_{NT}}{area_{NT}} \tag{2.10}$$

where $area_T$ is the area under the curve with transfer and $area_{NT}$ is the one with no transfer. The area under the learning curve represents the total reward accrued by the learning agent: more the area under the curve, higher reward will be accrued by the learning algorithm. The ratio r is not scale invariant.

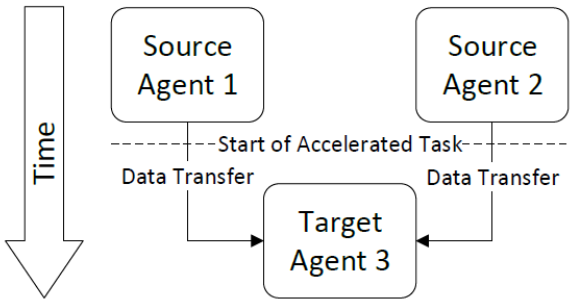
The last three metrics can be summarised in **learning speed** since they define the speed-up in a TL outcome. This concept considers these three metrics as partitions of a single whole.

2.5.3 Parallel Transfer Learning

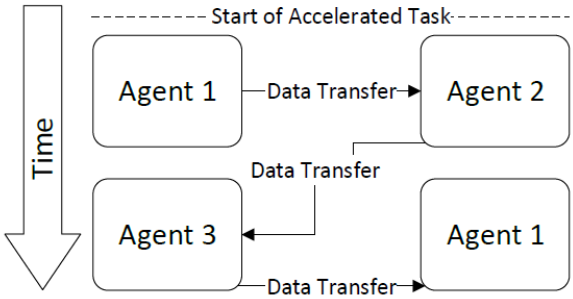
TL definition is open, it involves every kind of sharing among the agents. Many transfer approach have been developed and are present in literature in papers described in detail: according to [41] it is possible to define a TL implementation "Parallel Transfer Learning" (PTL) if the transfers are online, so the sharing can happen any time and the agents are learning at the same time, usually in TL implementation the source learning process has to be completed to perform the transfer to the target learning process as shown in the figure 2.7. For this reason the system is not properly closed and it allows other agents to join it anytime.

An agent is either sending or receiving an information, it cannot be certain of its accuracy, so PTL uses *source-driven transfer*, agents can transfer the information when they deem it representative. These information can be anything provides a kind of representation of the knowledge or is related directly to it.

In [41] the transfer technique is presented using Q-Learning, storing the q-values in a



(a) Transfer Learning (TL)



(b) Parallel Transfer Learning (PTL)

Figure 2.7: Difference between a standard TL and PTL process according to [41], in the second one shared data are exchanged during the learning process.

q-table, there is not an approximation system but a discretization of the state space is adopted. In the paper they wonder what, when and how to transfer, they tried to answer the questions studying a few transfer strategies in benchmark environments, it is possible to share the q-values among the agents since they capture agent's experiences gained by interacting with the environment, precisely $Q(S, A)$ represents a long-term value of taking the action A in the state S.

A PTL implementation based on q-table has some fixed components:

- **Set of agents** $\{A|A_1, ..., A_Z\}$ where Z is the number of agents in the system.
- **Set of neighbours** $\{N^a|N^a_1, ..., N^a_Y, a \in A\}$ where Y is the number of neighbour of the agent A in the system, the neighbours can communicate with the agent A.
- **Set of local learning processes** $\{LLP^a|LLP^a_1, ..., LLP^a_X, a \in A\}$ where X is the number objectives implemented by the agent A.

- **Transfer Size (TS)** is a parameter that determines how many states are shared in a single transfer of the process LLP^a.
- **Confidence Visit (CV)** determines the visit threshold to mark a state eligible to be shared.
- **Selection Method (SM)** is a function which filters the data and underlines which share in a LLP^a, a set of methods are presented later.
- **Merge Method (MM)** is a function that elaborates the local and the received q-values and performs a weighted merge in a LLP^a.

PTL increases the use of early acquired knowledge by enabling the online sharing between RL agents, and therefore aims to speed up the overall learning process. In the above-mentioned paper are explored five different selection methods, two of them rely on the number of state visits as *confidence*, the other three on the related q-values. In particular:

- **Most Visited States** shares the most visited states.
- **Visit Threshold**, the states are selected choosing from a subset that satisfies the visit-threshold condition.

The following three methods the confidence depends on the q-values:

- **Best/Worst States** shares the states with highest or lowest q-values.
- **Converged States** considers the difference of Q-value as confidence for a given state, so the states with the minimum Q-value difference between step t and $t-1$ are shared.
- **Greatest Change** is the opposite of the previous method, the greatest Q-value variation determines the shared states.

2.5.4 Distributed W-Learning

According to [13], Distributed W-Learning (DWL) is an RL-based algorithm for multi-policy optimisation in large-scale decentralised agent-based systems. It enables agents to engage in different levels of cooperation with each other in order to optimise the performance of the system towards its goals, while respecting the priority of these goals. It is a *distribute* technique since the required actions are performed on an agent level, removing the need for central control or global knowledge.
The collaboration mechanism is based on W-values to learn how agents actions affect their neighbours' policies: at each time step, each agent, each local and neighbour policy makes an action suggestion (based on learnt q-values), with an associated weight based on the state in which each policy is currently (i.e., a W-value).
The algorithm is introduced in [14] and is based on a **set of agents** and of **neighbours** as well as in PTL, then the other main components are:

- **Set of policies**, $LP_i = \{P_{i1}, ..., P_{ip}\}$ is deployed at each A_i. It is possible to refer to these policies as *local policies* of A_i.
- **Set of remote policies**, $RP_i = \{RP_{ij1}, ..., RP_{ijr}\}$ whose goal is to contribute to the implementation of each local policy LP_{jk} deployed at each $A_j \in N_i$
- **Collaboration coefficient C** is used to scale the importance of action suggestions by remote policies. C can be predefined to the same value for all agents in A, or can be learnt individually by each $A_i \in A$.

Each policy, both local and remote, is implemented as a combination of a Q-learning and a W-learning process. An agent's current action is denoted as a_i and its previous action a_{i-1}. A policy's current state is denoted as s_i and its previous state s_{i-1}.
The algorithm starts initialising Q-learning and W-learning processes, then each DWL agent exchanges state-space representations for each of its policies with each of its immediate neighbours.
The main point of the algorithm is the action selection. As represented in figure 2.8, each agent, using Q-learning and W-learning, learns q-values for its local-state/local-

action pairs and W-values for its local states; based on the reward, each policy updates its local q-values and W-values. Each agent learns q-values for its remote-policy-state/local-action pairs and W-values for its remote policies' states. To do this, an agent needs to receive information about its neighbours' current states for each of its policies and the rewards that they have received. Based on the rewards received, an agent updates q-values and W-values for its remote policies. The action that is executed by an agent is the one with the highest W-value (W_{win}), after remote policy W-values have been scaled by the collaboration coefficient C:

$$W_{win} = max(W_{il}, C \times W_{ijk}) \tag{2.11}$$

Where W_{il} are W-values associated with actions nominated by local policies of A_i and W_{ijk} are W-values associated with actions nominated by remote policies of the agent.

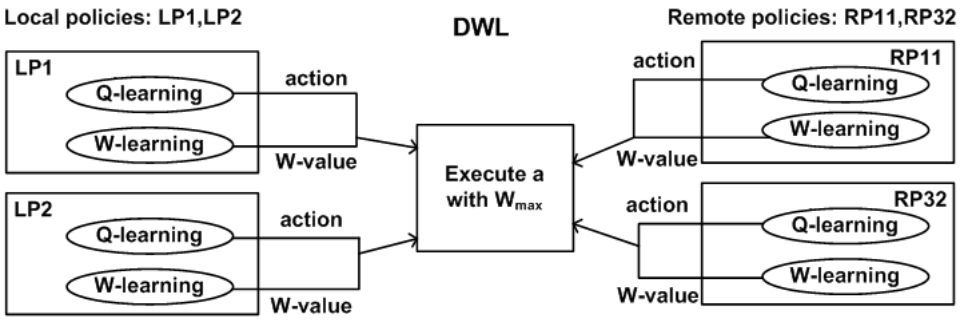


Figure 2.8: DWL action nomination

Chapter 3

Design

3.1 Introduction

The previous chapters addressed the need for transfer learning in RL to reach the agent's learning convergence and to discover new state-trajectories and policies more quickly. In section 2.5 the various methods of transfer learning have been discussed. The framework in this dissertation is inspired by one of the said transfer approach, Parallel Transfer Learning (PTL) which introduces multiple novel transfer mechanisms to achieve learning improvements. The new framework uses DQN to implement Q-Learning instead of q-table and is based on different mechanisms to get learning improvements.

The framework is designed to study how information perceived and learnt by an agent can affect another agent, in particular during the learning process the agents interact with the environment observing new states (s_t) and performing actions according to the action selection algorithm (a_t). These data are part of agent's *transition*, tuple (s_t, a_t, r_t, s_{t+1}), a set of these transitions represents agent's **experience**. In deep q-learning an agent updates its network's weights periodically using a set of transitions (batch) causing possible changes in the action selection method. For this reason it is possible to state that experience causes perspective changes in the agent affecting its action choices and consequently its behaviour.

Each agent acts differently in its environment though, creating different experiences

which usually are the main cause of the quick and correct learning of the best states trajectory to solve the task. The idea is to gather an improvement in learning speed and performances promoting the transfer of subsets of experiences among the agents in order to let them learn about those partial experiences which might not be well-known.

The chapter begins with a detailed description of Online Experience Sharing algorithm (OES) concerning the following topics:

1. First the definition of sending a receiving agents to clarify how agents are involved in transfers (discussed in section 3.2);
2. The introduced transfer mode and transfer buffer's role are concretely explained in section 3.3 with a particular attention to transfer frequencies and transferred data sizes and their variabilities;
3. Successively, in 3.4, the transfer selection methods are defined with a detailed description of the related agent models;
4. The last section (3.5) delineates the transfer hyperparameters tuning through Bayesian optimisation.

3.2 Sending and Receiving Agents

The method is developed specifically on deep q-learning implantation with the support of a replay buffer introduced in section 2.4.2: the "transferred experience" is generated by the sending agent and is pushed in new buffers of the receiving agent called **Transfer Buffer** (Tb), which is used as source of experience like the usage of replay buffer in a deep q-learning implementation.
The algorithm design based on deep neural networks starts with the actors definition

- **Set of agents** $\{\phi|\phi_1, ..., \phi_N, a \in \Phi\}$ where N is the number of agents in the system.

These agents can belong to the same environment in a multi-agent system or just being the protagonist of more single-agent systems. In any case the TL interactions are defined in the same way.
The common TL concept of agent's neighbourhood and neighbours is ignored since that definition implies the establishment of a bidirectional communication between to agents. This policy is replaced to a single-direction communication in order to have a specific distinction between agents that provide their information and agents that receive those information. The two agents groups can be considered as teachers and student but in this document they are named **senders** and **receivers**:

- **senders and receivers**: agents set S and R are defined as *partitions* of the set Φ, so $\bigcup_{I}^{\{S,R\}} I = \Phi$ and $S \cap R = \emptyset$. For this reason every agent in the system is considered either a sender or a receiver;
- **Set of agents senders** $\{\phi^s | \phi^s_1, ..., \phi^s_M, \phi^s \in S\}$ where M is the number of sender agents in the system;
- **Set of agents receivers** $\{\phi^r | \phi^r_1, ..., \phi^r_P, \phi^r \in R\}$ where P is the number of receiver agents in the system

3.3 Two Steps Experience Selection

As previously defined, a single transfer consists in a sending agent pushing data to a different receiving agent. Being these two agents separate entities with their own prospective (concretely in their experiences, networks and state confidence structures), it is possible to apply two different selection of the transferred data in order to enhance the transfer effectiveness. They consist in encouraging the sender to select states which have been explored many times which are supposed to be well known and the receiver to filter the transferred data choosing the states that are less visited and known. This double selection drives the sender to share states of its best trajectory, suggesting an agent which has not find it yet.
The transfer happens in two different phases, the sender pushes the selected data to the

receiver's transfer buffer, that can be consider as the receiver's intermediate transfer storage. This buffer is part of the receiver's structure and is thought in order to receive from anyone and being used as source just from the receiver itself. In this way the system is more versatile, it would be possible to implement easily a transfer system where more senders push data to the same receiver or just one sender pushes data to more receivers. Transfer buffer is a finite collection of transitions related to the receiving agent ϕ^r, $(s_t, a_t, r_t, s_{t+1}) \in Tb^r$ where

- $s_t \in S$ is a state of the agent at t step and S is the state space;
- $a_t \in A$ is the action at t step and A is the action space;
- $r_t \in \mathbb{Z}^+$ is the reward earned by the agent performing the action a_t from the state s_t;
- $s_{t+1} \in S$ is the next state performing the action a_t from the state s_t.

3.3.1 Transfer frequencies

- Sender's **Transfer Interval (TI)** $\in [1, \infty)$, $TI \in \mathbb{Z}^+$ is a hyperparameter which represents a temporal period in steps. Senders transfer data to the receiver's transfer buffer periodically, every TI steps. This parameter is decisive in a parallel implementation of the algorithm since it determines how often the communication between the processes (if a process handles a single agent) are established and used. There is the possibility exchange information intensively setting the TI parameter to a low value;
- Receivers pull data from the transfer buffer to build the DQN batch **every step**. The transfer effectiveness, at the receiver side, is conditioned by the amount of used transferred data, determined by $\hat{\theta}$ function which is introduced in the next section.

3.3.2 Transfer sizes

- Sender's **Transfer Size (TS)**: $\{T^s | T_1^s, ..., T_{TS}^s, TS \in \mathbb{Z}^+\}$ is the information set transferred by the sender to the receiver's transfer buffer. The amount of data is defined by the hyperparameter TS

- Receiver's **Transfer-Batch Size (TBS)**: $\{T^r | T_1^r, ..., T_{TBS}^r, TBS \in \mathbb{Z}^+\}$ is the set of transferred data used by the receiver to train its deep q-networks. The quantity is defined by the parameter TBS which is calculated in functions of the episode where the agent is acting. The size is calculated with the following procedure.

  The introduction of a new function θ is necessary to explain the variability of the TBS parameter, in equation 3.1 the function is introduced in its general form.

$$\begin{aligned} \theta &: (\mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+) \to \mathbb{R}^+ \\ \theta &: (\theta_{max} \times \theta_{min} \times \theta_{decay} \times \theta_{apex} \times ep) \to \overline{TBS} \\ \theta &= \begin{cases} \frac{\theta_{max} - \theta_{min}}{\theta_{apex}^2} \cdot ep^2 + \theta_{min} & ep < \theta_{apex} \\ \theta_{min} + (\theta_{max} - \theta_{min}) \cdot e^{-\frac{ep - \theta apex}{\theta_{decay}}} & \text{else} \end{cases} \end{aligned} \tag{3.1}$$

  The function θ is defined in the following variables' space:

  - $\theta_{max} \in [0, 1]$
  - $\theta_{min} \in [0, 1]$ where $\theta_{min} < \theta_{max}$
  - $\theta_{decay} \in (0, \infty)$
  - $\theta_{apex} \in (0, n\_episodes]$
  - $ep \in [TI, n\_episodes]$

  and the codomain is in $\mathbb{R}^+$ in the range $[0, 1]$, where BS is the DQN's batch size hyperparameter. In the first TI episodes the transfer buffer is empty so the function is never used. The outcome, value $\overline{TBS}$, represents the percent of the batch picked from the transfer buffer.

  The function is instantiated specifying the first four parameters and is used as

$\hat{\theta}$ by the agent as the equation 3.2, keeping as independent variable only the episode.

$$\hat{\theta} : \mathbb{Z}^+ \to \mathbb{R}^+ \tag{3.2}$$

The function graphically consists in a square function between the values 0 and $\theta_{apex} \in \mathbb{Z}^+$ which increases from $\hat{\theta}(TI) = \theta_{min}$ to $\hat{\theta}(\theta_{apex}) = \theta_{max}$, then it decreases exponentially asymptotically to $\hat{\theta} = \theta_{min}$ where the decreasing speed is determined by the parameter θ_{decay}. An analytic version of the function is shown in figure 3.1

The outcome $\overline{TBS}$ is multiplied to the batch size and rounded to get the right transfer batch size as in equation 3.3:

$$TBS = BS \cdot \theta(ep) \tag{3.3}$$

An average usage of six asynchronous agents training in cart pole environments is shown in figure 3.2 where 3 senders (blue line) transferred data to 3 receivers (red line) and the initialisation parameters are $< 0.5,\ 0.03,\ 125,\ 40 >$ with a DQN batch size of 64 transitions and 20 of TI, the shown trends are the average of the processes at i-step. The graph shows the amount of data pulled from the transfer buffer (TBS) to build the DQN batch, so the senders never use these data; the receivers start to use the transferred data after TI steps, because in the first TI steps the transfer buffer is empty, successively they reach the 50% percent of the batch size ($\theta_{max} = 0.05$) at the 40th episode ($\theta_{apex} = 40$) and then decrease asymptotically to 0.03 ($\theta_{min} = 0.03$).

When the transfer happens and the size of the transferred data have been discussed in the previous sections. In the next sections a set of selection algorithms will be introduced divided in two classes: **Sender Selection Method (SSM)** and **Receiver Selection Method (RSM)**.

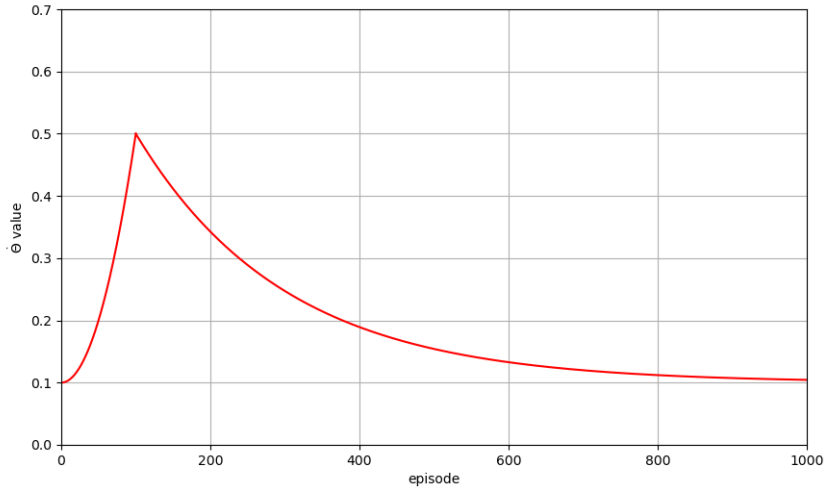


Figure 3.1: Analytic graph of θ function, which consists in a square function and a negative exponential. This plot depends on the given example set of initial parameters: $< 0.5,\ 0.1,\ 200,\ 100 >$

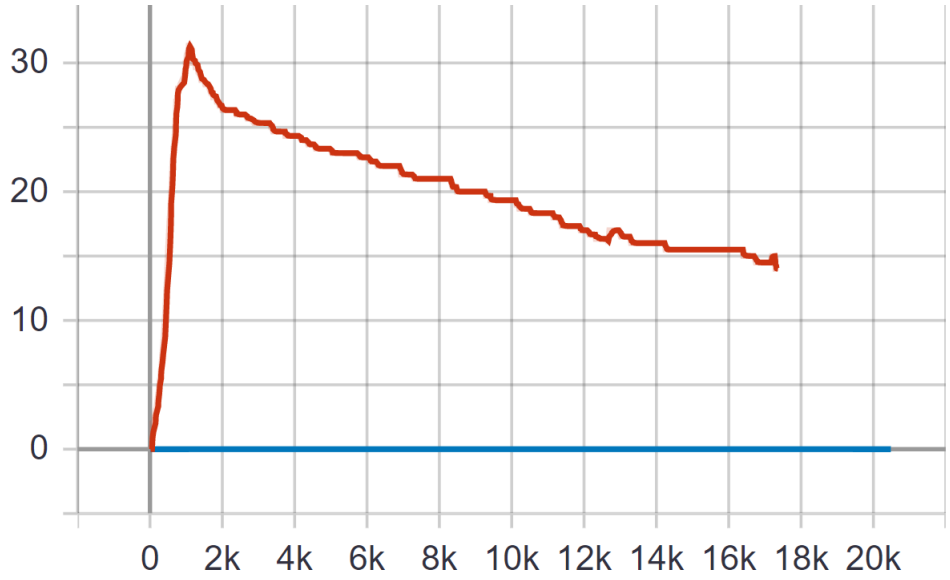


Figure 3.2: Amount of transferred data included in DQN batch at step variation of 6 asynchronous agents

3.4 Selection Methods

The introduced methods include trivial operations like random selection or most recent selection, but they consider more complex algorithms as well like operations on visiting tables to keep trace of the most and least visited areas of the state-space in section

3.4.1, or exploiting other neural networks to calculate the uncertainty of a found state (or state-action) in sections 3.4.2 and 3.4.3.

The general algorithm of transfer from sender (algorithm 1) performs the SSM from each sender to each related receiver's transfer buffer.

Algorithm 1: Push the selected data to the receiver's transfer buffer

Result: batch

```
// after DQL update;
if step mod TI == 0 then
    for s ∈ S do
        for r ∈ R^s do
            r.Tb.push(SSM(s.Rb, TS));
        end
    end
end
```

After the execution of these lines, if the step is a multiple of TI, the receivers' buffer is updated with new TS elements for each related sender. The general algorithm of transfer to receiver (algorithm 2) calculates, from the episode, the number of transitions sampled from the replay buffer and the number pulled from the transfer buffer. Then the batch is built sampling always randomly from the replay buffer and selecting with RSM from the transfer buffer.

Algorithm 2: Select batch from receiver's experience and transfer buffers

Data: Rb, Tb, ep

Result: batch

$\overline{TBS} = \theta(ep)$;

$TBS = BS \cdot \overline{TBS}$;

batch = Rb.sample($BS \cdot (1 - \overline{TBS})$);

$batch_T = \mathbf{RSM}(Tb, TBS)$;

batch.concat($batch_T$);

In the following sections a set of selection methods will be presented. Every section introduces a new SSM and RSM which rely on custom confidence source structures. For each confidence structure, two selection methods are introduced, for this reason it

is possible to evaluate the combination of the new selection methods with the trivial methods (that create the baseline presented in the section 5.4.1). These trivial methods do not exploit any insight:

- **Lasts** is the trivial SSM in which the sender selects the most recent transitions to transfer from its replay buffer;
- **Random** is the trivial RSM in which receiver picks randomly from its transfer buffer the transitions to use to updated its DQN.

3.4.1 State Visit Table

This first implemented state-confidence consists in storing the number of visits of a certain state: a state which has been visited more then another will be represented better by the q-networks and the chosen action will be more appropriate. The insight consists in count the state visits and to perform the selections based on this.
These first methods are inspired to the "Most Visited States" selection method of PTL (introduced in section 2.5.3): while that method works with discrete q-tables and keeps the state-visit traces with the same finite number of entries, this one works with deep q-networks and relies on a discretization of the state-space to keep trace of the visited states. Basically the state-space is divided in micro-areas, each state dimension is discretized in a number of bins set by the hyperparameter "STATE_BINS" and the entire state-space is divided in STATE_BINS $\cdot$ N_DIMS areas (where N_DIMS is the number of dimensions of the state-space). The *Visit Table* is a N_DIMS-dimensional table which keeps the trace of the total number of visit of each state. Every agent has its own instance of visit table in order to provide a first person interpretation of any state.
The proposed selection methods rely completely on the visit table, it consists in transferring the states in which the sender is more confident and to receive the states in which the receiver is less confident:

- **Most Visited** is a SSM which consists in the sending agent sampling from the visit table the most visited states to select the best action and to transfer them

to the receiver's transfer buffer. The transfer frequency and the size have been discussed in section 3.3;

- **Least Visited** is a RSM in which the receiver samples from its transfer buffer the states with the lowest number of visits.

Agents models in this specific scenario are proposed in two different representations: sender in figure 3.3 and receiver 3.4.
In VT scenario, sender and receiver are standard DQL agents defined first by their **RL component**: this abstraction includes *deep q-network* receiving the batch sampled from the *replay buffer* and producing the q-values used by *action selector* to pick the action. The agent interacts with the environment through the action and perceiving it through state and reward. The new component, **Transfer Learning Component**, in both the agent kinds consist in a *confidence source structure* which is implemented by the visit table and keeps track of the visited state for each step. Senders include the *sending selection method* which implements the designed methods, it selects data from the confidence structure and sends them to the receiver's transfer buffer; receivers have the *receiving selection method* which filters data, with the designed selection method, from the *transfer's buffer*, relying on its visit table as confidence source, and merges the standard batch with the selected data (transfer batch). The resulting batch is made by experience batch joined to transfer batch, the outcome size always respects the batch size parameter defined as configuration hyperparameter in order to use the same amount of data to update the network's weights.
A simulation implies parallel training of these two kinds of agent where the receiver updates its network using both its experience and sender's most confident experience.

Issues

As mentioned earlier, both replay buffer and transfer buffer are collections of transitions, for this reason it is not possible to transfer just the state. Once the state is selected, one of the two agents has to evaluate the best action and calculate the next state to finally build the tuple (s_t, a_t, r_t, s_{t+1}). The agent which performs the action

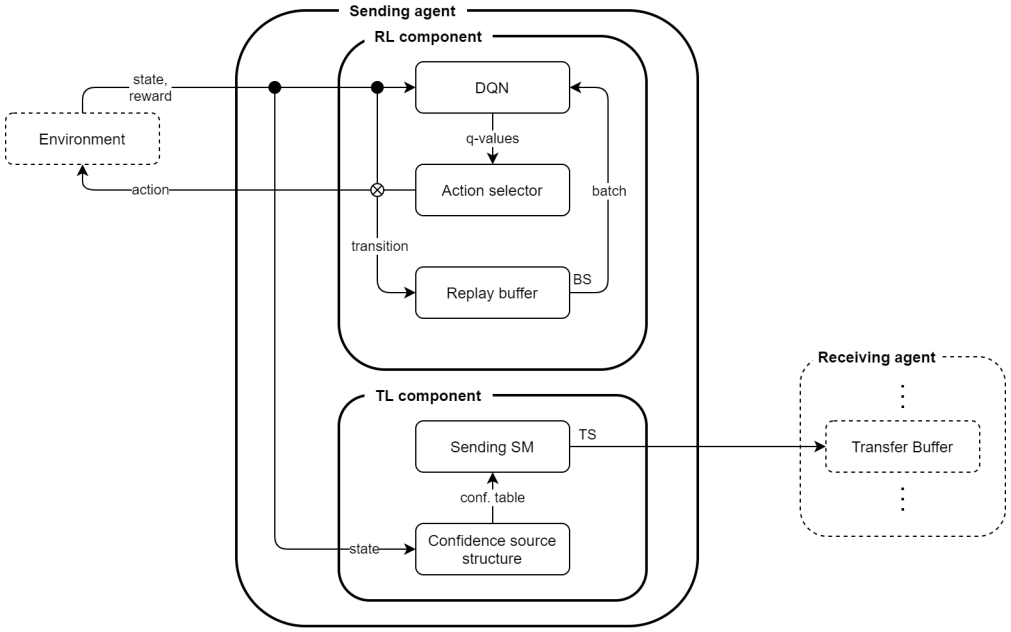


Figure 3.3: Agent sender model in "Visit Table" scenarios

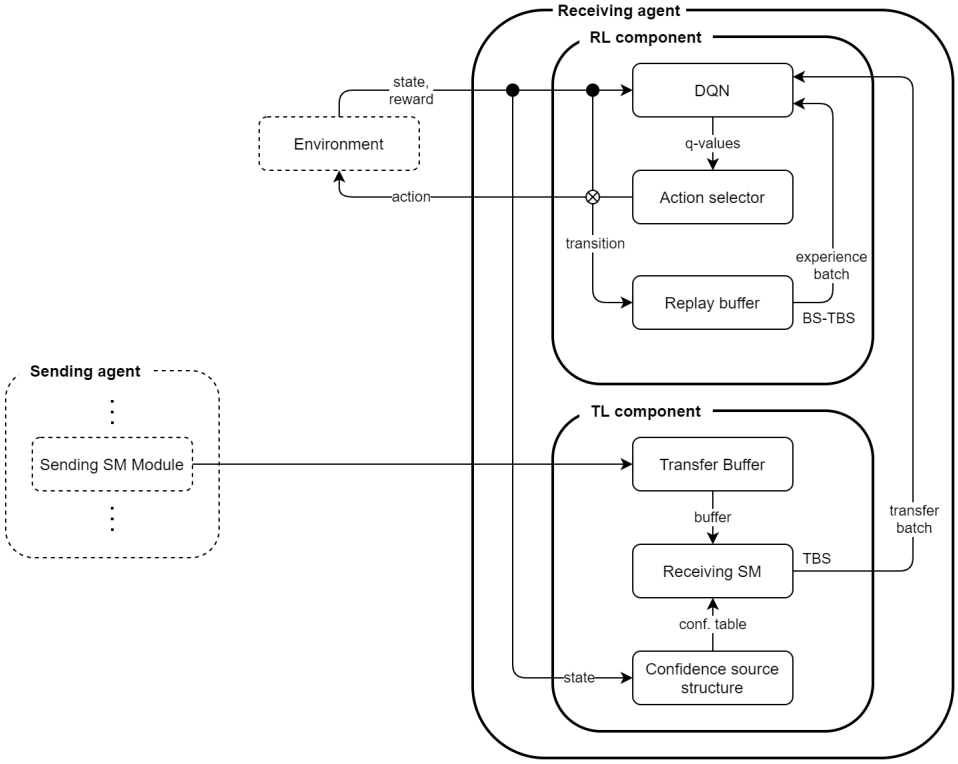


Figure 3.4: Agent receiver model in "Visit Table" and "State RND" scenarios

selection is the sender since its accuracy about the selected states is supposed to be higher then the receiver's one. Moreover, being the state-space discretized, the trans-

ferred data suffer the precision loss introduced by the dicretizer.
Another aspect is that the sender has to evaluate the next state from the state-action couple to build the transition tuple: this evaluation requires the possibility to set the state of either an extra virtual environment or to pause the execution of the current environment. A partial solution could consist in precalculating every possible state transition and store them in a structure, but this would prevent the proper functioning of the algorithm in non-stationary systems. This operation is part of the SSM so it does not happen each step but periodically every TI steps.

3.4.2 State RND

The second group of selection methods is called State RND (SRND) and includes a deep change of the confidence implementation, this is inspired by the curiosity methods which usually encourage the agent's exploration.

Random Network Distillation

A new possible representation of the state-confidence value can be the output of a Random Network Distillation algorithm (RND). It is a technique built to allow RL agents to have an intrinsic curiosity reward to improve their exploration capabilities.
RND has been introduced in [8] where they have studied the effects of the curiosity algorithm in classic Atari environments and in pure exploring situations (in the absence of any extrinsic reward). The paper suggests that the algorithm is sufficient to deal with local exploration like the consequences of short-term decisions, anyway global exploration that involves coordinated decisions over long time horizons is beyond the reach of the method. Another application of the algorithm is present in [24].
The algorithm consist in a random initialisation of two neural networks with the same input and output size, the first is called *Target* and is never updated, the second is *Predictor*. Because the target network's parameters are fixed and never trained, its feature representation for a given state will always be the same. Therefore, each state s_{t+1} is passed into both networks, and the prediction network is trained to minimise the difference mean square error between the target network's output and the predictor

network's output, as shown in figure 3.5.

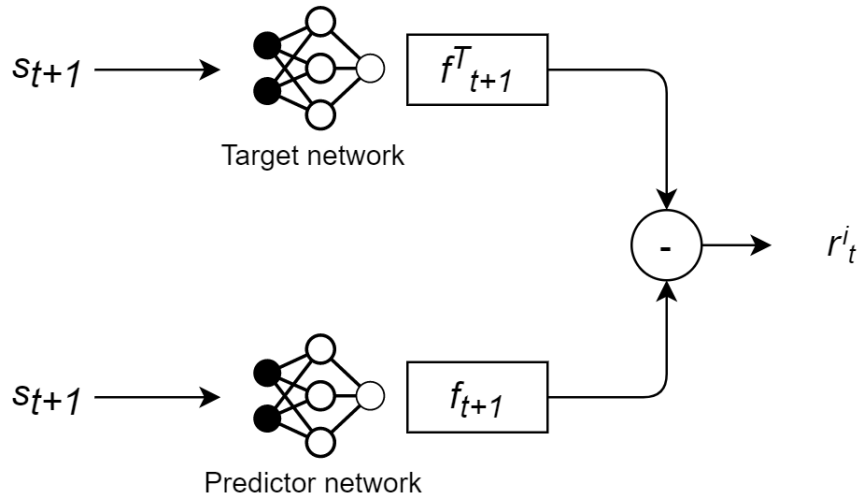


Figure 3.5: RND algorithm mechanism

The outcome r_t^i represents the uncertainty estimation of the given state s_{t+1}, its calculation is shown in the following equation (3.4).

$$r_t^i = ||f^T(s_{t+1}) - f(s_{t+1})||^2 \tag{3.4}$$

In these methods the opposite value of the calculated uncertainty, $-r_t^i$, represents the state confidence at a certain step. The predictor is updated each step using the current state in order to predict more accurate features by an often visited state then a less visited state.

The two networks are built considering the same input and output size, RND hyperparameters and in particular the output size are studied in details in evaluation chapter, in section 5.5.1.

The implemented selection methods rely on the RND mechanism as confidence structure to evaluate the states confidences. They consist in the following methods:

- **Less Uncertainty** is a SSM which consists in the sending agent sampling from the replay buffer the transitions with the lowest uncertainty to the receiver's transfer buffer;

- **Highest Uncertainty Difference** is a RSM in which the receiver calculates from its transfer buffer the states' uncertainty. In this case these values cannot be compared to the sender's ones in absolute terms since the sender's and receiver's RND networks are independent from each other. In this scenarios the sender transfers its confidence interpretation of the given state and the receiver selects the highest difference between the two interpretations (equation 3.5).

$$\begin{aligned} \Delta U_{s_t} &= U^R_{s_t} - U^S_{s_t} \\ &\underset{s_t}{\operatorname{argmax}}\, \Delta U_{s_t} \end{aligned} \tag{3.5}$$

The SRND models are designed starting from the same structure of VT models but the modifications are highlighted in figures 3.6 and 3.7 with colour red. The sender model, in figure 3.6, is characterised by a different **TL Component**: it implements as *confidence source structure* a random network distillation module instead of the visit table, which is updated by the states visited each step. Every time a new state is visited the confidence structure evaluates the related uncertainty and extracts the confidence pushing it to the transition in replay buffer. The *SSM* selects data from the sender's replay buffer, using the precalculated confidences as filter criteria, then it sends data to receiver's transfer buffer. Receiver, in figure 3.7, is quite similar to VT model. The only difference is the confidence source structure which is a RND module like the sender's model, so the agent evaluates the transferred data using its RND instead of the VT.

3.4.3 State-Action RND

RND is a technique whose main use is usually as curiosity module or intrinsic reward of an agent's model. Its aim is to estimate the relative degree of knowledge of the single state, it is able to distinguish a well-known state from a new one or a state that has been visited few times. The prediction network adjusts the weights and creates a state representation to imitate the target network. The basic idea of this scenario is to change this learned structure and to let RND to learn the representation of a *state-action* pair instead of the state only. In this way the network is able to provide

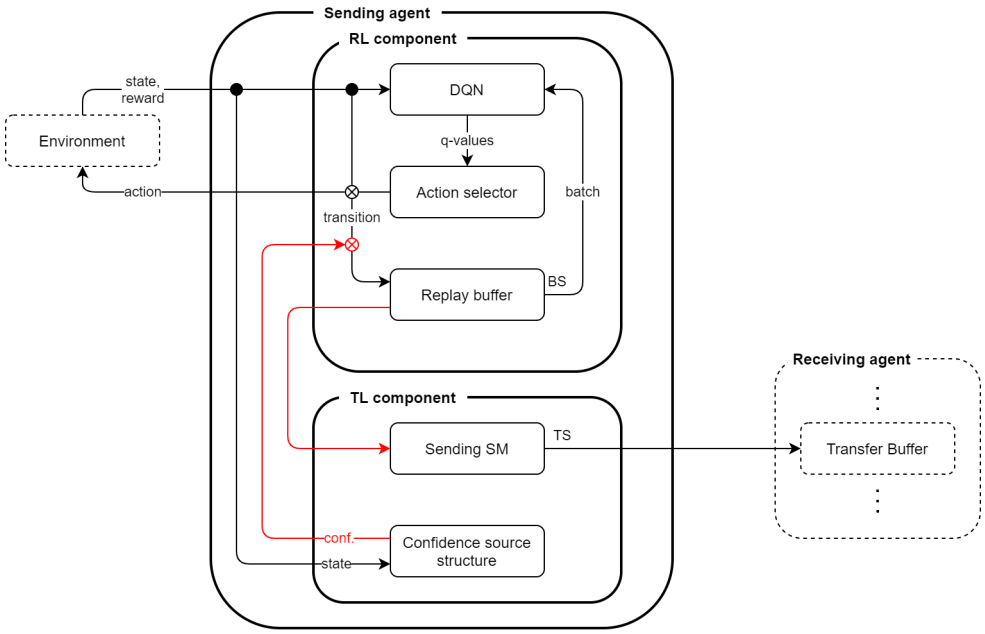


Figure 3.6: Agent sender model in "State RND" scenarios

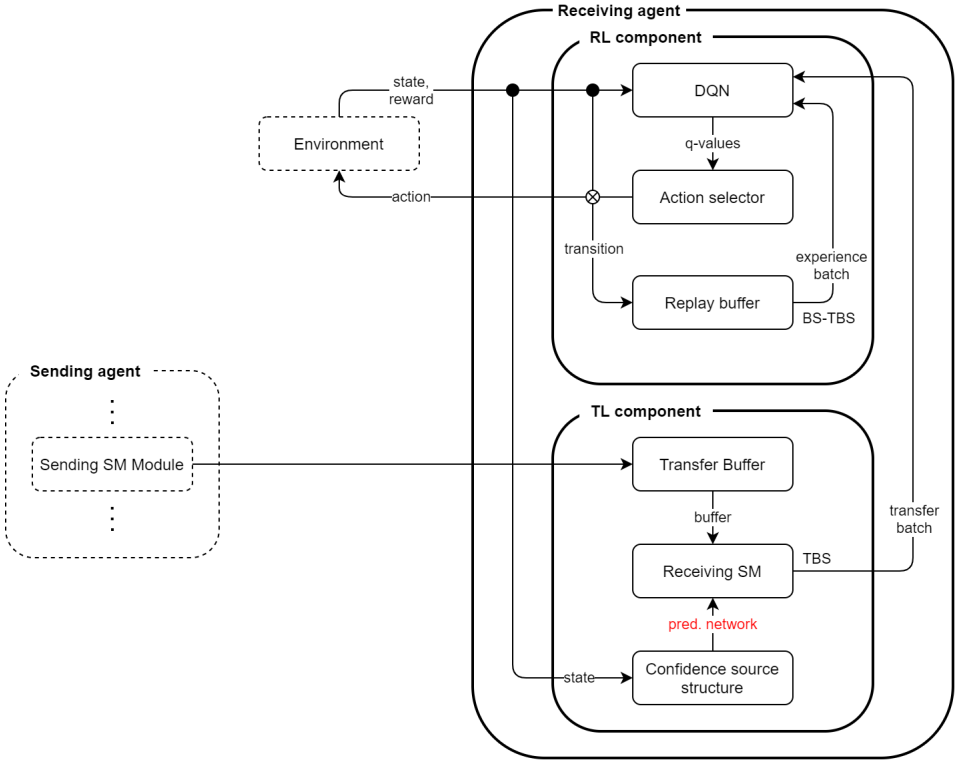


Figure 3.7: Agent receiver model in "State RND" scenarios

the confidence of the actions from the given state so from the curiosity prospective it can be considered an intrinsic estimation of the uncertainty on the state-action space.

The implemented selection methods, State-Action RND (QRND), are based on this state-action RND structure and they work at the same way of SRND:

- **Lowest Uncertainty** is a SSM which considers the lowest uncertainty as highest confidence and prioritise those states;
- **Highest Uncertainty Difference** is a RSM in which both confidence prospective are considered and the highest difference is priority criterion of selection (expressed in equation 3.5).

As for the previous models, the QRND models are designed starting from SRND's structure and the modifications are highlighted in figures 3.8 and 3.9 with colour blue. Both models implement a different mechanism to make the new *confidence source structure* work: they merge the perceived state with the selected action and send them to update the RND predictor's weights. Obviously when a transition has to be evaluated by the RNDs, as requested by the selection methods, both state and action are provided as network's input. Every other component works like in SRND model.

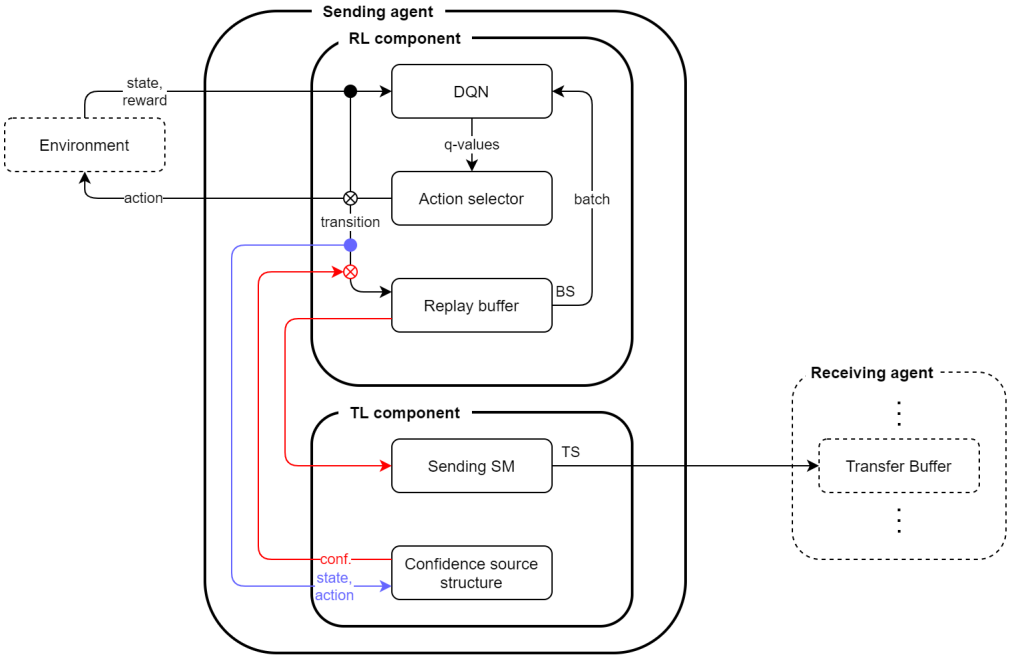


Figure 3.8: Agent sender model in "State-Action RND" scenarios

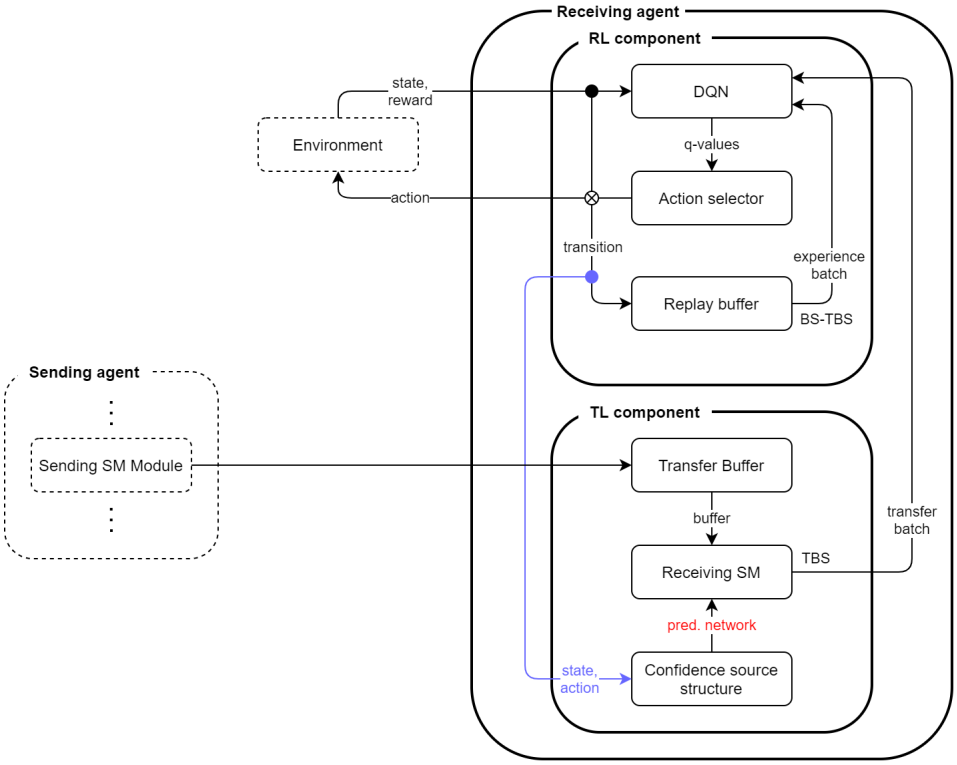


Figure 3.9: Agent receiver model in "State-Action RND" scenarios

3.5 Transfer Optimisation with Parameters Tuning

Usually machine learning algorithms involve many hyperparameters, in particular when they are implemented with neural networks, literally the whole network can be considered a set of parameters since every network's feature is customisable and can affect the outcome, for instance the number of neurons, of layers, the activation functions, the loss function and the optimiser. Once all the mentioned parameters are fixed, a deep q-learning algorithm has a minimum of three hyperparameters: the *learning rate* (α), the *discount factor* (γ) and the *batch size*. These parameters can be tuned using a Bayesian optimisation approach in which a surrogate model attempts to approximate the objective function and optimises it according to a chosen acquisition function. There are many frameworks and libraries that help to tune hyperparameters in complex machine learning models, a modern one is Vizier and is presented in [16]. Online Experience Sharing algorithm and methods involve many mechanisms, in par-

ticular the transfer frequencies and transfer sizes depend on a set of new hyperparameters. They have important roles in the algorithm's success and performances, so it is possible to tune these parameters attempting a automatic tuning through Bayesian optimisation, using as iterations evaluation criteria the total reward earned by the agents receivers. The transfer parameters tuning with Bayesian optimisation is explained in details in evaluation chapter, in section 5.5.2.

3.6 Summary

The designed algorithm is a specific transfer learning approach, in which a group of agents learn online how to solve their task, half of them are senders which share set of transitions to the other half the receivers, making independent systems composed by one sender and one receiver. Transfers are designed to implement every new feature twice, thanks to an auxiliary transfer buffer which stores data between the two transfers. Multiple significant features are designed to suite TL's goals, such as transfer frequency, size and the transitions selection method in order to extend the possible combinations of transfer methods and to enhance transfer's quality encouraging the agents to exchange data when they have the opportunity to help each other.
While the frequencies are constants, transfer size between buffer and receiver handled by a function designed to improve progressively the amount of shared transitions at the beginning of the learning process, reach a maximum and decay gradually to a minimum toward the end of the simulation.
Different agents have been designed, supporting the introduced selection methods and other features. The common point is that, they base the selection on a confidence source structure supplied to the agent. The aim of the introduced methods is to prioritise the transitions with more confidence then the others when the sender is selecting, and to choose the ones with less confidence when the receiver is selecting, to improve its knowledge about the less-known states. The confidence is provided by a table which keeps track of the state visits and two different RND curiosity modules, one keeps track of the states' uncertainty and the other of the pairs state-action's uncertainty.

Chapter 4

Implementation

The produced framework is a versatile tool capable of running many instances of the same system and establishing the communication among the agents.

4.1 Framework design

The developed framework[1] is a console application made in Python 3.8 using few machine learning libraries including one of the most widely used for machine learning *Pytorch*[2] to implement neural networks (tensors), loss functions and optimisers. A useful library to forward data easily to graphs is *Tensorboard*[3], it makes simple to monitor the learning processes' metrics. *OpenAI Gym*[4] is a library which simplifies agent abstraction and communication with the environments, it includes also some benchmark environments already implemented. A library to handle auto-ml processes is *pyGPGO*[5], it implements surrogates functions and other useful tools in Bayesian optimisation. The console application is not the only developed and used software, an external module has been built to compare the outcomes and produce graphs, a *Jupyter notebook*[6] is a set of text fields that groups code chunks and gives the possibility to

[1] Source code is available on Github: `https://github.com/FedericoBottoni/OES`
[2] `https://pytorch.org/`
[3] `https://tensorflow.org/tensorboard`
[4] `https://gym.openai.com/`
[5] `https://pygpgo.readthedocs.io/`
[6] `https://jupyter.org/`

organise them in simpler way, it is very useful when code is particularly linear. The libraries used in the graphs notebook are *Matplotlib*[7] and *Plotly*[8].
The built framework is extremely modular in order to replace any chunk with the minimum effort. The main modules that compose it are the following:

- **Agent** module includes the homonym class and the tied classes Buffer, DeepQNetwork and Environment, it represents the system as it is;
- **Online Learning Runner** module is responsible of instantiating the agents and the coordination of each other class, it occupies also of the systems' disposal at the end of the simulation;
- **Transfer Learning** module applies the introduced TL mechanisms among the agents during the simulation, it includes the Transfer class and the ConfidenceSource;
- **Tensorboard Plot** module gathers data like rewards, losses and transferred sizes and plots them in tensorboard;
- **Parameters Tuning** module runs the implemented automatic hyperparameters tuning through Bayesian optimisation.

The UML diagram is shown in figure 4.1 summarises the framework classes that compose the framework and their relationships.

4.1.1 Agent module

This module represents the system itself, it includes the agent, its environment, its learning and TL mechanisms. Every agent owns a replay buffer and a deep q-network to allow the learning process, only receiving agents own a transfer buffer too which is an instance of the same class of the replay buffer. With these structures the agent is capable of interacting with the environment, collecting states and rewards, calculating

[7] `https://matplotlib.org/`
[8] `https://plotly.com/`

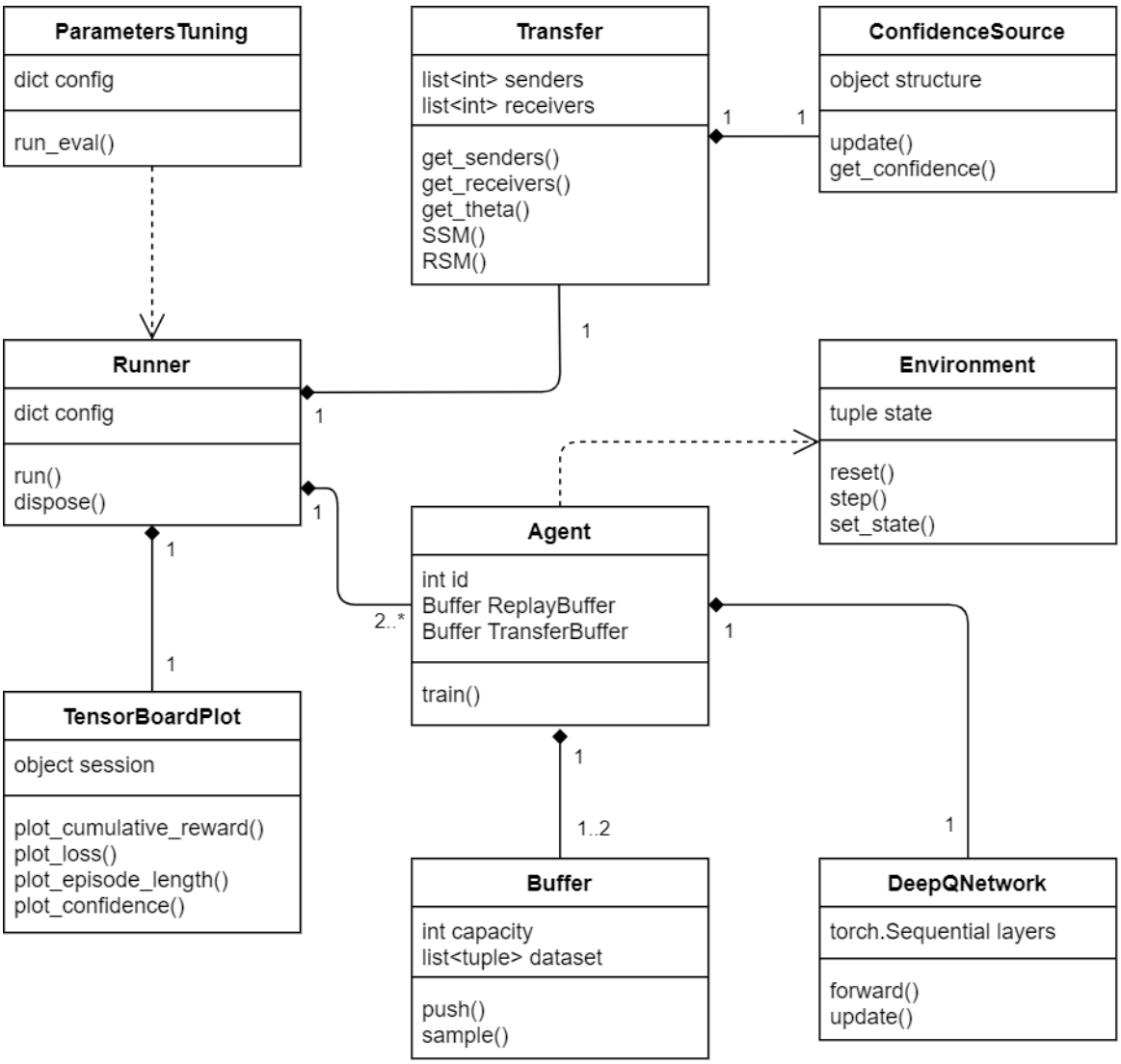


Figure 4.1: UML diagram of the framework

action through the DQN and storing transitions in the replay buffer. Agents' training processes are executed from the Runner class which handles the online execution and forwards collected data to TensorBoardPlot class.

4.1.2 Transfer Module

The transfer module is a python class instantiated at the beginning of the online simulation, it has methods to calculate the receivers from a given sender and vice-versa (the relationships among the agents), it contains the implementation of theta

function taking the episode number as input. It implements SSMs (trivial and the new one), RSMs, and it interacts with the ConfidenceSource class, which provides a method to update the structure and to evaluate the state confidence (or state-action's one), once the object is instantiated from the class, the SMs are set up and the structure is created. Three different implementations of the Transfer and ConfidenceSource classes have been created but only one pair of them is activated in a run, one for each agent model:

1. the first module implements VT, consisting in a multidimensional array with an entry for each discretized state of the state-space, in order to keep the count of the agent's visits of the given state-space area; SSM is *most visited* so it selects the first entries of the table with highest number of visits, while RSM is *less visited*, with the lowest visits;

2. another module is SRND where instead of the array, a RND module is instantiated, consisting in two torch networks, target and predictor, and in a set of APIs which allows to calculate the uncertainty through mean square error loss function and to update the predictor's weights. the implemented SMs are *lowest uncertainty* and *highest uncertainty difference*, the first prioritises the states from the replay buffer with the lowest RND output, while the second calculates the difference between the uncertainty got from sender's RND and receiver's and it selects the highest values;

3. the last TL module, QRND, implements the same SMs of SRND but the structure is slightly different because the RND module represents the action too, not only the states.

4.1.3 TensorBoardPlot Module

This module is a python class that gathers learning information from the runner, organises them and forwards to tensorboard and matplotlib to produce the appropriate graphs. In particular data every step consist in loss values and cumulative rewards,

while every episode the average value of the episodes are plot for the same metrics, than episode length, average confidence (if the scenario is SRND or QRND). The only metrics strictly tied to transfers are sending size and receiving size for each step, to allow the recognition of high transfer periods. Every graph is plotted in two versions: one with a line for each learning process and the other with the average values in step or episode among the agents, in order to perform reverse engineering on the "average line" and find the processes that caused the given behaviour.

4.1.4 Parameters Tuning Module

To perform the Bayesian optimisation a separated class has been developed, with a different entry point. Running the console app by "parameters tuning" entry point, an instance of pyGPGO is built including a *Gaussian Process* surrogate, a *Expected Improvement* acquisition function and *Squared Exponential* as covariance. The used evaluation function the call to "run" method of an instance of the Runner class which executes the online learning processes of the set of agents with their environments and DQNs according to the related configuration file and, at the end, returns the sum of the receiving agents that interacted with their environments. The GPGO instance runs 5 iterations of initialisation and successively other 20 iterations to search the max reward possible, saving the log in a log file.

4.2 Charts notebook

An external Jupyter notebook has been developed which processes the produced datasets (in json) outputs from tensorboard. The notebook allows to plot the datasets combining the experiments in order to allow the graphs overlap and to compare the results to find the improvements. It is made up by a first section where the main methods are declared, from the data acquisition function to the smooth methods and to the data aggregate functions. Successively a configuration section is executed where the lines to display are selected with the related parameters about the portion of graph to

zoom and emphasise and the smooth function's effectiveness. At the end a last section executes the methods building the graph and exporting it.

Chapter 5

Evaluation

Goal of this chapter is to present the results of the introduced algorithm studying the brought improvements for each scenario and selection method. Every experiment is presented and analysed in this chapter through the evaluation metrics of transfer learning. The second evaluated aspect concerns transfer hyperparameters tuning which results are presented and explained.
This chapter starts presenting the evaluation metrics that are used to evaluate each experiment, it continued with the definition of the considered environments to evaluate the algorithm, successively the experiments are introduced (from the baselines, deeply to every simulation), a quick look on the results of transfer hyperparameters tuning and a final discussion.

5.1 Evaluation Metrics

Every experiment is evaluated applying the metrics introduced in section 2.5.2. This metrics help to identify the success of a generic transfer learning algorithm, they can be applied in this experiments as well. They consist in:

1. **Asymptotic Performance** describes the experiment's trend if it has converged at the last episodes of a given simulation. If a scenario's performance converges at a higher cumulative reward value than the baseline, it will be considered an

improvement;

2. **Learning speed** considers how fast an experiment's performance increases compared to the baselines. The speed concept includes "Time of Threshold", "Total Reward" and "Transfer Ratio" introduced in the related section 2.5.2 and is visually evaluated checking how the graphs increase and decrease;

3. **Jumpstart** is an improvement that may be present at the first episode's performance of each experiment. If it starts from a higher cumulative reward than the baselines, in terms of this metric, it will be considered an improvement;

4. **Learning stability** is a metric that usually does not concern TL but it can describe better an experiment in particular during its convergence period. It consists in a visual evaluation of the amount of spikes that characterises the experiment while it should have low variance.

5.2 Environments

The experiments have been evaluated in a set of single-agent environments considered benchmarks since the methodologies are new, they need to be evaluated in the most simple environments possible. The two proposed environments are simple but hide some implicit mechanisms that have to been learnt by the agents to be solved, for this reason online experience transfer can be an interesting tool to encourage an agent to suggest to another one to face those mechanisms or to avoid some states and trajectories.

5.2.1 Mountain Car

In Mountain Car environment the task consists in driving an under-powered car up a steep mountain road. Being gravity stronger than the car's engine, even at full throttle the car cannot accelerate up to climb directly the mountain. The only possible solution is to first move away from the goal and approach the opposite slope to get

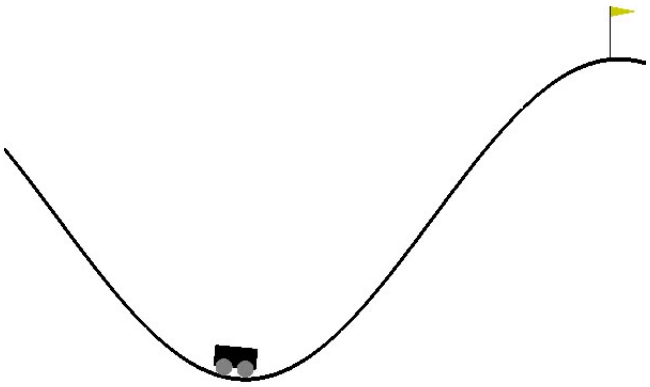


Figure 5.1: Mountain Car structure

enough potential energy. Then, by applying full throttle the car can build up enough kinetic energy to succeed at climbing it and reaching the goal. This environment is an example of a continuous control task where it is necessary to move away from the goal to reach it [40], so a greedy policy is not acceptable.

The parameters are presented in table 5.1, the initial position of the car is assigned a uniform random value in [-0.6, -0.4] and the starting velocity of the car is always assigned to 0.

Table 5.1: Mountain Car parameters

Observation/State	
Car Position	$\in [-1.2, 0.6]$
Car Velocity	$\in [-0.7, 0.7]$
Action	
Push Car to Left	0
Nothing	1
Push Car to Right	2
Episode termination conditions	
Car Position	> 0.5
Episode Length	> 2000
Reinforcement	
Reward	-1 for each step

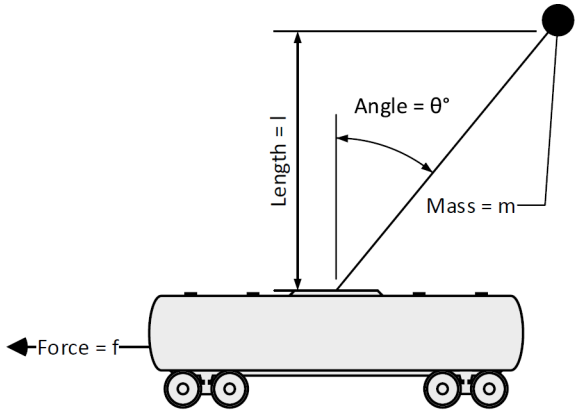


Figure 5.2: Cart Pole structure

5.2.2 Cart Pole

The objective here is to apply forces to a cart moving along a track so as to balance a pole which is attached on it and prevent it to fall over. A failure is said to occur if the pole falls past a given angle from vertical or if the cart runs off the track. The pole is reset to vertical after each failure. This task could be treated as episodic, where the natural episodes are the repeated attempts to balance the pole [40]. The agent needs to learn how to increase and reduce the cart's velocity to postpone the pole's fall. Parameters are presented in table 5.2.

5.3 Experimental Setup

All the outcomes presented in this chapter have been achieved on the same data and under same conditions to ensure fairness: in benchmark environments it is reflected only on the number of episodes and the number of online 1-to-1 systems that have run the simulation to get more accurate data, at DQL level every agent shared the same DQN shape (two hidden layers with 24 and 48 neurons and hyperbolic tangent as activation function) and the same hyperparameters. Each experiment has run the same number of episodes: 1000 episodes in the mountain car environment and 150 in cart pole. Each experiment has been run by the same number of online learning agents: three senders and three receiver in order to catch the high variability of these

Table 5.2: Cart Pole parameters

Observation/State	
Cart Position	$\in [-2.4, 2.4]$
Cart Velocity	$\in [-Inf, Inf]$
Pole Angle	$\in [\sim -41.8, \sim 41.8]$
Pole Velocity At Tip	$\in [-Inf, Inf]$
Action	
Push Cart to Left	0
Push Cart to Right	1
Episode termination conditions	
Pole Angle	$> 12^\circ$ or $< -12^\circ$
Cart Position	> 2.4 or < -2.4
Episode Length	> 500
Reinforcement	
Reward	1 for each step

environments.

Every produced graph has been analysed first through its raw data, then applying a smooth function such as *rolling* function from the library *pandas*, which provides a rolling window calculation replacing the samples with the average value of moving window with fixed size. When the function tries to process the first values, the rolling window lies partially outside the array, so these values are ignored. For this reason every graph has been analysed also through a second function, a custom *smooth* which implements the *Exponential Moving Average (EMA)* with weight between 0 and 1, it smooths the graph differently saving the first values. The following graphs are produced through one of these two methods with the related rolling window size or the smooth weight considering always the specific feature that has to be underlined.

The most important graphs are present in this section but some additional are located in appendix A.1 and referred in this chapter.

5.4 Evaluation and Analysis

This section is organised as follows:

1. Baselines presentation for each environment;

2. presentation of *State Visit Table (VT)* experiments where the state confidence is represented by the number of visits of each group of states. The experiments consist in different usage of the table to let the sender and receiver the transitions selection according to their confidence table;

3. *State Random Network Distillation (SRND)* experiments where the state uncertainty is represented by the mean square error between the RND predictor network and the RND target network (introduced in section 3.4.2). This outcome, the uncertainty, is multiplied by -1 to turn it in confidence value;

4. *State-Action Random Network Distillation (QRND)* experiments consist in a similar set compared to SRND but the networks accept an additive input, the action. In this way the calculated value describes the confidence in taking the action from a given state and no longer only the state;

5. *SRND and QRND Comparison* is a brief section in which the two scenarios' experiments are compared in order to identify similarities and differences.

5.4.1 Baselines Comparison

The baseline for this set of experiments consist in the following implementations:

- **No Transfer (NT)**

  A first execution of the parallel learning process with no sharing of any kind of information. In this case the systems are independent and run regular deep q-learning processes. The performances are related just on the local experience with no perturbation.

- **Random Transfer (RT)**

  In this online execution of a bunch of agents are divided in two groups, **senders** and **receivers** in 1-to-1 relationship (for each sender there is an only receiver and vice-versa). Senders run the same implementation of NT baseline, meanwhile

they fill their receiver's transfer buffer with the most recent samples from their replay buffer pushing a few of local transitions. The periodicity and the amount of transferred data are described by the hyperparameters (defined in section 3.3.1). While senders sample the batch to optimise their neural network's weights from the replay buffer, receivers sample randomly the batch in $\theta(ep) \cdot BATCH$ elements from the transfer buffer and $(1 - \theta(ep)) \cdot BATCH$ from the replay buffer, in order to let the transfer effects dependent by the θ function which variability is described in section 3.3.1.

In summary, sender selects the transitions to transfer getting the most recent data kept in its own replay buffer. The receiver samples randomly the transfer buffer where the amount of data depends by the current θ parameter.

- **Top Transfer (TT)**

  This baseline is thought as the best collaboration reachable among agents transferring any kind of information. This baseline is created assuming that the best possible outcome can be achieved by a group of entities sharing the same "brain", it means that every single information an agent gets is used to improve the only-brain abstraction, it can be interpreted like the transfer of every information with uniform weight. In agent systems controlled by a DNN this can be translated in a bunch of agents interacting with the environment and a single neural network which selects the action for each agent, which is optimised performing the DQN update by every agent.

Mountain Car

Figure 5.3 shows the convergences achieved by the three baselines in mountain car environment, it is evident that both NT and RT follow asymptotically the line at -175 of cumulative reward and look stable enough. They present periodically some negative spikes representing the agent either failing to solve its task or finding it hard. Analysing TT, it is possible to acknowledge that it reaches a better stability than the other two baselines and it tends asymptotically to a higher reward than the other two

(around -125). An interesting common aspect of NT and RT is that both drop after the major initial increase (around episode 125) and immediately before to the end of the simulation (around episode 925).

Figure 5.4 shows the first 150 episodes of the same online learning process, here it is possible to detect an important difference between NT and the other two baselines, in fact in this chart the transfer presence (RT) leads to a convergence speed-up which is evident between the episode 15 and 40 where the line follows more TT rather than NT. No baseline shows any jumpstart according to first episode of all the baselines, NT is the only line that starts above all the others.

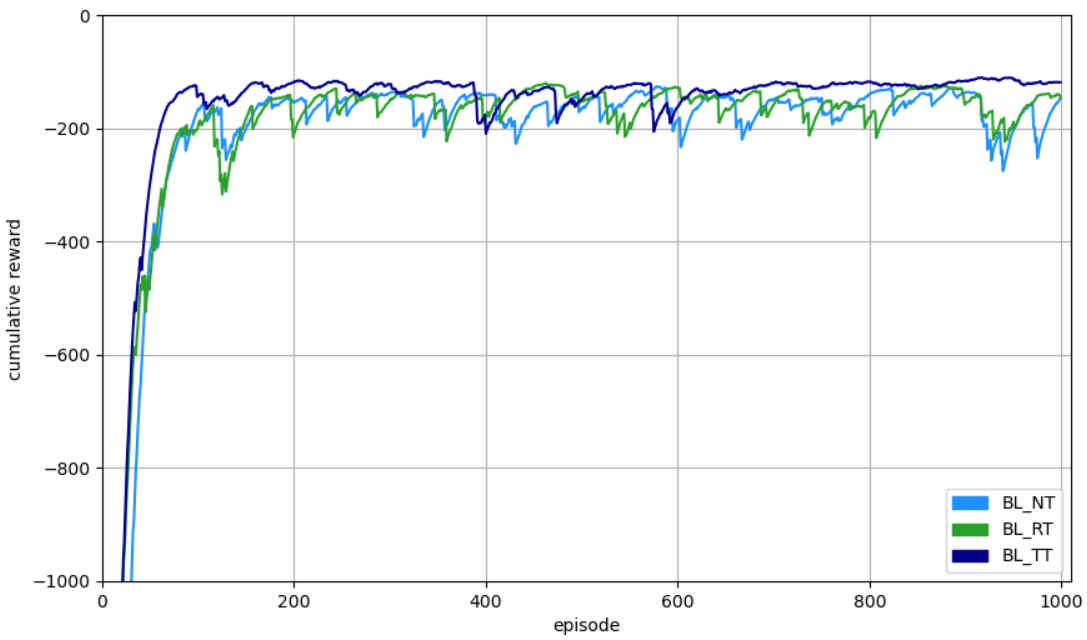


Figure 5.3: Mountain Car whole baseline

Cart Pole

Figure 5.5 shows the baselines' trends in 150 episodes of cart pole. Every line starts with a first episode failed in about 25 steps and do not increase within 15 episodes, so there are not any important jumpstarts. Both NT and RT have the most important improvement between the 40th and 75th episode, while TT between 15th and 30th

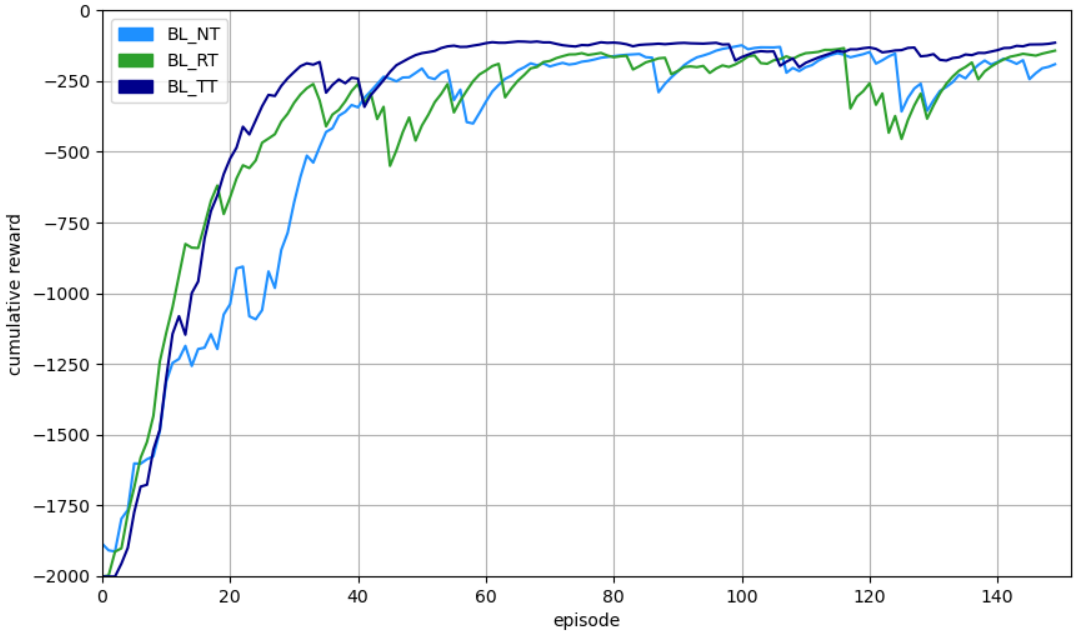


Figure 5.4: Mountain Car baseline in its first 150 steps

confirming itself again as the fastest to reach a first condition of stability.

NT finds its final stable convergence around 170 after 75 episodes, RT is slower since it reaches 150 after other 5 or 10 episodes, then it decreases for other 25 episodes to start to increase again, from 120th episode, until the end of the simulation. It is not stable since it does not find a proper convergence within 150 episodes but it reaches such high reward values that only TT has earned (after 140 episodes).

TT drops between 65th and 90th episode to start to increase until the end of the simulation reaching values higher than 225. Its stability is better than RT because although between 120th and 150th it floats around 210, it seems to be less stable than NT.

While TT brings an improvement in speed and asymptotic value reached, RT is slower than NT but reaches higher values, so it is possible to deduce that transferring enables the possibility to converge at higher values than NT in less episodes but with less stability.

Table 5.3 collects the total reward values related to each baseline of the two environments. These data can be used to compare the experiments with the usage of different

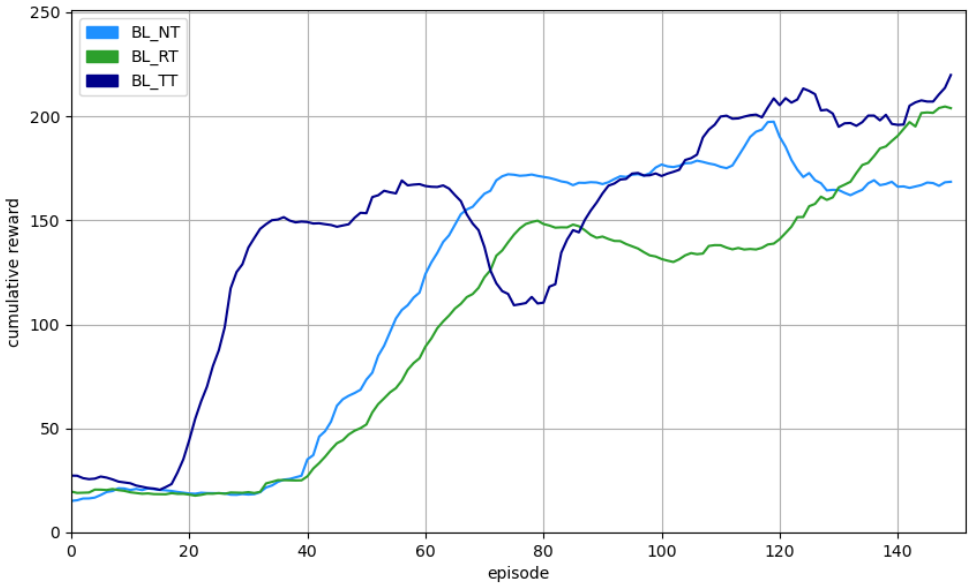


Figure 5.5: Cart Pole whole baseline

Scenario	Tot.Rew.
Mountain Car	
BL_NT	-196323
BL_RT	-182371
BL_TT	-151593
Cart Pole	
BL_NT	18932
BL_RT	16826
BL_TT	23477

Table 5.3: Total reward analysis of baseline

SMs in order to check the possible improvement in total reward.

5.4.2 State Visit Table SM

This first set of experiments consists in applying the State Visit Table (**VT**), presented in section 3.4.1, as transfer selection methods and confidence structure: three scenarios are presented according to the methodology of information selection on both sides, sender and receiver.

Sender can select picking the most recent available transitions (lasts) or by sampling

the most visited states (mostVisited), while receiver can select from the transfer buffer following a random sampling (random) or can choose the transitions related to the less explored states according to its own experience (leastVisited). This confidence implementation includes the following experiments:

1. **MostVisited-Random (Mv-R)**: the sender selects experience to share according to visits, while receiver follows a random sampling;

2. **Lasts-LeastVisited (L-Lv)**: the receiver picks experience related to state where it has a lack of information due to poor exploration, the sender provides the last stored transitions;

3. **MostVisited-LeastVisited (Mv-Lv)**: the introduced selection methods are applied in both the two sides.

In other words, when the table is not used the transitions are selected using the RT baseline's logic.

Mountain Car

Figure 5.6 shows that Mv-R in general seems slower than the baselines in overcoming the reward of -200, in figure A.1 present in appendix it is possible to observe how Mv-R increases quickly in the firsts 20 episodes and then slows down. It reaches high reward values just in few sparse episodes and in a stable period between 825th and 950th episode, then it performs a little bit worse than the baseline. It does not take any significant improvement in the described transfer metrics but shows a good trend in improving the stability.

According to figure 5.7 L-Lv seems similar to NT and RT baselines but it performs worse and with more spikes in some simulation periods like between 200th and 350th episode and between 500th and 575th episode. In other ranges it is more stable but globally it does not bring any asymptotic nor stability improvements. Observing the figure A.2 in appendix, it is evident that it increases quicker than NT in the first 35 episodes reaching -400 as quick as RT, successively it seems to find a stability around

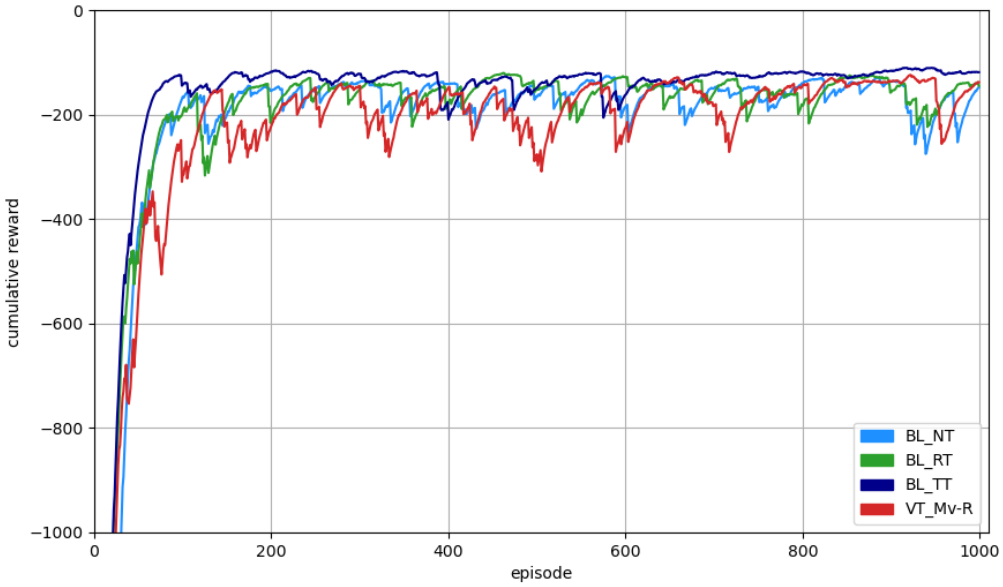


Figure 5.6: Mountain Car VT MostVisited-Random selection methods

-200 avoiding RT's drops around 45th and 125th episode. This experiment brings a slight speed-up in the first 150 episodes but it loses the advantage performing worse later.

Mv-Lv is not particularly interesting since it increases as quick as the NT baseline, shown in appendix figure A.3, with its drops and stability and is also similar in the rest of the simulation with some periodic drops and similar stability, shown in figure 5.8. It does not bring any kind of improvement.

A comparison of the three scenarios is available in appendix with figure A.4. It is possible to observe that Mv-R starts slower than the others but reaches the first interesting reward at 300th episode. After the 500th episodes it starts to increase gradually converging around the highest rewards; L-Lv is the quickest and brings just a speed-up in the first part of the learning process but periodically drops and increases again; Mv-Lv does not bring any interesting improvement compared to the other two experiments.

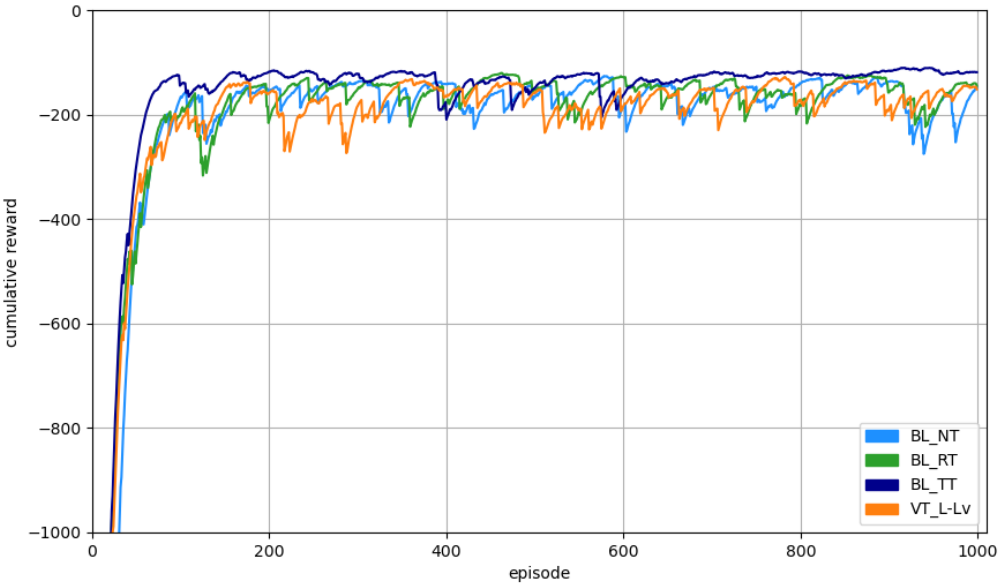


Figure 5.7: Mountain Car VT Last-LessVisited selection methods

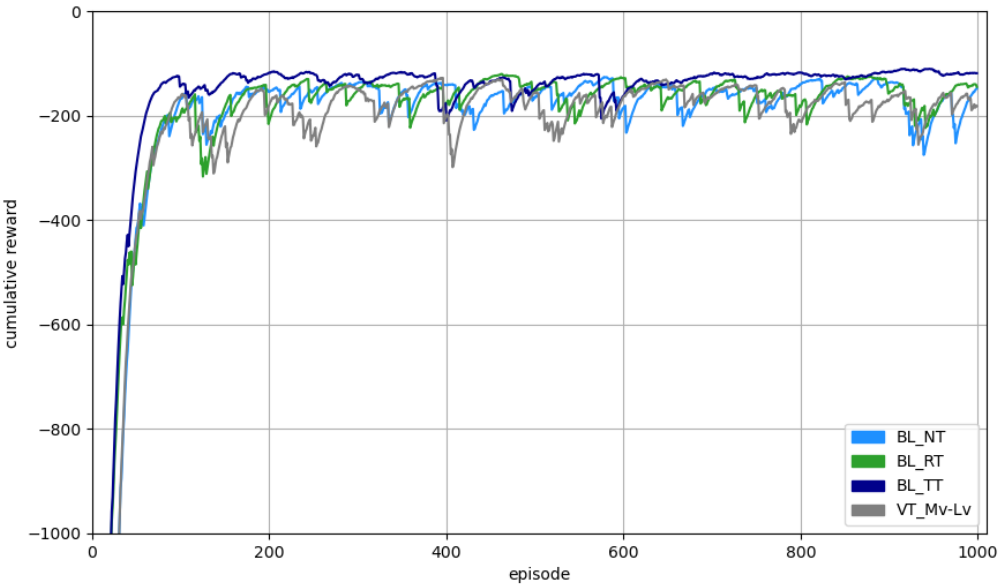


Figure 5.8: Mountain Car VT MostVisited-LessVisited selection methods

Cart Pole

Figure 5.9 shows how Mv-R starts to increase later than the baselines but it does it quicker (between 50th and 75th episode). It starts to stabilise slightly later than the

baselines but it does it around the high reward of 175 at the 75th episode. After the 100th episode it decreases slightly but immediately after it reaches a reward value as high as TT baseline, around 215 and, similarly to NT, it seems to converge again around 175. The behaviour is mostly stable, it presents the same behaviour of NT but with better performance, this suggests an asymptotic improvement of about 25 of reward compared to NT. The speed is a little bit inferior compared to NT and superior compared to RT while in terms of total reward it seems slightly worse than NT but always better than RT, especially considering the first 75 episodes.

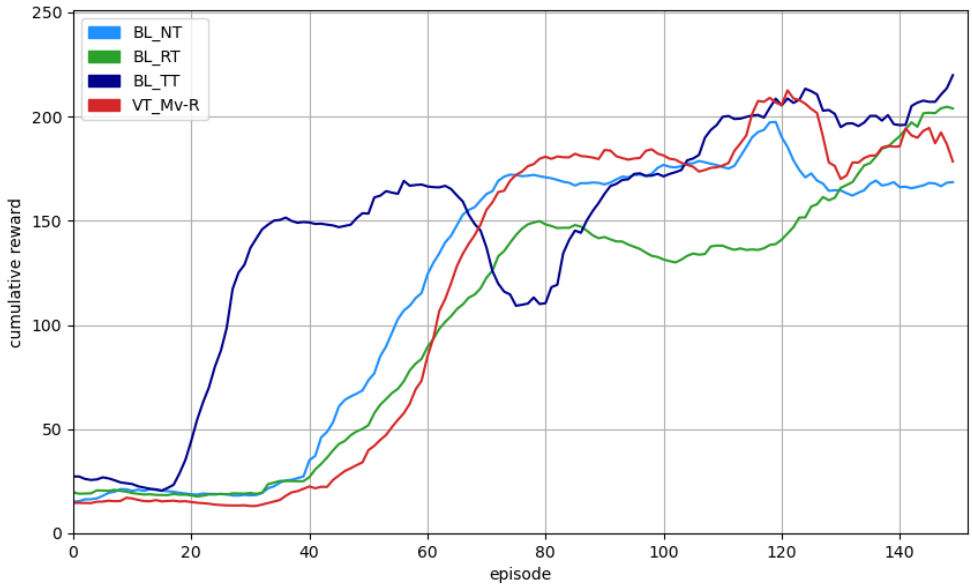


Figure 5.9: Cart Pole VT MostVisited-Random selection methods

In figure 5.10 L-Lv has a similar speed to the previous experiment Mv-R: it starts to grow later than the baselines but it reaches higher values reaching a kind of stability around 185 at 80th episode, it decreases between the 110th and 140th episode to start to increase quickly reaching performance similar to TT and RT. Its behaviour is really similar to RT but again reaching higher values. It does not find a true stable convergence but after the 80th episode it has not decreased under not-desired values of reward, for this reason it can not be considered an asymptotic improvement nor a significant speed-up, the only interesting aspects is the simulation's ending with a

positive final trend. Despite it does not converge in 150 episodes its trend does not present spikes or sudden changes and seems very smooth and stable.

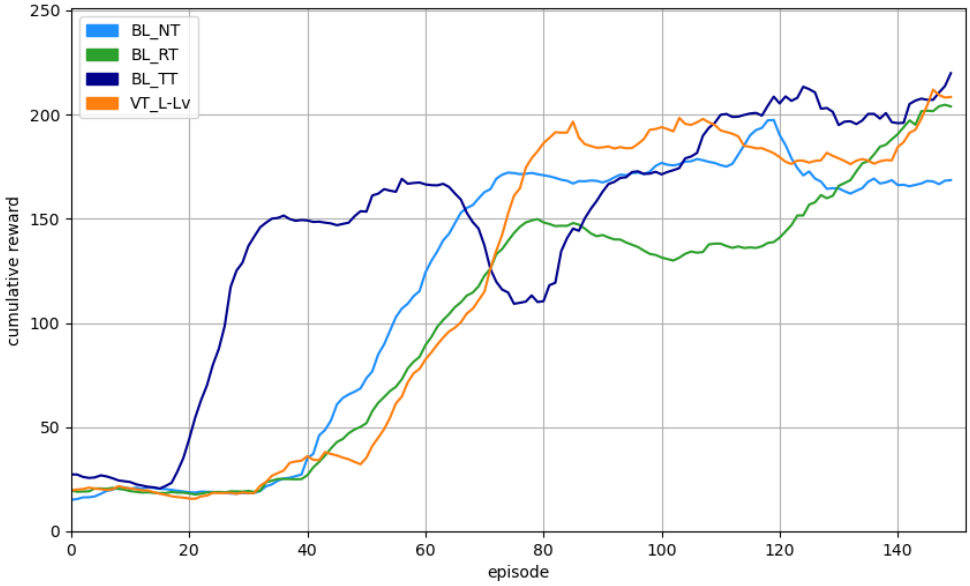


Figure 5.10: Cart Pole VT gather Last-LeastVisited selection methods

Mv-Lv (shown in figure 5.11) starts to increase before NT and RT with a speed similar to the no-transfer baseline. Again the agent reaches better performances than NT and earlier than the baseline, it reaches 175-180 at 70th episode. It behaves similarly to NT keeping the good performance until the slight decrease between 90th and 110th episode to reach 200 immediately after and, from the 120th episode, converge around 180. It brings an asymptotic improvement of about 20 of reward compared to NT and a speed-up in particular in terms of total reward, the transfer methods do not affect the agent's stability in the whole run and the speed is globally better than NT and RT baselines.

Comparing the experiments in figure A.6 present in appendix it is possible to observe that the quickest convergence is brought by Mv-Lv for sure. The highest reward (215) is reached by both Mv-R (120th ep.) and L-Lv (145th ep.), which is higher than the one reached by NT (195), but none converges at that value. At the end Mv-R and Mv-Lv are the ones that converge around 185 confirming the asymptotic improvement

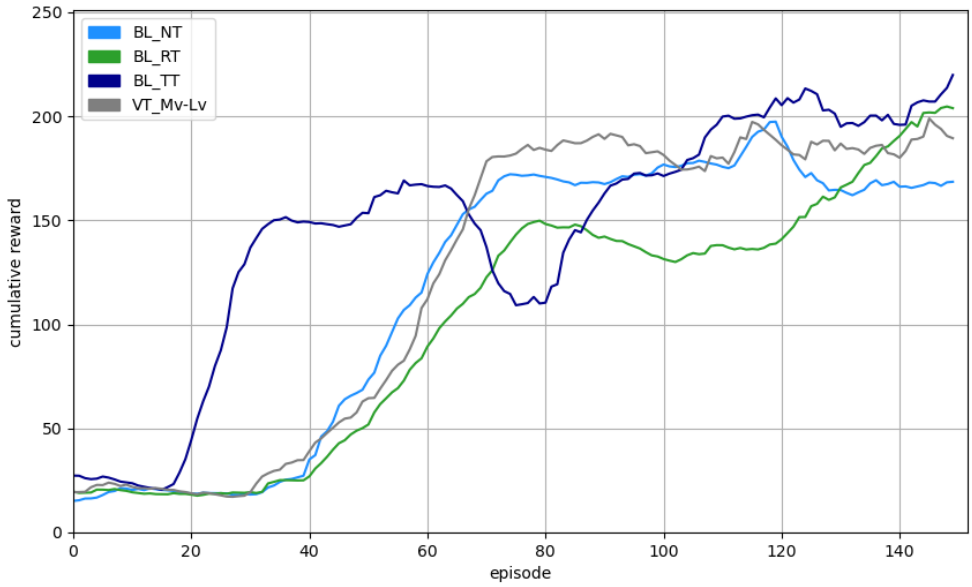


Figure 5.11: Cart Pole VT MostVisited-LeastVisited selection methods

Scenario/SMs	Tot.Rew.	Diff.NT	Diff.RT	% earned
Mountain Car				
VT_Mv-R	-213263	-16940	-30892	-0.38
VT_L-Lv	-194884	1439	-12513	0.03
VT_Mv-Lv	-207876	-11553	-25505	-0.26
Cart Pole				
VT_Mv-R	18819	-113	1993	-0.02
VT_L-Lv	19083	151	2257	0.03
VT_Mv-Lv	19947	1015	3121	0.22

Table 5.4: Total reward analysis in VT scenario

of about 20-25 reward units compared to the baseline NT. Each experiment's stability seems to not be affected by the transfers but L-Lv does not converge properly in 150 episodes. It is the one that might find the best late convergence since it finished its run at the highest reward value.

Summary

The total reward table 5.4 shows the values got from VT scenario's experiments, columns "Diff.NT" and "Diff.RT" express the margin got from the related baselines,

if the value is positive an improvement took place, "% earned" is the percentage of total reward margin of the range, created between NT and TT baselines, that has been earned with the brought improvement. Analysing the table VT seems to not have performed very well in mountain car since no interesting total reward improvements are brought compared to RT baseline, the best outcome is L-Lv that earns 3% of the margin from NT but does not overcome RT. Cart pole has two experiments, L-Lv and Mv-Lv, that bring improvement compared to both NT and RT, in particular the second one got the 22% of the considered improvement margin between NT-TT.
State Visit Table confidence implementation works definitely better in cart pole scenario, as explained, demonstrated by the asymptotic improvements of Mv-R and Mv-Lv and in general to the increasing of total reward of some experiments, while it seems to work worse in mountain car environment where the only relevant improvement is the speed-up in the first 100 episodes of the experiments and the small jumpstart by L-Lv (shown in figure A.4 and A.5 in appendix), where L-Lv performs better than NT but does not overcome RT. This suggests that in this environment the speed-up might be caused by sharing the transitions in general, and not by the table usage, while in Mv-R and Mv-Lv cases there was not a real speed-up.

5.4.3 State RND SM

The second set of experiments consists in replacing the visit table with the random network distillation mechanism as confidence source structure (**SRND**) presented in section 3.4.2 and, in addition, the application of the new transfer selection methods: three scenarios according to the methodology of information selection on both transfer sides.
Sender can again select the most recent transitions (lasts) or sample the states with lowest uncertainty according to the RND module (lowestUncertainty). Receiver can select from the transfer buffer through the trivial random sampling (random) or can choose the transitions with the highest difference between receiver's uncertainty evaluation and sender's according to their RND confidence structure (highestUncertaintyDifference). This scenario includes the following experiments:

1. **LowestUncertainty-Random (Lu-R)**: the sender selects experience to share considering the highest confidence, while receiver samples randomly;

2. **Lasts-HighestUncertaintyDifference (L-Hud)**: the sender provides the last stored transitions while the receiver selects experience from its transfer buffer using criteria of highest difference between receiver's and sender's uncertainty;

3. **LowestUncertainty-HighestUncertaintyDifference (Lu-Hud)**: the new selection methods are applied in both the two sides.

Mountain Car

Lu-R, in figure 5.12, increases as quick as RT in the first 40 episodes (evident in figure A.7 in appendix), than it does not present any drop until the 95th episode where it slightly decreases (before RT's drop). Successively it increases again to converge around -150 reaching a stability even better than the one of the baselines. Immediately before the end of the simulation, between 900th and 950th episode it drops another time decaying below -200 and increasing in the last 50 episodes. Globally this methods seem

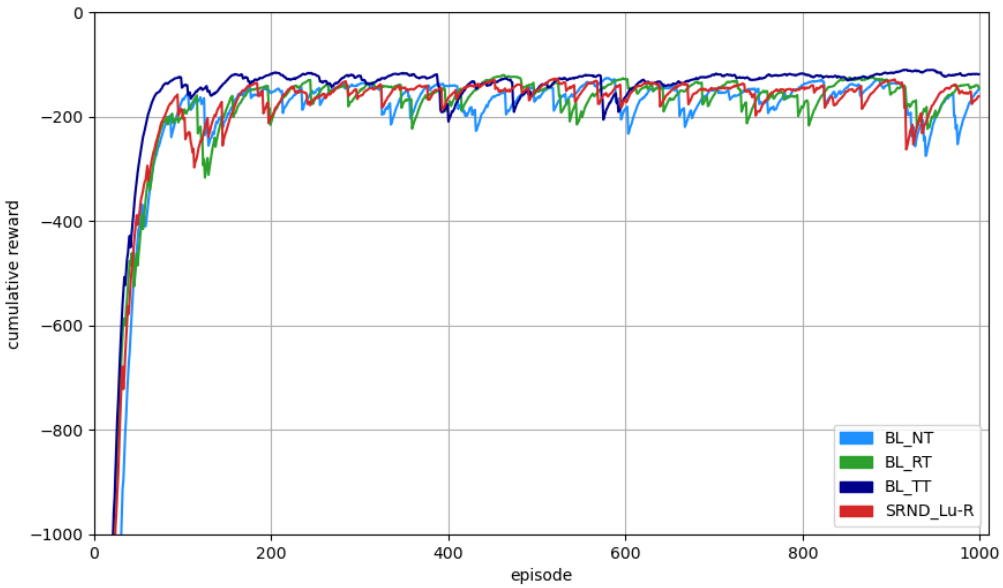


Figure 5.12: Mountain Car SRND Lu-R selection methods

to anticipate every behaviour shown by the baseline, which is translated in a better learning speed, it improves the agent's stability once it has reached the convergence, it does not bring any interesting asymptotic improvement since it drops in the last group of episodes reaching a final reward similar to RT.

L-Hud (figure 5.13) starts the same way TT does in the first 35 episodes (figure A.8 in appendix), it reaches as quick as the baselines -200 but it anticipates the first RT's drop around the 95th episode instead of the 115th. It seems as quick as the baselines in reaching the high reward of -150 but it does not converge immediately. Its stability seems similar to the baseline's in the first half of the simulation, but it improves in the last part with periodic drops alternating with stable periods. In particular between 600th and 700th episode the agent performs as well as TT (around -130) and is very stable. This methods seem to bring a speed-up in reaching -150 without affecting the simulation's stability in the first half of the simulation but it bring a slow stability improvement visible in the second half. It does not bring a proper asymptotic improvement but it might find a very late convergence higher than the baseline due to its great stability and performance similar to TT for about 100 episodes.

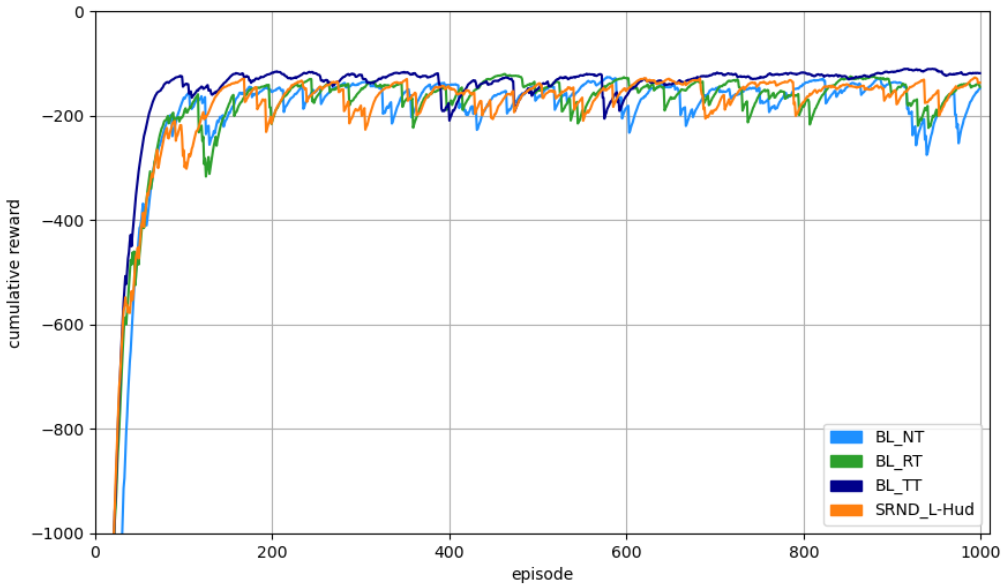


Figure 5.13: Mountain Car SRND L-Hud selection methods

Lu-Hud in figure 5.14 grows slowly compared to the baselines (figure A.9 in appendix), reaching the reward of -200 in late and -150 at the same time of NT and RT. It seems to be as stable as the baselines starting to improve from the 600th episode. The worst drop is around 970th episode, as late as NT baseline. Globally it does not bring any speed-up improvement because of the delays mentioned above, it seems really similar to NT and RT with a few periods of great performance alternating with some spikes, it does not bring any asymptotic improvement either.

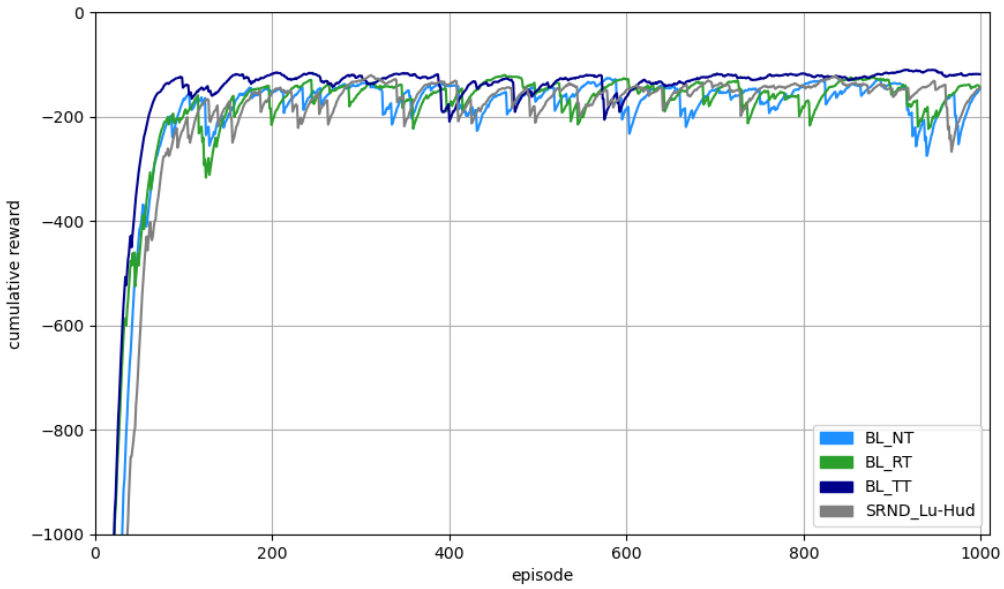


Figure 5.14: Mountain Car SRND Lu-Hud selection methods

The comparison between the experiments of this scenario is available in appendix figure A.10 and A.11, it is possible to observe that no method shows any jumpstart, Lu-R and L-Hud are both quick in the first 35-40 episodes and from the 300th everyone converges around -150. It is evident how the Hud method encourages to improve agent's stability as shown by the two experiments that use the selection method, while Lu-R is stable in the whole simulation.

Cart Pole

Lu-R is shown in figure 5.15 where it is evident its delay in the main performance improvement around episode 50 and 60, reaching the stability around the low reward of 160 after NT and RT baselines. The agent slightly increases from the 120th episode but drops in the last 10 episodes of the simulation. It does not enhance the baselines in every terms: speed, stability and asymptotic improvement.

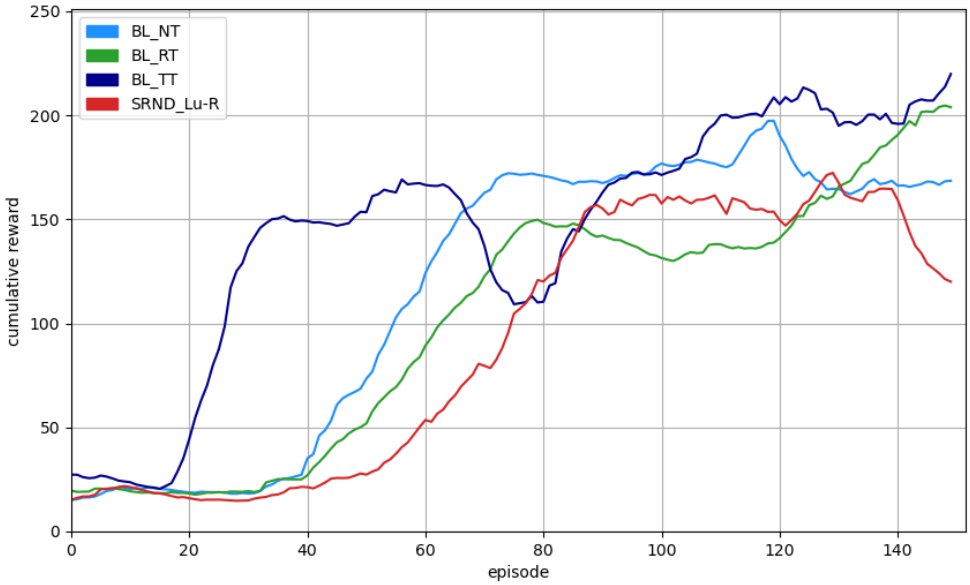


Figure 5.15: Cart Pole SRND Lu-R selection methods

L-Hud in figure 5.16 increases around 5 episodes later than RT, surpassing it and reaching 175 for few episodes. Between 95th and 125th episode it decreases to 140 and starts to increase again reaching 200 and, from the 140th episode, stabilising around 185. This experiment's behaviour is similar to RT but with a little delay, it does not bring any speed-up nor asymptotic improvement since it does not converge in 150 episodes. The total accumulated reward is similar to RT too, so these methods usage does not seem different from the trivial transfer methods.

Lu-Hud (figure 5.17) increases its performance with similar speed to RT but worse than NT. It reaches the high reward of 195 as first before every baseline, including

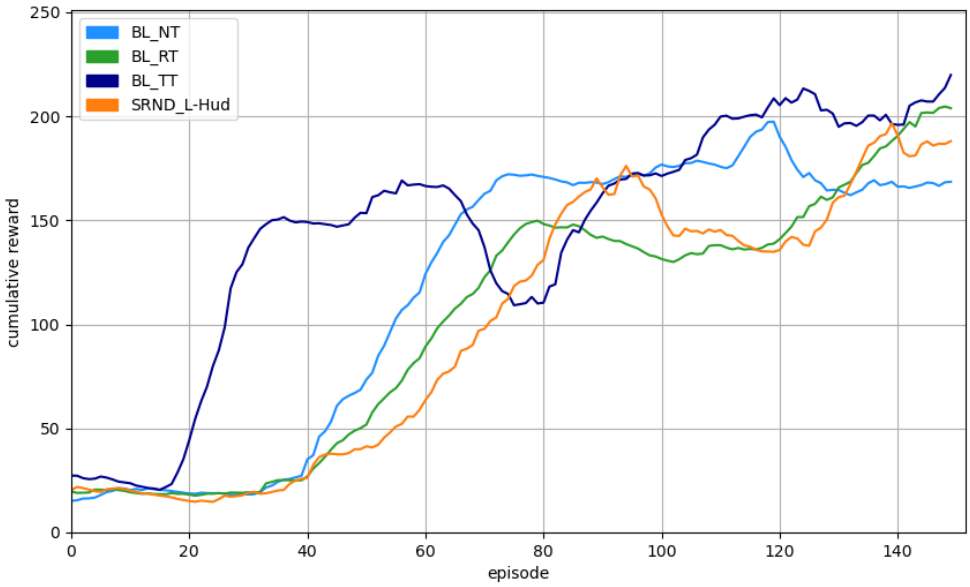


Figure 5.16: Cart Pole SRND L-Hud selection methods

TT, but it does not stabilise and slowly decays immediately after without finding a convergence in 150 episodes. Overall the speed is worse than every baseline since the earned advantage around 95th episode is successively lost in terms of total reward.
The comparison of this scenario's experiments is available in appendix figure A.12 which shows that the main difference between these methods is registered in the last 30 episodes. The usage of Lu method seems to cause a drop in the involved experiments around the end of the simulations, while L-Hud is the only experiments that ends with positive trend. No stability improvements, speed, nor jumpstarts are observable in these set of experiments.

Summary

According to total reward table 5.5, SRND got significant improvements in mountain car in Lu-R and L-Lu with an improvement of 27% in both the cases. Anyway these total rewards are not high enough to consider them as improvement compared to RT. In cart pole the got total rewards are lower than the baselines so they can not be considered interesting from total reward's point of view.

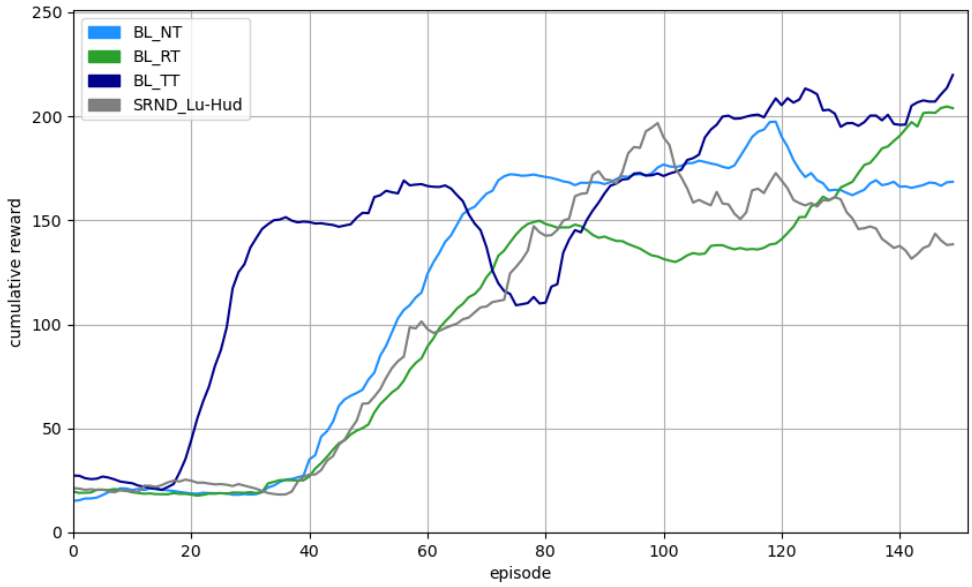


Figure 5.17: Cart Pole SRND Lu-Hud selection methods

Scenario/SMs	Tot.Rew.	Diff.NT	Diff.RT	% earned
Mountain Car				
SRND_Lu-R	-184215	12108	-1844	0.27
SRND_L-Hud	-184305	12018	-1934	0.27
SRND_Lu-Hud	-200814	-4491	-18443	-0.1
Cart Pole				
SRND_Lu-R	14577	-4355	-2249	-0.96
SRND_L-Hud	16209	-2723	-617	-0.6
SRND_Lu-Hud	16526	-2406	-300	-0.53

Table 5.5: Total reward analysis in SRND scenario

It is evident how State RND selection methods works better in mountain car rather than cart pole. Overall in the first environment the brought improvements include stability and speed, in particular Lu-R and L-Hud manage to anticipate the drops that characterise the baseline. In Cart Pole the experiments using Lu method perform globally worse than the baselines, while L-Hud has many similarities to RT.

5.4.4 State-Action RND SM

The third and last set of experiments consists in applying to the random network distillation mechanism a modification, it has to represent the action as well, building a state-action representation instead of just the state (**QRND**), presented in section 3.4.3. The confidence estimation does not concern only the state but the action chosen in the given state. The transfer selection methods are based on the same ideas of SRND: other three scenarios in which sender can select the lasts transitions available (lasts) or sample the states with lowest uncertainty (lowestUncertainty). Receiver can sample from the transfer buffer randomly (random) or can choose the transitions with the highest difference between receiver's uncertainty evaluation and sender's (highestUncertaintyDifference). This scenario includes the following experiments:

1. **LowestUncertainty-Random (Lu-R)**: the sender selects transitions based on the highest confidence, while receiver samples randomly;
2. **Lasts-HighestUncertaintyDifference (L-Hud)**: the sender provides the last transitions while the receiver selects experience with the highest difference between receiver's and sender's uncertainty;
3. **LowestUncertainty-HighestUncertaintyDifference (Lu-Hud)**: the new methods are applied in both the sides.

Mountain Car

Lu-R in figure 5.18 shows a jumpstart of about 30 rewards unit compared to NT in appendix figure A.13, it increases as fast as NT in the first 100 episodes, then it slows down and keeps increasing discontinuously to reach -130 before NT and RT (around 180th episode). Successively it alternates great performance periods with unstable and bad-performing ones for the whole simulation. This experiment's speed is similar to NT in the very first part of the simulation or even worse, it does not overcome baselines' stability and it does not bring an asymptotic improvement in agent's convergence.
L-Hud in figure 5.19 starts with a jumpstart of about 100 rewards unit (visible in figure

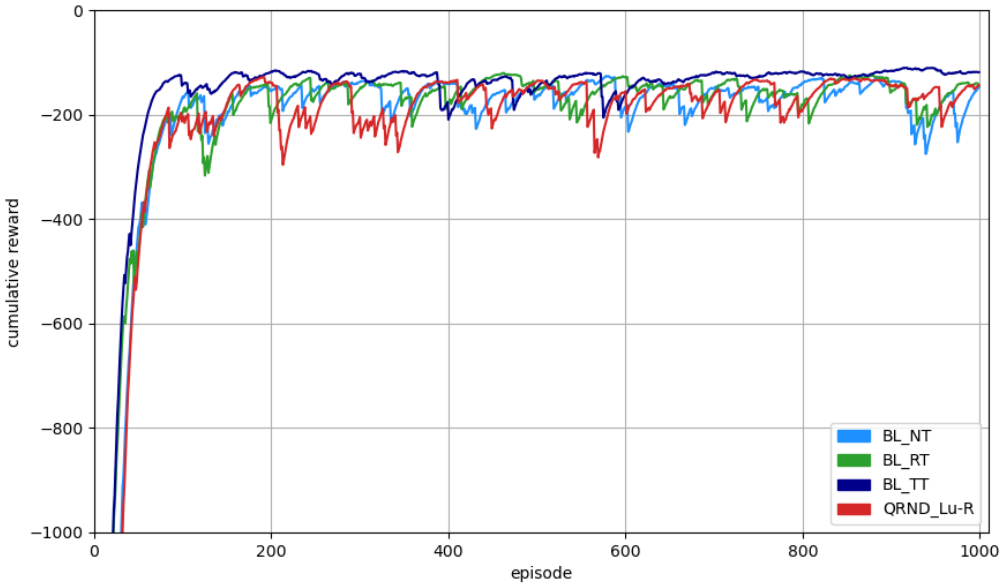


Figure 5.18: Mountain Car QRND Lu-R selection methods

A.14 in appendix), it increases as fast as RT with no drops in the first 100 episodes, it succeeds in earning the same performance that only TT has around 130th episode, reaching -115 of reward for few episodes. At the 200th episode it drops keeping the unstable and worse performance for the following 200 episodes until, about 380th episode, it stabilises again at TT's performance (around -110 of reward). From 600th episode to the end of the simulation its stability and performance are similar to the NT and RT's ones. The methods bring a good speed-up reaching a higher performance before the baselines and in total reward of the first 200 episodes. The agent does not converge at this great performance so it is not an asymptotic nor a stability improvement.

Lu-Hud is shown in figure 5.20 and in appendix figure A.15, it starts with no jumpstart but increases very quickly in the first 40 episodes (even more than TT), then it slows down but keeps increasing and converges at the same time of the baseline. It has just few slight drops including one around 600th episode and one around 700th. It is possible to observe its good performance and stability around the end of the simulation, it might be considered an asymptotic improvement compared to the two baselines' drop in the last 100 episodes. The agent is also more stable than NT and RT baselines and

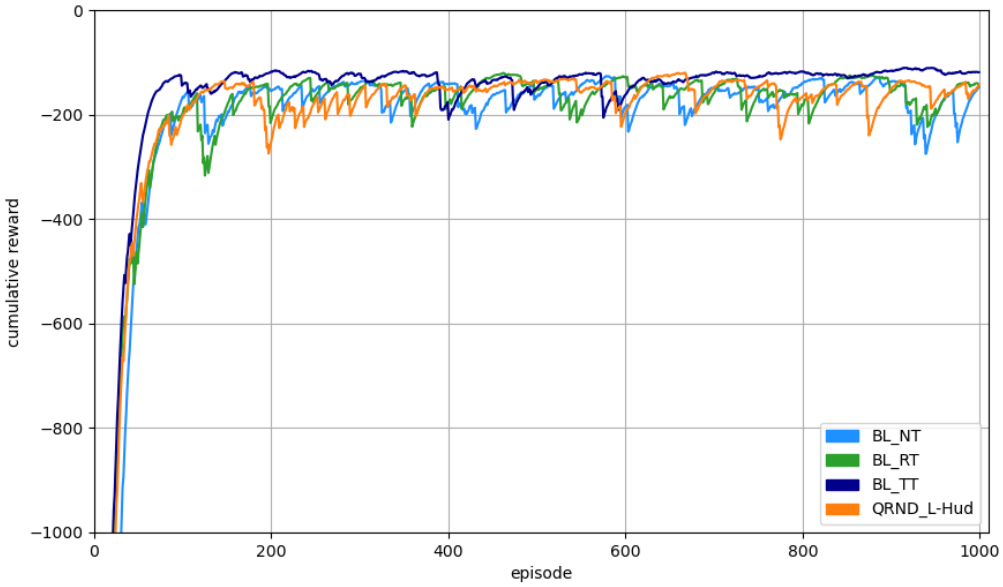


Figure 5.19: Mountain Car QRND L-Hud selection methods

it brings a visible speed-up in the very first part of the simulation, it gets also more total reward than the baselines (as shown in table 5.6).

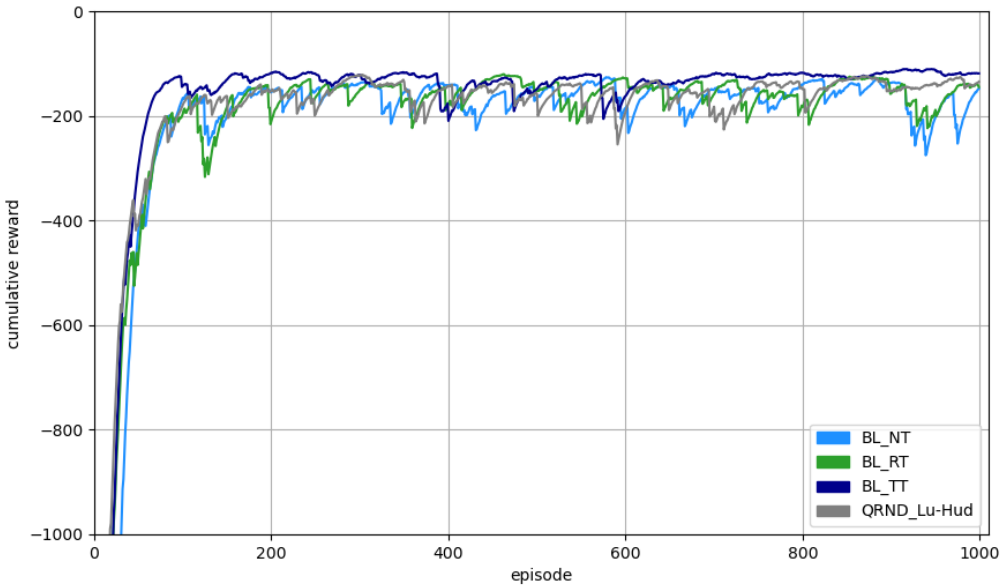


Figure 5.20: Mountain Car QRND Lu-Hud selection methods

The comparison of the scenario's experiments is available in appendix in figure A.16 and A.17 where is evident the differences of speed in the first 40 episodes and how the usage of Hud selection method makes the performance's growth smoother then Lu-R and baselines. Moreover the last 100 episodes can confirm the good stability kept by Lu-Hud at the end of the simulation.

Cart Pole

Lu-R in cart pole, in figure 5.21, starts to increase after the baselines (around episode 60) and reaches 185 of reward at 100th episode before NT, it does not converge and decreases immediately after until the end of the simulation. The experiment does not bring any improvement since it does not converge, it is not stable at all and in terms of speed and total reward it behaves worse than NT.

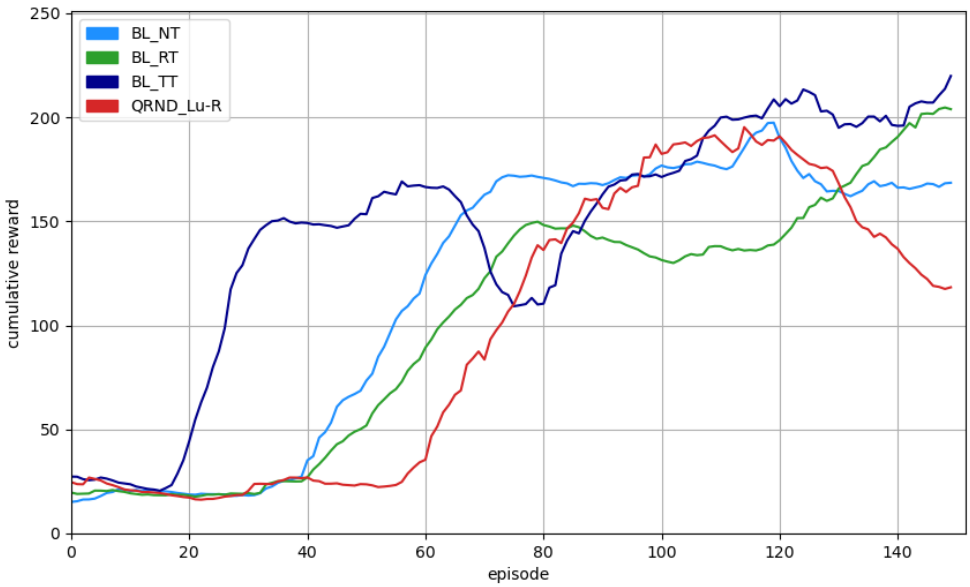


Figure 5.21: Cart Pole QRND Lu-R selection methods

L-Hud in figure 5.22 starts to increase few episodes after RT, it reaches 160 at the 80th episode, it immediately drops around 125 and keeps to grow slowly finishing the simulation around 200 of reward with a positive trend. The agent's behaviour is really similar to RT but while the baseline after have reached the first maximum value,

decays for about 25 episodes, L-Hud drops in 5 episodes to start to increase again. The agent does not seem very stable in the second major growth, after episode 85, but trend is almost always positive, it does not bring improvements in speed for sure, nor asymptotically since it does not converge and it performs slightly worse than RT at the end of the simulation.

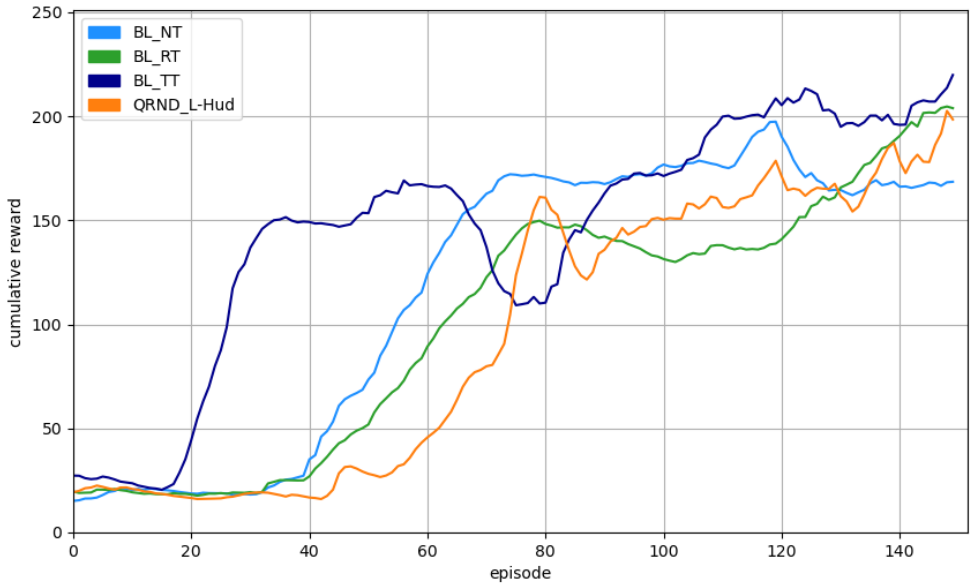


Figure 5.22: Cart Pole QRND L-Hud selection methods

Lu-Hud, in figure 5.23, increases very slowly reaching more than 200 of reward around 110th episode, before NT's best performance but it drops immediately decreasing of 30 reward units and increasing again to more than 200. From 130th episode it drops again under 150 and seems to start to grow again close to the end of the simulation. This agent has been stable only during the main increasing period (until 110th episode), after it is completely unstable, does not converge and drops before the end. It is not an improvement in speed since it is very slow and is not an asymptotic improvement because of its instability.

Figure in appendix A.18 shows the experiments of this scenario together, it is evident that there are no convergences and they are slow, in particular Lu-Hud. None is particularly stable and brings any jumpstart, none bring any interesting improvement.

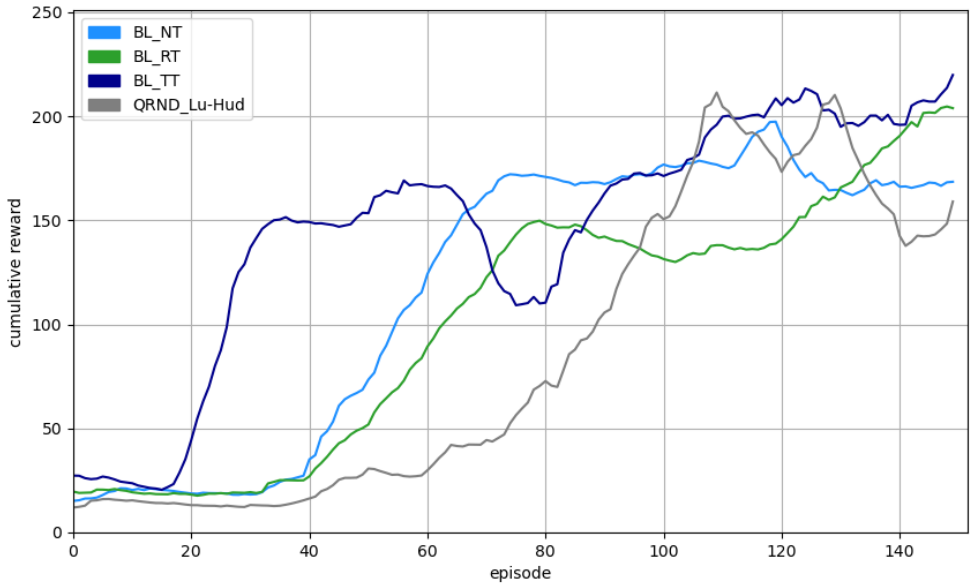


Figure 5.23: Cart Pole QRND Lu-Hud selection methods

Scenario/SMs	Tot.Rew.	Diff.NT	Diff.RT	% earned
Mountain Car				
QRND_Lu-R	-202545	-6222	-20174	-0.14
QRND_L-Hud	-183322	13001	-951	0.29
QRND_Lu-Hud	-174537	21786	7834	0.49
Cart Pole				
QRND_Lu-R	15425	-3507	-1401	-0.77
QRND_L-Hud	15914	-3018	-912	-0.66
QRND_Lu-Hud	14261	-4671	-2565	-1.03

Table 5.6: Total reward analysis in QRND scenario

Summary

Total reward table 5.6 shows the significant improvements brought by the usage of Hud selection method in mountain car context, in particular with Lu-Hud agent's performance is superior than both NT and RT baselines and the earned margin of improvement is 49%. In cart pole this scenario's experiments do not perform very well, they do not bring total reward improvements.

The selection methods seem to be very useful in mountain car systems, the main effect is to reduce the drop rate and to enhance the agent's stability, in particular the usage

of both Lu and Hud. Moreover this scenario is the only one that presents significant jumpstarts in addition to the great speed-ups detected especially with the usage of Hud. Cart pole is not affected by the same improvements with the usage of these SMs, here the the agents never converge.

5.4.5 RND Comparison

In figure A.19 (in appendix) and 5.24 are available comparisons among the experiments of SRND and QRND scenarios: mountain car's does not inform a lot, it is not possible to observe important differences between the scenarios' experiments, it is evident that in Lu-R they behave similarly to the baselines with slight drops and stability; in L-Hud they are more stable, in particular SRND drops once around the beginning and QRND a few more times during the whole simulation; in Lu-Hud SRND is a bit slower in the first episodes but both show a great stability in the whole simulation period, QRND seems really flat in the last 200 episodes.
Cart pole's comparison shows how QRND can reach higher reward levels then SRND, especially with the usage of Lu selection method. Presence of NT baseline highlights how in general performances of both SRND and QRND are not as good as the baseline.

5.5 Transfer Hyperparameters Tuning

Online Experience Sharing algorithm and methods involve many mechanisms which depend on a set of new hyperparameters. They have important roles in the algorithm's success and performances, so it is possible to tune these parameters either studying them manually sampling how they affect the agent's learning process or, exactly like every machine learning parameter, running a Bayesian optimisation. In particular, as presented in the next sections, the introduced parameter in RND module is analysed manually while a set of other transfer parameters are tuned through automatic processes.

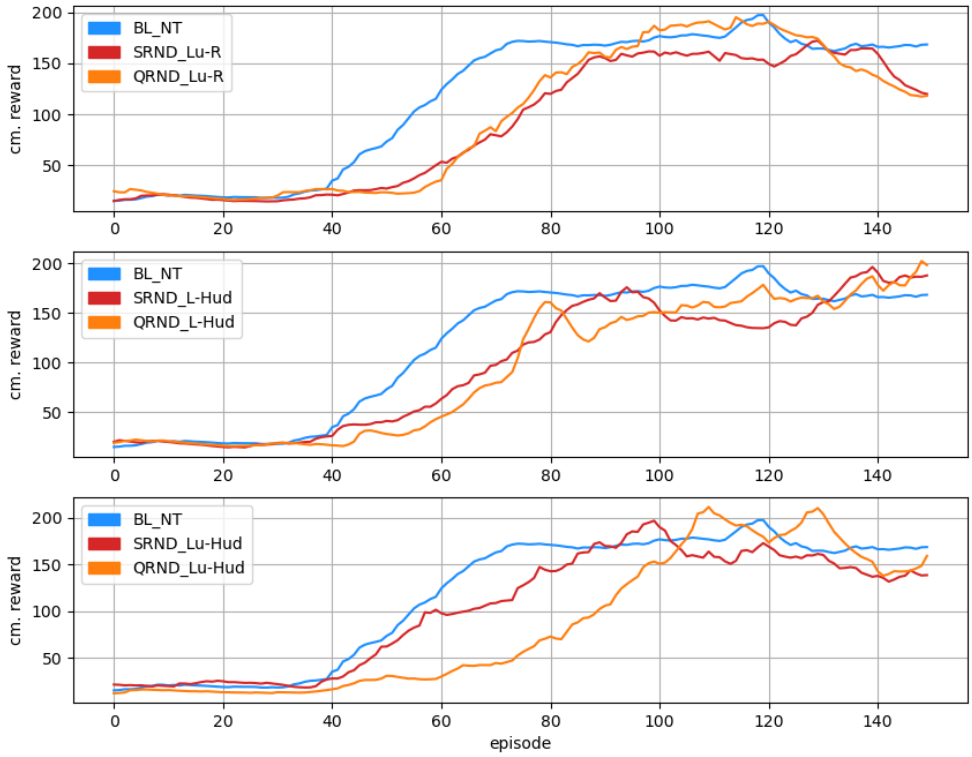


Figure 5.24: Cart Pole comparison between every scenario of SRND and QRND

5.5.1 RND Feature Extraction Size Analysis

The two RND networks are built considering the same input and output size, but the predictor has a extra hidden layer to allow its weights update every step of the learning process. The available customisation of the networks consist in the number of neurons of the hidden layer and the number of features in output: the first number has been considered constant at 64 - as average value of the other hyperparameter's range. The second parameter represents the network's capability of representation of the encoded state, for this reason the hyperparameter selection has been taken from the possible empirical values {32, 64, 128}, based on the graphs in figure A.20 and A.21 in appendix (Mountain Car and Cart Pole executions) which basically represent the average uncertainty of the visited states for each episode. The metrics to consider

interesting the studied graphs are:

1. uncertainty should decrease more slowly because in the first phase of learning, agents have to explore more, so the uncertainty is supposed to be higher;

2. it can present some spikes around the initial part, since in the initial phase the agent is exploring and should visit most of the undiscovered states;

3. it should decrease smoothly and should reach the lowest point at the end of the simulation, converging at the lowest uncertainty because at the last episodes the agent should follow the best trajectory (or trajectories) visiting only the well-known states.

An interesting aspect of the cart pole graphs in appendix figure A.21 is that around the end of the simulation (after the 140th episode) every line slightly increases demonstrating that the first found policy is exploited between the 80th and 140th episode (where the graphs are decreasing), after new trajectories are explored and agent's RND starts to evaluate these new states' confidences.
The final features size hyperparameters are 32 for mountain car and 128 for cart pole because they respect more the explained metrics.

5.5.2 Transfer VT Bayesian Optimisation

In section 3.5 a discussion about how hyperparameters are important for a good resulting policy has taken place and about the designed algorithm introducing new parameters that might be sampled. The involved hyperparameters are defined with their ranges in table 5.7, thought for an optimisation of cart pole VT scenario using as selection methods Mv-Lv.
TS represents the constant number of transitions that the senders push to the receivers' transfer buffer each transfer; *TI* is the constant frequency in which the senders transfer data; *Transfer Buffer Size* is the buffer's capacity, it has a correlation with the first two parameters since TS and TI determine in how many steps the buffer is filled by data and older data are discarded; θ_{decay} defines the decay speed of θ function (introduced

Table 5.7: Transfer hyperparameters' ranges for Bayesian optimisation

Hyperparameter	Min	Max
Transfer Size (TS)	5	100
Transfer Interval (TI)	10	500
Transfer Buffer Size	100	10000
θ_{decay}	50	500
θ_{apex}	1	150
VT Discretizer	7	15

in equation 3.1); θ_{apex} is the episode in which θ function returns the maximum value and when it starts to decay; *VT Discretizer* is the number of bins used to discretize every state's dimension.

Each iteration is evaluated with the sum of the total accumulated reward by three receivers of three different OES systems. Figure A.22 in appendix shows parallel coordinates plot of the 25 iterations of Bayesian optimisation: the selected methodology is *Gaussian Process* as surrogate model with *Squared Exponential* as covariance function and *Expected Improvement* as acquisition function, 5 iterations of initialisation and then other 20 iterations. The iterations are divided in three subsets with similar size: green lines represent iterations with lowest rewards, in yellow the mid rewards and red lines the best or the highest rewards (called class 0, 1 and 2). Correlations are not evident, nor patterns or just rules from this graph, it is possible to observe that the most part of the best runs have low TI and high θ_{apex} but these are not fixed patterns since there are good runs with high TI and low θ_{apex} too.

In order to quantify the detected correlation, it is possible to calculate the **Pearson correlation coefficient** for each parameter's distribution with the reward distribution, in equation 5.1

$$\rho_{h,r} = \frac{cov(h,r)}{\sigma_h \cdot \sigma_r} \tag{5.1}$$

where,

- $\rho_{h,r}$ is the coefficient of correlation between the hyperparameter h's distribution and the reward distribution:
- $cov(h,r)$ is the covariance between the two distributions:

- σ_h and σ_R are the standard deviations of the two distributions.

The coefficient assumes real values in range $[-1, 1]$ and represents direct correlations when is close to 1, inverse correlations when is close to -1 and uncorrelations when is close to 0.

Table 5.8: Transfer hyperparameters correlations to reward

H	$\rho_{h,r}$
Transfer Size (TS)	-0.27411
Transfer Interval (TI)	-0.25734
Transfer Buffer Size	-0.07111
θ_{decay}	-0.09749
θ_{apex}	0.474352
VT Discretizer	-0.02356

Table 5.8 shows that the correlation hypothesis can exist since θ_{apex} has a coefficient of 0.47 so it suggests a not very strong direct correlation and TI has -0.26 and suggests a weak inverse correlation, moreover a weak inverse correlation might exist in TS due to its -0.27 coefficient. Anyway these correlations are not strong enough to provide an important clue of how to model the transfer hyperparameters to maximise the earned reward, they just give suggestions that might work. In reinforcement learning, the hyperparameters tuning has always been a big issue, in particular in deep RL since the agents' performances change a lot in each simulation because neural networks need the exploration of different trajectories and strategies to reach the same result.

5.6 Discussion

It is evident that the algorithm performs differently depending on scenario, SMs, environment and task to learn, for instance it seems to show globally more improvements using state visit table in cart pole, while state and state-action RND in mountain car. The considered environments are two among the most simple single agent systems trying to solve their inside difficulties, so it is impossible to predict how the algorithm performs with other more or less complex single or multi agent systems.
The tasks of the studied environment require different skill from the agents, cart pole's

agent has to be stable and static around the best state, it starts close to the goal, while mountain car's agent needs to move away from the goal to reach it later, starting far away from it, the agent is required for a specific dynamic or trajectory. Another significant aspect might involve the agent's state-space: cart pole has 4 dimensional state, while mountain car has 2 dimensional state.

VT has the representation of state-space confidence in areas, in cart pole it manages better than RND structures to represent the stationary that the sender should suggest to another agent. SRND and QRND perform better in mountain car probably because the right trajectory is a single one, so the visited states tend to be always the same, despite the agent is requested for dynamicity. TL in mountain car can improve agent's speed encouraging it to start before to exploit the best trajectory and increase the reward, it can improve the stability too preventing the agent to drop as frequently as in NT and RT. In cart pole this TL implementation does not affect so much the learning speed like in TT where the agent finds a first convergence very early, it can cause a steeper rise of got reward which is output of a quick change of policy to a better one and it can bring an interesting asymptotic improvement compared to NT. These improvements depend directly on the confidence structure, the SMs implemented and the hyperparameter chosen.

Another evident aspect is how considering the action in confidence representation influences the agent's performances, QRND performs slightly better than SRND in few experiments in cart pole, this fact suggests that sometimes a representation of state-action confidence rather than a state confidence might be more effective. In some other cart pole experiments it was observable that the agent was starting to follow a certain convergence but 150 episodes are not always enough to find the right convergence. The purpose in cart pole environment is to observe how the agent changes its behaviour and if it enhances in the short period of 150 episodes using transferred experience from another agent.

Chapter 6

Conclusion and Future Work

This chapter contains the conclusive sections of the dissertation, the contribution of this work to the Research is summarised mentioning the addressed problems. The very last section presents some indications to possible future works.

6.1 Concluding Remarks

This dissertation presents a novel TL approach called Online Experience Sharing to improve the reinforcement learning process' speed and in general performance. The designed algorithm defines the transfer as a processes concerning two agents, a sender and a receiver. The transferred object consists in set of transitions representing agent's experience, transfer has been modelled as two-steps process in which first the sender selects the data to transfer and pushes them to receiver's specific buffer, transfer buffer - a structured inspired to q-learning's replay buffer that stores the exchanged transitions. Successively when the receiver samples the batch to update its DQN, it builds the structure starting partially from its replay buffer and the rest from the transfer buffer. The main points of the algorithm include the transfer frequencies and sizes handling the periodicity by inserting some parameters and a function to encourage the transfer at the beginning period of the agent's learning process - θ *function*. Another important aspect consists in establishing the transition selection criteria by both sender and receiver: the first prioritises data with stronger related knowledge, the most visited

ones, while the receiver prioritises newer data or with weaker related knowledge. In order to provide a numeric evaluation to the states' confidence a novel structure has been introduced: the confidence source structure. Three different scenarios with the related confidence structures and agent models have been introduced with the algorithm. The first implements a state visit table which keeps trace of the number of times that subsets of states have been visited, a second one uses a RND curiosity module which consists in two neural networks that arbitrarily encode the state and slowly decrease the difference in representation (uncertainty) adapting the predictor network's weights. Last, the same curiosity module has been implemented but encoding the action too, in order to calculate the uncertainty of a state-action pair instead of just the state. Multiple selection methods based on these confidence structure have been designed and they always exploit the idea of best knowledge priority by the sender and worst by the receiver.

This algorithm has been evaluated on benchmark environments mountain car and cart pole which although are between the most simple, the tasks are tricky since the policies to learn are not so immediate. The algorithm manages to encourage the exchange of significant transitions and in some particular configurations it brings interesting improvement in terms of TL metrics: speed-ups (which includes total reward earned), asymptotic improvements around the end of the simulation and in terms of learning stability. It is evident how a certain configuration improves these performance in a certain environment with a specific task but it does not produce the same improvements in the other environment, probably because the tasks, state-spaces and action-spaces are really important factors for the learning process, they are too relevant and keeping the same TL method is not enough the get the same outcome. Every task may need a different configuration of methods, confidence structure or hyperparameter to find the best TL improvements, the only possible solution seems to be to study the best configuration of parameters, confidence and selection methods to guarantee always the best result.

6.2 Limitations

Some details have not been considered to the realisation of this algorithm. Time, in particular, is represented by steps and episodes in this dissertation, so the computational cost and the overheads introduced by the communications among the agents have been considered irrelevant. The transfer frequencies and the agent model have been designed in order to minimise the number of effective communications but in TL the agents exchange data frequently, the fact is unavoidable. Another aspect of parallel computing that is not considered are the synchronisations among the processes which are not significant in the proposed agent designs. Some introduced selection methods have computational costs higher than other methods (for instance methods in SRND require more time than methods in VT), this aspect does not affect the learning speed since it is calculated in steps/episodes and not in effective time, for this reason it does not affect the described performances. The main advantage of not considering time as effective time is the possibility to ignore the power of the machine running the simulation.

6.3 Future Work

From the designed algorithm it is possible to carry out multiple further experiments aiming to test the introduced approach and find more potentialities and limits. The first set of these possibilities are quicker to implement and evaluate, the second part requires a more effort since they imply many changes of the developed framework.
The first set of quicker possible tests starts with the implementation of the algorithm in multi-agent systems in order to study if and how the behaviours can be different in agents interacting with each other. The most simple example might consider the implementation of a cooperative Predator-Prey environment to study separately how a collaboration among predators can take place and successively one among preys, but the most interesting experiment would be both the collaborations at the same time to check if one can change the other's performance and consequently the result of the

whole system. Another interesting test would be to find the improvements brought by a different relationship between senders and receivers, for instance for each receiver three senders could provide information. In environment with more complexity and with tougher tasks the effect might be amplified and more visible while in environments like mountain car and cart pole the policy is learnt relatively quickly. An interesting experiment involves the introduction of the role swap between agents, senders that become receiver and vice versa, which might be driven by some mechanism. Assuming that the sender should be the most skilled agent because it has to share its knowledge, the agent with more cumulative reward in last N episodes could be eligible as sender, while the others could be the direct receivers. In this way there would be a periodic role swap among senders and receivers, building a fair collaboration system.

The other set of tests, which require more time and effort to design and implement, includes the possibility to study a change in transferred object from set of transition to any other element, for instance they could be sub-trajectories of transitions to try to suggest not only a single state as confident but maybe few consecutive states and the related selected actions. Another possibility could be to transfer directly neural network's weights or any other detail concerning the neural networks to promote an exchange of policy representation instead of advice about experience. Another possible experiment could involve the study of the clear effect of transferred data, for example enabling few episodes of self-exploration of the state-space for each agent and successively preventing the receiving agent to interact autonomously with the environment. Every following interaction could be determined only by the transferred data in order to see exactly the transfer's effect with no bias. In this way the agent would not maximise the speed or the performance but it would make visible the clear effect of transfers directly on networks and on the agent.

From the experiments in QRND scenarios it is evident how the evaluation of state-action pair sometimes is turned out more effective then the representation of the only state. For this reason, the exploration of approaches and strategies concerning state-action pairs seems to be a good pathway given the results of some OES experiments, to get a confidence for data selection instead of considering only the state.

In light of the results achieved by the novel transfer methods proposed in this dissertation, experience sharing has proved better performance, as overall, compared to a no-transfer scenario. Furthermore, the proposed techniques overcome the cost of exploring characteristic of some tasks, in fact, exploration could be very expensive due to environment dynamics, such as huge state space or wide set of actions, or whenever a fail of the agent would result in waste of time. Eventually, to maximise the impact of the transfers both sender and receiver need to select the interactions according to some metrics as investigated in this work to improve the transfer impact through the quality of transferred data.

Appendix A

Additional Evaluation Charts

A.1 Experiment Charts

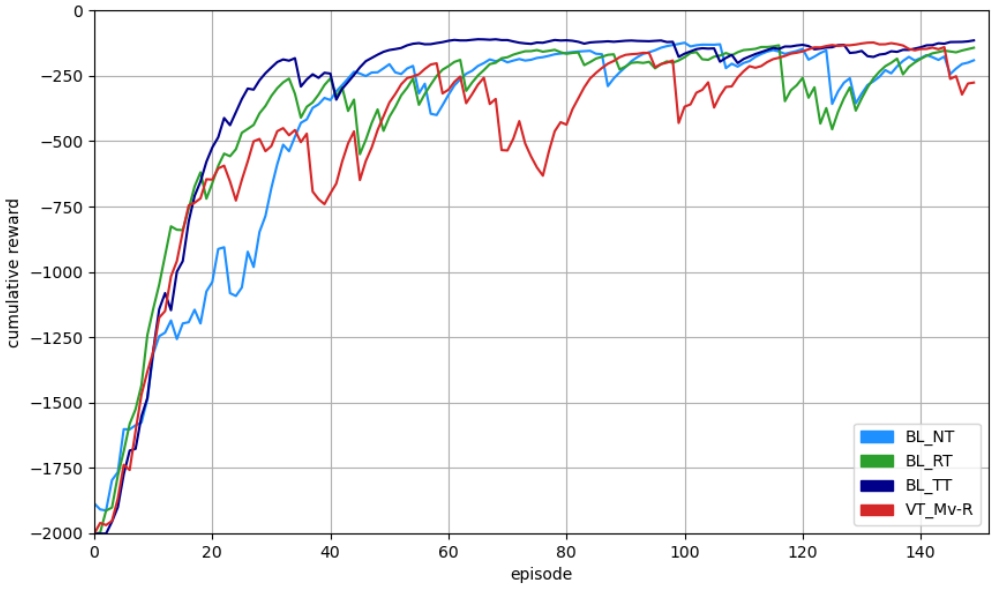


Figure A.1: Mountain Car VT Mv-R selection methods first 150 ep.

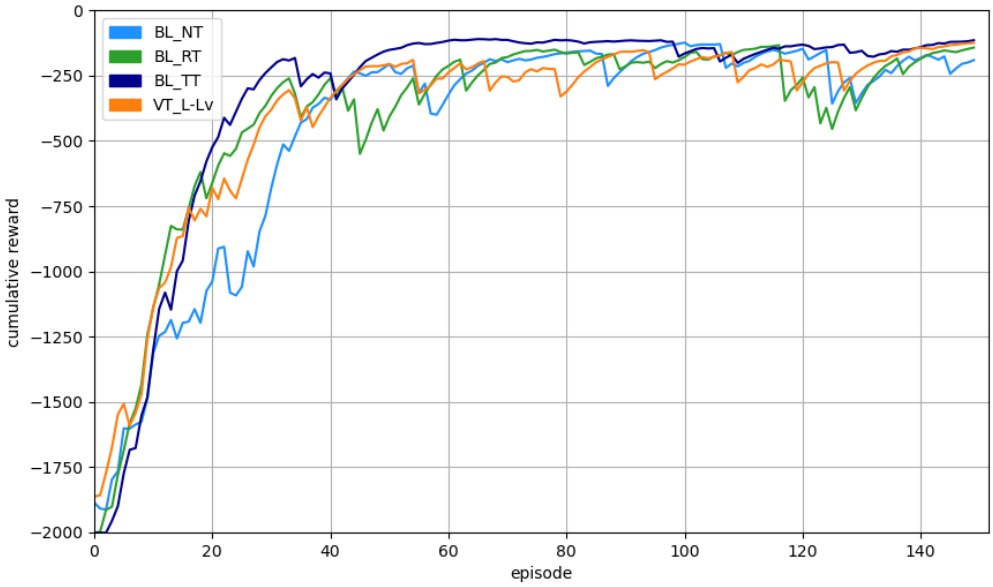


Figure A.2: Mountain Car VT L-Lv selection methods first 150 ep.

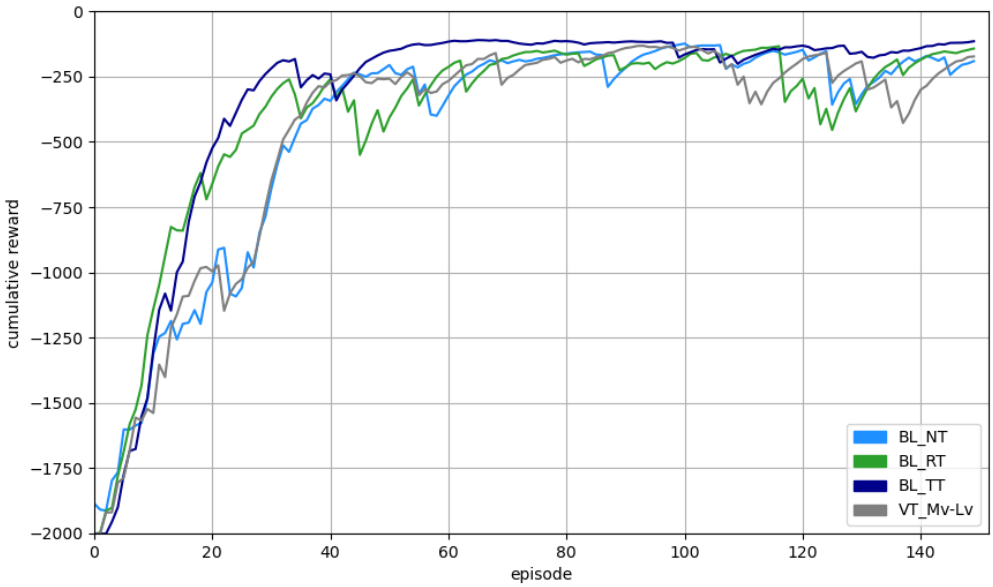


Figure A.3: Mountain Car VT Mv-Lv selection methods first 150 ep.

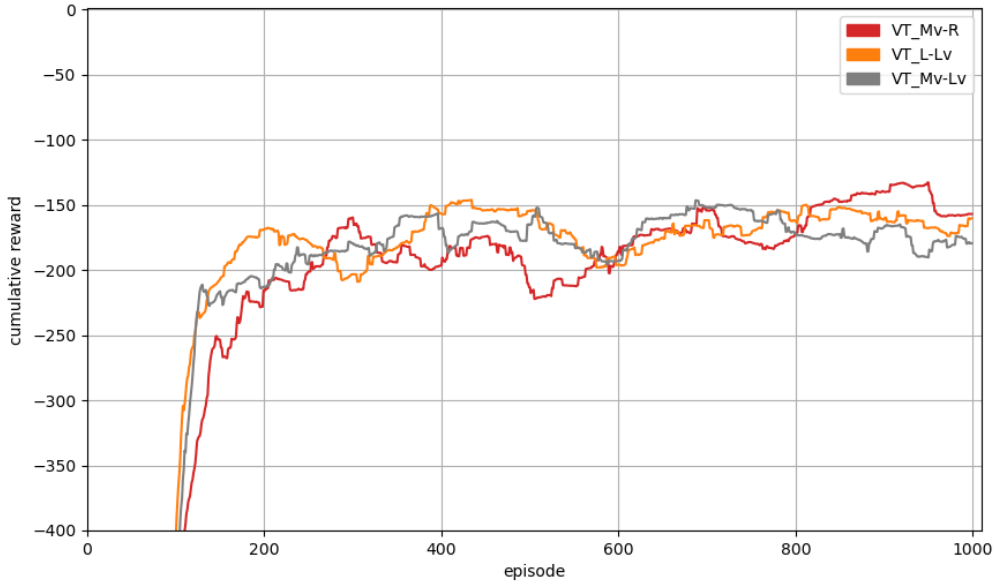


Figure A.4: Mountain Car VT all the selection methods

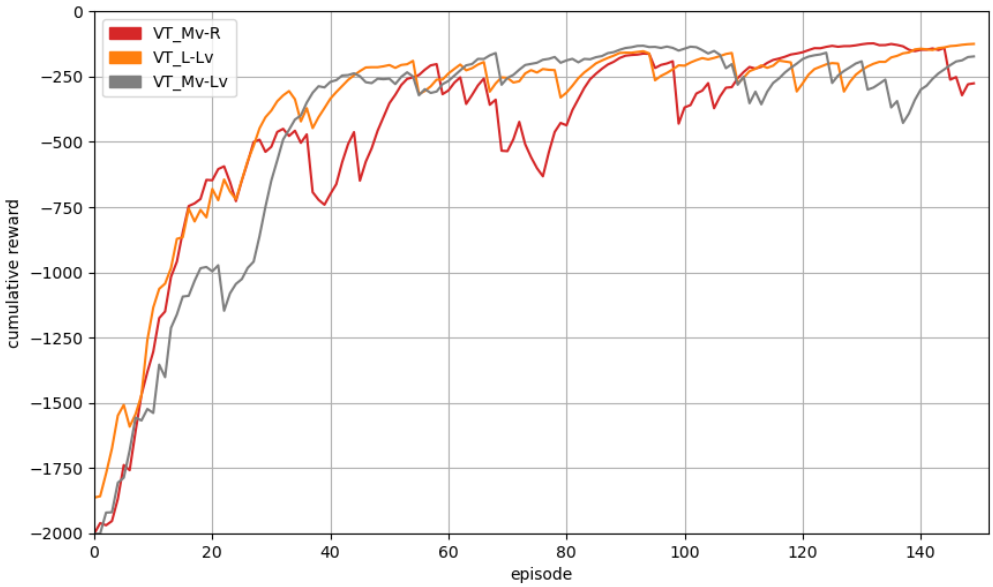


Figure A.5: Mountain Car VT all the selection methods first 150 ep.

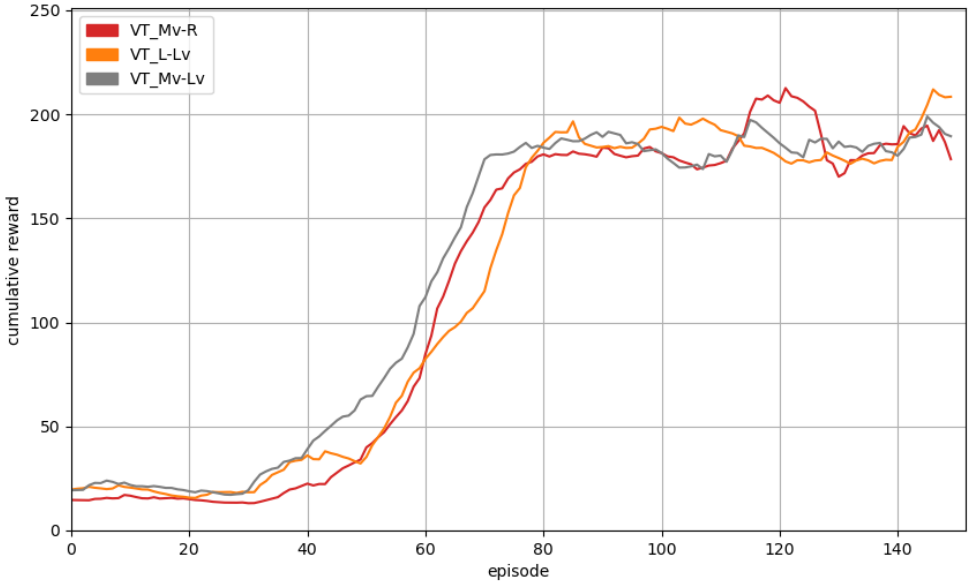


Figure A.6: Cart Pole VT all the selection methods

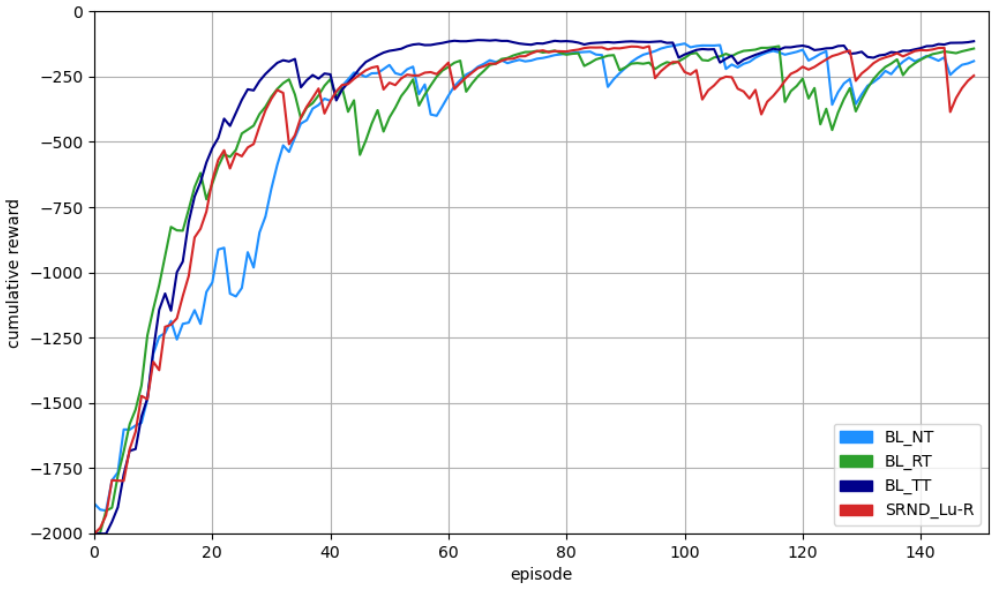


Figure A.7: Mountain Car SRND Lu-R selection methods first 150 ep.

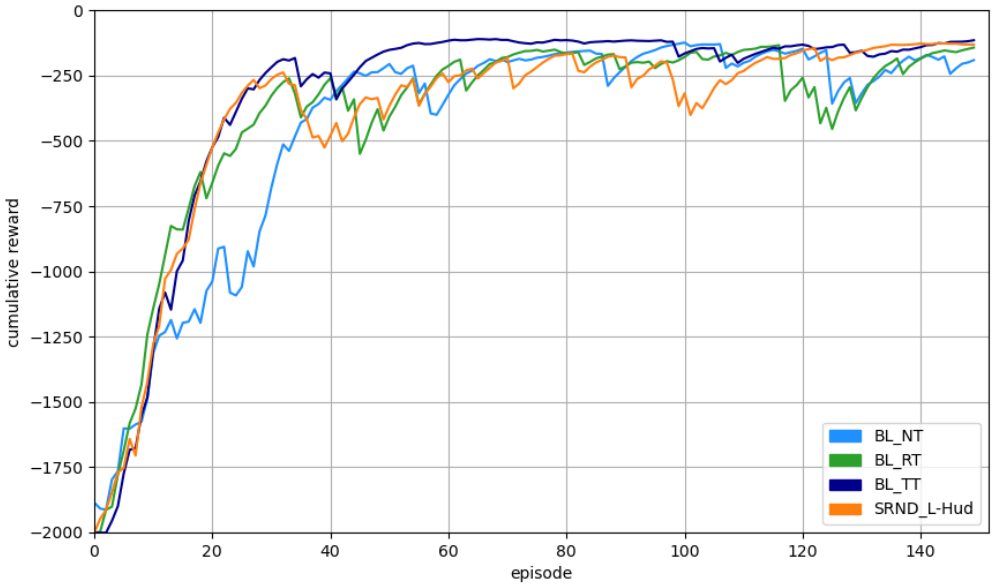


Figure A.8: Mountain Car SRND L-Hud selection methods first 150 ep.

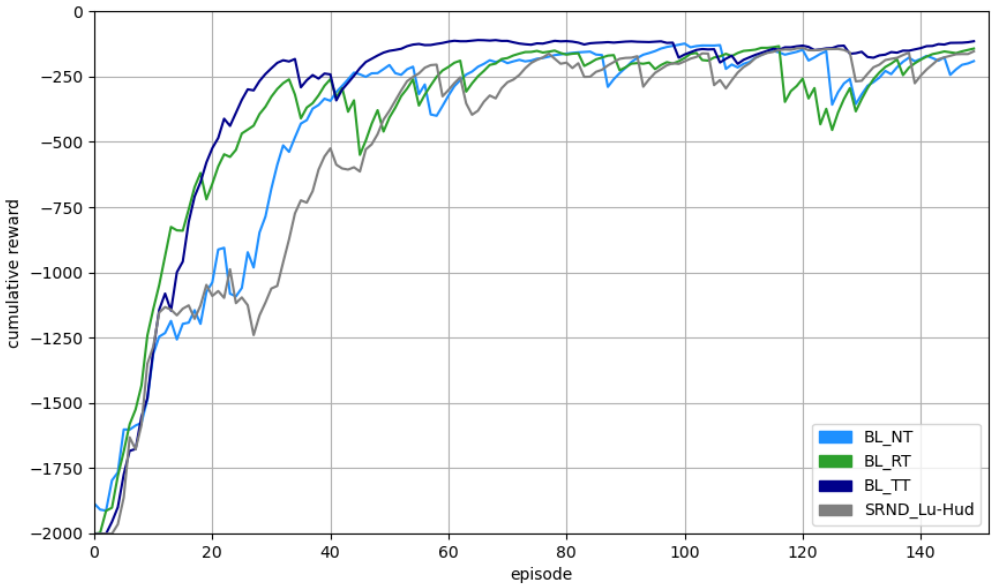


Figure A.9: Mountain Car SRND Lu-Hudselection methods first 150 ep.

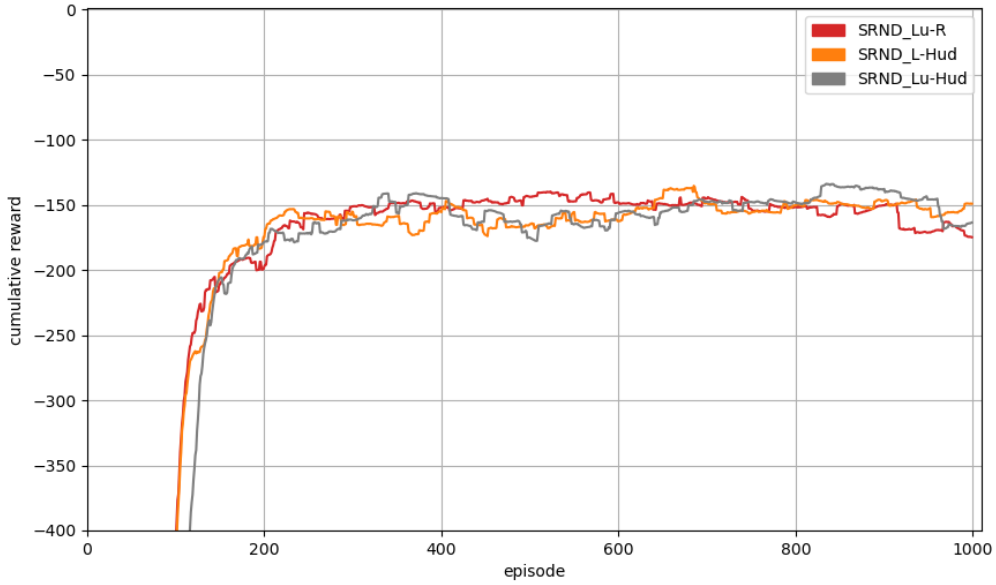


Figure A.10: Mountain Car SRND all the selection methods

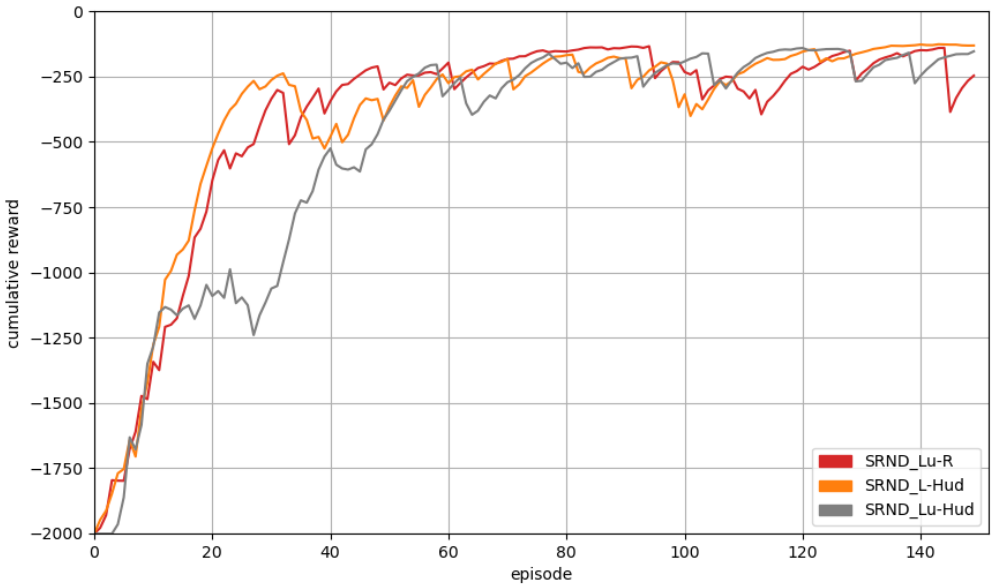


Figure A.11: Mountain Car SRND all the selection methods first 150 ep.

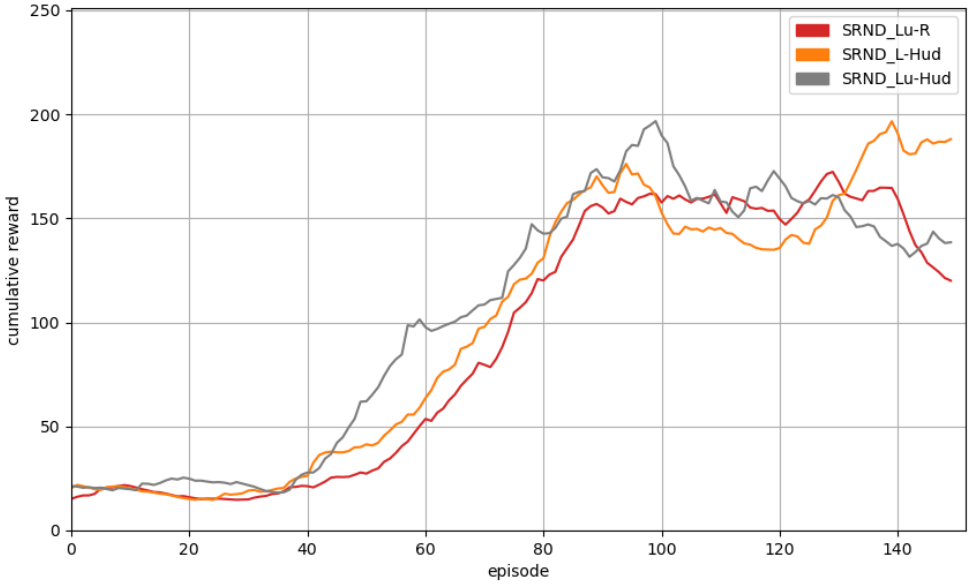


Figure A.12: Cart Pole SRND all the selection methods

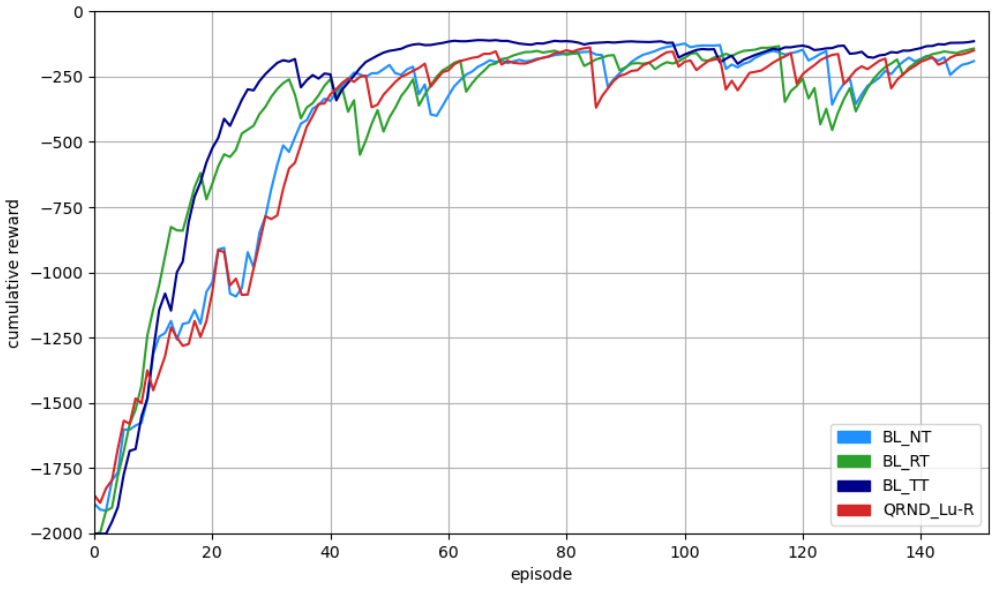


Figure A.13: Mountain Car QRND Lu-R selection methods first 150 ep.

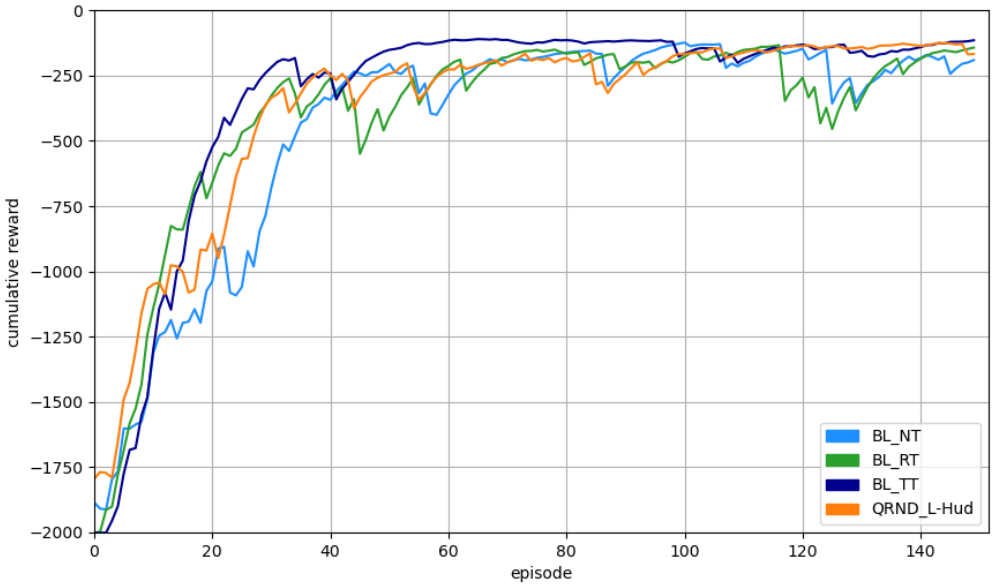


Figure A.14: Mountain Car QRND L-Hud selection methods first 150 ep.

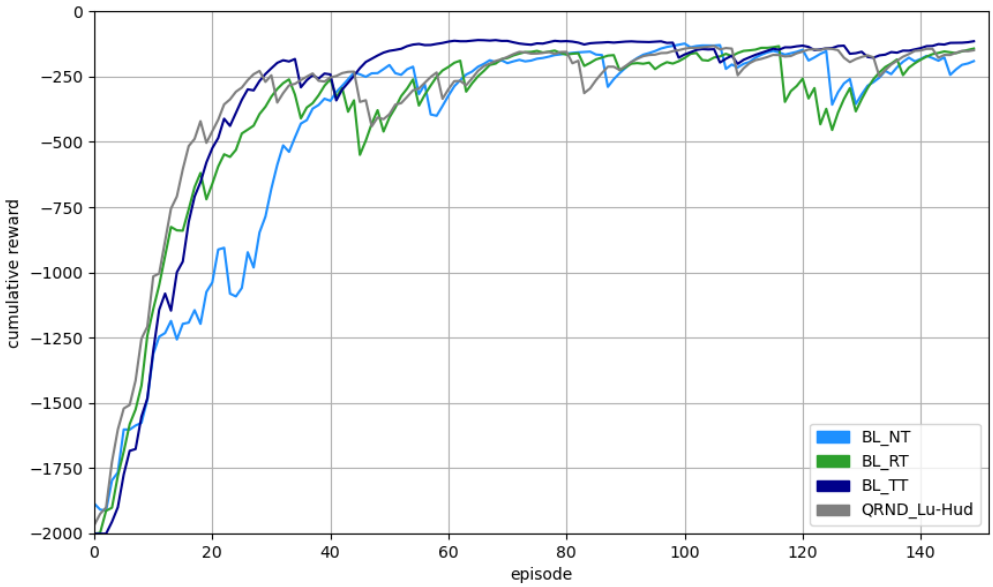


Figure A.15: Mountain Car QRND Lu-Hud selection methods first 150 ep.

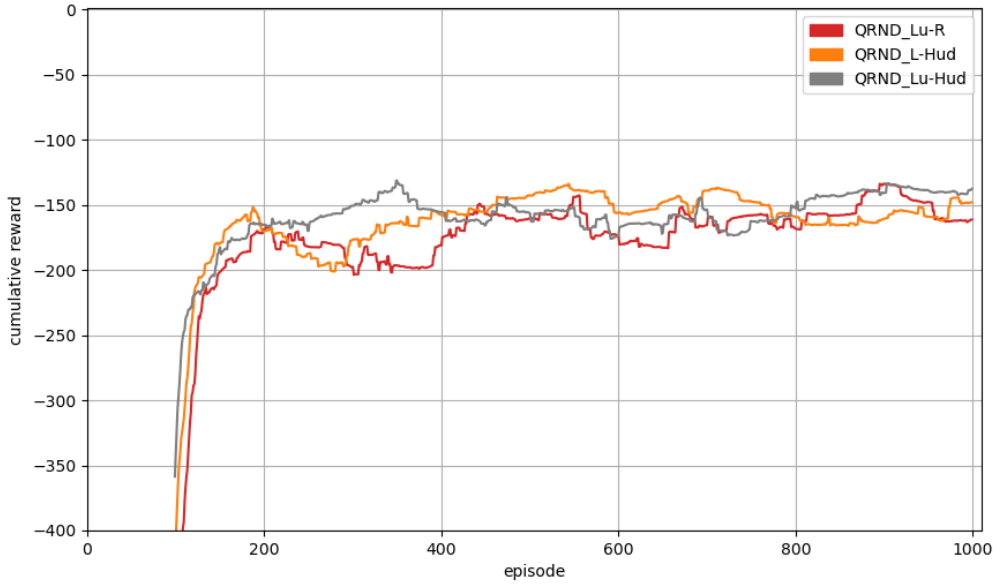


Figure A.16: Mountain Car QRND all the selection methods

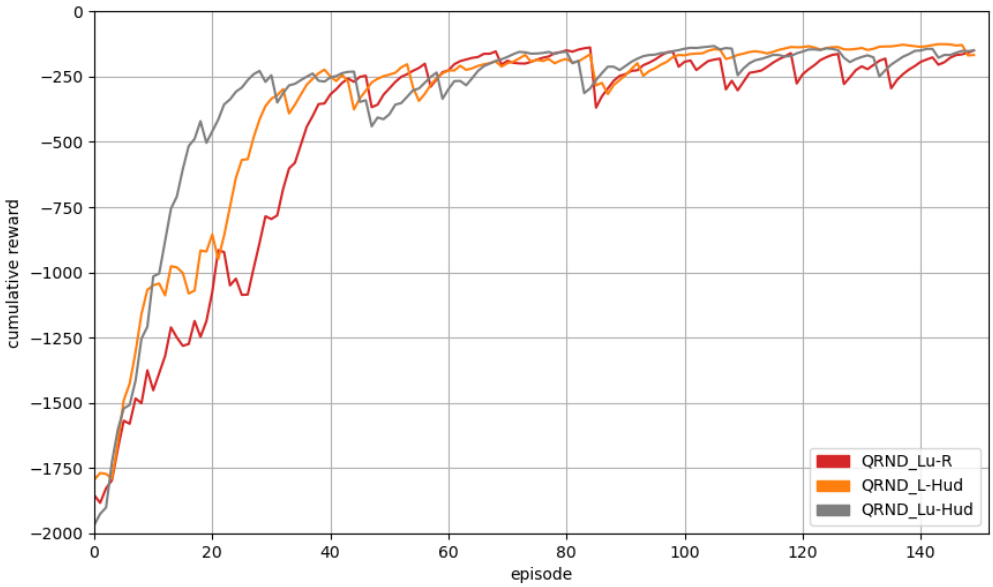


Figure A.17: Mountain Car QRND all the selection methods first 150 ep.

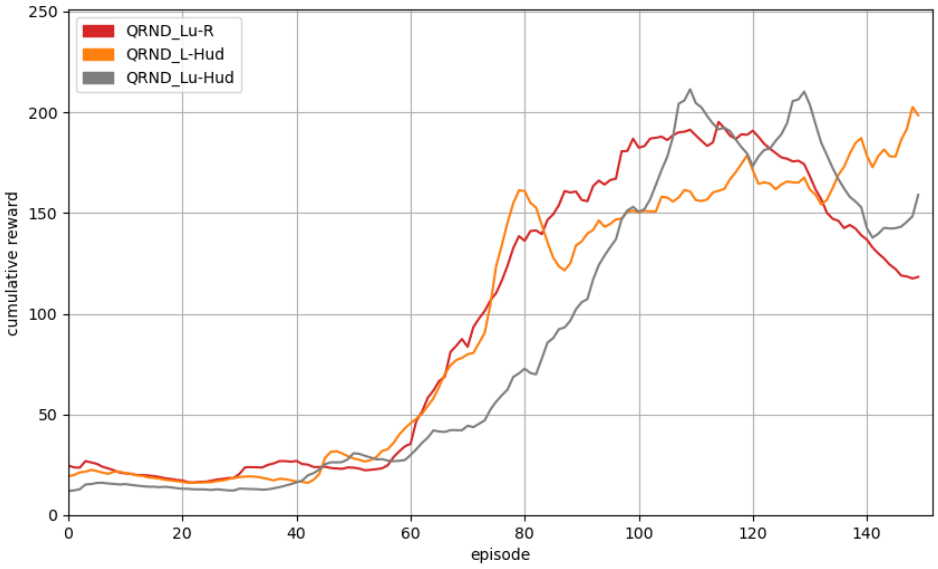


Figure A.18: Cart Pole QRND all the selection methods

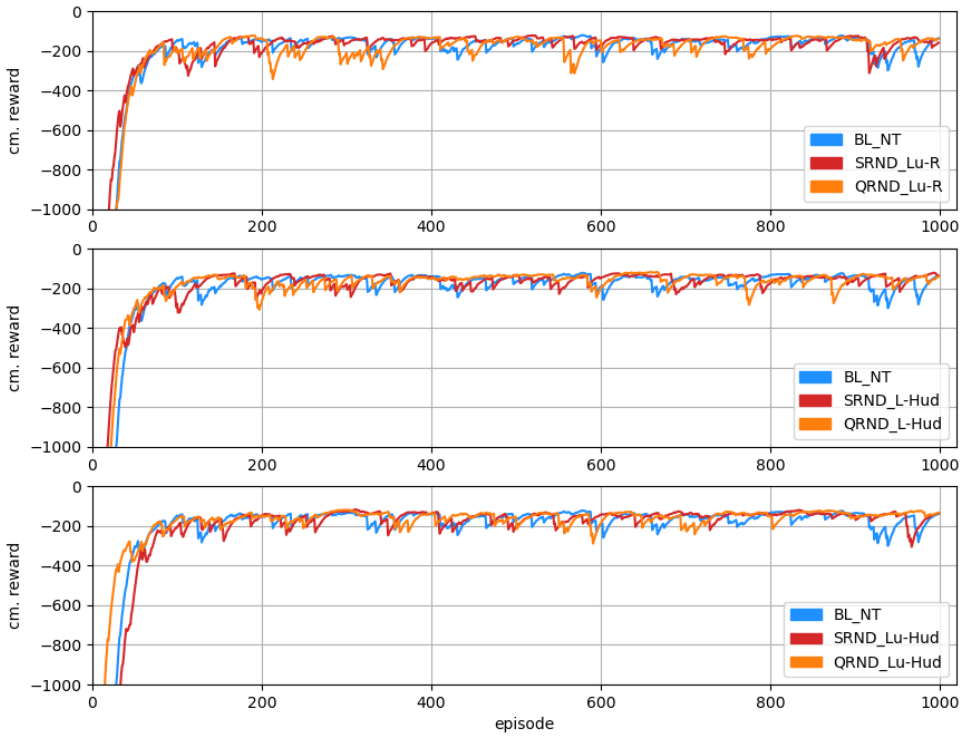


Figure A.19: Mountain Car comparison between every scenario of SRND and QRND

A.2 Transfer parameters Tuning Charts

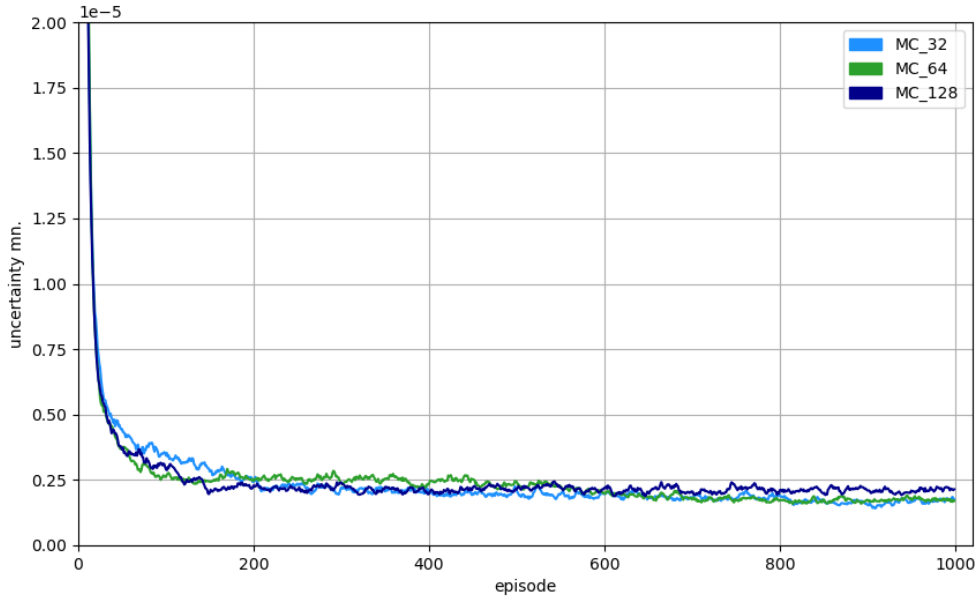


Figure A.20: Mountain Car RND features size sampling

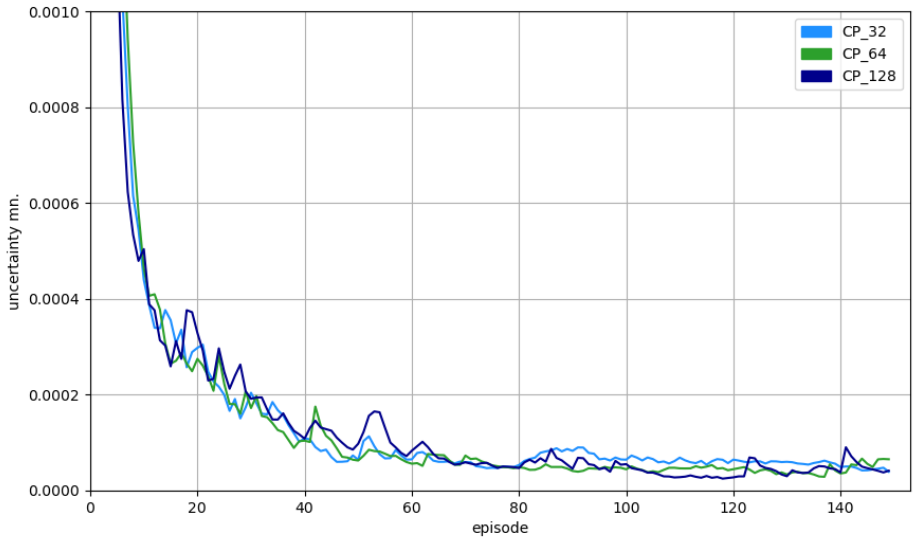


Figure A.21: Cart Pole RND features size sampling

Figure A.22: Transfer parameters tuning, 25 iterations of Bayesian optimisation using "Gaussian Process" as surrogate model

Bibliography

[1] Julie A. Adams. "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence". In: *AI Magazine* 22.2 (2001), pp. 105–108. URL: http://www.aaai.org/ojs/index.php/aimagazine/article/view/1567.

[2] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pp. 5048–5058.

[3] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2 (May 2002), pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352. URL: https://doi.org/10.1023/A:1013689704352.

[4] Daniel S Bernstein, Shlomo Zilberstein, and Neil Immerman. *The Complexity of Decentralized Control of Markov Decision Processes*. 2013. arXiv: 1301.3836 [cs.AI].

[5] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995. ISBN: 0198538642.

[6] Craig Boutilier. "Planning, Learning and Coordination in Multiagent Decision Processes". In: TARK '96. The Netherlands: Morgan Kaufmann Publishers Inc., 1996, pp. 195–210. ISBN: 1558604179.

[7] Justin A. Boyan and Andrew W. Moore. "Generalization in Reinforcement Learning: Safely Approximating the Value Function". In: *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*.

1994, pp. 369–376. URL: http://papers.nips.cc/paper/1018-generalization-in-reinforcement-learning-safely-approximating-the-value-function.

[8] Yuri Burda et al. *Exploration by Random Network Distillation*. 2018. arXiv: 1810.12894 [cs.LG].

[9] Shay Bushinsky. "Deus Ex Machina — A Higher Creative Species in the Game of Chess". In: *AI Magazine* 30.3 (2009), pp. 63–70. DOI: 10.1609/aimag.v30i3.2255. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/2255.

[10] Alberto Castagna et al. "Demand-Responsive Zone Generation for Real-Time Vehicle Rebalancing in Ride-Sharing Fleets". In: Apr. 2020.

[11] Rémi Coulom. "Reinforcement Learning Using Neural Networks, with Applications to Motor Control". PhD thesis. Institut National Polytechnique de Grenoble, 2002.

[12] Li Deng and Dong Yu. "Deep Learning: Methods and Applications". In: *Foundations and Trends in Signal Processing* 7.3-4 (2014), pp. 197–387. DOI: 10.1561/2000000039. URL: https://doi.org/10.1561/2000000039.

[13] I. Dusparic and V. Cahill. "Distributed W-Learning: Multi-Policy Optimization in Self-Organizing Systems". In: *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. 2009, pp. 20–29. DOI: 10.1109/SASO.2009.23.

[14] Ivana Dusparic and Vinny Cahill. "Autonomic Multi-Policy Optimization in Pervasive Systems: Overview and Evaluation". In: *ACM Trans. Auton. Adapt. Syst.* 7.1 (May 2012). ISSN: 1556–4665. DOI: 10.1145/2168260.2168271. URL: https://doi.org/10.1145/2168260.2168271.

[15] Pierre Yves Glorennec. "Reinforcement learning: An overview". In: *European Sym. on Intelligent Techniques*. Vol. 2000. Citeseer. 2000, pp. 17–35.

[16] Daniel Golovin et al., eds. *Google Vizier: A Service for Black-Box Optimization*. 2017. URL: http://www.kdd.org/kdd2017/papers/view/google-vizier-a-service-for-black-box-optimization.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[18] Geoffrey J. Gordon. "Reinforcement Learning with Function Approximation Converges to a Region". In: *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*. 2000, pp. 1040–1046. URL: http://papers.nips.cc/paper/1911-reinforcement-learning-with-function-approximation-converges-to-a-region.

[19] Geoffrey J. Gordon. "Stable Function Approximation in Dynamic Programming". In: *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*. 1995, pp. 261–268. DOI: 10.1016/b978-1-55860-377-6.50040-2. URL: https://doi.org/10.1016/b978-1-55860-377-6.50040-2.

[20] Carlos Ernesto Guestrin. "Planning under Uncertainty in Complex Structured Environments". AAI3104233. PhD thesis. Stanford, CA, USA, 2003.

[21] Hado van Hasselt. "Reinforcement Learning in Continuous State and Action Spaces". In: *Reinforcement Learning*. 2012, pp. 207–251. DOI: 10.1007/978-3-642-27645-3_7. URL: https://doi.org/10.1007/978-3-642-27645-3%5C_7.

[22] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN: 0132733501.

[23] Leemon C. Baird III. "Residual Algorithms: Reinforcement Learning with Function Approximation". In: *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*. 1995, pp. 30–37. DOI: 10.1016/b978-1-55860-377-6.50013-x. URL: https://doi.org/10.1016/b978-1-55860-377-6.50013-x.

[24] Ercüment İlhan, Jeremy Gow, and Diego Perez-Liebana. *Teaching on a Budget in Multi-Agent Deep Reinforcement Learning*. 2019. arXiv: 1905.01357 [cs.MA].

[25] Tommi S. Jaakkola, Michael I. Jordan, and Satinder P. Singh. "On the Convergence of Stochastic Iterative Dynamic Programming Algorithms". In: vol. 6. 6. 1994, pp. 1185–1201. DOI: 10.1162/neco.1994.6.6.1185. URL: https://doi.org/10.1162/neco.1994.6.6.1185.

[26] Maxim Lapan, ed. *Deep Reinforcement Learning Hands-On*. Packt Publishing Ltd, 2018. ISBN: 978-1-83882-699-4.

[27] Alessandro Lazaric. "Transfer in Reinforcement Learning: a Framework and a Survey". In: *Reinforcement Learning - State of the art*. Ed. by Martijn van Otterlo Marco Wiering. Vol. 12. Springer, 2012, pp. 143–173. DOI: 10.1007/978-3-642-27645-3_5. URL: https://hal.inria.fr/hal-00772626.

[28] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016. URL: http://arxiv.org/abs/1509.02971.

[29] Andrei Andreevich Markov. "The theory of algorithms". In: *Trudy Matematicheskogo Instituta Imeni VA Steklova* 42 (1954), pp. 3–375.

[30] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[31] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv e-prints*, arXiv:1312.5602 (Dec. 2013), arXiv:1312.5602. arXiv: 1312.5602 [cs.LG].

[32] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. "Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications". In: *IEEE Transactions on Cybernetics* 50.9 (Sept. 2020), pp. 3826–3839. ISSN: 2168-2275. DOI: 10.1109/tcyb.2020.2977374. URL: http://dx.doi.org/10.1109/TCYB.2020.2977374.

[33] Vassilis A. Papavassiliou and Stuart J. Russell. "Convergence of Reinforcement Learning with General Function Approximators". In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*. 1999, pp. 748–757. URL: http://ijcai.org/Proceedings/99-2/Papers/014%5C%20.pdf.

[34] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2003. ISBN: 0130803022. URL: http://www.worldcat.org/oclc/314283679.

[35] S. Sathiya Keerthi and B. Ravindran. "A tutorial survey of reinforcement learning". In: *Sadhana* 19.6 (Dec. 1994), pp. 851–889. ISSN: 0973-7677. DOI: 10.1007/BF02743935. URL: https://doi.org/10.1007/BF02743935.

[36] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144. ISSN: 0036-8075. DOI: 10.1126/science.aar6404. eprint: https://science.sciencemag.org/content/362/6419/1140.full.pdf. URL: https://science.sciencemag.org/content/362/6419/1140.

[37] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

[38] Peter Stone and Richard S. Sutton. "Scaling Reinforcement Learning toward RoboCup Soccer". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 537–544. ISBN: 1558607781.

[39] Richard S. Sutton. "Learning to Predict by the Methods of Temporal Differences". In: *Machine Learning* 3 (1988), pp. 9–44. DOI: 10.1007/BF00115009. URL: https://doi.org/10.1007/BF00115009.

[40] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[41] A. Taylor et al. “Parallel Transfer Learning in Multi-Agent Systems: What, when and how to transfer?” In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851784.

[42] Matthew E. Taylor and Peter Stone. “Transfer Learning for Reinforcement Learning Domains: A Survey”. In: *J. Mach. Learn. Res.* 10 (Dec. 2009), pp. 1633–1685. ISSN: 1532-4435.

[43] Gerald Tesauro. “Neurogammon Wins Computer Olympiad”. In: *Neural Computation* 1.3 (1989), pp. 321–323. DOI: 10.1162/neco.1989.1.3.321. URL: https://doi.org/10.1162/neco.1989.1.3.321.

[44] Gerald Tesauro. “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play”. In: *Neural Computation* 6.2 (1994), pp. 215–219. DOI: 10.1162/neco.1994.6.2.215. URL: https://doi.org/10.1162/neco.1994.6.2.215.

[45] Gerald Tesauro. “Temporal Difference Learning and TD-Gammon”. In: *ICGA Journal* 18.2 (1995), p. 88. DOI: 10.3233/ICG-1995-18207. URL: https://doi.org/10.3233/ICG-1995-18207.

[46] William R. Thompson. “On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples”. In: *Biometrika* 25.3/4 (1933), pp. 285–294. ISSN: 00063444. URL: http://www.jstor.org/stable/2332286.

[47] Michel Tokic. “Adaptive ε-Greedy Exploration in Reinforcement Learning Based on Value Differences”. In: *KI 2010: Advances in Artificial Intelligence: 33rd Annual German Conference on AI, Karlsruhe, Germany, September 21-24, 2010, Proceedings*. Vol. 6359. Springer Science & Business Media. 2010, p. 203.

[48] John N. Tsitsiklis. “Asynchronous Stochastic Approximation and Q-Learning”. In: *Machine Learning* 16.3 (1994), pp. 185–202. DOI: 10.1007/BF00993306. URL: https://doi.org/10.1007/BF00993306.

[49] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[50] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. King's College, Cambridge, 1989.

[51] Shangtong Zhang and Richard S. Sutton. "A Deeper Look at Experience Replay". In: *CoRR* abs/1712.01275 (2017). arXiv: 1712.01275. URL: http://arxiv.org/abs/1712.01275.