INSTITUTO
UNIVERSITÁRIO
DE LISBOA

my Human Brain Project (mHBP)

José António Guerreiro Nunes Sanches Salvador

Master's in Computer Science and Business Management

Supervisor:

PhD Maurício Breternitz Jr, Principal Researcher,

ISCTE

Co-Supervisor:

PhD João Pedro Afonso Oliveira da Silva, Assistant Professor,

ISCTE

September, 2021

TECNOLOGIAS
E ARQUITETURA

Departament of Information Science and Technology

my Human Brain Project (mHBP)

José António Guerreiro Nunes Sanches Salvador

Master's in Computer Science and Business Management

Supervisor:

PhD Maurício Breternitz Jr, Principal Researcher,

ISCTE

Co-Supervisor:

PhD João Pedro Afonso Oliveira da Silva, Assistant Professor,

ISCTE

September, 2021

To my family.

“Beware. We are in a pre scientific age in ML. We don’t yet have the right math. We are stumbling around in the dark, clutching at straws, waiting for our Darwin to show us the way…”

Sridhar Mahadevan

Acknowledgements

I would like to thank Professor Maurício Breternitz and Professor João Oliveira for all the guidance they provided in the making of this thesis.

Also a special thanks to Professor Luís Nunes for pointing out the existence of Hierarchical Reinforcement Learning, at the very beginning of the writing of this thesis, which led to a broader Literature Review and to an hierarchical approach in the creation of the mHBP architecture.

Lisbon, Sep 30th, 2021

José António Salvador

Abstract

How can we make an agent that thinks like us humans? An agent that can have proprioception, intrinsic motivation, identify deception, use small amounts of energy, transfer knowledge between tasks and evolve? This is the problem that this thesis is focusing on.

Being able to create a piece of software that can perform tasks like a human being, is a goal that, if achieved, will allow us to extend our own capabilities to a very high level, and have more tasks performed in a predictable fashion. This is one of the motivations for this thesis.

To address this problem, we have proposed a modular architecture for Reinforcement Learning computation and developed an implementation to have this architecture exercised. This software, that we call mHBP, is created in Python using Webots as an environment for the agent, and Neo4J, a graph database, as memory. mHBP takes the sensory data or other inputs, and produces, based on the body parts / tools that the agent has available, an output consisting of actions to perform.

This thesis involves experimental design with several iterations, exploring a theoretical approach to RL based on graph databases. We conclude, with our work in this thesis, that it is possible to represent episodic data in a graph, and is also possible to interconnect Webots, Python and Neo4J to support a stable architecture for Reinforcement Learning. In this work we also find a way to search for policies using the Neo4J querying language: Cypher. Another key conclusion of this work is that state representation needs to have further research to find a state definition that enables policy search to produce more useful policies.

The article “REINFORCEMENT LEARNING: A LITERATURE REVIEW (2020)” at Research Gate with doi 10.13140/RG.2.2.30323.76327 is an outcome of this thesis.

Keywords: reinforcement learning, sensorial data, actions as output

Resumo

Como podemos criar um agente que pense como nós humanos? Um agente que tenha propriocepção, motivação intrínseca, seja capaz de identificar ilusão, usar pequenas quantidades de energia, transferir conhecimento entre tarefas e evoluir? Este é o problema em que se foca esta tese.

Ser capaz de criar uma peça de software que desempenhe tarefas como um ser humano é um objectivo que, se conseguido, nos permitirá estender as nossas capacidades a um nível muito alto, e conseguir realizar mais tarefas de uma forma previsível. Esta é uma das motivações desta tese.

Para endereçar este problema, propomos uma arquitectura modular para computação de aprendizagem por reforço e desenvolvemos uma implementação para exercitar esta arquitetura. Este software, ao qual chamamos mHBP, foi criado em Python usando o Webots como um ambiente para o agente, e o Neo4J, uma base de dados de grafos, como memória. O mHBP recebe dados sensoriais ou outros inputs, e produz, baseado nas partes do corpo / ferramentas que o agente tem disponíveis, um output que consiste em ações a desempenhar.

Uma boa parte desta tese envolve desenho experimental com diversas iterações, explorando uma abordagem teórica assente em bases de dados de grafos. Concluímos, com o trabalho nesta tese, que é possível representar episódios em um grafo, e que é, também, possível interligar o Webots, com o Python e o Neo4J para suportar uma arquitetura estável para a aprendizagem por reforço. Neste trabalho, também, encontramos uma forma de procurar políticas usando a linguagem de pesquisa do Neo4J: Cypher. Outra conclusão chave deste trabalho é que a representação de estados necessita de mais investigação para encontrar uma definição de estado que permita à pesquisa de políticas produzir políticas que sejam mais úteis.

O artigo “REINFORCEMENT LEARNING: A LITERATURE REVIEW (2020)” no Research Gate com o doi 10.13140/RG.2.2.30323.76327 é um sub-produto desta tese.

Palavras-chave: aprendizagem por reforço, dados de sensores, ações como output

Contents

List of Figures

List of Tables

Acronyms

***Table 1**: Acronyms*

AGI	Artificial General Intelligence [38, p.16]
AI	Artificial Intelligence
ASI	Artificial Super Intelligence [38, p.16]
DRL	Deep Reinforcement Learning
FRL	Flat Reinforcement Learning
GAN	Generative Adversarial Network
HDSL	High-Dimensional Similarity Learning [87, p.5]
HRL	Hierarchical Reinforcement Learning
IM	Intrinsic Motivation
IRL	Inverse Reinforcement Learning
LSTM	Long Short Term Memory
MBRL	Model Based Reinforcement Learning
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MFRL	Model Free Reinforcement Learning
ML	Machine Learning
MRL	Meta Reinforcement Learning
NN	Neural Networks
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SARSA	State Action Reward State Action
SMDP	Semi-Markov Decision Process

1. Introduction

1.1. Motivation and Scope

Reinforcement Learning (RL) is an important area of study because it can enable society to automate tasks that we, in the past, never thought could be automated. Autonomous driving vehicles are one of the uses for RL. Other uses may comprise having robots that can perform tasks like preparing ingredients and cooking food, with the same robot preparing several different dishes without any human intervention or any specific programming for the executed tasks. Stock market trading can also be a task executed by a RL agent, that instead of being programmed with specific rules, can learn the rules of best trading by itself.

Producing a human-thinking like system has been one thought sticking for several years. The idea of having systems that can act and decide like we humans do is not only an awesome idea but also something that could enhance our own productivity by giving us a tool that performs tasks for us and therefore allows us to produce more in less time and with less effort.

If every neuron or equivalent structure in a human was accurately simulated in a computer, would it result in human-like consciousness? This is a question that also motivates us to explore this theme.

How much can a system be able to act as a conscious being, in a way that is believably natural? This thesis does not give an answer to this, but moves in the direction that allows us to, possibly, come to an answer to this question in the future.

The scope of this work is not to produce a system that is biologically plausible, but rather a system that is biologically inspired, not in its structure, but in the outcome it is able to produce.

Lastly a motivation and also part of the scope of this research is to be able to have a system that can fit into a small hardware using low computational power, making it more independent of a large computation infrastructure, so that it can be easily incorporated in an autonomous physical agent, without the need to be connected to a supercomputer or a cluster of computers.

This thesis achieves three goals. The 1st one is an architecture and framework to be able to test RL algorithms, the 2nd one is a proposal of episode and policy representations in a graph database, and the 3rd one is a proposal for an algorithm to search for similar states in a graph database.

The proposed mHBP architecture (Figure 26) is divided into:

- a virtual environment – used for testing of the mHBP RL agent.
- a graph database – used as memory to hold episodic data and as an interface between the RL Engines and the Controller.
- a Controller – used to gather perceptions and execute actions of the agent in the environment.
- a Babbling Engine – used to create simple macro-actions for exploration.
- a Brainstorm Engine – used to create more complex macro-actions for exploration.
- a Decision Engine – used to create decisions by exploiting gathered knowledge.

1.1.1. RL Application Domains

Any activity that has real-time decision making based on data, is a candidate for the application of RL. Here are some examples of potential applications of RL, amongst many others:

- Autonomous driving vehicles [138];
- Stock trading [139, 141];
- Ad serving and Marketing [65, p.451];
- Games AI [136, 144];
- Traffic Light Control [142];
- Robotics and Industry automation [140];
- Websites Personalized Recommendations of products and content [65, p.451];
- Bidding [145];
- Healthcare [143].

1.2. Preliminary Concepts

1.2.1. Machine Learning

Machine learning (ML) is a knowledge area that is divided into four main groups: Supervised Learning, Unsupervised Learning, Intrinsic Motivation (IM) and RL [56]. This thesis is mainly related to the later, but at some point there may be references to techniques of Unsupervised or/and Supervised Learning (in situations when sensorial data may be processed in batches where these techniques are potentially applicable and useful for state labeling or clustering) or/and IM.

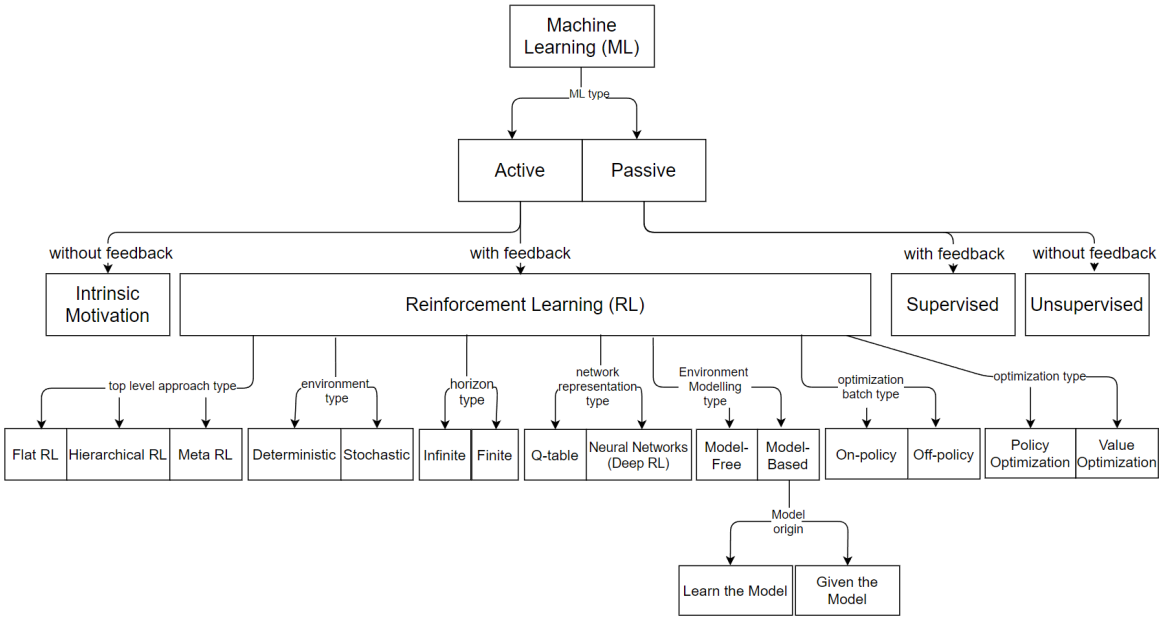


***Figure 1**: A diagram of Reinforcement Learning context and techniques.*

As seen in Figure 1, RL is considered an active type of ML [56]. IM is also considered as an active type of ML [56] but differs from RL because it has no feedback mechanism from a supervisor, which RL has. The feedback mechanism is also what creates a difference between Supervised and Unsupervised ML. While [56] considers IM as a separate field of ML, several works of RL use IM as a strategy to cope with complex, sparse reward environments.

Symbolic AI is one of the early approaches to create artificial intelligence reinforcement learning systems, giving later place to a stronger different stream of thought fundamentally focused on a non-symbolic neural network approach.

More recently some works propose a mix of symbolic and Deep Neural Networks, listing limitations of Deep Reinforcement Learning (DRL). In [85] a system like this is proposed, in order to overcome DRL shortcomings, such as the requirement of large datasets to work with, slow learning pace, inability of abstract, analogical and hypothesis-based reasoning and opaqueness to humans, not allowing easy verifiability. Hierarchical Reinforcement Learning (HRL) like seen in [80] is also an example of the effort to combine DRL with Symbolic AI. Perhaps this is the direction science can go to achieve more than the current level of RL, considered as Strict AI.

In theory RL can achieve a state of Artificial General Intelligence (AGI) [38, p.16], or even further, of Artificial Super Intelligence (ASI) [38, p.16], being the highest goal to have an agent perform any task any human performs but faster and with more precision.

As Greg Brockman, co-founder of OpenAI, writes in a statement as part of a hearing at the United States Senate, there is a movement from an era of narrow AI systems to AGI systems. Narrow AI systems typically do one type of task very well, like categorize an image, transcribe a speech, or play a computer game well. AGI systems will have different capabilities; they will be able to solve many tasks and improvise new solutions when they run into some kind of difficulty [37, p.4].

While surveys of RL algorithms are usually exclusive, either of Flat Reinforcement Learning (FRL) [91, p.1], Hierarchical Reinforcement Learning (HRL) [80, p.1] or Meta Reinforcement Learning (MRL) [128, p.1], they all propose to solve the problem of optimal control. Figure 2 shows, in a simplified way, a comparison between FRL, HRL and MRL methods. While FRL contains a core we call the “planner” that processes the observations and rewards to produce micro-actions (actions that are atomic and not divisible into sub-actions), HRL combines a planner level with a “Skills” level that we can also call macro-actions or options. Skills are a combination of micro-actions that can be used by the planner. In HRL the planner can also choose to perform micro-actions combined with macro-actions or other sub-planners. The planning with micro-actions may have a dimensionality higher weight, so it may also involve a higher computational cost. MRL is an approach that focuses on creating agents that learn how to learn, and that can model concepts. While we separate skills from concepts in the MRL sketch, a skill can be seen as a concept of an action, if we look at the adaptation a skill may get for every similar type of action.

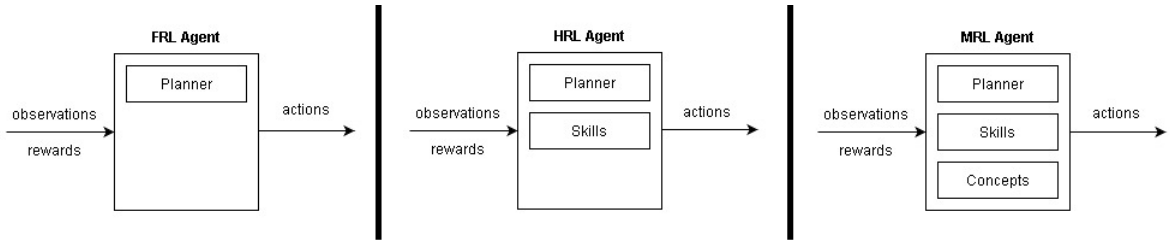


***Figure 2**: Flat Reinforcement Learning (FRL) vs Hierarchical Reinforcement Learning (HRL) vs. Meta Reinforcement Learning (MRL).*

1.2.2. Neural Inspired Computing

RL has been evolving in the last two decades to DRL, an approach that is supported in Deep Neural Networks, hence the name: Deep RL. This later philosophy is based on the usage of Neural Networks (NN), a neuromorphic structure that was conceived as an attempt to simulate the way our biological brains work.

DRL's future is closely linked to the developments of NN. The Curse of Dimensionality (see Glossary) is one of the limitations of tabular RL, and therefore NN started to be used as function approximators (see Glossary) and in this way cope with intractable optimal control problems [65] .

Brain-inspired devices, models and computers that contrast with the pervasive John von Neumann computer architecture, have been progressively known as Neuromorphic Computing (NC) [1, p.1]. NC is tightly coupled with NN based ML, because NC seems to be the platform better suited to implement future NN based ML algorithms [1, p.1-2].

Figure 3, shows some NN topologies, but these are only a small number of examples that can be considered when implementing a NN.

***Figure 3**: Some topologies that might be desired for neuromorphic systems [1, p.8].*

Figure 4 (next page) gives us a wider perspective of the possibilities of NN and also a view into the volume of papers that addressed each type of topology. Spiking NN are one type of NN that have more recently captured the attention of researchers because of the performance increase over GPU friendly NN [1, p.21]. ML algorithms have to evolve and integrate this new concept of Spiking NN to take better advantage of spiking able neuromorphic hardware [1, p.21].

The ML field greatly benefits from advancements on NC, not only in regard to Spiking NN but also with neuromorphic hardware architectures that by being designed to increase NN speed of computation can open the way for better performance of NN based ML algorithms [1, p.2].

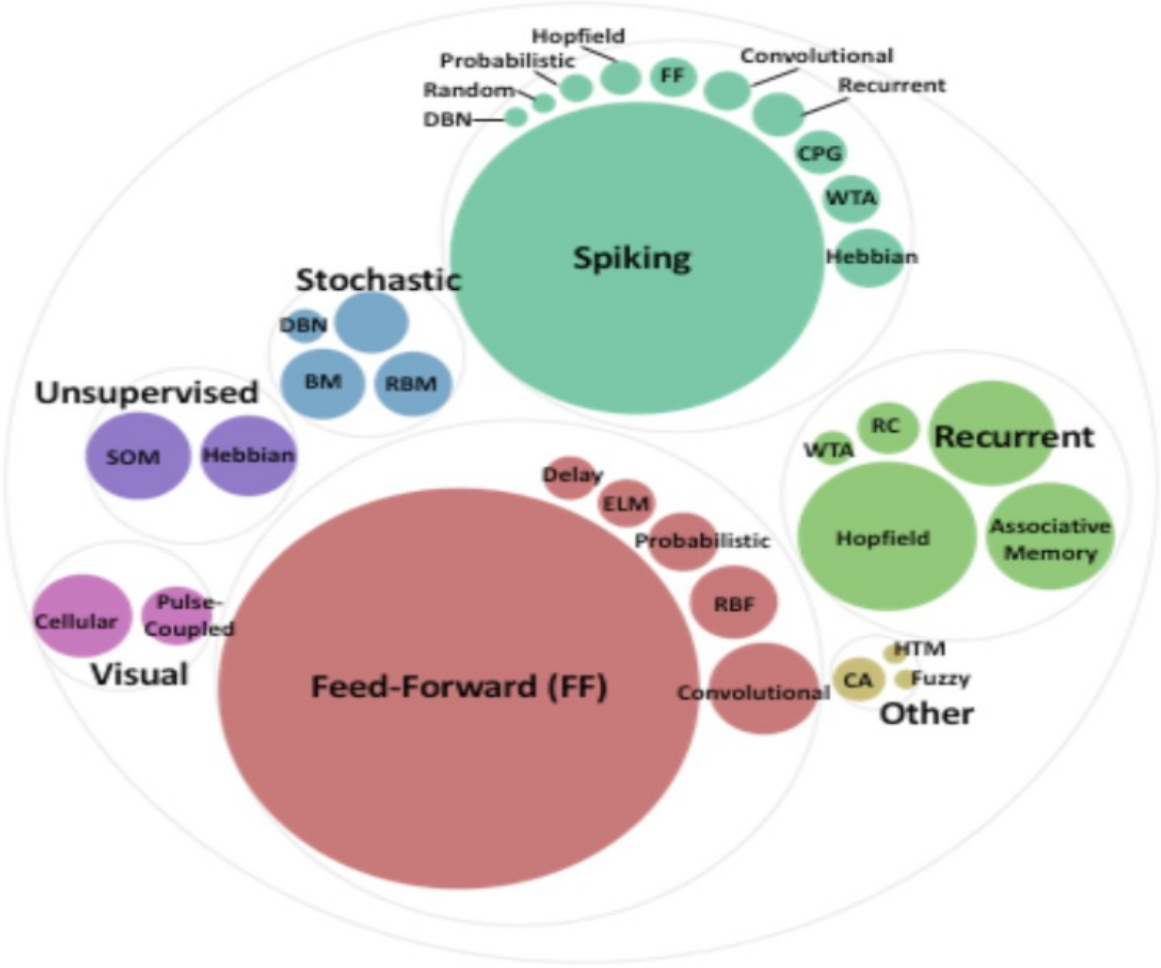


***Figure 4**: Neural Network models, grouped by type and sized by volume of neuromorphic field papers addressing each one [1, p.9].*

1.3. Research Problems & Question

The main problem addressed by this thesis is the optimal control of an agent that:

- can be fast enough to be used in real time;
- has low power consumption;
- Is able to run on limited hardware;
- can evolve and be able to learn and transfer learning between similar tasks;
- can learn its own and the environment dynamics;
- has proprioception,
- Is able to identify deception.

This problem leads us to the main **Research Question**:

How can we produce an agent that is capable of addressing all of the above requirements ?

1.4. Contributions

This work aims to make the following contributions:

- A reference tool of RL works with a chronological layout;
- A comparison between an ad-hoc literature review and a Systematic Literature Review (SLR);
- A Firefox extension to obtain results from Google Scholar without getting a blocked ip address;
- An architecture to implement Reinforcement Learning algorithms;
- A Graph representation of episodes and policies, as well, prior knowledge of the Agent;
- A method to retrieve and use information in the graph to perform Reinforcement Learning.

1.5. Dissertation Structure

Figure 5 presents how this dissertation is structured into 5 main parts.

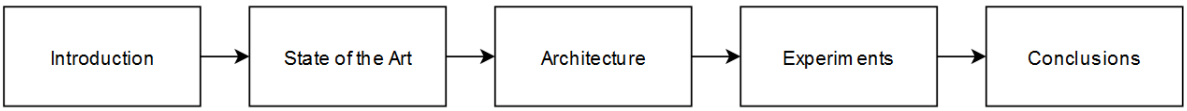


***Figure 5**: Dissertation structure diagram.*

"**Introduction**" (the current section) establishes the motivation to write this thesis, defines what RL is, where it is useful, how it is approached and which problems need to be addressed in the RL field.

The "**State of the Art**" section is where the methodology to search for RL literature is described, and where results of this search are presented and analyzed.

"**Architecture"** is the part where a RL system structure is presented and the **"Experiments**" part describes the experiments with that structure.

"**Conclusions**" presents the results of the experiments and confronts them with the state of the art This section also describes possible future work that is not addressed in this thesis.

2. State of the Art (SotA)

2.1. Introduction for SotA

This chapter presents:

- a set of questions to be answered by the literature search;
- a search methodology outline;
- a compilation of the latest work done in RL, where we present a list of algorithms published from 1989 to 2020. The oldest algorithm of this list is Q-learning, dated 1989;
- a summarization of the main aspects we can learn from the papers found;;
- the answers to the search questions.

Search is done mainly through the tracking of publications of RL algorithms throughout the last 31 years, starting in 1989. Algorithms of this period distance themselves a lot from the philosophy of an older approach to AI named Symbolic AI, pioneered by Allen Newell and Herbert A. Simon in the late 1960's. Symbolic AI looks at how human knowledge can be represented in a declarative form..

2.2. Literature Review

2.2.1. Search Questions

The search for literature related to the theme of this thesis starts with the following question:

- **SQ1:** What scientific studies exist about creating a human-like digital brain?

The first question, after some searching and counselling, gives place to another search question:

- **SQ2:** Which approaches and algorithms exist to control an agent in a given environment from a Reinforcement Learning perspective?

After analyzing OpenAI RL taxonomy and several algorithms we have the following questions:

- **SQ3:** Which Reinforcement Learning Algorithms have been published after 2018?

Because we have a motivation to use graphs, we must address the following question:

- **SQ4:** Are there any algorithms based on graphs?

Replicating our own brains means we also need to look into Intrinsic Motivation, so:

- **SQ5:** Which internal (environment independent) Rewards can be considered in RL?

After consulting peers working in RL, a new question emerges:

- **SQ6:** Which Hierarchical Reinforcement Learning algorithms have been proposed?

And lastly, considering the complexity of the found publications, we focused on the following two last questions:

- **SQ7:** Which insights does each paper bring to the field regarding strategies to handle Reinforcement Learning problems ?

- **SQ8:** How far back do we need to go to start analyzing papers that impacted the modern approaches to RL ?

2.2.2. Search Strategy Outline

We define the search strategy pipeline in Figure 6, starting with a **preliminary search** described in **2.2.3.**, where we get a first glimpse of the existing algorithms of RL, paper repositories where we can search for more RL literature and institutions working in RL.

While [112, p.14] describes an SLR preliminary search as a step to identify existing SLRs and evaluate the volume of studies that are potentially relevant, it doesn't specify a particular process to do this step, so we proceed to execute an **ad hoc Literature Review** to achieve the goals [112] set for a preliminary search.

The Ad hoc Literature Review is conducted as a second step and can be consulted in **2.2.4.**. This review results not only in a compilation of RL algorithms since 1989, but also in a set of papers that can be analyzed for word frequency statistics.

With the **text-mining** step we produce a search string that is used to perform a solid SLR[112] presented in **2.2.5.**

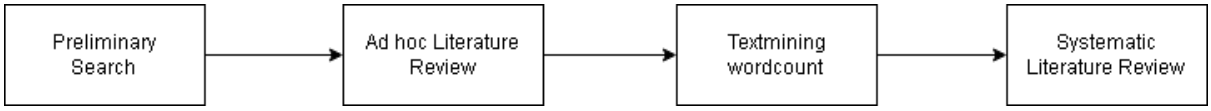


***Figure 6**: Search Strategy Outline.*

2.2.3. Preliminary Search

Preliminary Search process

Preliminary Search starts by the participation in online workshops about Deep RL, which leads to the identification of a taxonomy of Reinforcement Learning published by OpenAI at its website openai.com. Next step is to explore each of the algorithms in the referred taxonomy.

Now with a more complete list of RL algorithms, we move forward by identifying works done at ISCTE and contacting students and teachers involved in those theses in an effort to validate an improved taxonomy. This leads us to understand that HRL is a promising field of study within RL, so the next step is understanding which efforts exist in the HRL field, and which insights HRL research can offer.

Preliminary findings

Institutions

Universities have been the cradle for advancements in Artificial Intelligence (AI), and

the term AI was coined at a meeting at Dartmouth College in 1956 [38, p.19]. Deepmind was founded by three academics and later acquired by Google in 2014 [38, p.19].

Meanwhile we can see by the authorship of RL papers that academic scientists are increasingly being hired by companies, and this tendency is making companies investing in RL an important origin of newer advancements in RL.

Some top universities and other institutions researching on RL:

- Albert-Ludwigs-Universität Freiburg - http://www.cms.uni-freiburg.de/
- University of California Berkeley, Department of Electrical Engineering and Computer Science - https://eecs.berkeley.edu/
- Université de Montréal, Département d'informatique et de recherche opérationnelle - https://www.umontreal.ca/
- Stanford University, Department of Computer Science - https://cs.stanford.edu/
- University of Cambridge - https://www.cam.ac.uk/
- University of Toronto, Department of Computer Science - https://web.cs.toronto.edu/
- New York University, Department of Computer Science - https://cs.nyu.edu/home/index.html
- University College London - https://www.ucl.ac.uk/
- Alan Turing Institute - https://www.turing.ac.uk/
- The Swiss AI Lab IDSIA - http://www.idsia.ch/idsia_en/institute.html
- McGill University - https://mcgill.ca/
- University of Amsterdam - https://www.uva.nl/en?cb
- Tsinghua University - https://www.tsinghua.edu.cn/en/
- VUB Artificial Intelligence Lab - https://ai.vub.ac.be/
- Vrije Universiteit Amsterdam - https://www.vu.nl/en/
- University of Tokyo - https://www.u-tokyo.ac.jp/en/index.html
- Ntt Group - https://www.ntt.co.jp
- Narrative Nights - http://narr.jp/works.html
- Dwango - https://dwango.co.jp/english/index.html
- Deepsense.ai - https://deepsense.ai/
- Polish Academy of Sciences, Institute of Mathematics - https://www.impan.pl/en
- University of Warsaw, Faculty of Mathematics, Informatics and Mechanics - http://informatorects.uw.edu.pl/en/faculties/10000000/
- University of Illinois at Urbana–Champaign - https://illinois.edu/
- University of Alberta - https://www.ualberta.ca/index.html
- University of Pennsylvania - https://www.seas.upenn.edu/
- Fast.ai - https://www.fast.ai/
- CIFAR - https://www.cifar.ca

Some top companies researching on RL:

- Google Deepmind - https://deepmind.com/
- Google Brain - https://research.google/teams/brain/
- OpenAI - www.openai.com
- Amazon - https://www.amazon.science
- Alibaba - https://damo.alibaba.com/labs/ai
- Facebook - https://ai.facebook.com
- Microsoft - https://www.microsoft.com/en-us/research/theme/reinforcement-learning-group/
- IBM - https://www.ibm.com/blogs/research/tag/reinforcement-learning/

Algorithms in Preliminary Findings

OpenAI is one of the big players of the Reinforcement Learning scene, alongside DeepMind Technologies Limited (deepmind.com). Also Fast.ai, a small non-profit group has been making contributions to deep learning, being an example “super-convergence” [43]. Another player is Alibaba.

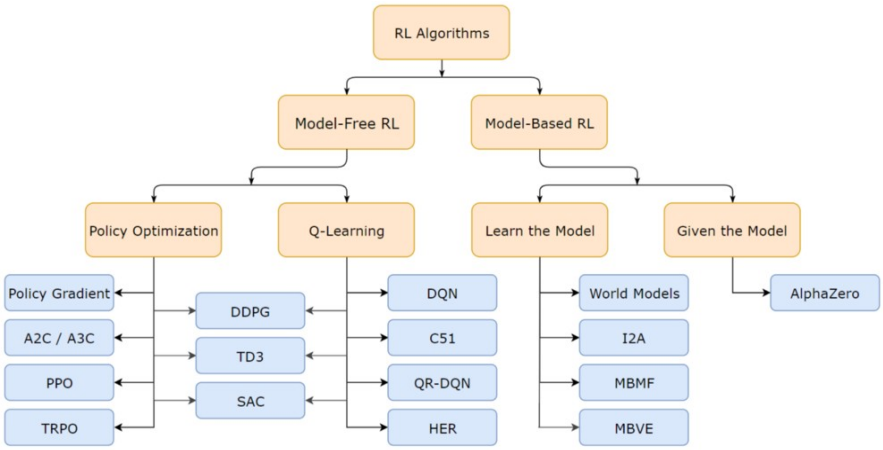


***Figure 7**: RL Taxonomy created by openAI .*

(https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#a-taxonomy-of-rl-algorithms)

One of the widely known algorithms of RL, called DQN [19] was published in 2013, but the pace of publication of RL algorithms was still slow at that time. It’s not until the year 2015 that more algorithms and at a faster pace start to be published (TRPO [15], DDPG [16], etc).

In 2016 the rhythm of publications doubled the number of 2015, and two major references known as A2C and A3C appear [12] .

The following year, 2017 was even more prolific, with PPO [14], I2A [23] and Alpha-Zero [26] algorithms seeing the light of day. Alpha-zero was trained exclusively using self-play [26, p.3]. 2017 was also the year when several model-based algorithms were being published setting an alternative approach to RL.

“World Models” [10], published in 2018 is also a model-based algorithm. While in 2018 there was a slowdown regarding the publishing of new algorithms, 2019 came with more algorithms in a competitive environment between Alibaba with its first published algorithm in RL called DVPG [29], Google Brain, and several universities like Berkeley, Vrije Universiteit Brussel and others partnering with the tech giants or publishing on their own.

In 2020, Amazon joined the race with its MQL algorithm [53], while openAi, Deepmind and Google Brain have also proposed more models.

Available scientific digital libraries

Below we present a list of digital libraries where a part of the preliminary search was conducted with arXiv being one of the main sources for the early search:

- arXiv
- ACM Digital Library
- SpringerLink
- Wiley Online Library
- ScienceDirect
- IEEE xplore
- B-on
- Scopus
- Web of Science
- Semantic Scholar
- Google Scholar

2.2.4. Ad hoc Literature Review

This ad hoc literature review focuses on answering all of the search questions presented in **2.2.1.** As stated before, RL algorithms are usually exclusive, either of Flat Reinforcement Learning (FRL) [91, p.1], Hierarchical Reinforcement Learning (HRL) [80, p.1] or Meta Reinforcement Learning (MRL) [128, p.1], our approach is to deal with all types in a chronological order (by year of print publishing date). Despite having fundamentally different approaches, all approaches propose to solve the problem of Optimal Control. One of the early papers that introduced Q-learning, a root for FRL approaches, also discusses how HRL methods can be used to solve the learning problem [107, p.1], but somehow, overtime some degree of separation of FRL from HRL became more noticeable, despite HRL algorithms implementation of planners sometimes resorting to FRL methods. MRL algorithms have been creating a thread separated from FRL and HRL, but we must say that in some aspects they leave a feel of being HRL.

With openAI taxonomy (Figure 7) in hand, some backward snowballing leads us to older RL algorithms and in this way have access to the fundamentals in which newer RL algorithms are based on, leading to a better understanding of newer algorithm proposals.

The next step is to search for RL algorithms dated after 2018, since the taxonomy does not refer to any algorithm after that year. A basic query done in search engines using “2019 Reinforcement Algorithms” results in an improved list of algorithms including newer ones. Also a 2020 paper is found [49].

This ad hoc Literature Review holds brief descriptions of the methods, algorithms, techniques and frameworks proposed in the selected publications, with focus on the insights each one brings. It also shows a trend in the interest of researchers, universities [48, 15, 31, 24, 34, 111, 52, 18, 25, 39, 54] and companies like Google [19, 16, 12, 61, 32, 40, 20, 21, 23, 26, 10, 49], OpenAi [111], Amazon [53], Alibaba [29] and others investing in the field of RL.

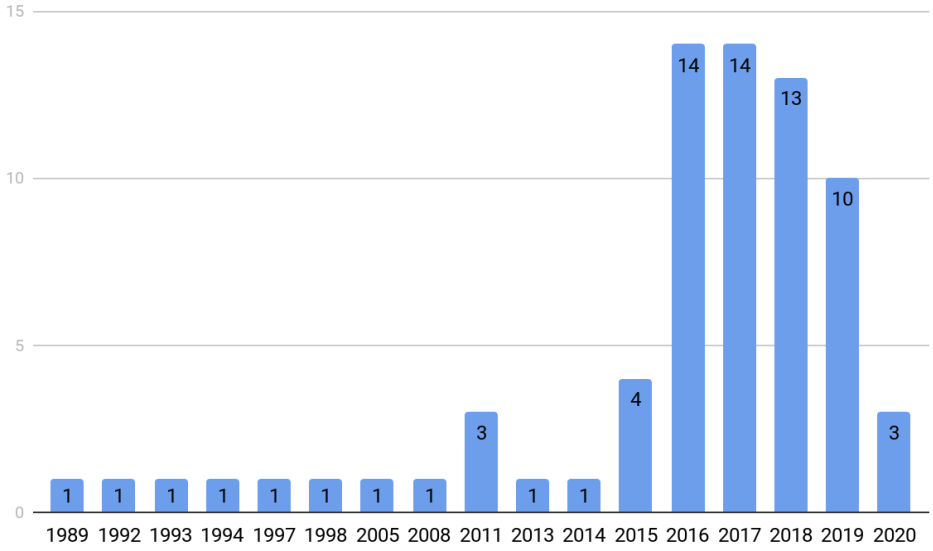


***Figure 8**: Distribution of papers, in the Ad hoc Literature Review, per year (2020 partial year).*

Check Appendix C for brief descriptions of the methods, algorithms and techniques proposed in the selected publications, with focus on the insights each one brings:

Summarization of the Analysis and Findings of the Ad hoc Literature Review (ALR)

Through the findings we can see several evidences worth of notice:

- Off-policy methods are more efficient in learning than On-policy ones. But off-policy methods tend to be less effective than on-policy methods, because they do not integrate the knowledge related to the policy in use at the moment [49, p.1].
- HRL allows us to address more complex tasks than FRL, with less learning episodes, but HRL algorithms are more complex due to the work with multiple layers [80, p.1].
- Neural Networks are a key component of the better performing models;
- High-dimensional continuous state/action spaces are much more difficult to address than their discrete counterparts;
- Model-free algorithms are able to generalize better than model-based ones, but are slower at learning policies [54, p.1]. Model-based algorithms also have the problem of possible model bias [29, p. 3317]. Model free algorithms are very sensitive to hyperparameters and are also very sample inefficient [46, p.3425]. Generally model based models are less efficient in asymptotic performance [114, p.9]. Model-based algorithms work well in environments where it is easy to model dynamic features, but on complex and noisy environments the learnt environment model will prove itself less performant [115, p.1].
- Deep NN have been successfully applied in model-free algorithms, but in model-based algorithms it hasn't been easy to apply them [132, p.3].

- In the face of dimensionality problems that demand more processing power, one approach has been to develop approximation methods, i.e. Stochastic Gradient Descent and NN.

Text-mining

With all the papers gathered with the ad hoc Literature Review we can now extract the "DNA" of an RL paper, by performing a word count of most relevant words. This count is done using NVIVO software (Figure 9), after importing all texts into the tool. Stop words are excluded in this word count. Stop words are common words in a language that have no importance in the definition of a context.

From the list of words with more than 4.500 hits we produce a search string that we can use in digital libraries search engines to support the SLR presented in 2.2.5. This search string identifies the word usage pattern an RL paper usually has, like if it was the RL "DNA". This way we can identify the papers that share the same "genome" of the papers already analyzed.

" learning AND function AND value AND state AND network AND policy AND reward AND reinforcement AND methods AND action AND algorithm AND gradient AND model AND space AND time AND training"

***Figure 9**: RL SLR Search String.*

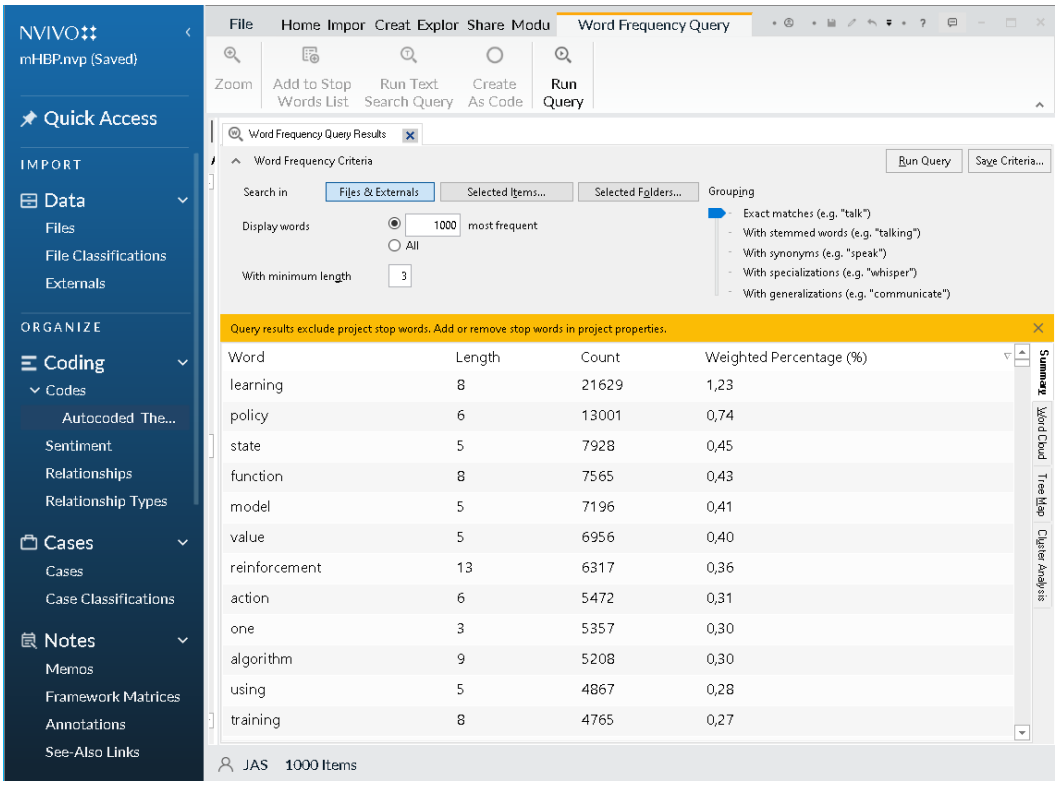


***Figure 10**: NVIVO Word Count.*

2.2.5. Systematic Literature Review

Selection Criteria

After the search in digital libraries (1.3.3.2.3.) is performed using the search string presented in Figure 9, the resulting papers from each digital library search are included or excluded using the criteria in table 2.

***Table 2**: Summary of Inclusion and Exclusion criteria.*

Inclusion Criteria	Exclusion Criteria
• English Language papers; • Papers from any date; • Available online; • Articles that relate to optimal control of an agent	• Papers that are not in English; • Unclear studies • Studies that do not pass in the filtration process presented in **Table 3**.

***Table 3**: SLR Exclusion filters (referred in Table 2).*

	Description
Filter 1	Exclude papers that can't have their bibtex extracted in bulk. Some databases have protection against automated reference extraction. For Google Scholar we created a Firefox extension to extract results (see **Appendix B**).
Filter 2	Filter out duplicate papers.
Filter 3	Filter out papers that are not related to Computer Science. Several papers from medical journals were excluded. Search results from the Wiley database have too many medical papers .
Filter 4	Exclude any bibliographic entry that is part of a book and accept only "Journal article" and "Conference paper" documents.
Filter 5	Exclude surveys or tutorials by finding papers with the following terms on their titles: ● "a review"; ● "survey"; ● "tour"; ● "tutorial"; ● "overview". The goal is to constrain the search to original papers only.
Filter 6	Exclude papers that only reuse RL algorithms and do not propose a new algorithm or technique for RL. Exclude papers with the following terms on the title: ● "with reinforcement learning"; ● "with deep reinforcement learning"; ● "reinforcement learning based"; ● "using reinforcement learning"; ● "using deep reinforcement learning"; ● "reinforcement learning approach"; ● "based on reinforcement learning"; ● "based on deep reinforcement learning"; ● "case study"; ● "case studies"; ● "application"; ● "communications"; ● "traffic".
Filter 7	Exclude papers related to computer vision by finding papers using the following terms on the title: ● "Vision"; ● "Image". We focus on papers for an agent optimal control and not computer vision only.

***Table 3** (continued).*

Filter 8	Exclude papers with less than 300 citations or that are not from the following conferences or journals: Conferences (see **Table 5** for details) ● International Conference on Machine Learning (ICML); ● Conference on Neural Information Processing Systems (NeurIPS); ● National Conference of the American Association for Artificial Intelligence (AAAI); ● European Conference on Machine Learning (ECML); ● International Joint Conferences on Artificial Intelligence Organization (IJCAI); ● Uncertainty in Artificial Intelligence (UAI); ● International Conference on Automated Planning and Scheduling (ICAPS); ● Knowledge Discovery and Data Mining (KDD); ● International Conference on Information and Knowledge Management (CIKM); ● IEEE International Conference on Data Mining (ICDM); ● SIAM International Conference on Data Mining (SDM); ● Conference on COmputational Learning Theory (COLT); ● Knowledge Capture Conference (K-CAP); ● European Symposium on Artificial Neural Networks 2010 (ESANN 2010); ● IEEE International Conference on Robotics and Automation (ICRA); ● International Conference on Learning Representations (ICLR). Journals (see **Table 6** for details) ● Journal of Machine Learning Research (JMLR); ● Journal of Artificial Intelligence Research (JAIR); ● Transactions on Knowledge and Data Engineering (TKDE); ● IEEE Transactions on Systems, Man, and Cybernetics; ● Machine Learning; ● Machine Learning Research; ● Neurocomputing - Artificial Intelligence; ● Artificial Intelligence; ● International Journal of Robotics Research; ● IEEE Transactions on Vehicular Technology; ● IEEE Transactions on Neural Networks and Learning Systems; ● European Review of Social Psychology; ● Frontiers in psychology; ● Nature (multidisciplinary); ● Business Horizons.

Results

***Table 4**: SLR filtration process summary.*

Database	No filter	Filter 1 Bulk extractable	Filter 2 Not duplicated	Filter 3 CS related	Filter 4 Scientific Article	Filter 5 Not Survey	Filter 6 Not an application of RL	Filter 7 Not about vision or images	Filter 8 > 300 citations or good conference
Google Scholar https://scholar.google.com	23.200	(1) 974	974	974	968	939	752	723	(2) 85
Springerlink https://link.springer.com	1.469	1.469	1.442	1.293	1.055	978	695	627	83
ACM https://dl.acm.org	693	693	690	689	661	635	524	507	74
Wiley https://onlinelibrary.wiley.com	710	710	705	173	113	105	71	68	1
Semantic Scholar https://www.semanticscholar.org/	463	0	0	0	0	0	0	0	0
Scopus	109	109	105	103	86	69	67	64	0
IEEE xplore https://ieeexplore.ieee.org	1	1	1	1	1	1	1	1	0
B-on (3) https://www.b-on.pt	0	0	0	0	0	0	0	0	0
EBSCO (4) https://www.ebsco.com	0	0	0	0	0	0	0	0	0
Web of Science (5)	0	0	0	0	0	0	0	0	0
Science Direct (6) https://www.sciencedirect.com	0	0	0	0	0	0	0	0	0
Total	**26.645**	**3.956**	**3.917**	**3.233**	**2.884**	**2.727**	**2.110**	**1.990**	**243**

1 Google Scholar search engine fails after page 49 presenting 20 results per page
2 More than 300 citations
3 Search didn't work
4 Search cannot be longer than 128 characters
5 No results
6 Can only use max 8 boolean operators

Quality Assessment

A convenient way of addressing the quality of found papers, is to use the ranking of the conferences where they were presented, and in this way have a criteria to exclude papers that have not been presented in high ranked conferences. Table 5 contains a list of conferences where RL papers are published, and as we can see:

- 15 conferences have A rank,
- 2 conferences have B rank (WSDM and AISTATS),
- 1 conference has C rank (IEEE-RAS),
- 3 conferences have no classification (ICLR, ICARSC and ACML).

While the ranking of a conference matters, we have one case of a conference without any classification, ICLR, that cannot be excluded from the list of important conferences.

Important researchers like Nicolas Heess, Volodymyr Mnih, David Silver, Andrew Levy, George Dimitri Konidaris and others have presented papers at ICLR, so despite it not having any ranking (confirmed with the organization of ICLR via email) we are still including papers of this conference.

***Table 5**: List of Conferences and their ranking.*

Conference Name	Rank [1]
International Conference on Machine Learning (ICML) - https://icml.cc/	A (ERA)
Conference on Neural Information Processing Systems (NeurIPS) - http://nips.cc	A (ERA)
National Conference of the American Association for Artificial Intelligence (AAAI) - https://aaai.org_	A (ERA)
European Conference on Machine Learning (ECML)	A (ERA)
International Joint Conferences on Artificial Intelligence Organization (IJCAI) - https://www.ijcai.org	A (ERA)
Uncertainty in Artificial Intelligence (UAI)	A (ERA)
International Conference on Automated Planning and Scheduling (ICAPS)	A (ERA)
Knowledge Discovery and Data Mining (KDD) - http://www.kdd.org	A (ERA)
IEEE Conference on Computer Vision and Pattern Recognition (CVPR) - http://cvpr2020.thecvf.com/	A (ERA)
International Conference on Information and Knowledge Management (CIKM)	A (ERA)
IEEE International Conference on Data Mining (ICDM)	A (ERA)
SIAM International Conference on Data Mining (SDM)	A (ERA)
Conference on COmputational Learning Theory (COLT)	A (ERA)
Knowledge Capture Conference (K-CAP)	A (ERA)
European Symposium on Artificial Neural Networks (ESANN 2010)	A (ERA)
IEEE International Conference on Robotics and Automation (ICRA)	A1 (Qualis)
ACM International Conference on Web Search and Data Mining (WSDM)	B (ERA)
International Conference on Artificial Intelligence and Statistics (AISTATS)	B (ERA)
International Conference on Humanoid Robots (IEEE-RAS)	C (ERA)
International Conference on Learning Representations (ICLR) - http://iclr.cc	N/A
IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)	N/A
Asian Conference on Machine Learning (ACML)	N/A

Regarding journals we can consider the ones presented in table 6. We include some

1 Ranks from http://www.conferenceranks.com/

journals that are not dedicated to Computer Science or Engineering, but that had some important papers published regarding interesting or important approaches to RL, namely intrinsic motivation, social influences on subjects, DQN algorithm, AlphaGo algorithm and AGI/ASI.

***Table 6**: List of Journals and their ranking.*

Journal Name	Scimago Classification [1]
Engineering and Computer Science Journals	
Journal of Machine Learning Research (JMLR)	Q1
Journal of Artificial Intelligence Research (JAIR)	Q1
Transactions on Knowledge and Data Engineering (TKDE)	Q1
IEEE Transactions on Systems, Man, and Cybernetics	Q1
Machine Learning	Q1
Machine Learning Research	Q1
Neurocomputing - Artificial Intelligence	Q1
Artificial Intelligence	Q1
International Journal of Robotics Research	Q1
IEEE Transactions on Vehicular Technology	Q1
IEEE Transactions on Neural Networks and Learning Systems	Q1
IFAC-PapersOnLine	Q2
Other Journals having intersection with RL	
European Review of Social Psychology	Q1
Frontiers in psychology	Q1
Nature (multidisciplinary)	Q1
Business Horizons	Q1

1 https://www.scimagojr.com/

Analysis and Findings in Reinforcement Learning with the SLR

***Table 7**: RL algorithms found with the SLR.*

Also present in the Ad hoc Literature Review	**Not present** in the Ad hoc Literature Review
Google Scholar	
• Actor-Mimic [117] (2016); • Twin Delayed Deep Deterministic policy gradient algorithm [17] (TD3) (2018); • Asynchronous Advantage Actor Critic (A3C) [12] (2016); • Q-learning with normalized advantage function (NAF) [76] (2016); • Bootstrapped-DQN [40] (2016); • Deterministic Policy Gradient Algorithms (DPG) [47] (2014); • FeUdal Networks (FUNs) [92] (2017); • Model-Agnostic Meta-Learning (MAML) [126] (2017); • Neural Fitted Q Iteration (NFQ) [48] (2005); • Model-Based and Model-Free (MBMF) [24] (2017); • Proximal Policy Optimization (PPO) [14] (2017) • Soft Q-Learning (SQL) [111] (2017) • RL2 [129] (2016) • Option-Critic Architecture [93] (2016) • Trust Region Policy Optimization (TRPO) [15] (2015)	• Adversarial REINFORCE algorithm [160] (2017); • Coordinated RL [150] (2002); • Counterfactual multi-agent (COMA) [151] (2017); • Deep Genetic Algorithm (Deep GA) [152] (2018); • Double Q-Learning [119] (2015); • NeuroEvolution of Augmenting Topologies + Q-Learning (NEAT+Q) [153] (2006); • Importance Weighted Actor-Learner Architecture (IMPALA/V-trace) [75] (2018); • GPOMDP [154] (2001); • Kernel-based RL algorithm [155] (2002); • Recurrent Reinforcement Learning (RRL) [156] (2001); • Least-Squares Policy Iteration (LSPI) [157] (2003); • AlphaGo [158] (2016); • MaxEnt [159] (2008); • NoisyNet-DQN, NoisyNet-Dueling and NoisyNet-A3C [161] (2018); • Hedger [162] (2000); • Rainbow DQN [60] (2018); • semi-Markov versions of TD(O), Q-Learning, RTDP, and Adaptive RTDP [163] (1994); • UNREAL [36] (2017) • A policy gradient reinforcement learning algorithm [164] (2004); • Extremely and totally randomized trees (2005);
SpringerLink	
• Neural Fitted Q Iteration (NFQ) [48] (2005); • Neural Fitted Q Iteration with Continuous Actions (NFQCA) [50] (2011);	• DQN Curiosity-driven Variational Autoencoder (DQN-CVAE) [167] (2020); • Active Reinforcement Learning with Demonstration [168] (2020); • Averaged-A3C [169] (2018); • Explore-then-Exploit (EE) [170] (2020); • Multi Agent DDPG (MADDPG) [71] (2017); • Continual, Hierarchical, Incremental Learning and Development (CHILD) [171] (1997); • Parameterized Auxiliary Asynchronous Advantage Actor Critic (PA4C) [172] (2018); • Advice RL [173] (1996); • Clustered Prioritized Sampling DDPG (CPS-DDPG) [174] (2018); • Symbiotic, Adaptive Neuro-Evolution (SANE) [175] (1996); • Fast Online Q [176] (1998); • Hybrid RL [177] (2006); • Induced Exploration on Policy Gradients [178] (2018); • Generator Constrained Q-learning [179] (GCQ) (2020); • Recursive Least-Squares Policy Iteration (RLSPI) [180] (2013); • Policy Gradient Actor-Critic (PGAC) [181] (2007); • Preference-based RL [182] (2012).
ACM	
	• Deep Active Learner [183] (2019); • EARLIEST [67] (2019); • Policy Optimizer for Exponential Models (POEM) [123] (2015); • Cognitive Structure Enhanced framework for Adaptive Learning (CSEAL) [45] (2019); • Relative Entropy Policy Search (REPS) [57] (2017).

With the results shown in Table 7, we can state that Google Scholar yielded the most balanced results, by having an almost equal number of algorithms already present in the

ALR and new ones.

Analysis of some of the new algorithms found with the SLR:

semi-Markov versions of TD(0), Q-Learning, RTDP, and Adaptive RTDP [163] (1994)

Proposes variants of TD(0), Q-Learning and RTDP to address the use of continuous time as defined in semi-Markov Decision Processes.

Hedger [162] (2000)

Presents a solution to approximate Q-learning value functions, while focusing on reducing the approximator's error. Hedger is based on locally weighted regression, a technique falling into Support Vector Machines realm.

Recurrent Reinforcement Learning (RRL) [156] (2001)

Proposes an algorithm that avoids the curse of dimensionality of the Bellman equation, and is robust to large amounts of noise in data. RRL achieves this by not using a value function, and instead learns policies directly from data.

Kernel-based RL algorithm [155] (2002)

Proposes a value iteration algorithm that uses kernel regression. This algorithm uses historic observations of transitions originating in the neighborhood of a state to calculate locally weighted averaging to assess the value of that same state.

Coordinated RL (CRL) [150] (2002)

Defines a set of RL algorithms where policies and value functions are represented in a structured and parameterized way. CRL aims at improving the action of several agents that are coordinated in action selection and parameter updates to perform as a whole.

Least-Squares Policy Iteration (LSPI) [157] (2003)

A model-free algorithm that combines value iteration with policy iteration. LSPI learns the state-action value function, which enables the algorithm to select actions without a model, and also to incrementally improve policies in a policy-iteration way. LSPI uses a linear approximation instead of NNs.

AlphaGo [158] (2016)

Proposes an algorithm that mixes MCTS and NNs to play the game of Go, being trained with gameplay. This algorithm addresses the curse of dimensionality in the game of Go.

Counterfactual multi-agent (COMA) [151] (2017)

Defines a multi-agent actor-critic algorithm, with a centralized critic and decentralized actors. The critic estimates the Q-function and the actors optimize the policies. Performs a sensitivity analysis by fixating all agents except one, in order to be able to calculate credit assignment per agent.

UNREAL [36] (2017)

Explores the use of intrinsic rewards that are also referred to as pseudo-reward functions. While extrinsic rewards may not exist at some point of the execution, the algorithm still keeps improving policies with pseudo-rewards. UNREAL adds auxiliary control and auxiliary prediction tasks to the base algorithm A3C, producing a mix of RL and IM.

Deep Genetic Algorithm (Deep GA) [152] (2018)

Applies evolution strategies to RL as an alternative to backpropagation and gradient optimization.

Importance Weighted Actor-Learner Architecture (IMPALA/V-trace) [75] (2018)

V-trace is an off-policy actor-critic RL algorithm.

Rainbow DQN [60] (2018)

It is a version of DQN that integrates several improvements made by the DRL community. The integrated extensions are Double Q-Learning, Prioritized Replay, Dueling Networks, Multi-step Learning, Distributional RL and Noisy Nets.

Reporting experience with the SLR method

Here we provide an experience report of the application of the SLR methodology to perform a literature review. While the SLR process sets well defined steps to search, filter and evaluate the available literature about any scientific theme, including RL, we can say that there are still some challenges to having a streamlined and easy implementation of a SLR:

1. [112] defines guidelines for performing an SLR in Software Engineering, but does not define how the preliminary search step should be executed;

2. Currently search tools available in the used digital libraries are lacking functionalities to bulk store search results inside each search platform where successive filtering could be applied more easily, if those platforms also developed the features to implement an SLR process;

3. Exporting bibliographic entries to softwares like Zotero, Mendeley or similar tools, faces the limitation of the number of automatic exports imposed by several digital libraries like Google Search, Semantic Scholar, EBSCO or Science Direct, and others, leading in some cases to ip address blocking;

4. Exporting of bibliographic entries into Zotero, Mendeley or similar tools sometimes ignores important information like the number of citations, creating difficulties in performing filtering by this indicator;

5. Automatic filtering of papers by title and/or abstract and even the body of the paper is an unreliable procedure and the results are highly biased by the style of writing of the paper's authors;

6. Tools like NVIVO that provide text mining abilities and automated tagging of papers may help reduce the error in filtering papers by content with mere search words. Future work in streamlining an SLR could involve evaluating NVIVO's efficacy in automated tagging;

7. Evaluation of the quality of found papers by the ranking of the conference where they were presented is not always possible, as we can see in the case of RL, where ICLR despite having papers with great importance to RL, does not have a conference rank;

Using the frequency of most important words in pre-gathered papers, to create the search string for the SLR, brought us papers that were either already present in the ALR or that were new and within the search scope of RL algorithms proposals. This is evidence that this approach to preliminary work for starting an SLR is a valid one.

Using the firefox extension (Figure 11) that we have created and that is shown in Appendix B, we get a list of papers from Google Scholar that provides not only the source digital library of the paper, but also the number of citations, two useful fields to perform easy filtering and that can potentially lead to Google Scholar as the main source for the making of an SLR in future literature reviews..

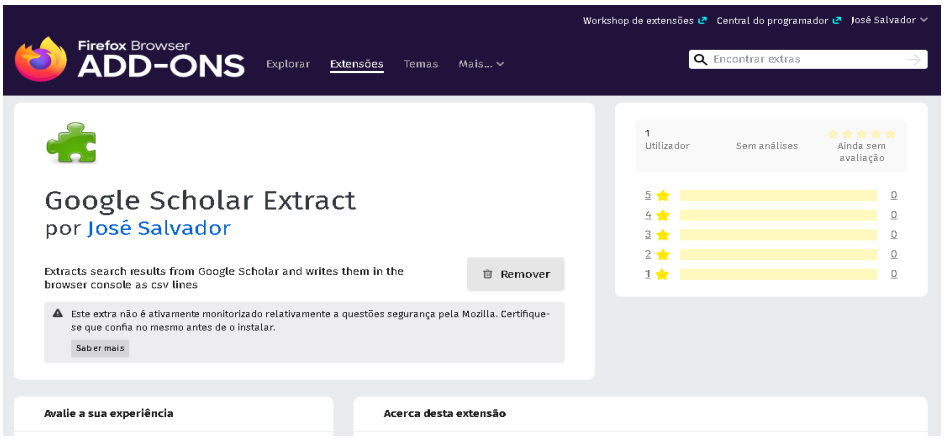


***Figure 11**: Google Scholar Firefox extension install page.*

2.2.6. Comparison between the ALR and the SLR

ALR and SLR are two very different approaches to performing a Literature Review. SLR is intended to be a more thorough process, structured, systematic and more mechanical, while ALR is less structured and more dependent on the researcher's style of search.

In this study we are able to compare both, because we implement both. The next table compares them.

***Table 8**: Ad hoc Literature Review (ALR) vs Systematic Literature Review (SLR).*

	ALR	SLR
Pros	• More flexible; • It is a more conscious process.	• More structured; • Gets more results and finds more different approaches to the field.
Cons	• Less structured; • Dependent of the researcher's search style; • Can omit important lines of thought in the field.	• Less flexible; • Mechanical process; • Digital libraries and search engines are not yet prepared to facilitate the execution of SLRs; • Defining filters is a subjective task and is not systematized; • Filtering by conference ranks is not always valid; • Creation of the search string is a subjective task.

2.2.7. Top Authors

Both the ALR and SLR lead us to top authors in the field of RL. Table 9 provides a list of authors and their respective h-index indicators.

***Table 9**: Top Authors in Reinforcement Learning.*

Name + eMail	h-index	Scholar profile
Sergey Levine - SLEVINE@EECS.BERKELEY.EDU	77	https://scholar.google.com/citations?user=8R35rCwAAAAJ&hl=en&oi=ao
David Silver - david@deepmind.com - DAVIDSILVER@GOOGLE.COM	61	https://scholar.google.com/citations?user=-8DNE4UAAAAJ&hl=en
Nicolas Heess - heess@google.com	36	https://scholar.google.com/citations?hl=en&user=79k7bGEAAAAJ
Timothy P. Lillicrap - countzero@google.com	44	https://scholar.google.com/citations?user=htPVdRMAAAAJ&hl=en
Martin Riedmiller - riedmill@informatik.uni-freiburg.de	47	https://scholar.google.com/citations?hl=en&user=1gVfqpcAAAAJ
Volodymyr Mnih - vlad@ deepmind.com VMNIH@GOOGLE.COM	28	https://scholar.google.com/citations?hl=en&user=rLdfJ1gAAAAJ
Koray Kavukcuoglu - koray@deepmind.com - KORAYK@GOOGLE.COM	59	https://scholar.google.com/citations?hl=en&user=sGFyDIUAAAAJ
Alex Graves - alex.graves@deepmind.com - GRAVESA@GOOGLE.COM - University of Toronto	52	https://scholar.google.com/citations?hl=en&user=DaFHynwAAAAJ
Daan Wierstra - daan@deepmind.com - wierstra@google.com	44	https://scholar.google.com/citations?hl=en&user=aDbsf28AAAAJ
Pieter Abbeel - PABBEEL@CS.BERKELEY.EDU	98	https://scholar.google.com/citations?hl=en&user=vtwH6GkAAAAJ
Rémi Munos - munos@google.com	60	https://scholar.google.com/citations?hl=en&user=OvKEnVwAAAAJ
R. Hafner - rhafner@informatik.uni-freiburg.de	12	https://scholar.google.com/citations?hl=en&user=HGXVByQAAAAJ
John Schulman - JOSCHU@EECS.BERKELEY.EDU - joschu@openai.com (co-founder of openai)	31	https://scholar.google.com/citations?hl=en&user=itSa94cAAAAJ
Michael Jordan - JORDAN@CS.BERKELEY.EDU	169	https://scholar.google.com/citations?hl=en&user=yxUduqMAAAAJ
Alexander Pritzel - apritzel@google.com	14	https://scholar.google.com/citations?hl=en&user=GPgAyU0AAAAJ
Adrià Puigdomènech Badia - ADRIAP@GOOGLE.COM	6	https://scholar.google.com/citations?hl=en&user=DcWRJW4AAAAJ
Will Dabney - Deepmind	17	https://scholar.google.com/citations?hl=en&user=dR-7QW8AAAAJ
Mohammad Azar - Deepmind	15	https://scholar.google.com/citations?hl=en&user=AITQrFcAAAAJ
Arthur Guez - DeepMind	20	https://scholar.google.com/citations?hl=en&user=iyD9aw8AAAAJ
Demis Hassabis - DeepMind	57	https://scholar.google.com/citations?hl=en&user=dYpPMQEAAAAJ
Justin Fu - justinjfu@eecs.berkeley.edu	10	https://scholar.google.com/citations?hl=en&user=T9To2C0AAAAJ

2.2.8. RL Formulation

Figures 12 and 13 (in the next pages) contain a set of RL basic formulas/equations and the pseudocode of tabular solution methods like Policy Iteration, Value Iteration, MC methods, Temporal Difference (TD) with Q-learning; and also one example of an approximate solution method with Deep Q Learning. Table 10 has a summary of the notation used in Figures 12 and 13.

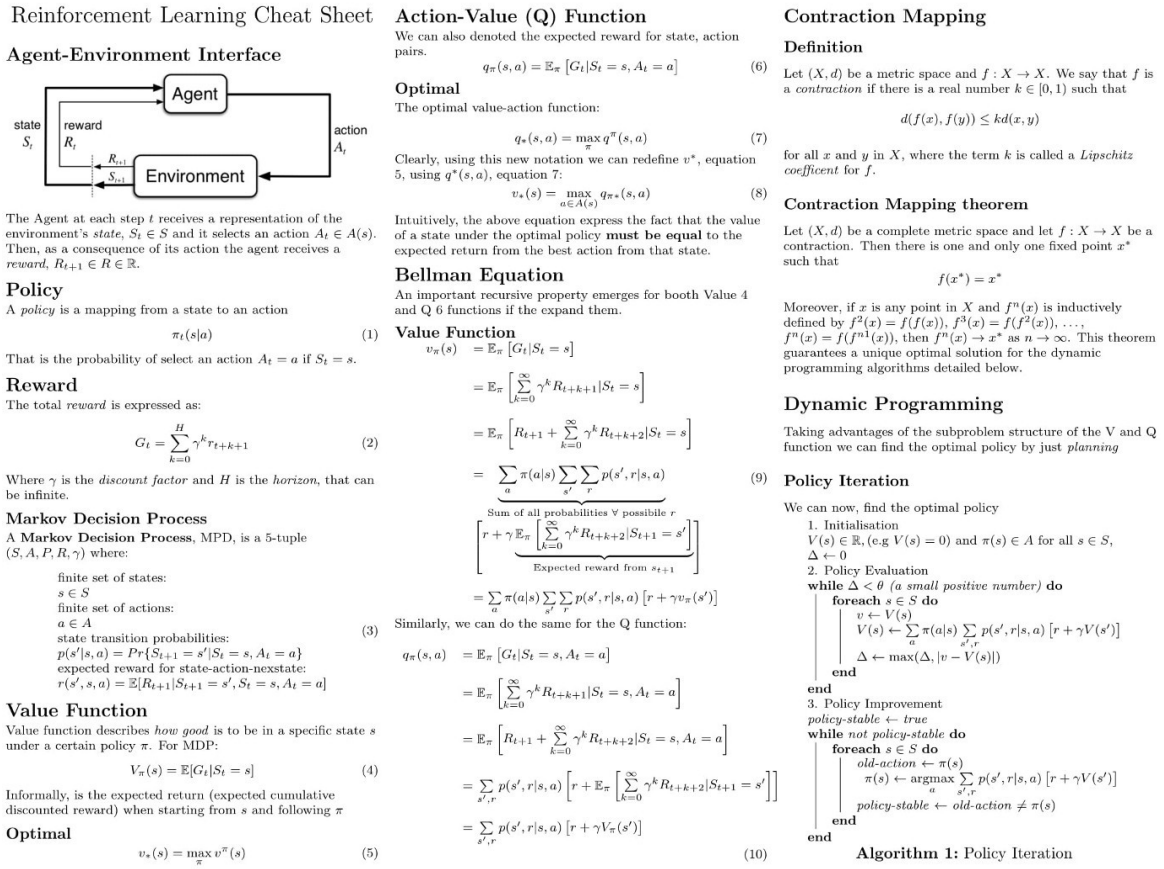
***Table 10**: Notation used in RL formulation.*

Symbol	Meaning
$s \in \mathcal{S}$	States.
$a \in \mathcal{A}$	Actions.
$r \in \mathcal{R}$	Rewards.
S_t, A_t, R_t	State, action, and reward at time step t of one trajectory. I may occasionally use s_t, a_t, r_t as well.
γ	Discount factor; penalty to uncertainty of future rewards; $0 < \gamma \leq 1$.
G_t	Return; or discounted future reward; $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.
$P(s', r\|s, a)$	Transition probability of getting to the next state s' from the current state s with action a and reward r.
$\pi(a\|s)$	Stochastic policy (agent behavior strategy); $\pi_\theta(.)$ is a policy parameterized by θ.
$\mu(s)$	Deterministic policy; we can also label this as $\pi(s)$, but using a different letter gives better distinction so that we can easily tell when the policy is stochastic or deterministic without further explanation. Either π or μ is what a reinforcement learning algorithm aims to learn.
$V(s)$	State-value function measures the expected return of state s; $V_w(.)$ is a value function parameterized by w.
$V^\pi(s)$	The value of state s when we follow a policy π; $V^\pi(s) = \mathbb{E}_{a\sim\pi}[G_t\|S_t = s]$.
$Q(s, a)$	Action-value function is similar to $V(s)$, but it assesses the expected return of a pair of state and action (s, a); $Q_w(.)$ is a action value function parameterized by w.
$Q^\pi(s, a)$	Similar to $V^\pi(.)$, the value of (state, action) pair when we follow a policy π; $Q^\pi(s, a) = \mathbb{E}_{a\sim\pi}[G_t\|S_t = s, A_t = a]$.
$A(s, a)$	Advantage function, $A(s, a) = Q(s, a) - V(s)$; it can be considered as another version of Q-value with lower variance by taking the state-value off as the baseline.

$\boldsymbol{\gamma}$ (gamma) is an important hyperparameter of RL, since it dictates the weight future rewards have in the calculation of the total reward at any given time of the agent's life. $\boldsymbol{\gamma}$ is part of a structural equation of RL: the Bellman equation. This equation integrates current and future rewards in a recurrent way, so that the agent can choose its actions without being shortsighted by only looking at the immediate reward. This lookahead behaviour is an attempt to create an intelligent agent.

None of the presented equations have Intrinsic Motivation into consideration, but introducing IM in the formulation is an easy step, since IM can be considered just another reward coming from the environment.

The reward formula introduces H, the horizon of the system, which can be infinite or finite. Naturally the bigger H is the more complex the RL problem becomes. H is also an important variable to allow the combination of model-free and model-based algorithms as you can see in the Literature Review part of this paper.



Reinforcement Learning Cheat Sheet

Agent-Environment Interface

The Agent at each step t receives a representation of the environment's *state*, $S_t \in S$ and it selects an action $A_t \in A(s)$. Then, as a consequence of its action the agent receives a *reward*, $R_{t+1} \in R \in \mathbb{R}$.

Policy

A *policy* is a mapping from a state to an action

$$\pi_t(s|a) \tag{1}$$

That is the probability of select an action $A_t = a$ if $S_t = s$.

Reward

The total *reward* is expressed as:

$$G_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{2}$$

Where γ is the *discount factor* and H is the *horizon*, that can be infinite.

Markov Decision Process

A **Markov Decision Process**, MPD, is a 5-tuple (S, A, P, R, γ) where:

$$\begin{aligned} &\text{finite set of states:} \\ &s \in S \\ &\text{finite set of actions:} \\ &a \in A \\ &\text{state transition probabilities:} \\ &p(s'|s,a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\} \\ &\text{expected reward for state-action-nexstate:} \\ &r(s',s,a) = \mathbb{E}[R_{t+1} | S_{t+1} = s', S_t = s, A_t = a] \end{aligned} \tag{3}$$

Value Function

Value function describes *how good* is to be in a specific state s under a certain policy π. For MDP:

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s] \tag{4}$$

Informally, is the expected return (expected cumulative discounted reward) when starting from s and following π

Optimal

$$v_*(s) = \max_\pi v^\pi(s) \tag{5}$$

Action-Value (Q) Function

We can also denoted the expected reward for state, action pairs.

$$q_\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{6}$$

Optimal

The optimal value-action function:

$$q_*(s,a) = \max_\pi q^\pi(s,a) \tag{7}$$

Clearly, using this new notation we can redefine v^*, equation 5, using $q^*(s,a)$, equation 7:

$$v_*(s) = \max_{a \in A(s)} q_{\pi *}(s,a) \tag{8}$$

Intuitively, the above equation express the fact that the value of a state under the optimal policy **must be equal** to the expected return from the best action from that state.

Bellman Equation

An important recursive property emerges for booth Value 4 and Q 6 functions if the expand them.

Value Function

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s \right] \\ &= \underbrace{\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)}_{\text{Sum of all probabilities } \forall \text{ possibile } r} \\ &\quad \left[r + \gamma \underbrace{\mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_{t+1} = s' \right]}_{\text{Expected reward from } s_{t+1}} \right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s') \right] \end{aligned} \tag{9}$$

Similarly, we can do the same for the Q function:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s, A_t = a \right] \\ &= \sum_{s',r} p(s', r|s, a) \left[r + \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_{t+1} = s' \right] \right] \\ &= \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_\pi(s') \right] \end{aligned} \tag{10}$$

Contraction Mapping

Definition

Let (X, d) be a metric space and $f : X \to X$. We say that f is a *contraction* if there is a real number $k \in [0, 1)$ such that

$$d(f(x), f(y)) \leq kd(x, y)$$

for all x and y in X, where the term k is called a *Lipschitz coefficent* for f.

Contraction Mapping theorem

Let (X, d) be a complete metric space and let $f : X \to X$ be a contraction. Then there is one and only one fixed point x^* such that

$$f(x^*) = x^*$$

Moreover, if x is any point in X and $f^n(x)$ is inductively defined by $f^2(x) = f(f(x))$, $f^3(x) = f(f^2(x))$, ..., $f^n(x) = f(f^{n1}(x))$, then $f^n(x) \to x^*$ as $n \to \infty$. This theorem guarantees a unique optimal solution for the dynamic programming algorithms detailed below.

Dynamic Programming

Taking advantages of the subproblem structure of the V and Q function we can find the optimal policy by just *planning*

Policy Iteration

We can now, find the optimal policy

1. Initialisation
   $V(s) \in \mathbb{R}$, (e.g $V(s) = 0$) and $\pi(s) \in A$ for all $s \in S$,
   $\Delta \leftarrow 0$
2. Policy Evaluation

while $\Delta < \theta$ *(a small positive number)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end

3. Policy Improvement

policy-stable $\leftarrow$ *true*
while *not policy-stable* **do**
  foreach $s \in S$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right]$
    policy-stable $\leftarrow$ *old-action* $\neq \pi(s)$
  end
end

Algorithm 1: Policy Iteration

***Figure 12**: RL Formulas/Equations and pseudocode [136].*

Value Iteration

We can avoid to wait until $V(s)$ has converged and instead to policy improvement and truncated policy evaluation step in one operation

Initialise $V(s) \in \mathbb{R}$, e.g$V(s) = 0$
$\Delta \leftarrow 0$
while $\Delta < \theta$ *(a small positive number)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end
ouput: Deterministic policy $\pi \approx \pi_*$ such that $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$

Algorithm 2: Value Iteration

Monte Carlo Methods

Monte Carlo (MC) is a *Model Free* method, It does not require complete knowledge of the environment. It is based on **averaging sample returns** for each state-action pair. The following algorithm gives the basic implementation

Initialise for all $s \in S, a \in A(s)$:
  $Q(s, a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s, a) \leftarrow$ empty list
while *forever* **do**
  Choose $S_0 \in S$ and $A_0 \in A(S_0)$, all pairs have probability > 0
  Generate an episode starting at S_0, A_0 following π
  foreach *pair* s, a *appearing in the episode* **do**
    $G \leftarrow$ return following the first occurrence of s, a
    Append G to $Returns(s, a))$
    $Q(s, a) \leftarrow average(Returns(s, a))$
  end
  foreach s *in the episode* **do**
    $\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$
  end
end

Algorithm 3: Monte Carlo first-visit

For no-stationary problems, the Monte Carlo estimate for, e.g, V is:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{11}$$

Where α is the learning rate, how much we want to forget about pass experiences.

Sarsa

Sarsa (State-action-reward-state-action) is a on-policy TD control. The update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

n-step Sarsa

Define the n-step Q-Return

$$q^{(n)} = R_{t+1} + \gamma Rt + 2 + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

n-step Sarsa update $Q(S, a)$ towards the n-step Q-return

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^{(n)} - Q(s_t, a_t)\right]$$

Forward View Sarsa(λ)

$$q_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

Forward-view Sarsa(λ):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^\lambda - Q(s_t, a_t)\right]$$

Initialise $Q(s, a)$ arbitrarily
foreach *episode* $\in$ *episodes* **do**
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  while s *is not terminal* **do**
    Take action a, observer r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end
end

Algorithm 4: Sarsa(λ)

Temporal Difference - Q Learning

Temporal Difference (TD) methods learn directly from raw experience without a model of the environment's dynamics. TD substitutes the expected discounted reward G_t from the episode with an estimation:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1} - V(S_t)\right] \tag{12}$$

The following algorithm gives a generic implementation.

Initialise $Q(s, a)$ arbitrarily
foreach *episode* $\in$ *episodes* **do**
  while s *is not terminal* **do**
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observer r, s'
    $Q(s, a) \leftarrow$
    $Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$
  end
end

Algorithm 5: Q Learning

Deep Q Learning

Created by *DeepMind*, Deep Q Learning, DQL, substitutes the Q function with a deep neural network called *Q-network*. It also keep track of some observation in a *memory* in order to use them to train the network.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\underbrace{\left(r + \gamma \max_a Q(s', a'; \theta_{i-1})\right)}_{\text{target}} - \underbrace{Q(s, a; \theta_i)}_{\text{prediction}} \right]^2 \tag{13}$$

Where θ are the weights of the network and $U(D)$ is the experience replay history.

Initialise replay memory D with capacity N
Initialise $Q(s, a)$ arbitrarily
foreach *episode* $\in$ *episodes* **do**
  while s *is not terminal* **do**
    With probability ϵ select a random action $a \in A(s)$
    otherwise select $a = \max_a Q(s, a; \theta)$
    Take action a, observer r, s'
    Store transition (s, a, r, s') in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s'_j) from D
    Set $y_i \leftarrow$
    $\begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{for non-terminal } s'_j \end{cases}$
    Perform gradient descent step on $(y_j - Q(s_j, a_j; \Theta))^2$
    $s \leftarrow s'$
  end
end

Algorithm 6: Deep Q Learning

***Figure 13**: RL Formulas/Equations and pseudocode [136].*

2.3. Conclusion of the State of the Art

We present an Ad hoc Literature Review of RL with a chronological order of the papers found. This Literature Review enables us to make conclusions about the current limitations of RL algorithms and also the techniques and strategies used to solve several RL limitations and their evolution over time.

We also present a Systematic Literature Review, that allows us to compare the process and results of an SLR versus an ALR. Concerning the SLR, we are able to draw several conclusions about this procedure and how it can be more streamlined. Using an ALR as a preliminary search for SLR is useful to produce a search string by applying text mining word frequency calculations. Google Scholar presents itself as a good candidate for the main source of an SLR.

Regarding the findings about RL, as [166] states, by referring to Wolpert and Macready theorems of “No Free Lunch”, algorithms to perform optimizations can achieve high performances in some types of problems at the expense of poorly solving a lot of other optimization problems. This is what is currently happening with RL, and the path to AGI seems still very uncertain.

We can’t know if there will ever exist an optimal control algorithm that can achieve top performance in many kinds of RL problems without needing modifications. Some approaches are focusing on having an algorithm that decides which algorithm should be used in any given problem. In some way this approach can be compared to our own brains which is thought to have specialized areas used in different types of tasks we perform.

3. Architecture Proposal

In this section we define the architecture of the “my Human Brain Project” (mHBP). Ultimately the mHBP architecture can execute in parallel not only one RL algorithm, but several algorithms that evaluate, modify and use data in their own particular way, and that come together to produce a multi-layered agent with several levels of decision making, that need to be coordinated and prioritized.

As stated before in the “Fundamentals” part of this thesis the goal is to produce a system that:

- can be fast enough to be used in real time;
- has low power consumption;
- Is able to run on limited hardware;
- can evolve and be able to learn and transfer learning between similar tasks;
- can learn its own and the environment dynamics;
- has proprioception;
- Is able to identify deception.

Successes in RL have mostly been limited to specific and a low number of different environments where the dynamics are simple and with a small number of degrees of freedom. In noisy, complex environments, it is difficult to learn an accurate model of the environment. If the model makes mistakes in this context, it can cause the wrong policy to be learned, hindering performance. mHBP is aiming at tackling the problems of a complex environment with a large number of degrees of freedom. Noise is also something that the environment may produce and the mHBP needs to process and identify as deception.

3.1. mHBP Project Architecture

From an higher view perspective, the mHBP architecture (Figure 14) takes into account 5 main parts for a complete RL setup:

1. Environment
2. Agent Controller
3. Memory
4. Parameters
5. RL Engines.

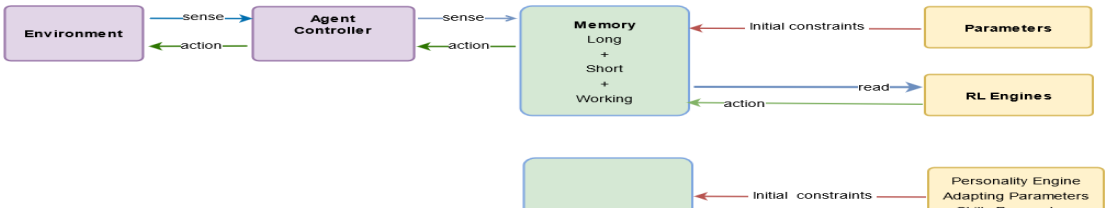


***Figure 14**: top level mHBP architecture.*

3.1.1. Environment

The environment is the origin of the sensorial data that is key to RL, since it represents the state, not only of the external environment, but also of the internal environment (the agent itself).

Sensors that get the environmental readings can be:

- joint angle indicators,
- image and sound feeds,
- Sonars,
- Lidars,
- Oscilloscopes,
- Other systems connectors,

and any other source of information and measurement to describe a state in continuous or discrete scales or even in categorical data.

3.1.2. Agent controller

The controller is responsible for conveying the sensor data into memory and for executing actions determined by the RL Engines.

This controller basically is a gateway between the mHBP core and the agent living in an environment. Actions can also change memory.

3.1.2.1. Actions

We can consider the following action types:

Per proactiveness:

- **Passive action** (do nothing). Like in other implementations of RL [39, p.7], mHBP also considers a type of action that is to do nothing.
- **Proactive action** - perform one micro or macro action or cancel an on-going action.

Per direction:

- **Inbound** - actions that do not lead the agent to interact with the environment but rather to change how it behaves internally, i.e:
    - Attention action - focus attention on specific sensory inputs - The attention

mechanism can be associated with actions to register data from specific sensory input.
    - Parametrization action - change specific parametrization values in memory.
- **Outbound** - Regular action - perform a regular task like picking up an object or walking.

Per scale:

- **Micro-action -** Micro Exploration Actions (MEA) are one type of Babbling [83, p.1] performed by mHBP. MEA can be performed to produce a sensitivity analysis, by freezing all the variables except one at a time. This way the output can better show the causality between actions and read sensorial data. This Micro Exploration also contributes for the agent to get to know the range each of its body parts have. Dealing with Micro-actions leads to a huge search space.
- **Macro-action** (a sequence of one or more micro-actions) - Using macro-actions is a way of reducing action space dimensionality [41, p.1], by dealing with less actions to create a complex behaviour.

Per state:

- **Execute asap** - the action needs to be executed as soon as possible.
- **Execute at** - the action needs to be executed at a specific time.
- **Executing** - the action is being executed.
- **Executed with success**
- **Executed with failure**

All actions can be represented in several different ways, depending on the type of process that will use them. mHBP uses the first 2 of the 3 possible following representations:

- **A JSON Object** - this type of representation enables mHBP Engines to more easily manipulate existing actions to create derived actions. This type of representation is useful to Engines like the Babbling and Brainstorm Engines;

```
[{          "enabled": True,
            "params": {
            "param1": "'RArmEly'",
            "param2": "random.random()",
            "param3": "4" },
            "action":"self.function_0001(param1, param2, param3)"
},{
            "enabled": True,
            "params": {
            "param1": "'RArmEly'",
            "param2": "random.random()",
            "param3": "5" },
            "action":"self.function_0002(param1, param2, param3)"}]
```

Figure 15*: A macro-action represented in JSON.*

- **A Set of instructions as Source code** - this type of representation is done with plain text of a programming language source code. This code is intended to be executed as is. This type of representation is an optimization that allows for an action to be

read and executed, by the Agent controller, without any additional processing.

```
param1= "'RArmEly'"
param2= random.random()
param3=4
self.function_0001(param1, param2, param3)
param1= "'RArmEly'"
param2= random.random()
param3=5
self.function_0002(param1, param2, param3)
```

***Figure 16**: A macro-action represented as Python source code.*

- **A Graph representation** - this type of representation is used to more easily identify patterns in macro-actions or to create relationships between macro-actions through the micro-actions they share. mHBP does not include this type of representation of actions in it's graph database because, it would increase the graph complexity too much. Macro-actions in mHBP are represented in JSON and Python Source Code Inside Action nodes of the graph database.

3.1.3. Memory

Memory in mHBP is used to establish a communication channel between the Engines and the Controller. Memory is the component where information about stimulation, perception, comprehension, rewards, penalties and actions is stored, for further, immediate or delayed, processing by the RL Engines. Actions in a "Execute asap" or "Execute at" state will be executed by the controller.

All of these need to stay in memory in a codified form to represent frequencies of appearance over time and planification. The information stored in memory must be bound to episodes (see Glossary) that represent conditional/contextual responses, behaviours and expressions of the agent.

Memory persistence is influenced by parametrization (personality traits) and by Engines that may condense, mutate or in any other way produce record loss.

Knowledge in Memory is also part of the state of the Agent, so the environment (Section 3.1.1.) is not the only source that contributes to the definition of a state.

mHBP memory is organized in a graph, and as [185, p.1] states, graphs are a non-Euclidean domain and good candidates for the application of Geometric Deep Learning techniques.

In [185, p.1], learning concepts by building a multilayer or hierarchical organization, where complicated concepts stand on top of simpler concepts is considered Deep Learning. Using Neural Networks is one way of achieving deep multilayer hierarchies [185. P.1]. From this we can say that using NN might not be the only valid technique, and other approaches are also eligible as deep learning.

mHBP memory is organized in several hierarchies, where macro-actions can be subdivided into micro-actions and where states can be subdivided into perception elements.

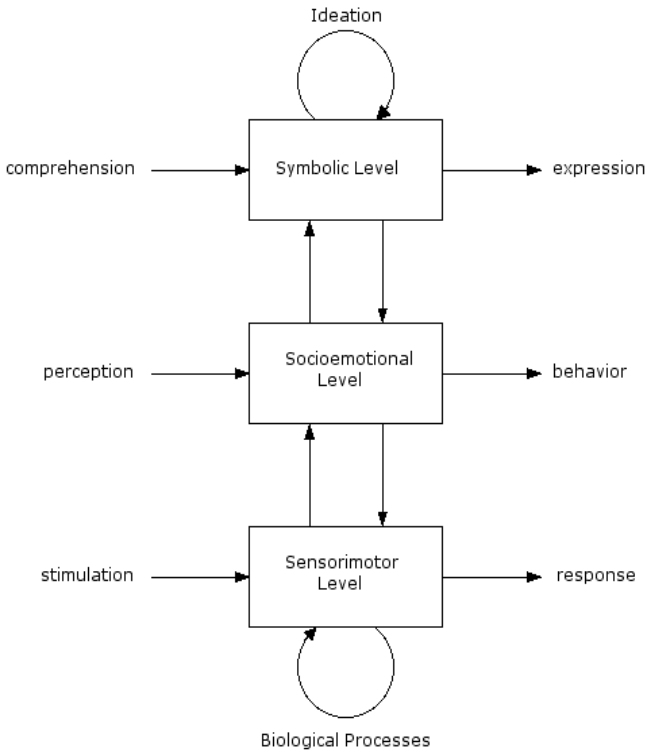


***Figure 17**: The three levels of the mind [184].*

3.1.4. Parameters

This component represents the parameters that will influence the functioning of the Engines. These parameters that can be static or dynamic, are registered in Memory and represent constraints for all occurring processes. Changing the values of some of the parameters are also actions available to mHBP Engines. mHBP does not include parameter handling in the baseline implementation, so the candidate parameters in table 11 are a guideline for future experimentation.

***Table 11**: mHBP example candidate parameters.*

Parameter Name	**Description**
BabblingDecreaseRate	Establishes how the Babbling Engines continues generating babbling actions throughout the life of the agent.
PerceptionFocus	Determines how much default perception mHBP demands to be executed by the Agent Controller.
BrainstormingPausing	The frequency the Brainstorming Engine generates new actions is determined by how many seconds it waits between brainstormed generated actions.
MacroActionMaxSize	Maximum number of Micro-actions contained in a Macro-action
DecisionEngineStateNodes	Number of nodes that the Decision Engine uses to support a given state

3.1.5. RL Engines

Each Engine in mHBP is responsible for handling data in its own way, becoming an implementation of a specific algorithm or a part of an algorithm. Below you will find examples of mHBP Engines. New engines can be created and added because the mHBP architecture allows the addition of new engines.

Babbling Engine

This engine can explore by using known micro-actions that it combines to produce macro-actions. It differs from the Brainstorm Engine, because it acts in a more restricted way.

Brainstorm Engine

This engine can explore by transforming previously known micro and macro-actions, giving them new values in parameters or rearranging them in a random way.

Understanding Engine

The understanding engine goal is to analyze data in the graph database in order to identify Rewards and/or to receive input from a tutor to register rewards or penalties.

The Decision Engine

The Decision Engine needs to take into account that the agent is permanently in a penalty state and it needs to proceed with actions that can bring rewards to go against the permanent penalty of not doing anything.

Additional engines can be created to process information in the graph database in ways that are different from the engines listed above.

3.2. Setting up the development environment

3.2.1. Python Interpreter and libraries/frameworks

Python is the chosen language for the implementation of mHBP due to the availability of several Machine Learning libraries/frameworks that may be used to implement parts of RL engines in future experimentation with mHBP. The work in this thesis does not use any of these libraries, but it is important to have access to them in possible future work.

Available Machine Learning libraries/frameworks include:

- Keras (https://keras.io/)
- scikit-learn (https://scikit-learn.org)
- pyTorch (https://pytorch.org/)
- Spark (https://spark.apache.org/docs/1.1.0/mllib-guide.html)
- TensorFlow (https://www.tensorflow.org/)
- Caffe (https://caffe.berkeleyvision.org/)
- Caffe2 (https://caffe2.ai/ - became part of pyTorch)
- Theano (http://deeplearning.net/software/theano/)

Python compatibility or interoperability needs to be guaranteed by the other tools chosen to be part of the needed development environment.

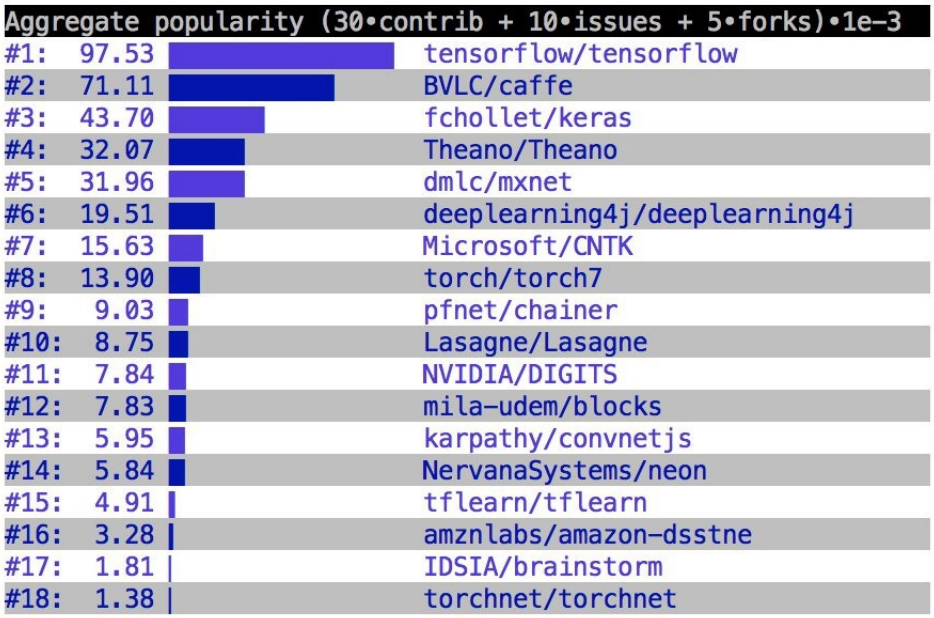


***Figure 18**: Deep Learning libraries/frameworks as per popularity (Source : Google).*

3.2.2. Physics Engine and Virtual World

Choosing a virtual environment with a physics engine was done by searching for several possible solutions and then testing until one of them shows to be easy to use with an external Python program.

List of possible environment and physics engines found: (see Annex A for screenshots)

- VRep/CoppeliaSim (https://www.coppeliarobotics.com/)
- Webots (https://cyberbotics.com/)
- Gazebo (http://gazebosim.org/)
- Unity + plugin (https://unity.com/)
- UE4 + plugin (https://www.unrealengine.com/en-US/)
- OpenAi Gym (https://gym.openai.com/)
- MuJoCo (http://www.mujoco.org/)
- DeepMind Lab (https://deepmind.com/research/open-source/deepmind-lab)
- Malmo Platform (https://www.microsoft.com/en-us/research/project/project-malmo/)

Testing started with CoppeliaSim formerly known as VREP. CoppeliaSim presents instability problems and difficult use of the remote API so we move to the next option: Webots. Both solutions are free (and open source) and both have Python api's. Webots opposite to CoppeliaSim is straight forward, despite some instability like random crashes of the GUI at times, problems that improve with an upgrade to a later version. Only VREP and Webots were tested.

Setting up an IDE environment with Visual Studio Code is also simple and allows for code debugging, which is not possible within Webots IDE.

With Webots it is possible to execute a robot controller in 3 ways:

- in a windows shell command line;
- in webots;
- or from an external IDE like pyCharm or Visual Studio Code.

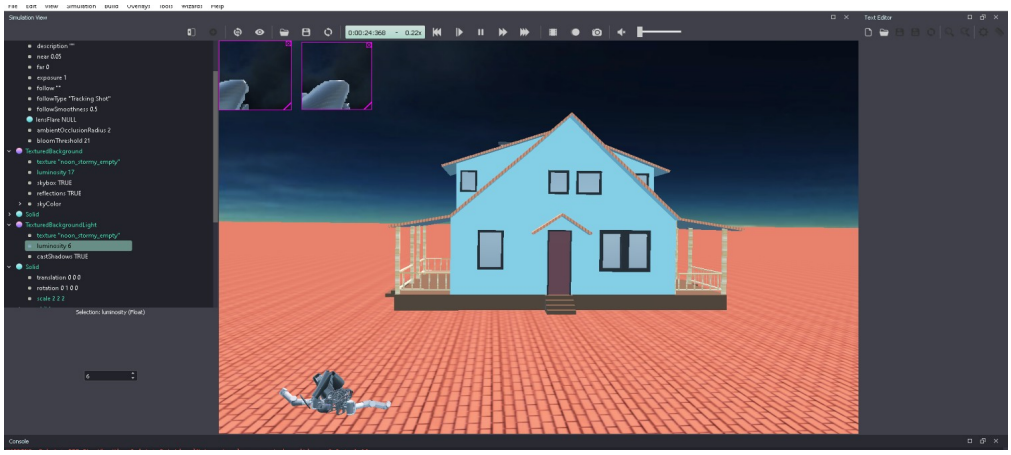


***Figure 19**: A test world in Webots.*

The test world (Figure 19) includes a humanoid robot that simulates an Atlas robot produced by Boston Dynamics. Why use a humanoid robot instead of another type of robot? Since the goal of this thesis is to create a thinking robot we consider it is easier for us, as humans, to evaluate the progression of a human inspired robot, than of another type of robot, because any manifestation of behaviour is more easily a target of critical thinking.

***Figure 20**: Atlas robot laying on the ground.*

Webots has several sample robots available. Work started with a model called “pedestrian” (see Annex A) that is used mainly to simulate people walking in the street to produce an autonomous car learning environment. While it is possible to control all the joints in this robot, its mechanical design is not good enough to have a proper physics response, due to the mismatch between bounding objects and the graphical object being bound. For this reason, the “Atlas” proto (Figure 20) from Boston Dynamics is used instead of the “pedestrian”. Atlas has 26 motors to control arms, legs, torso and head. For this project some sensors are added to its head like gyroscope, cameras, gpsa and distance sensor.

3.2.3. Graph Database

A major design decision and motivation of mHBP is to take advantage of a graph database to implement the memory component. As Emil Eifrem (Figure 21), Co-founder of Neo4j, states regarding AI, most AI problems can be represented in graphs. While most RL algorithms use Neural Networks (NN), structures that are represented by graphs, mHBP is using graphs not specifically to implement NN but to represent tuples of data. A graph representation is more flexible than a table representation because it allows better manipulation of relationships between tuples (rows, in a table), that may be essential to interconnect states, rewards, perceptions and episodes in general in mHBP.

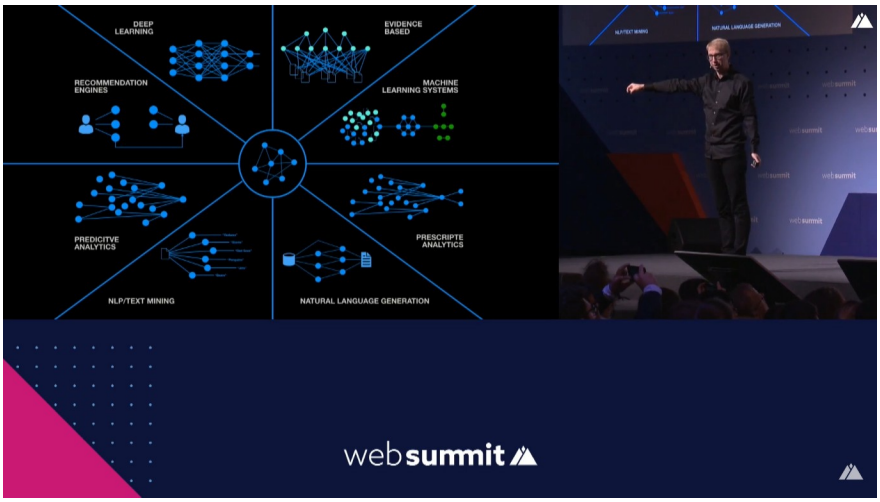


***Figure 21**: Emil Eifrem at Websummit [187]*

As a graph database we use Neo4j (neo4j.com). The choice to use Neo4j over other graph databases is based on a simple criteria. Neo4j seems to be a good fit for mHBP since it is possible to connect to it with a Python client named py2neo, and Neo4j already has several out-of-the-box algorithms to manipulate graphs (Figure 24). We are aware of the existence of other options (listed below) that are not evaluated here, because Neo4j is sufficient for our needs .

Other graph databases available in the market are:

- JanusGraph (janusgraph.org)
- Amazon Neptune (aws.amazon.com/pt/neptune)
- Orientdb (orientdb.com)
- Dgraph (dgraph.io)
- HyperGraphDb (hypergraphdb.org)
- TigerGraph (tigergraph.com)
- InfiniteGraph (objectivity.com/products/infinitegraph)
- Sparksee (sparsity-technologies.com/#sparksee)
- Graphbase (graphbase.ai)
- Memgraph (memgraph.com)
- AnzoGraph (cambridgesemantics.com/anzograph)
- TinkerPop (tinkerpop.apache.org)
- Cayley (github.com/google/cayley)

For interaction with Neo4j we set up pyCharm (Figure 22) as an IDE to execute cypher queries (cypher is a query language for Neo4j), because it is a tool with a better user experience when compared to Neo4j Browser (Figure 23), regarding storing and re-running queries. Being able to inspect the graph database in a convenient way can speed up the evaluation of results in experiments.

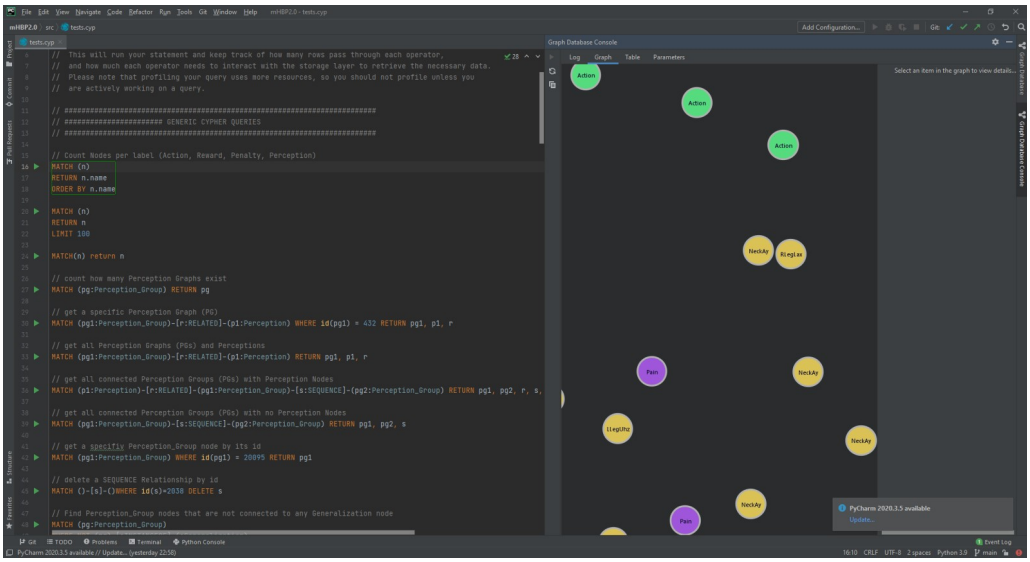


***Figure 22**: Cypher queries in pyCharm.*

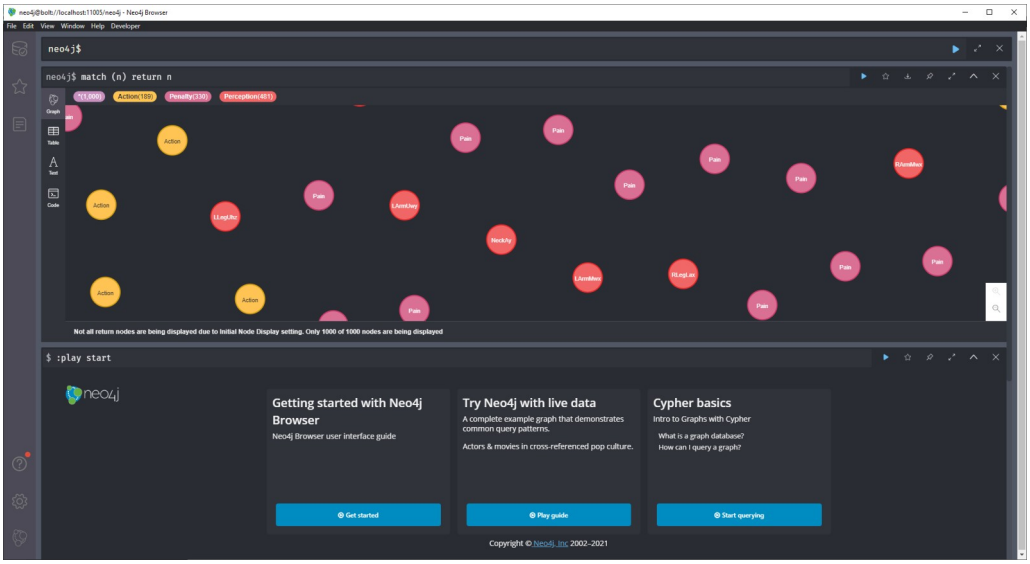


***Figure 23**: Neo4j browser.*

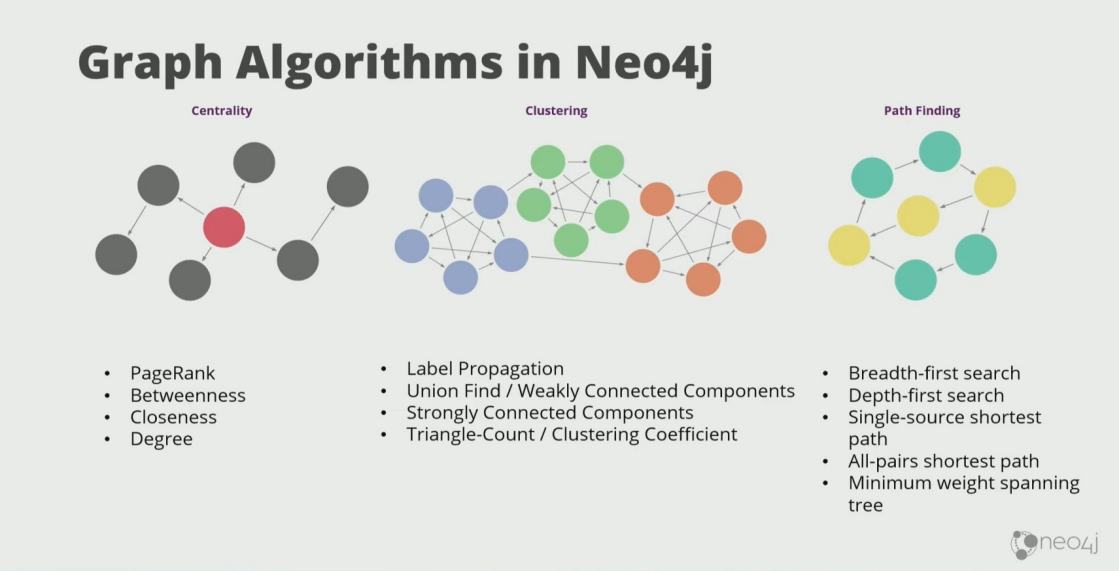
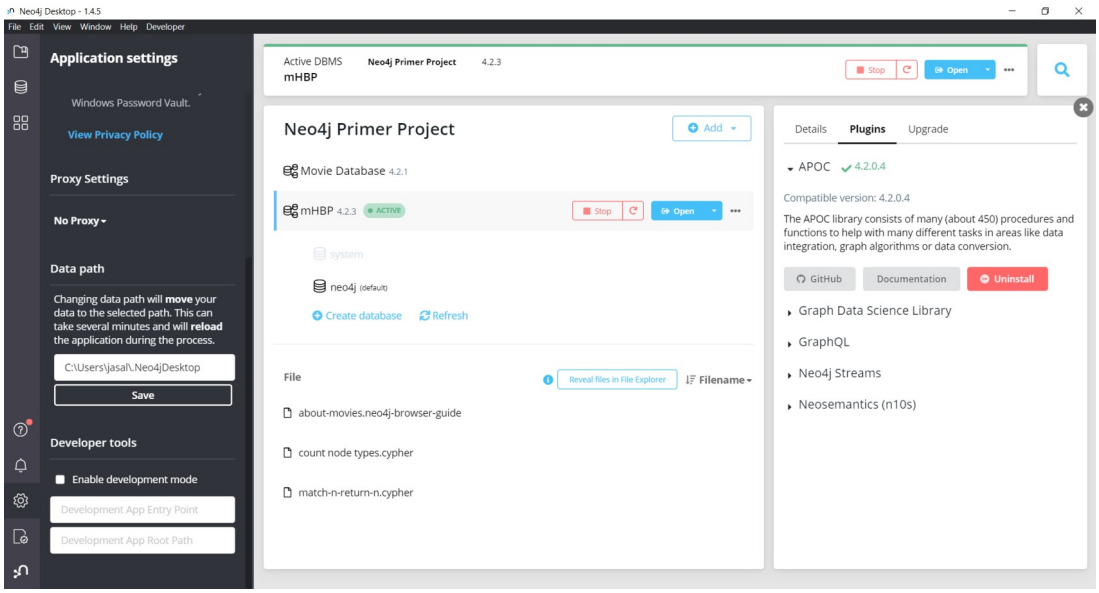


***Figure 24**: Neo4j out-of-the-box graph algorithms.*



. ***Figure 25**: Installation of Neo4j apoc plugin.*

Figure 25 shows the installation screen of the Neo4j apoc plugin. This plugin supplies a library of graph manipulation functions, some of them useful for mHBP.

3.3. Modules Development

In this section we describe how the baseline version of each module of mHBP is implemented. Any modifications to the baselines are explored in section 4. Figure 26 derives from figure 14 and includes several specific engines that are considered at the baseline implementation of mHBP.

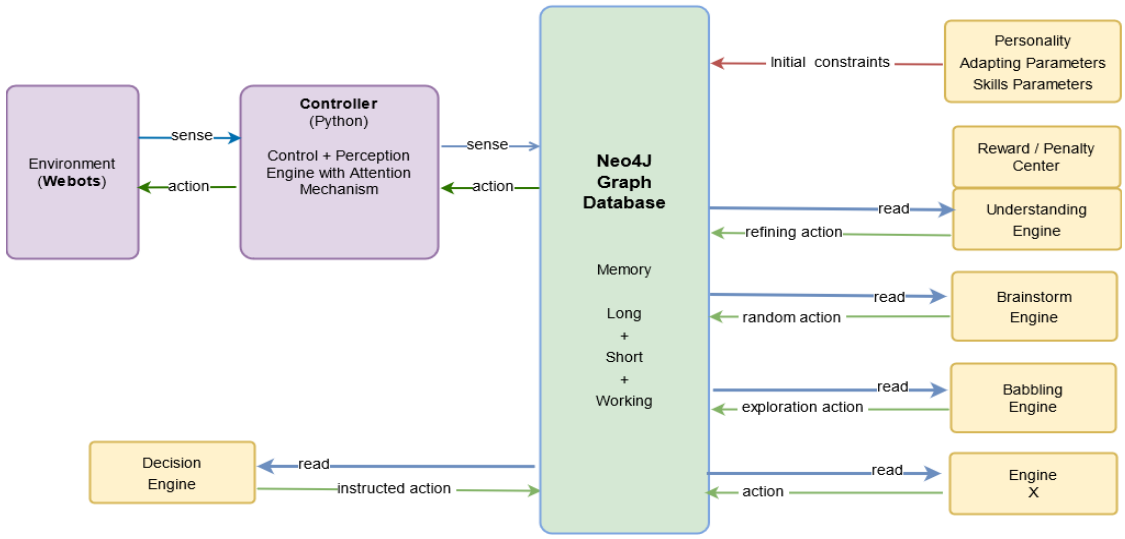


***Figure 26**: Detailed mHBP architecture.*

This design allows flexibility in the use of several algorithms, each one represented by an engine and is generic, because it is designed to accommodate any type of robot or combination of sensors and actuators / tools. Engine X represents every future engine that may be added to the system.

The Controller contains all the available micro-actions functions. These functions are triggered by the Babbling, Brainstorm or Decision Engines when they register macro-actions in the Graph Database (acting as memory).

3.3.1. Sensors and Motors

How many sensors does a robot need to have and of what kind? The answer lies in a complex causality between action and sensorial data. Sensorial data must be enough to register logical change produced by actions of the robot. It also needs not to be excessive in order for rewards and penalties to be computable, avoiding the curse of dimensionality. Actions can be physical or the creation of more concepts in the brain.

Sensorial input is key to the system since it provides integration with the Environment and is one pillar of Reinforcement Learning. Sensorial data may also be “made up” by the system itself as a way to mimic self consciousness.

One goal of this thesis is to elaborate a protocol to allow engines to have access to sensorial data concurrently. By sensorial data we mean all types of inputs. A camera is seen as a sensor, a microphone and joint positions also, and so on. This protocol is used by the

Webots controller and in each engine.

Sensorial input includes “proprioception” of the joint positions of the agent (robot). “Proprioception” is also known as “kinaesthesia “ and is a concept explored in [96]. “Proprioception” concerns the ability to evaluate body position and self movement, but can be extrapolated to include more than the physical sensorial readings.

Sensor data is realtime (no past or future data). Sensor Data is written to the Neo4j graph database respecting a timeline.

In order to allow scalability of the number of input sensors, and tackle the curse of dimensionality, most sensorial data is read in a need to read basis, and not read by default.

Figure 27 shows a part of the code of the Agent Controller accessing the devices available in the Atlas Robot at the Webots environment.

```
self.root_node_ref = self.getSelf()
##### Get sensor nodes
self.LeftEyeCamera    = self.getDevice("LeftEye")
self.RightEyeCamera   = self.getDevice("RightEye")
self.InnerEarGyro = self.getDevice("InnerEarGyro")
self.Gps = self.getDevice("Gps")
self.Inertial_Unit = self.getDevice("InertialUnit")
```

***Figure 27**: Example of device getter functions in Webots.*

3.3.2. Agent Controller with Webots

The Webots software package allows us to create a module, named a controller, that as the name indicates can be used to control an agent in the Webots world (Figure 19).

The Webots controller executes actions registered in memory with a status to be executed. This module is a multithreaded server that awaits alerts from engines, via a TCP/IP socket (Figure 28). These alerts inform the controller that new actions to be performed are available, and are treated by the thread illustrated in Figure 29.

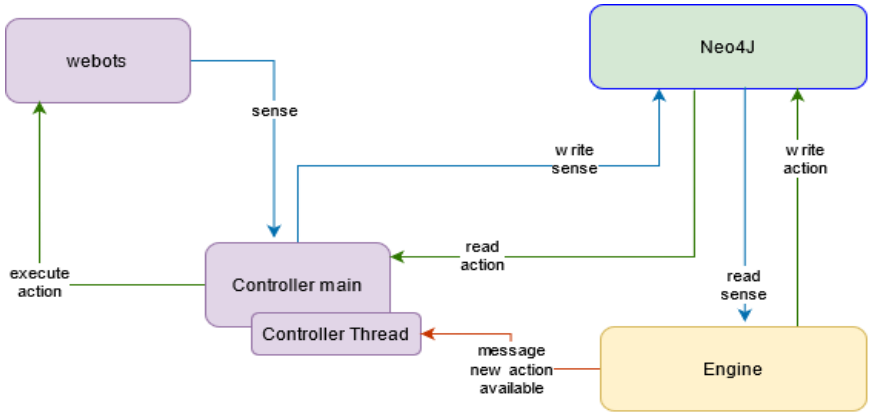


***Figure 28**: Webots controller multithreading and alert mechanism.*

```
def on_new_client(conn,addr,self):
    global event
    threadId = threading.current_thread().ident
    safeLog(str(datetime.now()),"- thread Id="+str(threadId)+ " - New thread started")
    try:
        safeLog(str(datetime.now()),"- thread Id="+str(threadId)+ " - Waiting for data in socket (1)")
        data = conn.recv(1024)
        safeLog(str(datetime.now()),"- thread Id="+str(threadId)+ " - received message ["+ data.decode("utf-8") +"] (1)")
        self.strMessage= data.decode("utf-8")
        conn.sendall(b"Received by Server")
        if self.strMessage=="Leaving":
            safeLog(str(datetime.now()),"- thread Id="+str(threadId)+ " - stopping thread (1)")
            conn.close()
            event.set()
        event.set()
    except:
        safeLog(str(datetime.now()),"- thread Id="+str(threadId)+ " - could not read from socket")
```

***Figure 29**: Source code of Webots controller thread to receive an alert.*

The multithreading design and event based mechanisms are used to avoid having a controller that needs to perform polling of memory (Neo4j graph database) searching for new actions to execute. In this way there is not an excessive use of CPU or I/O operations in the main controller cycle.

Possible basic actions that the Agent can perform are defined in the controller in the way shown in Figure 30.

```
############################
##### available Micro-actions
def function_0001(self, param1):
    """
    prints a text to the console
    param1: string: text to print
    """
    if self.stopExec == True:
        return
    print(param1)
```

```
def function_0002(self, param1, param2, param3):
    """
    sets the position of a specific motor
    param1: string: motor name
    param2: float: position
    param3: float: position multiplier
    """
    if self.stopExec == True:
        return
    self.setMotorPosition(param1, param2*param3)
```

***Figure 30**: Sample source code of Webots controller to define micro-actions.*

Like in [106, p.2] where HAM's are defined as "programs", in mHBP we also define "programs" that can be used by the agent.

mHBP uses Programmatic Actions (PA) because it has simple and complex actions it can execute and then evaluate at which state it arrives from a given state. mHBP develops programs (metaprogramming) that executes against the world. So by using python exec() method (or importing modules on-the-fly), mHBP can have modifications of its behaviour.

Like in HAMs, mHBP will use programs associated with skills/options, except that, differently from HAMs, it will have no program calling another like HAMs, but only one level macro-action and one level of planning.

The Agent Controller can include micro-actions that allow Parameter Exploration, which consists in the trial of settings the agent uses to produce its behaviour. The full mHBP Webots controller source code can be found at:
https://github.com/a10554/mHBP2.0/blob/main/src/mHBPAtlasController.py

3.3.3. Babbling Engine

The Babbling Engine simulates the way a baby explores its own physics, by performing uncoordinated actions whose sole purpose is to create a starting point for the system to have enough information to build more complex actions. Babbling can consist of micro-actions or macro-actions that result from a very simplistic merging of micro-actions. The agent learns by executing programs in a babbling logic to register the implications of its actions in itself and in the environment. The Brainstorm engine can be seen in a similar way. Both the Babbling and Brainstorm engines exist to create behavior policies (see glossary) as defined by [65, p.103]

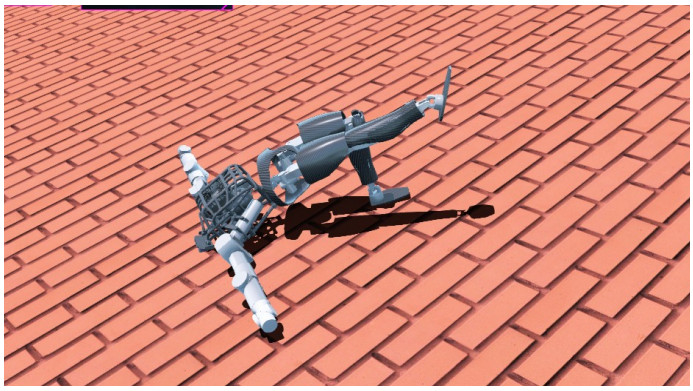


***Figure 31**: Atlas babbling.*

```
# repeat the chosen micro-action n times to create a macro-action
for i in range(1, n):
    # creating macro action in actionsArray
    actionsArray.append(config.Actions[nChosenAction])
```

***Figure 32**: Babbling baseline strategy.*

The full mHBP Babbling Engine source code can be found at:
https://github.com/a10554/mHBP2.0/blob/main/src/mHBPBabblingEngine.py

See the Babbling Engine in action at: https://www.youtube.com/watch?v=Cm9yPDxEf5c

3.3.4. Brainstorm Engine

The Brainstorm engine creates random macro-actions but differs from the Babbling engine because it explores not only the combination of parameter values for actions but also the combination of actions. [103, p.2] provides a compilation of synonyms for the term macro-actions, which can also be called options or skills depending on the author.

The combination of elements of different macro actions leads to a richer set of possible tested actions, allowing the system to have more information when evaluating the possible paths of actions to a reward when in a particular state.

The combineActionsArray is implemented as seen in Figure 33, but can be implemented in other ways, for instance by using genetic algorithms logic.

```
def combineActionsArrays(actionsArray1, actionsArray2):
    combinedActionsArray = []
    for action1 in actionsArray1:
        includeAction = randint(0,1)
        if includeAction == 1:
            combinedActionsArray.append(action1)
        for action2 in actionsArray2:
            includeAction = randint(0,1)
            if includeAction == 1:
                combinedActionsArray.append(action2)
                break
    return combinedActionsArray
```

***Figure 33**: Brainstorming baseline strategy for mixing macro-actions.*

```
# modify parameters of a macro-action
def actionsArrayChange(combinedActionsArray):
    for action in combinedActionsArray:
        for param in action["params"]:
            paramValue = action["params"][param]
            try:
                paramValue=float(paramValue)
                action["params"][param]=str(paramValue+randint(0,1)-0.5)
            except:
                pass
    return combinedActionsArray
```

***Figure 34**: Brainstorming baseline strategy for modifying micro-actions parameters.*

The “actionsArrayChange” function shown in Figure 34 is a baseline version, and can be implemented in many other ways.

The full mHBP Brainstorm Engine source code can be consulted at:
https://github.com/a10554/mHBP2.0/blob/main/src/mHBPBrainstormEngine.py

See the Brainstorm Engine in action at: https://www.youtube.com/watch?v=b2VCHDTIak8

3.3.5. Understanding Engine

The main role of the mHBP Understanding Engine (mUE) is to register Rewards or Penalties. Reward and Penalty nodes are then created in Neo4J with a timestamp near to the originating group of perceptions. With this, the Decision Engine can then find policies that take the agent from one state to another where there is a Reward. With the found policies the Decision Engine can also take notice of intermediate rewards and penalties. For further details on the Decision Engine check section 3.3.6.

The mUE can either get a Reward/Penalty signal from a human supervisor/tutor or explore the sensory data registered as nodes in Neo4j and identify patterns that trigger the creation of a Reward or Penalty node. Basically, the identification of specific patterns give place to the labeling of perception clusters or contexts.

The mUE baseline implementation only gets a Reward/Penalty signal from a human

operator/tutor.

Though not implemented, the Understanding Engine is able to use not only external rewards but also incorporate intrinsic rewards or pseudo-rewards like they are called in [36]. An Intrinsic reward can be viewed as a dopamine reward [3] that happens in specific states of the agent. This intrinsic reward can be linked to curiosity, since being able to explore may also be computed as a reward.

A function of mUE is to identify the sensory values (Perceptions) or a combination of sensory values that are associated either to a benefit or a loss Rewards and Penalties come not only from the environment but also from inside the system. The pace, scope and amplitude of the gathered or synthesized knowledge can also induce Reward or Penalty oriented behaviour.

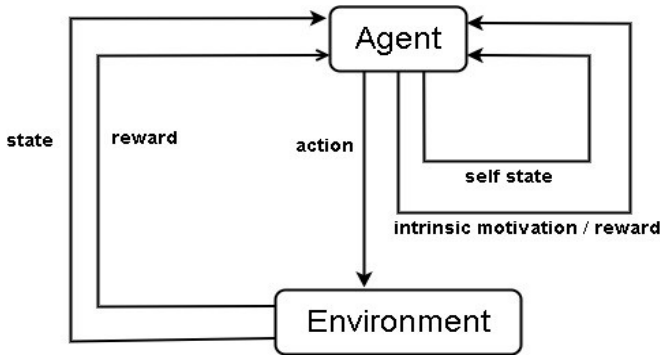


***Figure 35**: A Redesign of the RL interaction diagram including Intrinsic Motivation / Reward. [108].*

As [56, p.3] states, Intrinsic Motivation is the concept of, without having any feedback from the environment, getting the agent to perform a specific behavior. Having no feedback from the environment, implies that the agent does not have any received observations a priori semantics [56, p.4].

Intrinsic motivation may also be the drive to take the agent to a state that is not motivated by a need to get a reward from the environment, but rather to fulfill a specific agent need related to itself. Why do we humans exercise? Part may be because we have an intrinsic motivation to well maintain or improve our own bodies.

As [118, pp.171-172] refers, many authors consider emotions as a motivational system that produces behaviours in individuals helping them to adapt to the environment, alter attention and activate relevant associative networks in memory.

Attention is a mechanism that needs to be well balanced. Take for example Attention Deficit Hyperactivity Disorder (ADHD) in humans. There is a sweet spot for attention intensity. Too many readings of the environment (internal and external), and the agent gets to be fidgety and impulsive, not planning ahead, and too few and the agent becomes vegetative and not paying any attention to the context.

The attention mechanism allows the agent to perceive the environment with a manageable number of properties that can be used to identify Rewards or Penalties. This identification is compromised if there is an insufficient number of perceived elements.

A well-balanced attention mechanism allows a level of control that enables the reduction of the curse of dimensionality, but can also prevent the activation of perception patterns in mUE.

RL is considered a type of supervised Machine Learning. Supervision is widely used,

across several algorithms, and is translated into a set of expert defined rewards that enable the agent to progress in the right direction to achieve the expected goals. Now, if we compare this with how a human learns things, there is something more we can add to the previously described RL approach.

Lets see, a child has in almost every case, a mother or any kind of tutor, that contributes to signaling, through rewards and penalties (verbal and non-verbal), what goals and what skills the child is encouraged to pursue. So, an RL agent should count with the figure of a tutor to help it progress in the right direction. Not in a pre-programmed set of rules but as a continuous process during the lifespan of the agent. mUE allows a human supervisor or tutor to signal a Reward positive or negative (Penalty) at any moment.

***Table 12**: Intrinsic Motivation [108] / Rewards candidate variables.*

	Reward	Penalty	Variable
1	No pain in perception	Pain in perception	out of range action values
2	Being able to sleep when time arrives	Not being able to sleep when time arrives	difference between expected time to sleep and effective time to sleep
3	Getting attention / interaction	Not getting attention / interaction	feedback rate from other agents
4	Being able to perceive	Not being able to perceive	Number of Perceptions created in last minutes
5	Long paths of nodes (more exploration was done)	Short paths of nodes (less exploration was done)	
6	Dopamine release	No dopamine release	dopamine released quantity
7	Prediction confirmation	Prediction non confirmation	
8	Better reward than expected (advantage)	Worse Reward than Expected	Difference between estimated and effective rewards.
9	Being able to perform the same way as peers	Not being able to perform as peers	similarity of results or procedures
10	Being able to perform better than peers	Not being able to perform better than peers	speed of procedures and higher frequency of rewards
11	Less energy spent per goal	More energy spent per goal	number of action until a goal x consumption per action (1)
12	Doing less used functions once in a while	Doing the same function excessively too many times in a long period of time	number of times each function is used
13			Distance between current state and goal state
14	Executing commitment actions	Not executing commitment actions	Number of commitment action performed (a human will try to do what he/she committed to doing and feel bad if things don't happen)

Table 12 shows a list of candidate variables to generate intrinsic motivation. While they are not used in the baseline of mHBP, they can be introduced further ahead in experimentation.

The full mHBP Decision Engine source code can be consulted at:
https://github.com/a10554/mHBP2.0/blob/main/src/mHBPUnderstandingEngine.py

3.3.6. Decision Engine

The mHBP Decision Engine (mDE) enables the agent to make informed decisions based on past experiences recorded as Neo4j nodes of **the following types**: **Action**, **Perception**, **State**, **Parameter**, **Penalty (implemented as a Reward with negative value)** and **Reward**. In Neo4j mHBP database all the stimulation and perception data can be classified into "**State**" nodes that will be used by the Decision Engine to find which possible future rewards may come from a specific starting state (basically a state equal or similar to the current state). A state node is an abstraction supported by the nodes of basic types.

The mDE evaluates a policy on a as needed basis, so there is no evaluation of policies unrelated to the current state. The evaluation of policies not related to the current state approach can be executed by an additional engine or by the Understanding Engine, in a background process.

While these evaluations unrelated to the current state can be useful to improve performance of decision making in realtime, because all path (policies) values calculations would be done in advance, there are some caveats:

- scenarios with a high number of states and policies create a computationally very demanding problem;

- evaluations do not have the access to the exploration of a current state. The current state of the agent can have partial observations which we are able to improve on, like when we as humans focus our attention on additional perception variables in order to make a decision. While we can have a system which may be thorough in registering all available sensor variables at each step, in a setting with a large number of degrees of freedom, this approach represents a really high overload of the system.

Both foreground (on-policy) and background (off-policy) methods can be combined and be used according to the state of the agent. If the current state allows for an extended time of policy evaluation the agent can use a foreground approach, if not it can use policies resulting from background methods. Background calculations can also be done while the system is less loaded or idle having more available computational resources.

Figure 36 shows the Neo4j simplified cypher query that allows the mDE to find the possible paths between the starting state and possible known future Rewards.

```
MATCH p = (s: STATE) - [*] -> (r: REWARD)
RETURN p
```

***Figure 36**: Simplified Cypher Query to find all paths between a given State and possible future rewards.*

The previous cypher query is further improved to guarantee there are no circular paths. The improved version can be seen in Figure 37.

```
MATCH path = (s:State)-[:SEQUENCE*]-(r:Reward)
  UNWIND NODES(path) AS n
   WITH path,
            SIZE(COLLECT(DISTINCT n)) AS testLength
  WHERE testLength = LENGTH(path) + 1
RETURN path
```

***Figure 37**: Cypher Query to find paths between a given State and possible future rewards with guarantee of no circular paths.*

Figure 38, shows a representation of possible paths. This representation is only related to an episode, but other episodes can be related to the one shown.

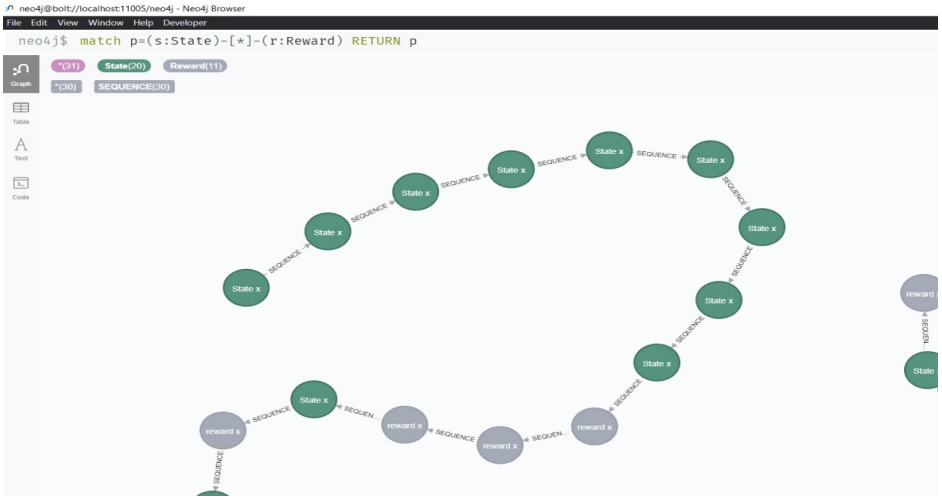


***Figure 38**: Neo4j Browser showing several possible paths between a starting State of an episode and future possible Rewards (Green=State, Gray=Reward).*

The mDE may find several other possible starting States, originating more possible paths.

In Figure 39 we show the pseudocode of the Decision Engine ideal algorithm. Each mDE cycle starts by checking the agent current state and looking for states in memory (Neo4j) that are similar to the current state (this similarity checking can be done using Unsupervised ML techniques of cluster analysis or any equivalent techniques). From similar states mDE can then identify paths from these states to possible rewards (intrinsic or extrinsic). The getPathsToRewards function is implemented by using the cypher query in Figure 37.

```
while alive == TRUE:
   pathValue = 0
   maxValuePath = None
   pathsMaxValue = - costOfDoingNothing
   currentState = getCurrentState()
   similarStates = getSimilarStates(currentState)
   pathOfMaxValue = None
   for similarState in similarStates:
      possiblePathsToReward = getPathsToRewards(similarState)
      for path in possiblePathsToReward:
         pathValue = calculatePathValue(path)
         if (pathValue > pathsMaxValue):
            maxValue = pathValue
            pathOfMaxValue = path
   startingState = getStartingState(pathOfMaxValue)
   actionsToApproximateCurrentState = getApproximationActions(
      currentState, startingState)
   for action in actionsToApproximateCurrentState:
      executeAction(action)
      expectedState = getExpectedState(action)
      currentState = getCurrentState()
      if differTooMuch(currentState, expectedState):
         continue
   pathActions = getActions(pathOfMaxValue)
   for action in pathActions:
      executeAction(action)
      expectedState = getExpectedState(action)
      currentState = getCurrentState()
      if differTooMuch(currentState,  expectedState):
         continue
```

***Figure 39**: Decision Engine pseudocode of ideal algorithm (path=policy).*

Note that there is a cost for doing nothing, that we set to a negative value. This cost of doing nothing adds a new perspective to the calculation of the value of the State. This constant basically tells the Engine that if it only finds a path with a penalty outcome less negative than the cost of doing nothing, that path is still worth following, because it is better to keep thriving than to do nothing.

The more stochastic an environment is, the less sense it makes for the agent to plan too much ahead (using a long path), because any long plan in a fast changing environment will be compromised very quickly. Any stochastic effects can be tackled in future plans. On the other hand in more stable environments the length of the path between the current state and a future reward can be longer.

After choosing the best path, mDE ideally would execute actions to try to approximate the current state to the first state of the path. This is not done in the baseline version of mDE. If this approximation is not achieved mDE should restart the main cycle. Otherwise mDE needs to execute the first macro-action in the chosen path and evaluate if the achieved intermediate state is in sync with what is expected, if it is not then the main cycle must be restarted. The macro-action implements an upper hierarchic level relative to elementary or micro-actions. Ultimately a macro-action can contain only one elementary action like considered in the Options Framework [90, p.181].

Interrupting the plan in mid execution is a necessary feature to account for stochastic environments that may change during the execution of the plan/policy, creating a need for a new plan. Our insight is the same as the Interrupt Options operation in [90, p.196].

The Decision Engine, like suggested in [92, p.2] implements a “policy-over-options”

that evaluates and modifies paths of execution and consequently options at each step, performing behaviour micro-management.

All the learning provides a base for the simulation of actions in a “Mind World”, so that the system can project what output may come from each action it is proposing itself to do. It creates a plan also considered a policy as [90, p.192] rightly states. Herein, the terms, path, policy and plan are interchangeable.

The regular check of the differTooMuch function is a step in controlling deception scenarios where the agent starts to progress with certain assumptions that later are debunked by the unexpected development of the agent state.

The calculatePathValue function can integrate a mechanism of Preference, to accomodate a strategy of Preference RL [182]. The baseline version only adds rewards in the path and subtracts the penalties in the same path.

The value of the current state is the value of the path with maximum value, and while we do not use a constant like gamma (γ) in figure 40, that can represent inflation over time and/or a cost of each action to be executed in the future to achieve a reward, the calculatePathValue function integrates, with experiment 3, a cost per action that is customized for each macro-action. In this way, mHBP explores a variant of the discounted rewards logic.

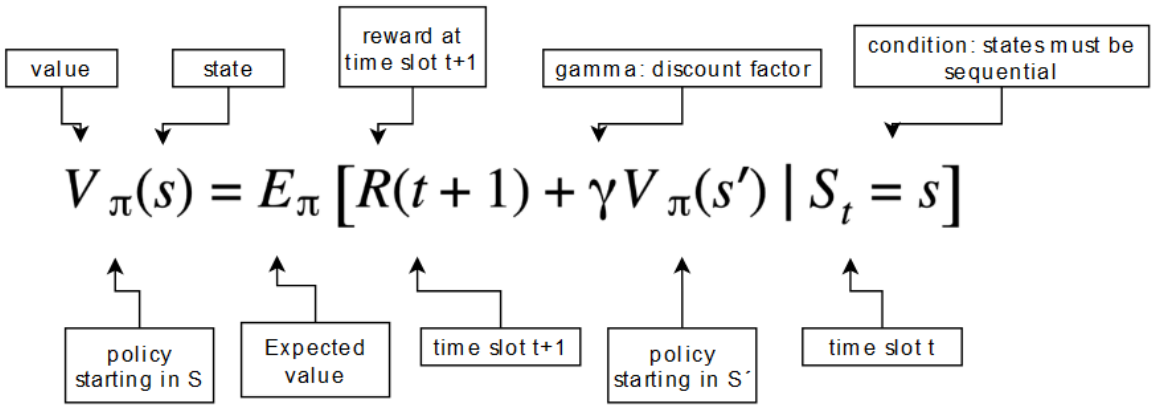


***Figure 40**: the Bellman Equation annotated- calculating the value of a policy for the current state depends only on the possible next states.*

Figure 41 illustrates, in a bidimensional space, the problem of data clustering, the same problem we face when trying to figure out if there is a previous state that with the current state, both belong to the same cluster.

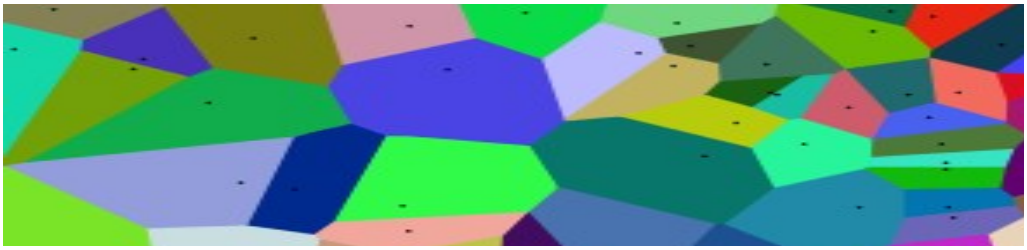


***Figure 41**: Voronoi diagram [186].*

The agent needs to identify deception to tackle deceptive domains [152, p.4]. Some perceptions may include or be a full deception, so the agent must break down the observation and try to separate the true observations from the deceptive ones. This can happen after the feedback the agent gets with each action towards the reward.

Key elements in controlling an agent with RL, as said in [72] ”is the ability to create control systems that can deal with a large movement repertoire, variable speeds, constraints and most importantly, uncertainty in the real-world environment in a fast, reactive manner.”

Model based architectures create a model of the environment, which is impractical for an infinite horizon setting with a big and changing environment. mHBP does not create a model of the environment, but it tries to understand the dynamics of the agent and the environment, the contexts identifiable within the environment.

mDE also faces the Credit Assignment Problem, that is the problem of knowing which contribution each action gave to the accomplishment of a specific Reward. This problem makes it difficult to unselect actions in a given path of actions that for sure would not lead to the expected reward, and are only an origin of cost.

We consider that running into equal decision paths in different episodes is useful to separate the factors that contributed to success from the ones that didn’t, because a frequency pattern can be obtained from the actions in common.

The full mHBP Decision Engine source code can be consulted at:
https://github.com/a10554/mHBP2.0/blob/main/src/mHBPDecisionEngine.py

See the Decision Engine in action at: https://www.youtube.com/watch?v=tcCROB4CT8I and https://www.youtube.com/watch?v=XdsWvFeNS8s

4. Experimental Results and Discussion

In this chapter we experiment with variations to the baseline of mHBP described in table 13. The experiments and obtained results are documented in the next subsections..

***Table 13**: mHBP Engines baselines.*

Babbling Engine	**Brainstorm Engine**	**Understanding Engine**	**Decision Engine**
Babbling is done by creating macro-actions from the repetition of the same micro-action with the same parametrization, originating a sequence that explores the same dynamics variable.	Brainstorm is done by combining random elements of macro-actions already in memory and also selected randomly. Numeric parameters of micro-actions are modified by small increments or decrements.	This engine basically registers Reward nodes in the mHBP memory. These Reward nodes are created when an external feedback is sent to the system or when a specific perception pattern exists in memory. The Reward nodes are the key to how the Decision Engine will guide its action from a specific state.	Decisions are performed by selecting the maximum reward path given by the following cypher query **MATCH** p = (s: **STATE**) - [*] - (r: **REWARD**) **RETURN** p The state in memory is selected by a function that maps the current state to a similar state in the past. Classifies perception sequences into state labels. Connects the states to adjacent posterior feedbacks (Rewards or Penalties), creating a chronological path of states and feedback nodes. Does not discount rewards.

All the performed experiences are cumulative except when we explicitly state in the experiment that:

- We will revert the source code changes occurred during the experiment, or
- We will disable the experiment and leave its source code in the main source code.

With these experiments we utilize the mHBP framework to search for insights on how to:

- Enable the agent to perform meaningful tasks for us humans
- Clearly show the agent is deciding based on Rewards and Penalties
- Work towards the Intrinsic Motivation mechanism as described in 3.3.5.
- Produce a system that:
    - Can be fast enough to be used in real time;
    - Has low power consumption;
    - Is able to run on limited hardware;
    - Can evolve and be able to learn and transfer learning between similar tasks;
    - Can learn its own and the environment dynamics;
    - Has proprioception;
    - Is able to identify deception.

In the following experiments we test:

- enabling and disabling some micro-actions;
- the modification of the engines to increase integration of Bablling, Brainstorming and Decision processes;
- using different values for Rewards and Penalties;
- the search for policies with different max graph relationships number;
- balancing between exploitation (Decision) and exploration (Babble/Brainstorm);
- using different values for the cost of doing nothing;
- creating additional relationships between Action and Perception Nodes to improve visual graph analysis;
- changing the number of recorded perception variables;
- organizing micro-actions in inbound and outbound groups to better control the perception mechanims;
- automatic rewards for painless moves;
- ways to deal with episode contradictions due to asymmetric tutor rewarding.

4.1. Experiment 1

Disable micro-action function_0006 that sets all motors to a default position. This micro-action can prevent the agent from creating more long running exploration. This long running exploration is important to let the agent achieve states that can be of more value than the states achieved with an interrupting exploration micro-action like function_006 is. For example, if the agent is trying to run and we call a micro-action that resets all the motors, we may never achieve a running policy.

The execution of function_0006 can be avoided through the feedback of the reward and penalty stream, so this function may be reactivated in future experiments after the referred stream is working efficiently.

The behavior resulting from the execution of function_0006 can be seen after second 59 in the video https://www.youtube.com/watch?v=Cm9yPDxEf5c&t=59s

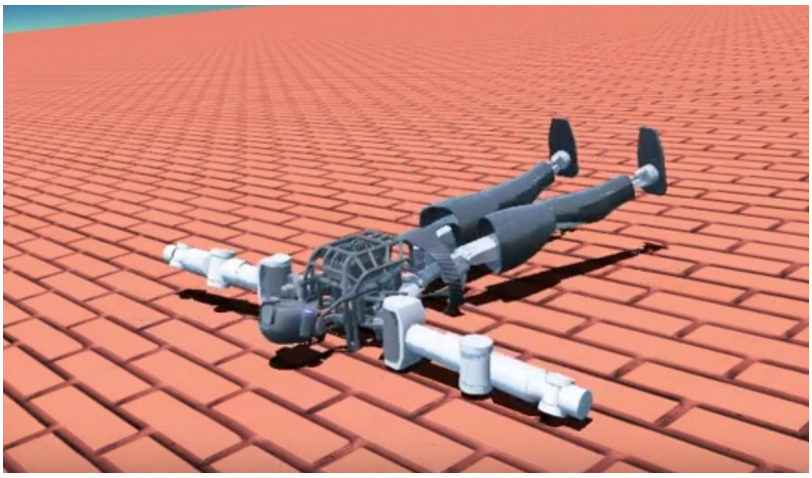


***Figure 42**: Agent state after the execution of function_0006.*

Results of experiment 1: The agent gets to more complex movements. Check behaviour in the following video:
https://www.youtube.com/watch?v=xxfkZWFwQac

4.2. Experiment 2

Babbling and Brainstorming in the baseline of mHBP occurs in a way not coordinated with the Decision Engine. Experiment 2 consists of integrating the Babbling and Brainstorming Engines with the Decision Engine. In this experiment whenever the Decision Engine can't make a decision it will resort to Babbling and Brainstorming in order to explore more.

Results of experiment 2: The agent starts from zero knowledge state and progressively and slowly gathers a repertoire of babble and brainstorm macro-actions. Check behaviour in the following video: https://www.youtube.com/watch?v=a_IzUre1qCM. For example, the video shows the progression of the robot state from a static position to a sequence of simple actions, involving several components of the robot, such as arms, legs, torso and head.

4.3. Experiment 3

This experiment adds discounted rewards in a very simple way. For every step in the evaluated policy, the Decision Engine discounts a constant value to the sum of rewards of the policy. A more complex way of adding a discount is to evaluate the cost of each action to be executed and then discount that cost from the total rewards of the policy (this is left for future work).

Taking customized action costs into account is essential to make the agent act with an efficient behavior. Let's look at the following example: when we walk we move our arms back and forth in a straight movement. We can also move our arms back and forth in a circular motion. In both cases we get a reward for being able to walk, but in the second case the circular arm movement may have a higher cost than the straight movement, thus in transitions with the same reward, the cost of the action is an important factor to make the Decision Engine choose the policy that has a lower cost and thus is more efficient.

Results of experiment 3: In this experiment the agent starts with an empty memory. The agent is now acting with a very simple implementation of discounted rewards. It gradually builds macro-actions while it is receiving positive or negative feedback from a human supervisor.

https://www.youtube.com/watch?v=3CDfNhSPczM

4.4. Experiment 4

Babbling and Brainstorming must also occur after the Decision Engine finishes executing a policy arriving at a rewarded state. This allows the agent to explore more using this state as a starting point. Babbling happens when Brainstorming is not possible.

In this experiment we also tighten the acceptable minimum distance between the current state and states in past episodes in order to search for policies to apply. We change the initial value of “maxPerceptionNameCount” from 0 to 5 in the “getBestSimilarStates” function.

Other changes made in this experiment:

- Maximum number of hops from states to rewards changed from 10 to 50 hops.
- Human supervisor reward changed from 1 to 10 units and human supervisor penalty changed from -1 to -10 units.
- Decision Engine output log is simplified to allow better identification when the system is Deciding, Brainstorming or Babbling.

Results of experiment 4: The Decision Engine is not able to find past states similar to the current state (Figure 43). This shows that the perception micro-actions are not activated with enough frequency to allow a good comparison of states. The result of this experiment is also an indication of the robustness of this framework to allow further research and refinement of the learning strategies, and also that a key problem in RL is how states are defined so that:

- They can be efficiently and easily compared with each other, and also that
- They can translate context in a way that allows the system to decide effectively and in a useful way.

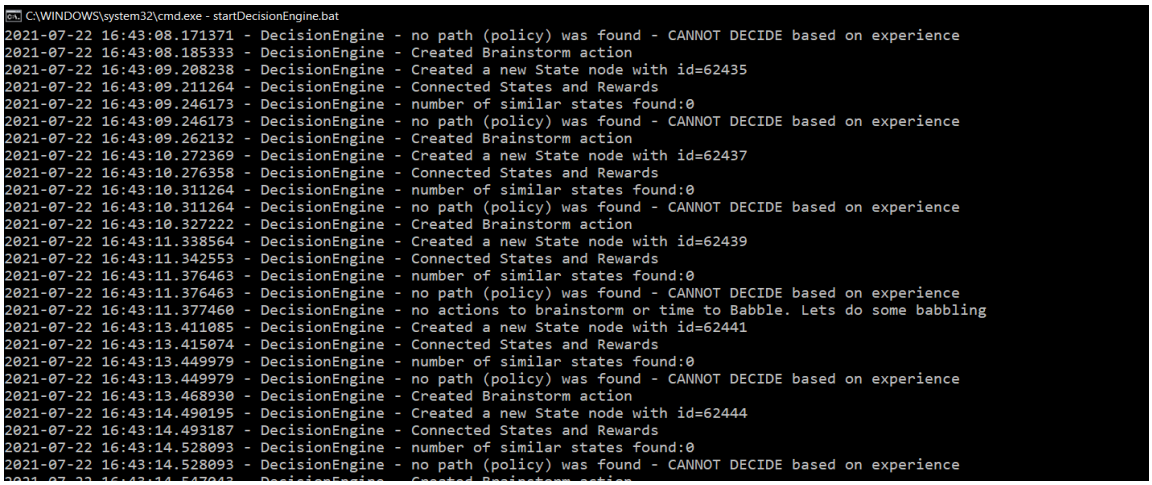


***Figure 43**: Decision Engine log in experiment 4.*

4.5. Experiment 5

In this experiment we Increase the number of perception variables registered at every moment and also the frequency of perception micro-actions.

Results of experiment 5: Despite the changes made with this experiment we still can see from the Decision Engine log (Figure 44), that the Decision Engine is still not able to make decisions. Facing this result this experiment will be disabled before experiment 6, since it did not address the indecisiveness of the Decision Engine.

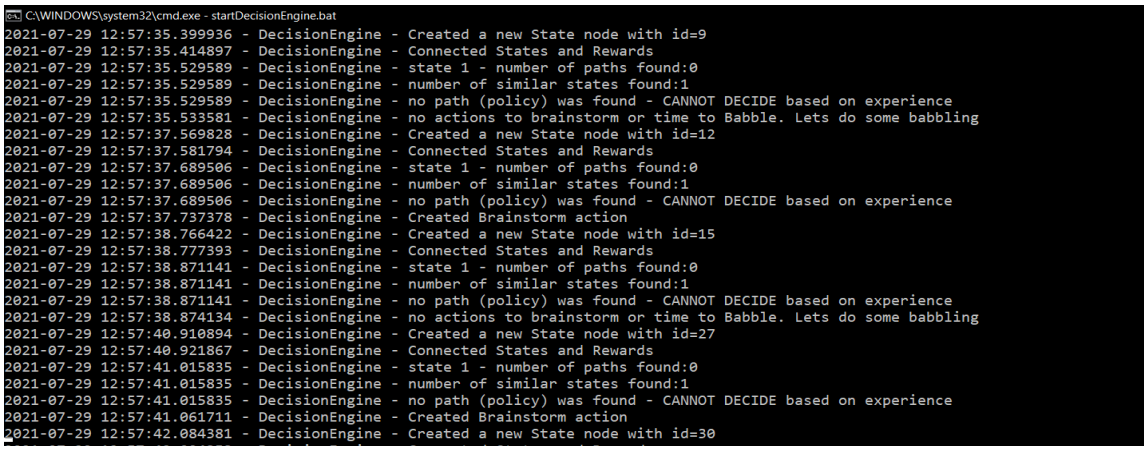


***Figure 44**: Decision Engine log in experiment 5.*

4.6. Experiment 6

This experiment aims to improve the organization of nodes in memory by:

- Creating relationships between actions and perceptions (relationship named “CREATED”);
- Creating relationships between action, states and reward nodes (relationship named “GenericSEQUENCE”);

- Renaming the relationships between States and Rewards from “SEQUENCE” to “SpecificSEQUENCE”.

These relationships are meant to provide a better visualization and consequently a better understanding of the data being registered in memory. They can also be used, in the future, by the Decision Engine to improve performance.

Results of experiment 6: With the implementation of the referred changes it is now possible to better visualize the content of the Graph database.

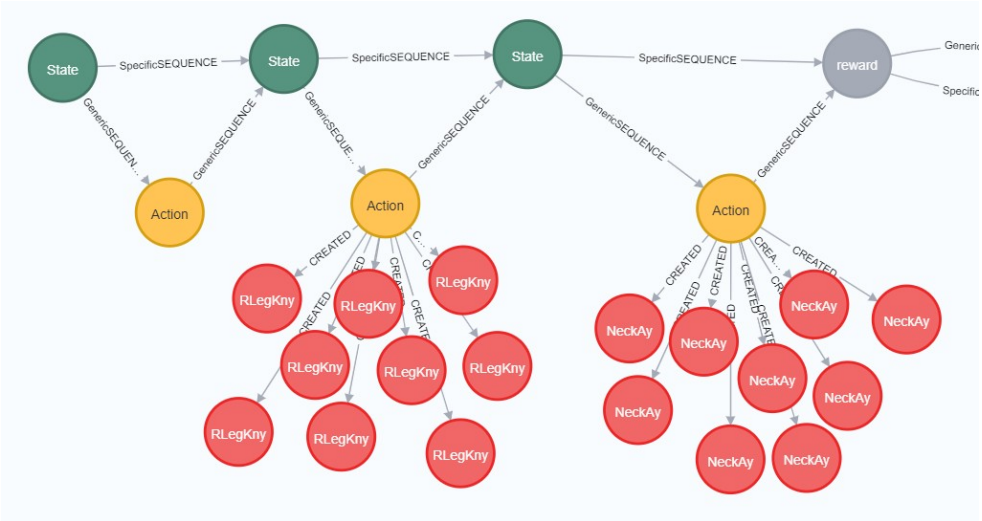


***Figure 45**: Neo4j database with new relationships.*

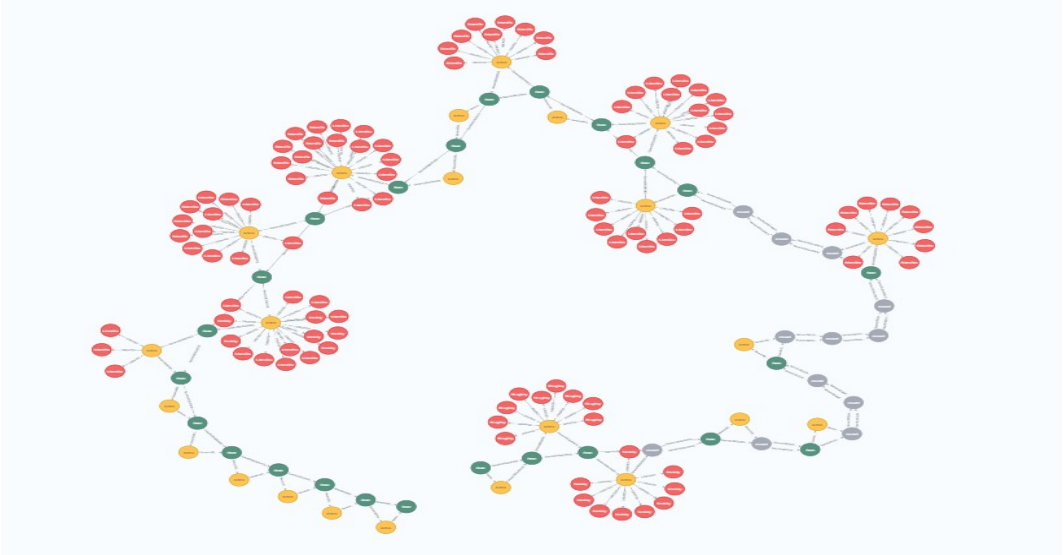


***Figure 46**: Neo4j database with new relationships (wider view).*

4.7. Experiment 7

In this experiment we separate micro-actions into different sets “in” and “out”. The “in” actions are associated with micro-actions that execute perceptions and the “out” actions with all other types of micro-actions.

With this separation it is possible to babble with these “in” and “out” micro-actions in different ways. In Babbling different “perception”, micro-actions are combined to produce a richer set of perceptions while with the “out” micro-actions babbling is done by choosing one micro-action and repeating it several times with different random parameter values.

Also during the course of this experiment we found out that “Pain” (a Penalty registered as a “Reward” node with negative value) was not being recorded in the graph database due to a syntax error in a cypher query.

Results of experiment 7: Figure 47 illustrates the results of this experiment. We are now able to better understand data being saved in Neo4J and how specific States, Actions, Perceptions and Rewards (or Penalties) are related to each other.

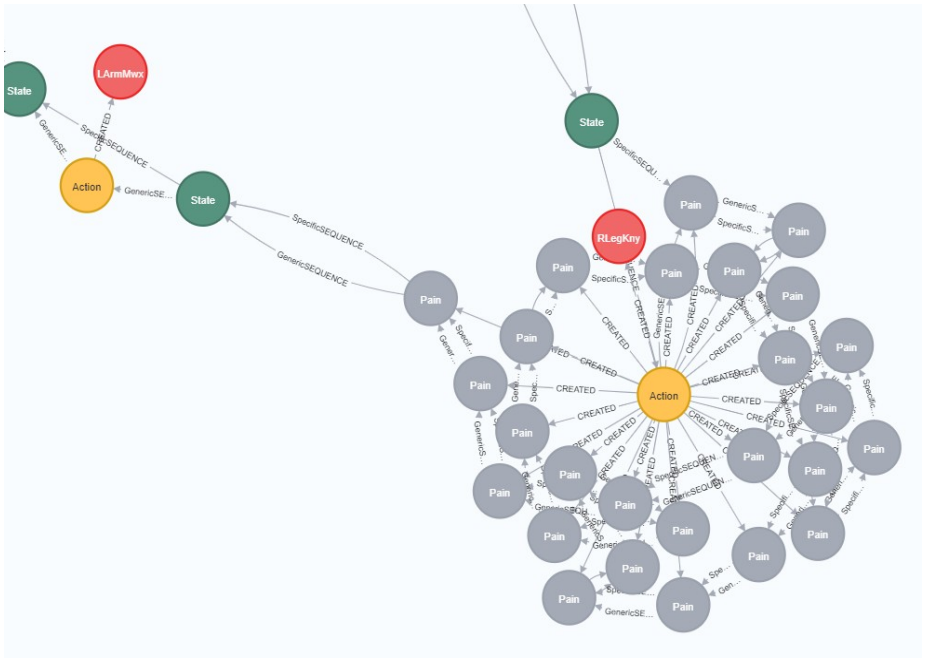


***Figure 47**: Neo4j database with “Pain” “Reward” nodes.*

4.8. Experiment 8

The Decision Engine needs to find past states that are equal or similar to the current state. The search for similar states in the graph database is executed by the getSimilarStates function implemented in the Decision Engine. The search is performed with static thresholds or boundaries in each node value comparison. The resulting nodes are then ordered to identify clusters within the same time window. The cluster with the larger node count is considered the best match, so it will identify a point in time where the agent had a state with the most similarity to the current state. Other similarity measurement approaches can be used.

In the current experiment, the goal is to enable the Decision Engine to get more

perception data after it couldn't make a decision with already read perception data. This is achieved by enabling the Decision Engine to increase, at the current state, the execution of micro-actions that allows it to extend the search of past similar states.

In this experiment we separate the babbling function into the babbling function of output actions and the babbling function of input actions. This allows us to invoke input actions babbling to help the decision engine better characterise (increase the number of percepted variables) the current state whenever it cannot decide.

Also in the AtlasController the function_0005 (register perception) is now also called inside function_0007 (move a motor). This way every time a specific micro-movement is executed there will be a corresponding perception reading of the activated motor.

During the current experiment we found a bug in function_0007, where the motor positions were not being saved into the controller memory, so any further perception of any motor was not being done correctly.

Results of experiment 8: In Figure 48 we are able to see that the system is now able to find several states in the past and policies that start at those states. All the policies found during this experiment have negative values, not allowing the system to make a decision because they are more negative than the cost of doing nothing (-1).

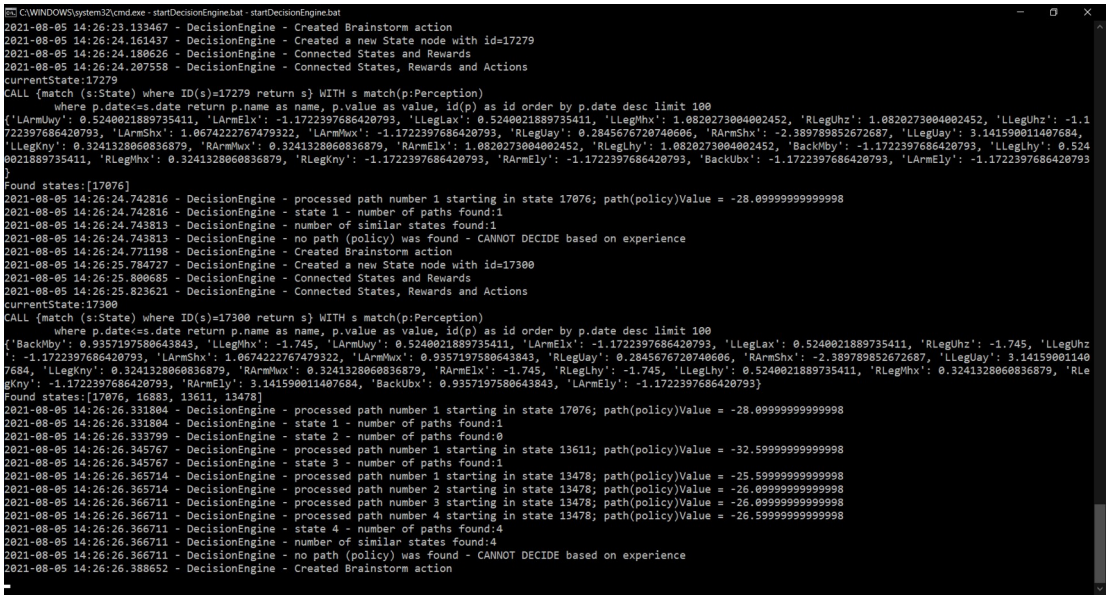


***Figure 48**: Decision Engine log after experiment 8.*

4.9. Experiment 9

In experiment 9 we add an automatic reward of 0.5 when the agent performs a micro-action that does not generate pain (pain penalty happens when a micro-action parameter is outside the acceptable physical capacities of the part of the agent to be moved). This change modifies the rewards/penalties generated by function_0002 and function_0007 in the mHBP Controller.

In this experiment we also reduce the path length (or policy length) of paths obtained by the mHBP Decision Engine. We move from 50 to 10 relationships between a state and a reward.

Results of experiment 9: as a result, the Decision Engine is now able to find paths with positive values. With the change from 50 to 10 relationships between starting states and future rewards, the number of replayed macro-actions are reduced, letting the system assess conditions more frequently.

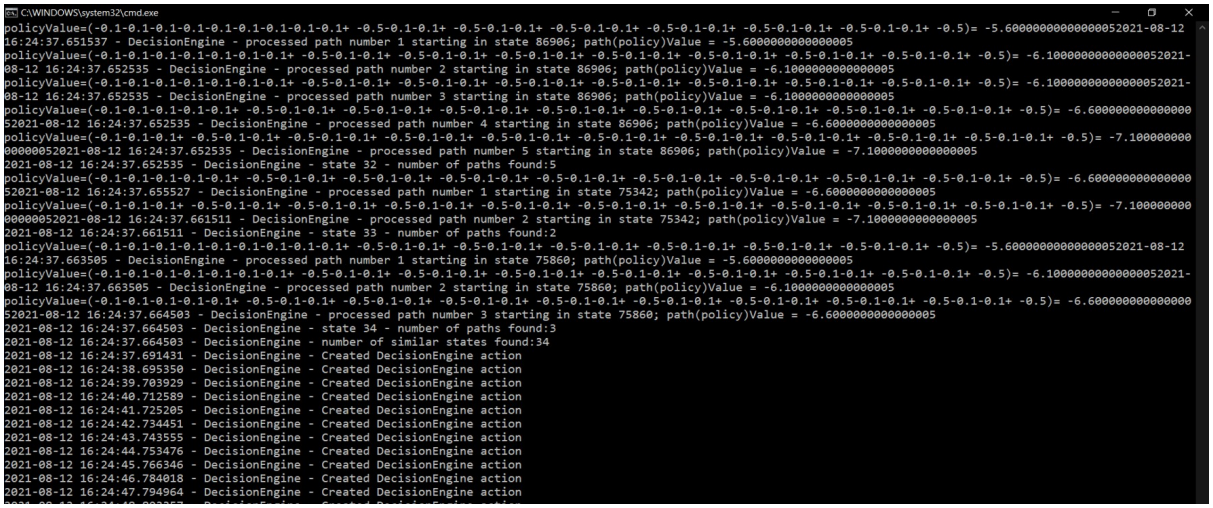


***Figure 49**: Decision Engine log after experiment 9.*

4.10. Experiment 10

In this experiment we reduce the degrees of freedom of the Atlas robot, disabling output micro-actions not related to the robot right arm. All input micro-actions (perceptions) related to the right arm are still enabled, but all others are disabled in this experiment.

Results of experiment 10: the agent performs movements only with its right arm. These movements are being generated by micro-actions 0002 and 0007. While 0002 creates a movement with an absolute value, 0007 creates a movement with a value relative to the current position. During the experiment no distinction between the two can be seen. We also detect some rolling of the body of the agent, not generated directly by any micro-action but created by the Webots physics engine when the center of mass shifts after the arm is moved. https://www.youtube.com/watch?v=K7p6M5N4HXQ

4.11. Experiment 11

Exploration before this experiment is being done at every cycle of the Decision Engine. In this experiment we run the Decision Engine for 3 hours, and as a tutor give it a Reward signal sometimes when the Atlas robot has the arm over its head, and we activate exploration only when the Decision Engine cannot decide. In this experiment we also increase the threshold to compare the current state with states in the past - change from 0.01 to 0.1 in the getSimilarStates function in the Decision Engine. This way we can increase the number of states that match or are similar to the current state.

A Reward signal from a Tutor is different from a Reward signal from an automatic Reward signal. In the first case, the tutoring reward signal has some degree of randomness, because the tutor may signal, sometimes, that an upper position of the arm is better and sometimes the tutor may not give a feedback signal. In an automatic Reward signal, the

agent has information for every time it moves the arm up, so there are no contradictions in the value of goal states. Figure 50 shows the effects of tutoring in recorded rewards. In Episode 1 the tutor didn't give any feedback, while on Episode 2 the tutor rewarded the agent for achieving the goal state.

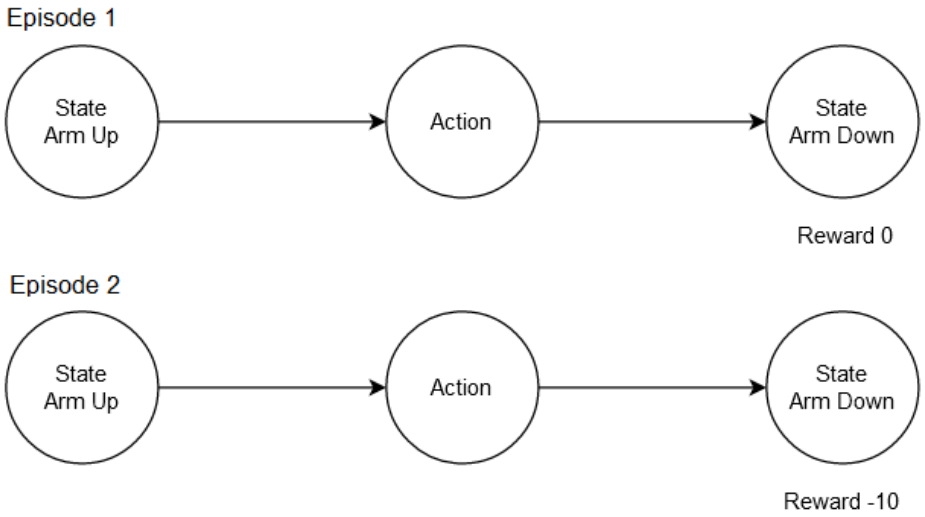


***Figure 50**: Episodes with contradicting Reward/Penalties.*

.

Results of experiment 11: the Atlas robot still makes movements that do not avoid moving the arm down. This can happen when exploration leads to a down position of the arm. We also observe in the Decision Engine log, that it is not using the policies with the largest rewards. https://youtu.be/jo8aORJxSNw

5. Conclusions and Future Work

5.1. Conclusions

In this thesis we explored an approach that is different from the current state of the art: our approach is to use a graph database, not only because it allows us to interconnect episodic data easily but also because it can hold a considerable amount of data even in a limited hardware (because it writes data to disk and is not a volatile-memory only storage). The ability to create relationships has also the potential to increase similar state search, besides being a way of interconnecting state and reward nodes.

The design of the mHBP architecture was instantiated in a software framework that enables us to perform several experiments that are executed using our multithreaded webots controller. Integration with Neo4J is also effective.

Manipulation of data in the graph database is kept simple and data structuring and search for similar states in a graph are promising proposals, in spite of our yet-to-be-improved results for meaningful control of the agent.

Figures 51 and 52 ilustrate equivalent ways to structure episodic data. The first one represents data in a tabular format, that is a more common format in the field of ML. The second one represents episodic data in a graph format, the way we implement mHBP.

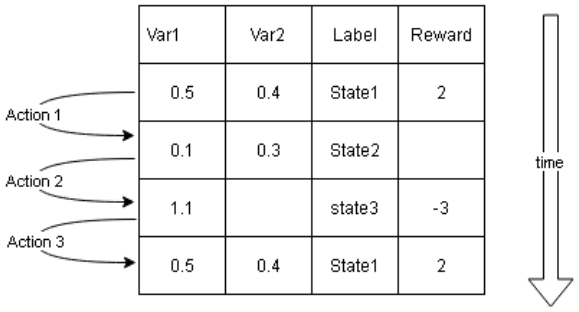


***Figure 51**: Tabular Representation of States, Actions, Rewards and Perceptions.*

Figure 51 - Tabular Representation of States, Actions, Rewards and Perceptions.

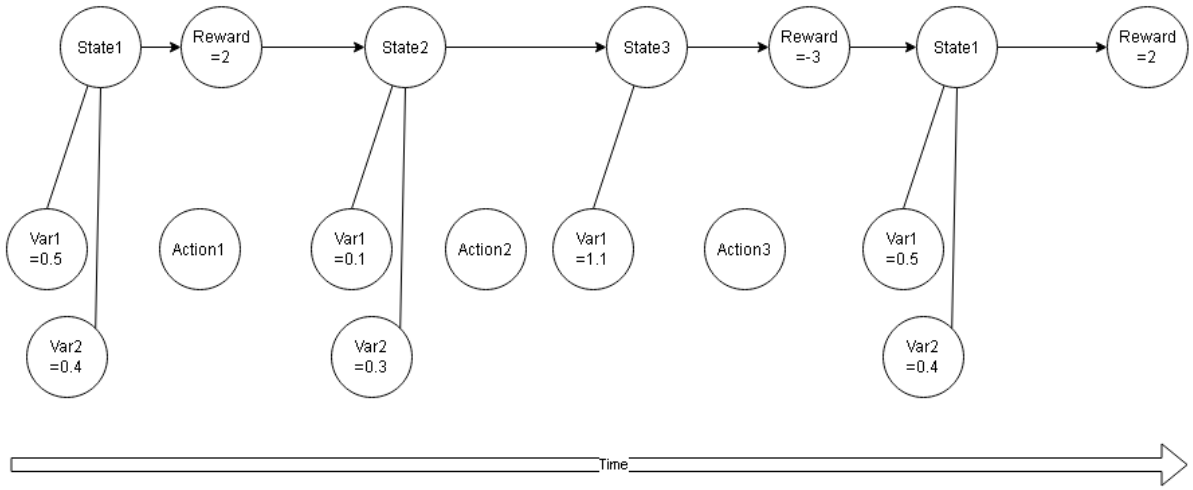


***Figure 52**: Graph Representation of States, Actions, Rewards and Perceptions.*

The hierarchical approach in spite of looking like a promising approach, may not be the ideal approach to be able to identify action effects beforehand, because the number of possible macro-actions is far larger than the number of possible micro-actions, so gathering knowledge about that larger number of possibilities is a combinatorially more demanding task than gathering knowledge only about micro-actions.

Regarding on-policy versus off-policy RL, during the current work we get the insight that while the first one is more capable of taking the current state into account which is important for a precise action of the agent, the second one may be viewed as a background process to speed-up the work of the first, which is important in realtime settings.

5.2. Future Work

After the approach we started with mHBP we can list several ways to move forward. Future work can include the topics below:

- Using a different graph database like orientdb - The performance of mHBP with Neo4J can be compared with the performance of mHBP with other graph database engines. While, with the tested workload, we didn't see any performance degradation with Neo4J, the goal is to have as low computational cost as possible.

- Improving the similarity function to find similar states in the Decision Engine - Finding similar states in the past is increasingly harder due to accumulated data in memory over time. While mHBP proposes a method to perform this search throughout the graph database, during experimentation, we were not able to prove the viability of this method, so future work may include revising this method and correct flaws or improve visualization that allow us to come to a conclusion of its effectiveness.

- Using Unsupervised ML techniques to assign a group/cluster id to states, based on the perceptions that compose each state.

- Improving the graph database design to have relationships between Actions, States and Rewards - The Decision Engine can take a better advantage of the capabilities

of graph databases, by having more interconnected episode data. If similar states are already connected, the search for related states is much faster. If micro-action nodes are created to describe a macro-action, then macro-actions can be related between themselves by the micro-actions they share.

- Improve Intrinsic Motivation (IM) in mHBP – While we implement “Pain” in mHBP as an IM variable, there are other IM candidate variables that can be implemented (see table 12).

- Categorization of Rewards/Penalties – Rewards can come from Intrinsic Motivation or tutor actions. Each type of Reward/Penalty can be categorized so that they can be treated accordingly to their specific stochasticity. Exploration of the frequency of relationships between specific types of Rewards/Penalties and states can lead to more knowledge.

- Determine which criteria needs to be used to define a state – the way a state is defined is of paramount importance to have a viable RL agent. The variables that compose the state “open doors” to the identification of patterns that really matter for a valuable decision making process.

- Find a mechanism to avoid mHBP exploring into a state that is already known to have an associated penalty – Babbling and Brainstorming, without taking into account already known states, leads the agent to have an erratic behaviour to the outside viewer and make it seems it hasn’t learned something. This work may include changing the hierarchical approach of mHBP to the flat approach to more easily predict results of actions.

- Connecting one mHBP system to several independent robots to try to achieve collaborative actions – This experiment can assess if a system that controls one agent is also capable of controlling several agents as if they are one. We can also see this in a reverse way. Each body part of an agent can be considered a robot by itself.

- Use genetic algorithms to generate agents living in the environment and have evolution dictate selection of the most viable agents (corresponding to specific sets of personality parameters).

- Comparison of mHBP with other RL algorithms performance in environments like MuJoCo.

Bibliography

[1] C. D. Schuman et al., “A Survey of Neuromorphic Computing and Neural Networks in Hardware,” CoRR, vol. abs/1705.0, 2017.

[2] P. Dayan and Y. Niv, “Reinforcement learning: The Good, The Bad and The Ugly,” Current Opinion in Neurobiology, vol. 18, no. 2. Elsevier Current Trends, pp. 185–196, 01-Apr-2008.

[3] P. W. Glimcher, “Understanding dopamine and reinforcement learning: The dopamine reward predictio*n error hypothesis,” Proc. Natl. Acad. Sci., vol. 108, no. Supplement_3, pp. 15647–15654, 2011.

[4] M. Danel and M. Skrbek, “Reinforcement learning for humanoid robot control,” POSTER, May, 2017.

[5] F. Qi and W. Wu, “Human-like machine thinking: Language guided imagination,” CoRR, vol. abs/1905.0, 2019.

[6] S. Gu, E. Holly, T. Lillicrap, S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in 2017 {IEEE} International Conference on Robotics and Automation, {ICRA} 2017, pp. 3389-3396.

[7] S. Iida, S. Kato, K. Kuwayama, T. Kunitachi, M. Kanoh, and H. Itoh, “Humanoid robot control based on reinforcement learning,” in Micro-Nanomechatronics and Human Science, 2004 and The Fourth Symposium Micro-Nanomechatronics for Information-Based Society, 2004., 2004, pp. 353–358.

[8] Z. Sun and N. Roos, “An energy efficient dynamic gait for a Nao robot,” in 2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), 2014, pp. 267–272.

[9] O. Tutsoy, D. Erol Barkana and S. Colak, "Learning to balance an NAO robot using reinforcement learning with symbolic inverse kinematic", Transactions of the Institute of Measurement and Control, vol. 39, no. 11, pp. 1735-1748, 2016. Available: 10.1177/0142331216645176.

[10] D. Ha and J. Schmidhuber, “Recurrent World Models Facilitate Policy Evolution,” in Advances in Neural Information Processing Systems 31, Curran Associates, Inc., 2018, pp. 2451–2463.

[11] M. Otsuka, J. Yoshimoto, and K. Doya, “Free-energy-based Reinforcement Learning in a Partially Observable Environment,” in Proceedings of the European Symposium on Artificial Neural Networks (ESANN2010), 2010.

[12] V. Mnih et al., “Asynchronous Methods for Deep Reinforcement Learning,” in Proceedings of The 33rd International Conference on Machine Learning, 2016, vol. 48, pp. 1928–1937.

[13] D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel, “Reinforcement Learning Versus Model Predictive Control: A Comparison on a Power System Problem,” IEEE Trans. Syst. Man, Cybern. Part B, vol. 39, no. 2, pp. 517–529, 2009.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms.” 2017.

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust Region Policy Optimization,” in Proceedings of the 32nd International Conference on Machine Learning, 2015, vol. 37, pp. 1889–1897.

[16] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” CoRR, vol. abs/1509.0, 2015.

[17] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” in Proceedings of the 35th International Conference on Machine Learning, 2018, vol. 80, pp. 1587–1596.

[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” in Proceedings of the 35th International Conference on Machine Learning, 2018, vol. 80, pp. 1861–1870.

[19] V. Mnih et al., “Playing Atari With Deep Reinforcement Learning,” in NIPS Deep Learning Workshop, 2013.

[20] M. G. Bellemare, W. Dabney, and R. Munos, “A Distributional Perspective on Reinforcement Learning,” in Proceedings of the 34th International Conference on Machine Learning, 2017, vol. 70, pp. 449–458.

[21] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, “Distributional Reinforcement Learning With Quantile Regression,” in Proceedings of the Thirty-Second {AAAI} Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th {AAAI} Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New, 2018, pp. 2892–2901.

[22] M. Andrychowicz et al., “Hindsight experience replay,” in Advances in Neural Information Processing Systems, 2017, vol. 2017-Decem, pp. 5049–5059.

[23] S. Racanière et al., “Imagination-Augmented Agents for Deep Reinforcement Learning,” in Advances in Neural Information Processing Systems 30, I. Guyon, U. V Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5690–5701.

[24] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning,” in 2018 {IEEE} International Conference on Robotics and Automation, {ICRA} 2018, Brisbane, Australia, May 21-25, 2018, 2018, pp. 7559–7566.

[25] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, “Model-

Based Value Estimation for Efficient Model-Free Reinforcement Learning," CoRR, vol. abs/1803.0, 2018.

[26] D. Silver et al., "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," CoRR, vol. abs/1712.0, 2017.

[27] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.

[28] L. Kaiser et al., "Model Based Reinforcement Learning for Atari," in 8th International Conference on Learning Representations, {ICLR} 2020, Addis Ababa, Ethiopia, April 26-30, 2020, 2020.

[29] Q. Cai, L. Pan, and P. Tang, "Deterministic Value-Policy Gradients," in The Thirty-Fourth AAAI Conference on Artificial Intelligence, {AAAI} 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, {IAAI} 2020, The Tenth {AAAI} Symposium on Educational Advances in Artificial Intelligence, {EAAI, 2020, pp. 3316–3323.

[30] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," in 4th International Conference on Learning Representations, {ICLR} 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[31] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 4565–4573.

[32] Z. Wang et al., "Sample Efficient Actor-Critic with Experience Replay," in 5th International Conference on Learning Representations, {ICLR} 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[33] P. W. Battaglia et al., "Relational inductive biases, deep learning, and graph networks," CoRR, vol. abs/1806.0, 2018.

[34] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," in Advances in Neural Information Processing Systems 30, I. Guyon, U. V Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5279–5288.

[35] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in Proceedings of the 32nd International Conference on Machine Learning, 2015, vol. 37, pp. 448–456.

[36] M. Jaderberg et al., "Reinforcement Learning with Unsupervised Auxiliary Tasks," in 5th International Conference on Learning Representations, {ICLR} 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[37] G. Brockman, "Hearing on The Dawn of Artificial Intelligence," US Committee on Science, Space and Technology, 2018.

[38] A. Kaplan and M. Haenlein, “Siri, Siri, in my hand: Who’s the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence,” Bus. Horiz., vol. 62, no. 1, pp. 15–25, 2019.

[39] D. Steckelmacher, H. Plisnier, D. M. Roijers, and A. Nowé, “Sample-Efficient Model-Free Reinforcement Learning with Off-Policy Critics,” in Machine Learning and Knowledge Discovery in Databases - European Conference, {ECML} {PKDD} 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part {III}, 2019, vol. 11908, pp. 19–34.

[40] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, “Deep Exploration via Bootstrapped DQN,” in Advances in Neural Information Processing Systems 29, D. D. Lee, M. Sugiyama, U. V Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4026–4034.

[41] H. Kim, M. Yamada, K. Miyoshi, and H. Yamakawa, “Macro Action Reinforcement Learning with Sequence Disentanglement using Variational Autoencoder.” 2019.

[42] S. Ivanov and A. D’yakonov, “Modern Deep Reinforcement Learning Algorithms.” 2019.

[43] L. N. Smith and N. Topin, “Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.” 2017.

[44] A. Vaswani et al., “Attention is All you Need,” in Advances in Neural Information Processing Systems 30, I. Guyon, U. V Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008.

[45] Q. Liu et al., “Exploiting Cognitive Structure for Adaptive Learning,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 627–635.

[46] W. Shi, S. Song, and C. Wu, “Soft Policy Gradient Method for Maximum Entropy Deep Reinforcement Learning,” in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, {IJCAI-19}, 2019, pp. 3425–3431.

[47] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic Policy Gradient Algorithms,” in Proceedings of the 31st International Conference on Machine Learning - Volume 32, 2014, pp. I–387–I–395.

[48] M. Riedmiller, “Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method,” in Machine Learning: ECML 2005, 2005, pp. 317–328.

[49] N. Y. Siegel et al., “Keep Doing What Worked: Behavioral Modelling Priors for Offline Reinforcement Learning,” CoRR, vol. abs/2002.0, 2020.

[50] R. Hafner and M. Riedmiller, “Reinforcement learning in feedback control,” Mach. Learn., vol. 84, no. 1, pp. 137–169, 2011.

[51] S. Lange, T. Gabel, and M. A. Riedmiller, “Batch Reinforcement Learning,” in Reinforcement Learning, vol. 12, M. Wiering and M. van Otterlo, Eds. Springer, 2012, pp. 45–73.

[52] T. Anthony, Z. Tian, and D. Barber, “Thinking Fast and Slow with Deep Learning and Tree Search,” 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. 5360–5370.

[53] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola, “Meta-Q-Learning,” in International Conference on Learning Representations, 2020.

[54] M. Janner, J. Fu, M. Zhang, and S. Levine, “When to Trust Your Model: Model-Based Policy Optimization,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 12519–12530.

[55] C. Lyle, M. G. Bellemare, and P. S. Castro, “A Comparative Analysis of Expected and Distributional Reinforcement Learning,” in The Thirty-Third {AAAI} Conference on Artificial Intelligence, {AAAI} 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, {IAAI} 2019, The Ninth {AAAI} Symposium on Educational Advances in Artificial Intelligence, {EAAI} , 2019, pp. 4504–4511.

[56] A. Aubret, L. Matignon, and S. Hassas, “A survey on intrinsic motivation in reinforcement learning,” CoRR, vol. abs/1908.0, 2019.

[57] H. Van Hoof, G. Neumann, and J. Peters, “Non-Parametric Policy Search with Limited Information Loss,” J. Mach. Learn. Res., vol. 18, no. 1, pp. 2472–2517, 2017.

[58] R. M. Viccari, P. A. Jaques, and R. Verdin, Agent-based tutoring systems by cognitive and affective modeling. Information Science Reference, 2008.

[59] D. Freeman, D. Ha, and L. Metz, “Learning to Predict Without Looking Ahead: World Models Without Forward Prediction,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 5379–5390.

[60] M. Hessel et al., “Rainbow: Combining Improvements in Deep Reinforcement Learning,” in Proceedings of the Thirty-Second {AAAI} Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th {AAAI} Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New, 2018, pp. 3215–3222.

[61] W. B. Knox and P. Stone, “Interactively shaping agents via human reinforcement: the TAMER framework,” in Proceedings of the 5th International Conference on Knowledge Capture {(K-CAP} 2009), September 1-4, 2009, Redondo Beach, California, {USA}, 2009, pp. 9–16.

[62] D. W. F. Jardim and L. M. M. Nunes, “Hierarchical reinforcement learning: learning sub-goals and state-abstraction.” 2011.

[63] M. Hausknecht and P. Stone, “On-Policy vs. Off-Policy Updates for Deep Reinforcement Learning,” in Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop.

[64] A. Y. Ng and S. J. Russell, “Algorithms for Inverse Reinforcement Learning,” in ICML, 2000.

[65] R. S. Sutton and A. G. Barto, Reinforcement learning: an introduction, Second edi. The MIT Press, 2018.

[66] Y. Li, “Deep Reinforcement Learning,” arXiv1810.06339 [cs, stat], 2018.

[67] T. Hartvigsen, C. Sen, X. Kong, and E. Rundensteiner, “Adaptive-Halting Policy Network for Early Classification,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 101–110.

[68] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, “Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 11784–11794.

[69] R. Agarwal, D. Schuurmans, and M. Norouzi, “An Optimistic Perspective on Offline Reinforcement Learning,” arXiv1907.04543 [cs, stat], Mar. 2020.

[70] G. Barth-Maron et al., “Distributional Policy Gradients,” in International Conference on Learning Representations, 2018.

[71] R. Lowe, Y. I. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments,” in Advances in Neural Information Processing Systems 30, I. Guyon, U. V Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6379–6390.

[72] J. Peters, S. Vijayakumar, and S. Schaal, “Reinforcement learning for humanoid robotics,” Proc. third IEEE-RAS Int. Conf. humanoid Robot., pp. 1–20, 2003.

[73] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized Experience Replay,” in 4th International Conference on Learning Representations, {ICLR} 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[74] Y. Liu, P. Ramachandran, Q. Liu, and J. Peng, “Stein Variational Policy Gradient,” in Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, {UAI} 2017, Sydney, Australia, August 11-15, 2017, 2017.

[75] L. Espeholt et al., “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures,” in Proceedings of the 35th International Conference on Machine Learning, 2018, vol. 80, pp. 1407–1416.

[76] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous Deep Q-Learning with Model-based Acceleration,” in Proceedings of The 33rd International Conference on Machine Learning, 2016, vol. 48, pp. 2829–2838.

[77] R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist

Reinforcement Learning," Mach. Learn., vol. 8, pp. 229–256, 1992.

[78] V. Mnih et al., "Human-level control through deep reinforcement learning", Nature, vol. 518, no. 7540, pp. 529-533, 2015. Available: 10.1038/nature14236.

[79] B. Kim, K. Lee, S. Lim, L. P. Kaelbling, and T. Lozano-Pérez, "Monte Carlo Tree Search in Continuous Spaces Using Voronoi Optimistic Optimization with Regret Bounds," in The Thirty-Fourth {AAAI} Conference on Artificial Intelligence, {AAAI} 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, {IAAI} 2020, The Tenth {AAAI} Symposium on Educational Advances in Artificial Intelligence, {EAAI, 2020, pp. 9916–9924.

[80] A. Levy, G. D. Konidaris, R. P. Jr., and K. Saenko, "Learning Multi-Level Hierarchies with Hindsight," in 7th International Conference on Learning Representations, {ICLR} 2019, New Orleans, LA, USA, May 6-9, 2019, 2019.

[81] C. Szepesvári, Algorithms for Reinforcement Learning. Morgan & Claypool Publishers, 2010.

[82] V. Kuzmin and A. Panov, "Hierarchical Reinforcement Learning with Options and United Neural Network Approximation: Volume 1," 2019, pp. 453–462.

[83] T. Aoki, T. Nakamura, and T. Nagai, "Learning of Motor Control from Motor Babbling**This research is supported by CREST, JST.," IFAC-PapersOnLine, vol. 49, no. 19, pp. 154–158, 2016.

[84] L. N. Kanal, "Perceptron," in Encyclopedia of Computer Science, GBR: John Wiley and Sons Ltd., 2003, pp. 1383–1385.

[85] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards Deep Symbolic Reinforcement Learning," arXiv1609.05518 [cs], 2016.

[86] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in 2009 IEEE 12th International Conference on Computer Vision, 2009, pp. 2146–2153.

[87] K. Liu and A. Bellet, "Escaping the Curse of Dimensionality in Similarity Learning: Efficient Frank-Wolfe Algorithm and Generalization Bounds," Neurocomputing, vol. 333, pp. 185–199, Mar. 2019.

[88] Y. LeCun, Y. Bengio, and others, "Convolutional networks for images, speech, and time series," Handb. brain theory neural networks, vol. 3361, no. 10, p. 1995, 1995.

[89] P. Dayan and G. E. Hinton, "Feudal Reinforcement Learning," in Advances in Neural Information Processing Systems 5, {[NIPS} Conference, Denver, Colorado, USA, November 30 - December 3, 1992], 1992, pp. 271–278.

[90] R. S. Sutton, D. Precup, and S. P. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," Artif. Intell., vol. 112, no. 1–2, pp. 181–211, 1999.

[91] T. G. Dietterich, “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition,” J. Artif. Intell. Res., vol. 13, pp. 227–303, 2000.

[92] A. S. Vezhnevets et al., “FeUdal Networks for Hierarchical Reinforcement Learning,” in Proceedings of the 34th International Conference on Machine Learning, {ICML} 2017, Sydney, NSW, Australia, 6-11 August 2017, 2017, vol. 70, pp. 3540–3549.

[93] P.-L. Bacon, J. Harb, and D. Precup, “The Option-Critic Architecture,” in Proceedings of the Thirty-First {AAAI} Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, {USA}, 2017, pp. 1726–1734.

[94] O. Nachum, S. Gu, H. Lee, and S. Levine, “Data-Efficient Hierarchical Reinforcement Learning,” in Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada, 2018, pp. 3307–3317.

[95] B. O’Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih, “Combining policy gradient and Q-learning,” in 5th International Conference on Learning Representations, {ICLR} 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[96] N. Heess, G. Wayne, Y. Tassa, T. P. Lillicrap, M. A. Riedmiller, and D. Silver, “Learning and Transfer of Modulated Locomotor Controllers,” CoRR, vol. abs/1610.0, 2016.

[97] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation,” in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 3675–3683.

[98] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, “Meta Learning Shared Hierarchies,” in 6th International Conference on Learning Representations, {ICLR} 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[99] A. Pashevich, D. Hafner, J. Davidson, R. Sukthankar, and C. Schmid, “Modulated Policy Hierarchies,” CoRR, vol. abs/1812.0, 2018.

[100] A. Vezhnevets et al., “Strategic Attentive Writer for Learning Macro-Actions,” in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 3486–3494.

[101] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, “A Deep Hierarchical Approach to Lifelong Learning in Minecraft,” in Proceedings of the Thirty-First {AAAI} Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, {USA}, 2017, pp. 1553–1561.

[102] N. Gopalan et al., “Planning with Abstract Markov Decision Processes.” 2017.

[103] D. J. Mankowitz, T. A. Mann, and S. Mannor, “Iterative Hierarchical Optimization for Misspecified Problems (IHOMP),” CoRR, vol. abs/1602.0, 2016.

[104] S. Sukhbaatar, E. Denton, A. Szlam, and R. Fergus, “Learning Goal Embeddings via Self-Play for Hierarchical Reinforcement Learning,” CoRR, vol. abs/1811.0, 2018.

[105] J. Rafati and D. C. Noelle, “Learning Representations in Model-Free Hierarchical Reinforcement Learning,” in The Thirty-Third {AAAI} Conference on Artificial Intelligence, {AAAI} 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, {IAAI} 2019, The Ninth {AAAI} Symposium on Educational Advances in Artificial Intelligence, {EAAI} , 2019, pp. 10009–10010.

[106] R. Parr and S. J. Russell, “Reinforcement Learning with Hierarchies of Machines,” in Advances in Neural Information Processing Systems 10, {[NIPS} Conference, Denver, Colorado, USA, 1997], 1997, pp. 1043–1049.

[107] C. Watkins, “Learning From Delayed Rewards,” King’s College, 1989.

[108] G. Baldassarre, T. Stafford, M. Mirolli, P. Redgrave, R. Ryan, and A. Barto, “Intrinsic motivations and open-ended development in animals, humans, and robots: An overview,” Front. Psychol., vol. 5, p. 985, 2014.

[109] A. A. Rusu et al., “Policy Distillation,” in 4th International Conference on Learning Representations, {ICLR} 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[110] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, “Safe and Efficient Off-Policy Reinforcement Learning,” in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 1046–1054.

[111] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement Learning with Deep Energy-Based Policies,” in Proceedings of the 34th International Conference on Machine Learning, {ICML} 2017, Sydney, NSW, Australia, 6-11 August 2017, 2017, vol. 70, pp. 1352–1361.

[112] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.

[113] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, “Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play,” in 6th International Conference on Learning Representations, {ICLR} 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[114] K. Chua, R. Calandra, R. McAllister, and S. Levine, “Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models,” in Advances in Neural Information Processing Systems 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 4754–4765.

[115] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, “Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion,” in Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing

Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada, 2018, pp. 8234–8244.

[116] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma, “Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees,” in 7th International Conference on Learning Representations, {ICLR} 2019, New Orleans, LA, USA, May 6-9, 2019, 2019.

[117] E. Parisotto, L. J. Ba, and R. Salakhutdinov, “Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning,” in 4th International Conference on Learning Representations, {ICLR} 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[118] A. H. Fischer, A. S. R. Manstead, and R. Zaalberg, “Social influences on the emotion process,” Eur. Rev. Soc. Psychol., vol. 14, no. 1, pp. 171–201, 2003.

[119] H. van Hasselt, “Double Q-learning,” in Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada, 2010, pp. 2613–2621.

[120] S. Sharma, A. S. Lakshminarayanan, and B. Ravindran, “Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning,” in 5th International Conference on Learning Representations, {ICLR} 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[121] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, “Learning Continuous Control Policies by Stochastic Value Gradients,” in Advances in Neural Information Processing Systems 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2944–2952.

[122] J. Oh, S. Singh, and H. Lee, “Value Prediction Network,” 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. 6118–6128.

[123] A. Swaminathan and T. Joachims, “Batch Learning from Logged Bandit Feedback through Counterfactual Risk Minimization,” J. Mach. Learn. Res., vol. 16, no. 1, pp. 1731–1755, 2015.

[124] G. Rummery and M. Niranjan, “On-Line Q-Learning Using Connectionist Systems,” Tech. Rep. CUED/F-INFENG/TR 166, 1994.

[125] Z. Lin, G. Thomas, G. Yang, and T. Ma, “Model-based Adversarial Meta-Reinforcement Learning,” CoRR, vol. abs/2006.0, 2020.

[126] C. Finn, P. Abbeel, and S. Levine, “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks,” in Proceedings of the 34th International Conference on Machine Learning, {ICML} 2017, Sydney, NSW, Australia, 6-11 August 2017, 2017, vol. 70, pp. 1126–1135.

[127] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, “Efficient Off-Policy Meta-

Reinforcement Learning via Probabilistic Context Variables," in Proceedings of the 36th International Conference on Machine Learning, {ICML} 2019, 9-15 June 2019, Long Beach, California, {USA}, 2019, vol. 97, pp. 5331–5340.

[128] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "ProMP: Proximal Meta-Policy Search," in 7th International Conference on Learning Representations, {ICLR} 2019, New Orleans, LA, USA, May 6-9, 2019, 2019.

[129] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning," CoRR, vol. abs/1611.0, 2016.

[130] M. Dudik, J. Langford, and L. Li, "Doubly Robust Policy Evaluation and Learning," in Proceedings of the 28th International Conference on Machine Learning, {ICML} 2011, Bellevue, Washington, USA, June 28 - July 2, 2011, 2011, pp. 1097–1104.

[131] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011, 2011, pp. 465–472.

[132] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-Ensemble Trust-Region Policy Optimization," in 6th International Conference on Learning Representations, {ICLR} 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[133] "Part 2: Kinds of RL Algorithms", 2020. [Online]. Available: https://web.archive.org/save/https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html. [Accessed: 01- Dec- 2020].

[134] Y. Flet-Berliac, "The Promise of Hierarchical Reinforcement Learning." 2020.

[135] C. Balkenius, A. Förster, B. Johansson, and V. Thorsteinsdottir, "Anticipation in Attention," in The Challenge of Anticipation: A Unifying Framework for the Analysis and Design of Artificial Cognitive Systems, G. Pezzulo, M. V Butz, C. Castelfranchi, and R. Falcone, Eds. Berlin, Heidelberg: Springer, 2008, pp. 65–83.

[136] "AlphaGo: The story so far", Deepmind, 2020. [Online]. Available: https://web.archive.org/web/20201011134603/https://deepmind.com/research/case-studies/alphago-the-story-so-far. [Accessed: 01- Dec- 2020].

[137] Q. Lan, Y. Pan, A. Fyshe, and M. White, "Maxmin Q-learning: Controlling the Estimation Bias of Q-learning," in 8th International Conference on Learning Representations, {ICLR} 2020, Addis Ababa, Ethiopia, April 26-30, 2020, 2020.

[138] B. R. Kiran et al., "Deep Reinforcement Learning for Autonomous Driving: A Survey." 2020.

[139] A. Azhikodan, A. Bhat, and M. Jadhav, "Stock Trading Bot Using Deep Reinforcement Learning," in Lecture Notes in Networks and Systems, 2019, pp. 41–49.

[140] J. Kober, J. Bagnell, and J. Peters, “Reinforcement Learning in Robotics: A Survey,” Int. J. Rob. Res., vol. 32, pp. 1238–1274, 2013.

[141] T. G. Fischer, “Reinforcement learning in financial markets - a survey,” Friedrich-Alexander-Universität Erlangen-Nürnberg, Institute for Economics, Nürnberg, 2018.

[142] X. Liang, X. Du, G. Wang, and Z. Han, “A Deep Reinforcement Learning Network for Traffic Light Cycle Control,” IEEE Trans. Veh. Technol., vol. 68, no. 2, pp. 1243–1253, 2019.

[143] C. Yu, J. Liu, and S. Nemati, “Reinforcement Learning in Healthcare: A Survey.” 2020.

[144] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, “A Survey of Deep Reinforcement Learning in Video Games.” 2019.

[145] H. Cai et al., “Real-Time Bidding by Reinforcement Learning in Display Advertising,” in Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, 2017, pp. 661–670.

[146] E. Thorndike, Animal Intelligence. The MacMillan company, 1911.

[147] M. Henne, A. Schwaiger, K. Roscher, and G. Weiss, “Benchmarking Uncertainty Estimation Methods for Deep Learning With Safety-Related Metrics,” in SafeAI@AAAI, 2020.

[148] T. Salimans, J. Ho, X. Chen, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,” CoRR, vol. abs/1703.0, 2017.

[149] S. Kakade, “A Natural Policy Gradient,” in Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, 2001, pp. 1531–1538.

[150] C. Guestrin, M. G. Lagoudakis, and R. Parr, “Coordinated Reinforcement Learning,” in Proceedings of the Nineteenth International Conference on Machine Learning, 2002, pp. 227–234.

[151] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual Multi-Agent Policy Gradients,” in Proceedings of the Thirty-Second {AAAI} Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th {AAAI} Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New, 2018, pp. 2974–2982.

[152] F. Such, V. Madhavan, E. Conti, J. Lehman, K. Stanley, and J. Clune, “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning,” 2017.

[153] S. Whiteson and P. Stone, “Evolutionary Function Approximation for Reinforcement Learning,” J. Mach. Learn. Res., vol. 7, pp. 877–917, 2006.

[154] J. Baxter and P. L. Bartlett, “Infinite-Horizon Policy-Gradient Estimation,” J. Artif. Intell. Res., vol. 15, pp. 319–350, 2001.

[155] D. Ormoneit and Ś. Sen, “Kernel-Based Reinforcement Learning,” Mach. Learn., vol. 49, no. 2, pp. 161–178, 2002.

[156] J. E. Moody and M. Saffell, “Learning to trade via direct reinforcement,” {IEEE} Trans. Neural Networks, vol. 12, no. 4, pp. 875–889, 2001.

[157] M. G. Lagoudakis and R. Parr, “Least-Squares Policy Iteration,” J. Mach. Learn. Res., vol. 4, no. null, pp. 1107–1149, 2003.

[158] D. Silver et al., “Mastering the game of Go with deep neural networks and tree search,” Nat., vol. 529, no. 7587, pp. 484–489, 2016.

[159] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum Entropy Inverse Reinforcement Learning,” in Proceedings of the Twenty-Third {AAAI} Conference on Artificial Intelligence, {AAAI} 2008, Chicago, Illinois, USA, July 13-17, 2008, 2008, pp. 1433–1438.

[160] J. Li, W. Monroe, T. Shi, S. Jean, A. Ritter, and D. Jurafsky, “Adversarial Learning for Neural Dialogue Generation,” in Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, {EMNLP} 2017, Copenhagen, Denmark, September 9-11, 2017, 2017, pp. 2157–2169.

[161] M. Fortunato et al., “Noisy Networks For Exploration,” in 6th International Conference on Learning Representations, {ICLR} 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[162] W. D. Smart and L. P. Kaelbling, “Practical Reinforcement Learning in Continuous Spaces,” in Proceedings of the Seventeenth International Conference on Machine Learning {(ICML} 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000, 2000, pp. 903–910.

[163] S. J. Bradtke and M. O. Duff, “Reinforcement Learning Methods for Continuous-Time Markov Decision Problems,” in Advances in Neural Information Processing Systems 7, {[NIPS} Conference, Denver, Colorado, USA, 1994], 1994, pp. 393–400.

[164] R. Tedrake, T. W. Zhang, and H. S. Seung, “Stochastic policy gradient reinforcement learning on a simple 3D biped,” in 2004 {IEEE/RSJ} International Conference on Intelligent Robots and Systems, Sendai, Japan, September 28 - October 2, 2004, 2004, pp. 2849–2854.

[165] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-Based Batch Mode Reinforcement Learning,” J. Mach. Learn. Res., vol. 6, pp. 503–556, 2005.

[166] S. Kalyanakrishnan and P. Stone, “Characterizing reinforcement learning methods through parameterized learning problems,” Mach. Learn., vol. 84, no. 1, pp. 205–247, 2011.

[167] G.-J. Han, X.-F. Zhang, H. Wang, and C.-G. Mao, “Curiosity-Driven Variational Autoencoder for Deep Q Network,” in Advances in Knowledge Discovery and Data Mining, 2020, pp. 764–775.

[168] S.-A. Chen, V. Tangkaratt, H.-T. Lin, and M. Sugiyama, “Active deep Q-learning with demonstration,” Mach. Learn., vol. 109, no. 9, pp. 1699–1725, 2020.

[169] S. Chen, X.-F. Zhang, J.-J. Wu, and D. Liu, “Averaged-A3C for Asynchronous Deep Reinforcement Learning,” in Neural Information Processing, 2018, pp. 277–288.

[170] C.-Y. Kang and M.-S. Chen, “Balancing Exploration and Exploitation in Self-imitation Learning,” in Advances in Knowledge Discovery and Data Mining, 2020, pp. 274–285.

[171] M. B. Ring, “CHILD: A First Step Towards Continual Learning,” Mach. Learn., vol. 28, no. 1, pp. 77–104, 1997.

[172] C. Yin, R. Zhang, J. Qi, Y. Sun, and T. Tan, “Context-Uncertainty-Aware Chatbot Action Selection via Parameterized Auxiliary Reinforcement Learning,” in Advances in Knowledge Discovery and Data Mining, 2018, pp. 500–512.

[173] R. Maclin and J. W. Shavlik, “Creating advice-taking reinforcement learners,” Mach. Learn., vol. 22, no. 1, pp. 251–281, 1996.

[174] W. Wu, F. Zhu, Y. Fu, and Q. Liu, “Deep Deterministic Policy Gradient with Clustered Prioritized Sampling,” in Neural Information Processing, 2018, pp. 645–654.

[175] D. E. Moriarty and R. Miikkulainen, “Efficient reinforcement learning through symbiotic evolution,” Mach. Learn., vol. 22, no. 1, pp. 11–32, 1996.

[176] M. Wiering and J. Schmidhuber, “Fast Online Q(λ),” Mach. Learn., vol. 33, no. 1, pp. 105–115, 1998.

[177] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, “Improvement of Systems Management Policies Using Hybrid Reinforcement Learning,” in Machine Learning: ECML 2006, 2006, pp. 783–791.

[178] A. B. Labao, C. R. Raquel, and P. C. Naval, “Induced Exploration on Policy Gradients by Increasing Actor Entropy Using Advantage Target Regions,” in Neural Information Processing, 2018, pp. 655–667.

[179] C. Wang, T. Zhou, C. Chen, T. Hu, and G. Chen, “Off-Policy Recommendation System Without Exploration,” in Advances in Knowledge Discovery and Data Mining, 2020, pp. 16–27.

[180] Y. Zhu and D. Zhao, “Online Model-Free RLSPI Algorithm for Nonlinear Discrete-Time Non-affine Systems,” in Neural Information Processing, 2013, pp. 242–249.

[181] D. Wierstra and J. Schmidhuber, “Policy Gradient Critics,” in Machine Learning: ECML 2007, 2007, pp. 466–477.

[182] J. Fürnkranz, E. Hüllermeier, W. Cheng, and S.-H. Park, “Preference-based reinforcement learning: a formal framework and a policy iteration algorithm,” Mach. Learn., vol. 89, no. 1, pp. 123–156, 2012.

[183] H. M. S. Hossain and N. Roy, “Active Deep Learning for Activity Recognition with Context Aware Annotator Selection,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery &; Data Mining, 2019, pp. 1862–1870.

[184] C. George Boeree, "The three levels of the mind", 2021. [Online]. Available: http://web.archive.org/web/20210108224128/http://webspace.ship.edu/cgboer/levelsofmind.html. [Accessed: 14-Apr- 2021].

[185] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," in IEEE Signal Processing Magazine, vol. 34, no. 4, pp. 18-42, July 2017, doi: 10.1109/MSP.2017.2693418.

[186] "Voronoi diagram", 2021. [Online]. Available: https://en.wikipedia.org/wiki/Voronoi_diagram. [Accessed: 17- May- 2021].

[187] "Youtube Web Summit", 2021. [Online]. Available: https://www.youtube.com/watch?v=2ZzGMzitNgo, Minute: 17, [Accessed: 18- May- 2021].

Appendices

Appendix A - Glossary

A

Ablative Analysis - This type of analysis allows one to know the contribution of each part of a RL model [17, p.8][92, p.1], by disabling at a time, each optional step of the pipeline of that model.

Action - something an agent can do in an environment. An action may consist of a modification of the environment or of the agent state.

Action-Value function - same as Q-function [91, p.233]. Starting from a specific state, calculates an expected return after executing an action and then following a specific policy [16, p.2].

Actor-Critic methods - a type of RL algorithm that has a component, called a critic, that evaluates the actions selected by the main system called the actor [65, p.331]

Advantage - the extra reward an agent has when choosing one action over another or several other possible actions.

Agent - it is an entity that performs actions in an environment where it has existence, an environment it can sense, modify and control [58, p.3] totally or partially.

Artificial General Intelligence (AGI) - it stands for the artificial intelligence of a machine that is capable of understanding or to learn tasks that involve the ability to improvise solutions [37, p.4] or to reason and plan actions it was never designed to do [38, p.16]. This type of AI is still theoretical as per the current state of the art, and can be considered a second generation of AI performing at a human level [38, p.16].

Artificial Narrow Intelligence (ANI) - also called weak or below human level AI, can perform only specific problems and is unable to, by itself, solve other types of problems. Nevertheless it performs better or at the same level as humans in the specific area it was designed to operate [38, p.16]. Usually it is referred to as just AI.

Artificial Super Intelligence (ASI) - It is considered a theoretical third generation of AI, with a self-aware AI, having consciousness and able to perform in any area of human knowledge such as social skills, scientific creativity, etc, and at a level above humans [38, p.16].

Asymptotic - when a curve approaches a line, but never touches that line.

AutoML - a set of techniques of ML to automatically design and test a NN model, in order to decide which model is best suited for a specific type of problem.

B

Babbling - consists of a random exploration of possible actions an agent can perform. This way the agent gets to know its physiognomy or its processual capabilities. [83, p.1]

Backpropagation algorithm - an algorithm that goes back and forward in a neural network, computing activation of each neuron while moving forward and computing a partial derivative for each weight when moving backward [65, p.225]. It is a gradient descent technique and therefore an online technique [48, pp.319-320].

Batch Normalization (a.k.a. Batch Norm) - it is a technique, that consists of a normalization step that corrects the means and variances of layer inputs (data whitening) and the add scale and shift parameters [35, p.5], to improve operation of Neural Networks [35, p.2] by:

- Avoiding gradient explosion [35, p.5];
- Reducing internal covariate shift [35, p.2];
- Having a beneficial effect on the gradient flow [35, p2];
- In some cases eliminate the need to Dropout [35, p.1];
- Avoid gradients getting stopped at bad local minima [35, p.5];
- Making training less affected by parameter scale [35, p.5].

As a result it accelerates the training of Neural Networks [35, p2]. With batch normalization, it is possible to learn effectively across several different tasks with different types of units [16, p.4].

Batch Reinforcement Learning - when a set of transitions is stored to be processed together [51, p.1] and in this way learn a policy from offline data, without getting new information from the environment [49, p.1]. Other RL settings alternate between interaction with the environment and policy improvement [49, p.1].

Bias (of an estimator) - It consists of the difference between an expectation given by an estimator and the real value of the data [27, p.116]. An unbiased estimator is the one where this difference is zero, and an asymptotically unbiased estimator the one where that difference tends to zero when the number of samples grows towards infinite [27, p.116]. Bias is one of the sources of error of an estimator, alongside with variance [27, p.127]

Biologically inspired models - models that are inspired by biology, but not mimic biology [1, p.4].

Biologically plausible models - models that mimic biology [1, p.4].

Bayes' theorem - expresses the probability of an event A occurring if another event B happens. $P(A \vee B) = \frac{P(B \vee A) P(A)}{P/B¿}$¿, $P(B) \neq 0$

Bootstrapping - consists of a technique that allows one to have less interaction with the environment or to explore more [40]. Estimations, instead of actual rewards and returns, are used to update model targets [40].

Bidirectional Recurrent Neural Networks (BRNN) - a type of NN topology where hidden layer neurons are split to pass values in two directions. One forward, called the positive time direction, and another one backward, called the negative time direction.With this arrangement it is possible to calculate an output with data from current and previous time frames.

Baseline (function)- a learned function known as a baseline that can be used to reduce the variance of an estimator and still keep it unbiased [12, p.3].

Baseline Algorithm - Often referred to as the algorithm or algorithms to be compared with a new algorithm and in this way assess the performance of the latter, enabling the researchers to prove the state-of-the-art was improved.

Black-box optimization - an alternative to RL in optimal control problems [148] .

Behavior Policy - it is a policy that is intended to explore the environment or the agent dynamics, and as [65, p.103] it is used to generate behavior (see Target Policy).

C

Clipped Gradient - it is a technique for limiting a soft policy gradient to a range of values, so that there is less chance of the gradient explosion problem occuring [46, p.3429].

Credit Assignment Problem - when you arrive at a decision that involves many component structures, how can you assign credit for success or failure to each part that contributed to the decision? [65, p.385].

Continuous-action - it is an action that can be represented by a value that varies within a certain range of possible values. Controlling an agent with four possible directions (up/down/right/left) Is an example of a discrete-action. If we add “how much” to the action then we have a continuous-action. Example: move an agent to the left by 10.2 centimeters.

Continuous-state - it is a state that is not discrete. For example, a robot agent can have its right leg forward in a 27 degrees angle relative to the torso. In a discrete-state we would only have the information that the right leg is in a forward position without any indication of the angle relative to the torso.

Convolutional Neural Networks (CNNs) - they are a specialized type of NN used to process data with a known grid-like topology, being examples, image data, that is associated with a 2D grid of pixels, or a time series seen as a 1D grid of data [27, p.326]. The name convolutional comes from the fact that it implements, at least in

one of its layers, a mathematical type of linear operation named convolution, instead of using matrix multiplication [27, p.326]. CNNs are commonly used for feature extraction [88, p.10].

Critic (in Actor-Critic) - a critic assesses policies defined by the actor, by estimating their value [81, p.64]. A critic can either be a NN separated from the actor NN or can share the actor NN.

Convergence - An algorithm converges when as the iterations increase its output gets closer to a minimum error.

Cross-validation (in Trading off Bias and Variance) - It is a technique used to choose between two estimators [27, p.121]. Bias and variance are two sources of error of an estimator, and if we have to choose between two estimators, one with large variance and another with large bias, cross-validation is one of the options to use (another option is using the mean squared error (MSE)) [27, p.121]. Cross validation consists of cycling through data and processing it in different arrangements of train / test subsets, being k-Fold one the possible arrangements [27, p.127].

Clustering - it consists of a procedure to divide a group of tuples, objects or concepts into meaningful subsets that can represent groups which differ between themselves by some extent in a given property or properties [27, p.148]. Clustering performed by ML algorithms may not be perfect, since each cluster may not represent well our own vision of how the world can be organized into groups of tuples, objects or concepts [27, p.148]

Curse of dimensionality - it is a term created by Richard Bellman in 1957 [65, p.90], meaning that the bigger the number of dimensions, the harder ML problems are to solve [27, p.152]. The increase of the number of variables, results in an exponential increase of the number of possible distinct configurations of those variables [27, p.152]. This curse becomes a problem because applications commonly have state-representations with hundreds of dimensions [81, p.27]. The curse may be reduced depending on how many variables are irrelevant to the state representation [81, p.27]. This is true both for state spaces and for action spaces [41, p.1]. Training gets harder, because costs in used memory and computational power gets higher, and also generalizations become weaker [87, p.1]. Real world lifelong learning systems are examples that suffer from the curse of dimensionality [101, p.1]. The curse of dimensionality in state spaces can be reduced by using Convolutional NN (CNN), and in action spaces that reduction can also be done by using macro-actions [41, p.1].

D

Degrees of freedom - it is the number of independent variables an agent can modify [81, p.27]. A human arm has 7 degrees of freedom [16, p.1], 3 at the shoulder, 1 at the elbow, 1 at the forearm and 2 at the wrist.

1. Shoulder flexion or extension
2. Shoulder abduction or adduction
3. Shoulder medial rotation or lateral rotation
4. Elbow flexion or extension
5. Forearm pronation or supination
6. Wrist flexion or extension
7. Wrist abduction or adduction

Deep Reinforcement Learning - Consists in the use of Neural Networks with a considerable number of layers to perform Reinforcement Learning. Deep learning creation happens, partially, to overcome the limitations of traditional algorithms that are not able to generalize properly [27, p.152].

Deep Neural Networks - they are Neural Networks that have more than one hidden layer. They are very capable function approximators and can generalize well, producing the same output for similar state inputs [63, p.2].

Deep Q-learning - a group of algorithms that combines q-learning with NN, by using the latter to substitute the q-table.

Deterministic Policy - for every state there is a very well defined action to take. It differs from Stochastic Policies where there is no well defined action to take when being at a specific state, but rather there is a set of possible actions, each with a certain probability to be chosen.

Discrete-action - differs from continuous-action. Discrete-action is an action that is categorical, i.e. moving an agent up, down, left or right. In a continuous-action there is the “how much” property, i.e. the agent moves 10 centimeters to the right. In the discrete-action there is no “how much”.

Distributional Reinforcement Learning - consists of RL that instead of calculating a value function as an expected average value, condensing the stochasticity of the environment, by using Bellman equation or derivatives, it calculates a distribution of the returns so that the value of an action can better relate to the reality of the environment, and avoid values that, by being averages, may not exist in reality [20, p.1][21, p.1].

Discounted Return/Reward - future rewards are seen as less valuable than immediate rewards, so when calculating the sum of possible rewards in a given policy (return), rewards are multiplied by a discount percentage that is proportional to how far in the future the rewards will happen if following the selected policy [65, p.55].

Dilated Long Short Term Memory (dLSTM) - a type of NN that while being able to learn from every input experience, is also able to keep memories for a long time [92, p.4].

Dynamic programming - a type of algorithmic approach, developed by Bellman in the 1950’s that can be used to solve many different types of problems, including RL problems, despite that in too complex scenarios it can suffer from the curse of

dimensionality [65, p.14]. Dynamic programming algorithms can often be represented in tabular presentations where a problem is divided into sub-problems to help achieve the answer to the main problem with a series of recursive and repetitive operations.

Dynamics model - a dynamics model expresses the rules of how elements in an environment or an agent can change states in a model-based algorithm setting, and can be deterministic or probabilistic, being the latter a better option to deal with uncertainty and avoid model-based algorithm bias [131, p.7].

Double Q-learning - it is an algorithm based on Q-learning that uses two estimators to control the problem of overestimation of action values at Q-learning in stochastic environments [119]

Dropout - it is a regularization strategy that consists of dropping some neurons randomly [27]. For large NN, dropout provides an approximation to evaluation and training of models, so that computational costs can be reduced [27].

Dueling Networks - consists of one NN with two streams. One of the streams estimates the value of a state, and the other the advantages of each action. When both calculations are combined we get a state-action-value tuple, and a better generalization. Dueling networks architecture can improve the performance of an algorithm without changing the algorithm logic.

E

Elementary Action - Represents a type of action referred to, also as, a primitive action, that is associated with simple actions that may involve contracting a muscle, moving a joint, typing a character, setting a parameter, etc, [90, p.181] and that in essence represents a lower-level action that can be sequenced to build up a macro-action.

Episode - it is the sequence of algorithm cycles registered between the initial and the terminal state. A sequence of control cycles defines an episode, starting with an initial state and ending by fulfilling some termination condition or getting to a maximum number of cycles [48, p.322].

Environment - everything that can be sensed and acted upon. Environments can either be fully or not fully observable. The latter happens when only a portion is visible to the agent.

Energy Based Model - an approach to RL that aims at reducing state dimensionality by concentrating efforts in certain states that have low energy function values [11, 111].

Epsilon-Greedy Strategy - it is an RL strategy to balance the agent's execution between exploration and exploitation actions. When an agent has not yet explored much of the environment, epsilon-greedy strategy gives priority to exploration, but as exploration actions accumulate, the agent will exploit more of the knowledge that has already learned, and will explore less.

Experience Replay - a technique where an RL agent will use data from previous policies, kept in memory, to choose the policy for the next transition (results in an higher computational cost) [39, 12]

Exploration-Exploitation Dilemma - an agent needs to explore an environment in order to learn about it and then be able to make decisions to accomplish the goals it's meant to achieve guided by the rewards it gets. The dilemma results from the problem the agent faces, that is, should it explore more to have more knowledge and become more able to decide better which policies it needs to execute, or should it exploit more the knowledge it already has and go on to get the rewards it knows it can get.

Episodic Environment - execution of the agent is divided into episodes, which means that actions in one episod are important in the scope of that episode and have no consequence in other episodes, because each episode has its own context. While actions in an episode don't relate to actions in another episode, the agent can still apply what it learned in one episode to decide what to do in another episode, as an ability to generalize and to transfer learning.

Exploding Gradients - an RL agent learns through optimization often performed with gradients. The gradient calculations are propagated through the neural network, and sometimes this propagation can result in the accumulation of values that at some point of the network start to be outside of the expected values for neurons, resulting in the NN instability.

F

Feature extraction - the identification of features (objects, shapes, boundaries,...) from an image in order to have a classification of what an agent is seeing through a camera view or a screen feed [92, p.4].

Function estimation - same as Function approximation.

Function approximation - normally implemented using NN, allows to calculate an output from an input, like what could be obtained with a function [65]. Function approximation is used when the transformation between input and output is too complex to be modeled by a common function, due to too many outputs and conditional rules.[65]

Feedforward neural network - it is a topology type of NN that has no cycles between nodes and any connection forward only moves computation closer to the output layer

Flat Reinforcement Learning - refers to normal RL algorithms as opposed to Hierarchical RL algorithms, because the previous methods see space as a big flat search space [91, p.228]. Flat RL methods have limitations because of the basic

actions orientation of models in this set of approaches.[97, p.1].

G

Generalization Error - it is the estimated error of a machine learning model when being tested with new data, that is different from the data used to train it [27, p.108].

Goal Space - the set of possible goals for an agent [104, p.2].

Gradient ascent - it is the opposite of gradient descent. With gradient ascent calculations it is possible to solve a maximization problem. Gradient descent can be used to solve minimization problems.

Generalization - it is the ability of a machine learning algorithm to perform correctly when processing new inputs that differ from the inputs used in training, and it is a central challenge in machine learning [27, p.108].

Gradient descent - used to solve minimization problems. It is used in RL to minimize the cost function so that the agent can choose the optimal policy at each step. Types of gradient descent:

- Standard Gradient Descent;
- Stochastic Gradient Descent (SGD)
- Momentum
- Nesterov Accelerated Gradient (NAG)
- Adaptive Gradient (AdaGrad)
- AdaDelta
- Adaptive Moment Estimation (AdaM)

Generative Adversarial Network (**GAN**) - two NN face one another with opposite goals. One NN tries to generate an output that is as realistic as possible, and the other NN tries to identify which outputs are not real. With this game between the two NN, the system is able to synthesize something that resembles a realistic perception of the world.

H

HER - Hindsight Experience Replay - a technique that avoids wasting computational efforts when an agent fails to achieve a goal. When the agent chooses a policy and the result is a failure, the goal is changed to match the policy [22]. This way any effort results in better learning [22].

Hyperparameters - these are settings that allow us to control an algorithm's behaviour. Their values are not changed by the algorithm itself, but there may exist an architecture where one ML algorithm calculates hyperparameters for another ML

algorithm. [27, p.118]. Tuning hyperparameters is one of the challenges in some RL algorithms [18, p.7]. Hyperparameters tuning in real world settings is usually infeasible [39, p.6].

Hierarchical Reinforcement Learning (HRL) - ir is an approach to RL that combines a planner level with a “Skills” level that we can also call macro-actions or options. Skills are a combination of micro-actions that can be used by the planner. In HRL the planner can also choose to perform micro-actions combined with macro-actions or other sub-planners.

I

Initial state - the state from which a RL agent starts its execution.

Inverse Reinforcement Learning - instead of producing behaviour of an Agent, the goal is to evaluate observed behaviour of an Agent, and then figure out what is the Reward function that can explain that behaviour [64, p.3] It can be applicable in getting skilled behaviour, in an apprenticeship type of learning by observing a natural system [64, p.1]. A lot of IRL algorithms have an extremely high computational cost [31, p.1].

Internal covariate shift - it is the change in the distribution of network activations caused by the modification in network parameters during training [35, p.2]. This change in distribution makes it harder for the model to converge. See Batch Normalization.

L

Law of Effect - it is a behaviour law that states that an animal will bond strongly to situations where it can get satisfaction and bond weakly with situations that bring it discomfort [146, p.244].

Long Short-Term Memory (LSTM) Network - a type of Recurrent Neural Network where each neuron has a memory, useful for processing time series.

Loss Function - a function to calculate the deviation of the real results from the prediction of results made by the agent.

Learning Rate - how fast a NN abandons old beliefs for new ones.

M

Manifolds - a type of space that allows an approach to manage high dimensionality by separating that high dimensionality space into smaller dimensionality maps [27, p.521], like an earth globe (high dimensionality continuous space) can be divided into several maps (smaller dimensionality continuous spaces) of portions of the surface of that globe. Some spaces may not allow this in some parts of their topology, so they are not considered manifolds.

Maximum Entropy Reinforcement Learning - it is a technique of RL that aims to optimize a policy in order to maximize the expected entropy and the expected return of that policy [18, p.2].

Maximum Likelihood Estimation (MLE) - a process to maximize the parameter Θ of a likelihood function, in a way that the function can estimate with more precision. This function is used to perform inference statistics to, given a random independent, identically distributed sample of data, infer characteristics of the population distribution [27].

Meta-RL (MRL) - a type of approach to RL that focuses on creating agents that learn how to learn and that learn from multiple tasks the ability to adapt to never seen tasks in an efficient way [53, 125, 128]. MRL tries to improve that sample efficiency by creating a shared structure for each family of tasks [125, p.1].

Model Predictive Control (MPC) - a model based non-RL technique to perform optimal control of a system, that is based on an expert modelling and [13, 24].

Model Ensemble - A model that results from a mix of model-free and model-based methods in an attempt to get the best from each type of approach and reduce problems from the type of approach [115, p.9].

Markov Decision Process (MDP) - an MDP consists of a network of nodes that represent states and connections that represent transitions (between states) that can occur with a given probability [65, p.52].

Model Free RL - a type of RL that does not use predictions of rewards from the environment. Instead it makes decisions based on real data gathered from trial and error [95]..

N

Nested policies - consists of policies that nest primitive actions and in this way reduce the size of the sequence of actions to be learned [80, p.5] like what happens with HRL when micro-actions are aggregated into skills.

Noisy Networks - Neural networks with noise (stochasticity) added to its weights, gradients or activation functions to avoid overfitting and generalize better.

Neural Network (NN) topologies - NN can be of several different types depending on how many input cells, output cells and hidden layers it has, and how each

different layers are connected. Known types include “Recurrent Neural Network” (RNN), “Deep Feed Forward” (DFF), “Deep Convolutional Network” (DCN), Long Short-Term Memory(LSTM) , etc.

Neural Network - also known as Artificial Neural Network (ANN), it is a digital representation of a biological neuron network, and is commonly used as a function approximator when the search space of a problem is too large to use a function [105, p.10009].

O

On-policy - Q-Learning is an example of an on-policy method [65, p.100]. A method that uses on-policy will try to improve the policy currently being used [65, p.100], which can make these methods impractical for real world applications because of its inefficiency in high-dimensional continuous state or action spaces [94, pp.1-2].

Off-policy - off-policy RL algorithms reuse the agent's previous experiences [18, p.1]. Opposite to methods using on-policy, in off-policy the policy being used currently is not the one being improved [65, p.100]. Off-policy methods are more sample efficient than on-policy methods and are also more suitable to real world applications [94, pp. 1-2].

Overestimation bias - it is the effect on the value the agent calculates for the current state and action with the observed reward [137, p.1]. This overestimation occurs due to the maximization process the agent performs when it tries to find the action with the maximum value in the next state [137, p.1]. One example of an algorithm with overestimation bias is Q-Learning [137, p.1].

Overfitting - if there is a large gap between the training and test errors, then there is an overfitting [27, p.110]

Optimal Control - refers to the optimal control of the dynamical behaviour of a system [65, p.14]. It is a term that started to be used in the end of the 1950s, and was viewed as a problem which was addressed by a class of methods that became known, as stated by Bellman, as dynamic programming [65, p.14].

Option - it consists of a sub-policy that has a termination condition [100, p.2].

Optimism in face of uncertainty - an heuristic for general decision making and that in RL will guide an agent to choose the policy that will make more exploration, hoping to find better rewards than the ones already known.

P

Policy Distillation - a process of selecting good policies to create a new NN that can perform more efficiently or to merge several policies into one [109]

Policy - a strategy for an RL agent to execute actions that will make it closer to its goals.

Perceptron - it is a simple feedforward neural network proposed by Frank Rosenblatt in 1957, that acts as a pattern classification system [84, pp.1383-1385].

Prioritized Experience Replay - an approach to use recorded transitions of an agent, that are more important than other transitions. This approach differs from a simple experience replay that uses recorded transitions in the sequence they were recorded.

Policy Gradient methods - continuous spaces RL methods that use gradients to solve optimization problems for choosing the best policies.

Policy Iteration - an approach of improving decision making by iterating through possible policies, evaluating and improving each one. This approach is less computationally expensive than the value iteration approach when we are dealing with continuous spaces, because with these types of spaces actions and/or states are too diverse.

Pseudo-reward - when some of the agent's properties values are treated as rewards from the environment, so that their modification can enter the decision process [36, p.1]. Intrinsic Motivation can also be viewed as a pseudo-reward approach, since the agent will consider some rewards not coming from the environment but from the agent itself [56, p.3].

Policy Optimization Algorithms - most of them can be classified into 3 main categories: Policy Iteration Methods, Policy Gradient Methods and Derivative-free optimization Methods [15, p.1].

Policy-based methods - methods where actions are selected with policies mapping states directly to actions, instead of calculating a value function that can provide the expected sum of rewards to help choose the action to execute.

Partially Observable MDP (POMDP) - a Markov Decision Process where the agent doesn't know exactly what state it is in, but instead has a distribution probability for the possible states after each transition it chooses and based on what it can observe or, guess what it is observing based on another distribution probability.

Q

Q-table - a table used to store reward calculations for every state-action pair. Columns are actions and rows are states [107].

Q-function - it gives the cumulative expected reward of performing some action while being in a specific state and then following another sequence of policies thereafter [91, p.233].

R

Reinforcement Learning - it consists of an not supervised mechanism to enable an agent to learn how to act in a given environment in order to get the maximum possible reward according to some reward calculation criteria [60, p.3216].

Receptive Field (CNN receptive field) - a window that is moved around a matrix input, like an image, and is used to apply convolution operations to parts of the input instead to the whole input at once. This area in the input space that a particular kernel of a CNN is looking at is named Receptive Field.

Reward - a number that measures the agent's performance in achieving its goals in the environment.

Replay Memory - a record of past experiences of an agent [16, p.4]. These experiences can be represented as tuples of previous states (state at moment t), following states (state at moment t+1), actions, transitions, rewards, policies in whatever combination is considered by each algorithm [16, p.4].

Replay Buffer - same as Replay Memory, except that it is a limited size memory, that when it is full, older samples are discarded [16, p.4].

Reward Hacking - when an algorithm exploits the flaws in the design of the environment rewards by finding a loophole and producing an undesired outcome in the control of the agent.

Rollout - usage of a RL model for a certain number of steps [29, p.1][54, p.1]. To roll out a model for a certain number of steps..

Root Mean Squared Propagation (RMSProp) - a variation of Stochastic Gradient Descent (SGD) optimization method.

Restricted Boltzmann Machine (RBM) - a type of NN able to learn a probability distribution from its inputs. In Boltzmann Machines all neurons connect to each other, but this creates a computationally excessive cost, because for each new neuron, connections will grow exponentially. RBM's simply define that hidden nodes do not connect between themselves and visible nodes (input and output) also do not connect between themselves.

Recurrent Neural Network (RNN) - a type of NN able to perform temporal learning. RNNs use an internal state memory to process sequences of variable length.

Recursive Neural Network (RvNN) - a type of NN with an hierarchical structure representation. RvNNs differ from RNNs because the latter has more of a chain structure.

S

Self-Supervised learning (SSL) - an approach to unsupervised ML, where the algorithm analyzes data to create its own labels.

Semi-Markov Decision Process (SMDP) - a variation of an MDP, with a transition dynamics that includes the notion of time. In MDPs, time in transitions is always discrete.

SARSA - an RL algorithm that proposes a method to address continuous state spaces with high-dimensionality by using NN as function approximators and backpropagation [124, p.1]. It is an on-policy algorithm that fits the action-value function to the current policy, and then refines it in a greedy manner regarding those action-values [95, p.1].

State space - the set of possible states the agent can observe from the environment or from itself (proprioception).

State - the observable part of the environment for an agent. Can also include the observable part of the agent itself (proprioception).

Sparse Rewards - opposed to dense rewards, it means that the agent may have to perform several actions until it can get a reward [56, p.4].

Sigmoid Function - a function $S(x)=\frac{1}{1+e^{-x}}$ that can be used to trigger neurons to pass values to the next neurons to which it is connected. With a Sigmoid function it is possible to establish a threshold after which the neuron will activate the next neurons.

Stochastic Policy - used when the environment is not fully observable like in an POMDP, and there is no certainty in which state the system is in. So a policy cannot be deterministic, but either probabilistic, because the agent can only choose a policy that has an x probability of being the optimal one.

Stochastic Gradient Descent - SGD is an approximation, with less convergence, to the Gradient Descent optimization method. SGD and its derived or extended methods are computationally less costly, because only a random sample of data is considered for the optimization process, instead of all the data [27, pp.149-150].

Subgoal discovery - a process of HRL to divide a goal into subgoals, in order to break a problem into smaller problems.

Super-convergence - a method to increase the speed of learning by feeding data to the model in a bottom-up logical order to increase the chances of a good generalization in the NN [43].

State-value Function - determines the value of a policy by transitioning from a particular state to a path of future states. Unlike the state-action value function, it doesn't take an action as a parameter, only the state.

State-action value Function - also named a Q-function, determines the value of a policy by following a specific action being at a particular state. It takes both state and action as parameters.

T

Thompson sampling - it is a method to balance exploration with exploitation[39, p.5], like epsilon greedy also is .

Transfer Learning - it is a machine learning method to reuse knowledge obtained in executing one task, to execute another relatable task.

Terminal state - it is a state that ends an episode [52, p.2].

Training Error - the computed error measure on a training data set being used to train an ML model [27, p.108].

Target Network - also known as a target model, it is a copy or snapshot of the main NN (model) to stabilize training. The copy is done every n number of steps. This is a way of getting some stability in the prediction operation, that instead of being done against the main model which changes at each step is done against the target model.

Target Model - see Target Network.

Target Policy - it is a policy being learned and that is intended to create a meaningful course of action of the agent towards an expected reward [65, p.103]. Basically it is a policy that exploits existing knowledge, as opposed to a policy that further explores the environment (see behavior policy).

Temporal Difference (**TD**) **learning** - a class of RL algorithms that are model free and that learn from complete or incomplete episodes of experience to make predictions of reward coming from future steps [65, p.119]. TD methods make estimates based on other estimates in a bootstrap approach, and constantly recalculate estimates after each transition that brings relevant new information associated with the new state [65, p.119].

Temporal Difference Error - the difference between the better estimate of TD and the value of the current state (example of TD error for TD(0): $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$) [65, p.122].

TD(0) - a temporal difference algorithm that only looks ahead the transition between the current and the next state [65, p.121].

Transition - the change process from one state to another, by executing a chosen action.

O

Observational Dropout - Experiments with “World Models” by changing methods to see how much “World Models” algorithms are able to work without looking ahead [59].

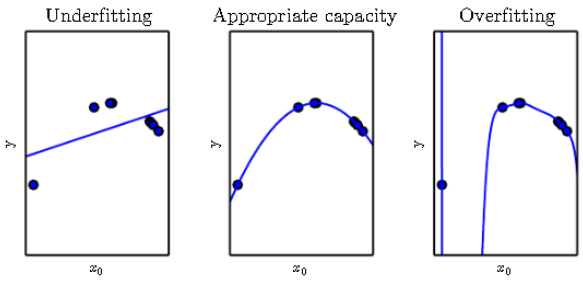
R

Regret - it is the opportunity loss for one step.

U

Uncertainty Estimation Methods - methods to calculate the uncertainty in the output of a NN model in order to evaluate its suitability for safety-critical contexts[147].

Underfitting - when the model is unable to get a significant low error value on the training set then an underfitting situation occurs [27, p.109], meaning that the model hasn't enough complexity to be able to reliably evaluate input data to produce a usable output.



Underfitting vs Good fitting vs Overfitting [27, p.111]

V

Value Iteration - an algorithm that uses value iteration (as opposed to policy iteration) will use a value function it can optimize and in this way find a policy that is already optimal. In policy iteration the algorithms just cycle through the available policies evaluating and improving each one. Value iteration has two basic steps of 1) finding an optimal value function and 2) extract a policy. Value iteration is not as fast as policy iteration when the search spaces are big.

Value Function - a defined function that can be optimized in order to help an agent choose the best action to perform [81, p.12]

Vanishing Gradients - the opposite effect of exploding gradients. In the vanishing gradients problem the propagation of values in a NN can result in so small values that they start not having any relevance to the NN output.

Variance (of an estimator) - measures the variation of an estimate if we independently use a different sample from the dataset as input [27, p.125]

Vanilla Policy Gradient (VPG) - a term used to name a simple policy gradient RL algorithm, with a one step iteration and that tends to suffer from high variance and noise with unstable learning. A VPG algorithm does not resort to RL more advanced techniques.

Z

Zero-shot learning - the ability to perform a task that has never been experienced before.

Appendix B - Firefox extension for Google Scholar

Manifest.json file

```
{
  "manifest_version": 2,
  "name": "GSEFE",
  "version": "1.0",

  "description": "Extracts search results from Google Scholar",

  "icons": {
        "48": "icons/GSEFE-48.png",
    "96": "icons/GSEFE-96.png"
  },

  "content_scripts": [
        {
        "matches": ["*://*.scholar.google.com/*"],
        "js": ["GSEFE.js"]
        }
  ],

    "applications": {
         "gecko": {
         "id": "a10554@iscte-iul.pt"
         }
    }
}
```

GSEFE.js file

```
//
// Copyright José Salvador b21665@iscte-iul.pt (b=a;n=n-1)
// License: GPLv3
// Google Scholar Extract Firefox Extension (GSEFE)
//
//
// 1. enter Google Scholar (GS)
// 2. open your web console
// 3. turn on persistent records at your web console
// 4. activate the extension
// 5. enter your search string at GS and press enter
// 6. click next on GS until you want
// 7. select "registers" on the web console
// 8. export lines from web console to file or clipboard and save as csv
// 9. open file in excel or another spreadsheet sw -
        string delimiter = ' , field delimiter = ;
// 10. remove last column and lines of search string info
// 11. deactivate the extension



fcMain();


function fcMain(){
    fcDebug("fcMain");
    var oGSResult, oGSResultTitle, snapshotGSResultAuthors,
        oGSResultAuthorsRaw; oGSResultBody, oGSResultCitations,
        oGSResultDigitalLibrary, arrCitations, arrDigitalLibrary;
    var regexCitations;
    var regexDigitalLibrary;
    var strLine;
    for (i=1;i<=20;i++){
         strLine = "";
```

```
                // extract data with xpath
                      oGSResultTitle=fcGetElementByXpathAndObj('/html/body/div/div[10]/div[2]/
div[2]/div[2]/div['+i+']/div[2]/h3/a',document);
                if(!oGSResultTitle){

oGSResultTitle=fcGetElementByXpathAndObj('/html/body/div/div[10]/div[2]/div[2]/div[2]/
div['+i+']/div/h3/a',document);
                            snapshotGSResultAuthors=fcGetElementsByXpathAndObj('/html/body/div/
div[10]/div[2]/div[2]/div[2]/div['+i+']/div/div[1]/a',document);

oGSResultAuthorsRaw=fcGetElementByXpathAndObj('/html/body/div/div[10]/div[2]/div[2]/div[2]/
div['+i+']/div/div[1]',document);

oGSResultBody=fcGetElementByXpathAndObj('/html/body/div/div[10]/div[2]/div[2]/div[2]/
div['+i+']/div/div[2]',document);
                           oGSResultCitations=fcGetElementByXpathAndObj('/html/body/div/div[10]/
div[2]/div[2]/div[2]/div['+i+']/div/div[3]/a[3]',document);
                            oGSResultDigitalLibrary=fcGetElementByXpathAndObj('/html/body/div/
div[10]/div[2]/div[2]/div[2]/div['+i+']/div[1]/div/div/a',document);
                } else {
                            snapshotGSResultAuthors=fcGetElementsByXpathAndObj('/html/body/div/
div[10]/div[2]/div[2]/div[2]/div['+i+']/div[2]/div[1]/a',document);

oGSResultAuthorsRaw=fcGetElementByXpathAndObj('/html/body/div/div[10]/div[2]/div[2]/div[2]/
div['+i+']/div[2]/div[1]',document);

oGSResultBody=fcGetElementByXpathAndObj('/html/body/div/div[10]/div[2]/div[2]/div[2]/
div['+i+']/div[2]/div[2]',document);
                           oGSResultCitations=fcGetElementByXpathAndObj('/html/body/div/div[10]/
div[2]/div[2]/div[2]/div['+i+']/div[2]/div[3]/a[3]',document);
                            oGSResultDigitalLibrary=fcGetElementByXpathAndObj('/html/body/div/
div[10]/div[2]/div[2]/div[2]/div['+i+']/div[1]/div/div/a',document);
                }
                // save data
                fcDebug(oGSResultTitle.textContent);
                strLine+="'"+oGSResultTitle.textContent+"'"+";";
                fcDebug(oGSResultTitle.href);
                strLine+="'"+oGSResultTitle.href+"'"+";";
                if (snapshotGSResultAuthors.snapshotLength==0){ // no authors
                                                                        with links
                        strLine+=";";
                }
                for ( var ii=0 ; ii < snapshotGSResultAuthors.snapshotLength; ii++ )
                {
                        fcDebug(snapshotGSResultAuthors.snapshotItem(ii).textContent);
                        strLine+=snapshotGSResultAuthors.snapshotItem(ii).textContent+" - ";
                        fcDebug(snapshotGSResultAuthors.snapshotItem(ii).href);
                        if(!snapshotGSResultAuthors.snapshotItem(ii).href){
                                if (ii==snapshotGSResultAuthors.snapshotLength-1)
                                        strLine+=";";
                                else
                                        strLine+=",";
                        } else {
                                if (ii==snapshotGSResultAuthors.snapshotLength-1)

strLine+=snapshotGSResultAuthors.snapshotItem(ii).href+";";
                                else

strLine+=snapshotGSResultAuthors.snapshotItem(ii).href+",";
                        }
                }
                fcDebug(oGSResultAuthorsRaw.textContent);
                strLine+="'"+oGSResultAuthorsRaw.textContent+"'"+";";
                fcDebug(oGSResultBody.textContent);
                strLine+="'"+oGSResultBody.textContent+"'"+";";
                fcDebug(oGSResultCitations.textContent);
                strLine+=oGSResultCitations.textContent+";";
                regexCitations = /[0-9]+/g;
                arrCitations = regexCitations.exec(oGSResultCitations.textContent);
                if(arrCitations!=null){
                        fcDebug(arrCitations[0]);
```

```
                        strLine+=arrCitations[0]+";";
                } else {
                        strLine+="0;";
                }
                if(!oGSResultDigitalLibrary){
                        strLine+=";;"
                } else {
                        fcDebug(oGSResultDigitalLibrary.textContent);
                        strLine+=oGSResultDigitalLibrary.textContent+";";
                        regexDigitalLibrary = /\s(.*)/g;
                                                arrDigitalLibrary                               =
regexDigitalLibrary.exec(oGSResultDigitalLibrary.textContent);
                        fcDebug(arrDigitalLibrary[0]);
                        strLine+=arrDigitalLibrary[0]+";";
                }
                strLine = strLine.replace(/(\r\n|\n|\r)/gm, ""); // removing line breaks
                fcWriteToConsoleLog(strLine);
            } /* for */
            fcGoToEndOfPage();
        } /*** fcMain ***/

        function fcGetElementsByXpathAndObj(path,obj){
           var nodesSnapshot = document.evaluate(path, obj, null,
              XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null );
           return nodesSnapshot;
        } /*** fcGetElementsByXpathAndObj ***/

        function fcGetElementByXpathAndObj(path,obj) {
           return document.evaluate(path, obj, null,
              XPathResult.FIRST_ORDERED_NODE_TYPE, null).singleNodeValue;
        } /*** fcGetElementByXpath ***/

        function fcGoToEndOfPage(){
           window.scrollTo(0, document.body.scrollHeight ||
              document.documentElement.scrollHeight);
        } /*** fcGoToEndOfPage ***/

        function fcWriteToConsoleLog(strLine){
           console.log(strLine);
        } /*** fcWriteToConsoleLog ***/

        function fcDebug(strMessage){
           //alert(strMessage);
        } /*** fcDebug ***/
```

Appendix C – RL papers analysis

Year of 1989

FRL - Q-learning - Chris Watkins, does not coin Q-learning term [107, Preface], but proves that one-step Q-learning method can converge to optimal value function and policy [107, p.112]. Addresses estimation of q* with action-value functions, that are now often called “Q-functions”. [65, p.71]. Watkins makes a parallel with animal conditioning/learning and learning algorithms [107, p.3]. Explores learning as a problem of obtaining delayed rewards [107, p.24], in which the agent does not try to maximize immediate rewards [107, p.41]. Defines Q-functions as the expected return from starting at a given state and following a sequence of policies [107, p.46] by searching in a look-ahead tree, recognizing the problem of combinatorial explosion with bigger depths of the tree [107, pp.59-60]. Establishes the principles of a Q-table, by describing a way to describe a policy as being a single action for each state, being stored as a function from states to actions [107, p.102]. This dependence of a Q-table naturally induces a limitation to finite and discrete states. Q-learning stands for an off-policy algorithm [95, p.1] [90, p.206] and uses the greedy policy [16, p.3]. A limitation of using one-step methods consists of obtaining a reward that only directly influences the value of the state action pair that gave place to the reward, while the other pairs are only indirectly influenced, which can make learning slow, because several updates will be needed to propagate a reward to the relevant previous actions and states [12, p.2].

Year of 1992

FRL- REINFORCE - describes a set of model-free algorithms with a gradient-based approach compatible with backpropagation [77, pp.23-24]. These algorithms are described as belonging to an associative RL class [77, p.1]. The agent is considered to be a feedforward network with learning units that can be identified as a neural network with the respective neurons [77, p.6].

Year of 1993

HRL - Feudal Reinforcement Learning - this method explores a way to speed up RL, by creating multiple Q-learning layers at different resolutions, and in this way divide RL problems into layers that know how to define tasks and layers that know how to solve them, in a lords/serfs type of relationship [89, p.1]. Each command at an upper layer is associated with a reward that the lower layer tries to maximize, and authors claim that this method is more efficient than flat Q-learning [89, p.1].

Year of 1994

FRL - Modified Connectionist Q-Learning (MCQ-L) - also known as SARSA, this algorithm proposes a method to address continuous state spaces with high-dimensionality by using NN as function approximators and backpropagation [124, p.1]. It is an on-policy algorithm that fits the action-value function to the current policy, and then refines it in a greedy manner regarding those action-values [95, p.1].

Year of 1997

HRL - Hierarchical Abstract Machines (HAM) - uses a library of plans with description of the decomposition of higher level activities into lower level activities [106, p.1]. HAMs are finite state machines/programs and work in a non-deterministic way [106, pp.1-2]. States that with HAMs, knowledge can be reused across different problems and that it only implies a recombination of component solutions to attack a larger more complex problem [106, p.1]. Machines are associated with skills, like when finding a wall the current machine can call a backoff machine or a follow-wall

machine as a policy [106, p.3]. Authors of HAM also propose HAMQ, a crossing between HAM and Q-Learning. [106, p.5].

Year of 1998

HRL - Options-Framework - Defines the term “options” that includes actions like picking an object, going to have a meal, and traveling to some place, as well as micro / primitive actions such as muscle contractions and joint movements, as a way to provide temporal abstraction [90, p.181]. All types of options are used in planning a task, and can also be changed during the execution of a plan that by getting changed at mid-execution can be corrected to perform better [90, p.181]. This is an approach that is well suited to stochastic changing environments [90, p.207]. Options are evaluated during their application to produce more learning [90, p.181]. Besides the options mechanism, this paper also explores the concept of sub-goals [90.181]. Options are given, defined by the programmer [98, p.2].

Year of 2005

FRL - Neural Fitted Q Iteration (NFQ) - it is an improvement on Fitted Q Iteration algorithms by proposing the use of a Neural Network to train a Q-value function and in this way take advantage of NN ability to approximate nonlinear functions [48, p.317]. In [48] learning failures and learning speed problems of NN are discussed and proposes a mechanism of storing previous experiences (a sort of Experience Replay) to deal with the aforementioned problems. It uses off-line batch learning to be able to use advanced supervised learning techniques that can converge faster than on-line learning [48, p.319], and also avoid the destruction of learning the latter approach can produce [48, p.327].

Year of 2008

HRL - MAXQ framework - uses an approach of divide and conquer by decomposing an MDP into smaller MDPs [91, pp.227-228] and acting recursively [91, p.229]. It depends on the identification of goals and subgoals by the programmer [91, p.227]. Focuses itself on both state abstraction [91, p.227][91, p.260] and temporal abstraction [91, p.237]. It is an online, model-free algorithm [91, p.227].

Year of 2011

FRL - Neural Fitted Q Iteration with Continuous Actions (NFQCA) - this work approach consists of using batch learning of a Q-function based on experiences of state transitions, with a learning process for learning Neural Network based controllers comprising continuous action values [50, p.144]. Solves the limitation of Q-learning, that can only be used with discrete actions [50, p.147]. Uses a Critic implemented as a Neural Q-function, and an Actor represented as neural policy function [50, p.147]. It is similar to the Deterministic Policy Gradient (DPG) algorithm [16, p.2].

FRL - Doubly Robust (DR) - proposes a method to perform optimal control in a partially observed environment [130, p.1]. Focuses on evaluating policies taking context into account, as well as previous actions and rewards [130, p.1]. Tackles bias and variance problems deriving from models of rewards and models of past policies respectively [130, p.1]. Defines a Doubly robust estimator that uses the estimate of expected reward and the estimate of action probabilities [130, p.3].

FRL - Probabilistic Inference for Learning Control (PILCO) - proposes a sample efficient model-based RL algorithm for high dimensional problems, with a probabilistic dynamics model to reduce model bias (by including uncertainty) and the use of approximate inference to perform policy evaluation [131, p.1]. PILCO has problems in getting stopped at local optimas [131, p.7].

Year of 2013

FRL - Deep Q Network (DQN) - addresses the problem of processing high-dimensional sensory input with RL [19, p.1]. Uses a convolutional neural network that estimates future rewards by processing raw pixels [19. p.1]. It achieves high performance and a new level, by being able to perform across a set of different problems, without needing to use problem-specific features, using the same network architecture, raw input, and parameter values like the step size, rate of discount, exploration parameters [65. p.437]. While it is able to process high-dimensional observation spaces, when it comes to action spaces, it can only process discrete and low dimensional ones, so if it is to be applied to continuous action-spaces, a discretization has to occur (but brings the problem of the curse of dimensionality) [16, p.1]. Can be compared with NFQ, except that NFQ uses a batch update that has a cost per iteration that is proportional to the data set size, whereas DQN uses stochastic gradient updates that have a small cost per iteration and scales well with bigger data sets [19, p.4].

Year of 2014

FRL - Deterministic Policy Gradient (DPG) - proposes an algorithm, for continuous action spaces, that is based on the expected gradient of the action-value function, which can be calculated more easily than the stochastic policy gradient [47, p.1]. DPG uses an off-policy optimization with an actor-critic architecture and by employing an exploratory behaviour policy learns deterministic target policies [47, p.1]. Shows that deterministic policy gradient algorithms are better suited for high-dimensional action spaces than stochastic alternatives [47, p.1].

Year of 2015

FRL - Generalized Advantage Estimation (GAE) - makes advancements in high-dimensional continuous control with RL [30, p.2] by using state-value functions, which have a smaller dimensional input than state-action value functions, and so it is easier to learn than the later [30, p.12]. https://github.com/higgsfield/RL-Adventure-2/blob/master/2.gae.ipynb

FRL - Trust Region Policy Optimization (TRPO) - It provides a method to optimize large non-linear policies like NN [15, p.1]. This method is similar to policy gradient methods [15, p.1]. It has a good performance in physical control tasks, like walking, swimming, etc [15, p.1][15, p.6]. TRPO does not need to learn an action-value function [16, p.8]. Establishes boundaries that limit updates of the policy parameters to avoid the new policy to diverge too much relative to the existing policy [16, p.8], making training easier, and avoiding inability of Gradient Descent to make progress in more complex tasks [15, p.7].

FRL - Deep Deterministic Policy Gradient (DDPG) - extends DQN ideas, and can operate in continuous action spaces which DQN can't, being DDPG more suitable for physical control tasks [16, p.1] and also able to perform better with fewer experience steps (but still a lot) than DQN [16, pp.8-9]. It is an actor-critic algorithm and is able to generalize well and operate without having a model of the environment [16, p.1] across a diversity of domains [16, p.8]. DDPG combines the actor-critic structure with NN function approximation, by following strategies (i.e. off-policy training with a replay buffer, and a separate target network) implemented in DQN to make those NN more stable; it also takes advantage of the Batch Normalization technique [16, pp.2-3]. (see Glossary for Batch Normalization). DDPG is able to learn good policies from pixels of a camera image or/and from joints angles or other low dimensional inputs, and operate in a stochastic environment [16, p.2]. It can learn in big state and action spaces, and uses mini-batches to be able to operate in a large network [16, p.3]. DDPG is not sample efficient [29, p.1]. https://github.com/higgsfield/RL-Adventure-2/blob/master/5.ddpg.ipynb

FRL - Stochastic Value Gradient (SVG) - defines a framework for continuous control policies that use backpropagation [121, p.1]. SVG proposes the use of a deterministic function with external noise so that stochasticity can be integrated into the Bellman equation [121, p.1]. With the SVG framework several algorithms are presented and named SVG(∞), SVG(0) and SVG(1), and each

one has a different Bellman equation recursion depth [121, p.3].

Year of 2016

HRL - Option-Critic Architecture - proposes an option-critic architecture that can learn internal policies and termination conditions of options [93, p.1726] from [90]. Presents an alternative approach, that mixes the problem of discovering options with the problem of learning options, which works with linear and nonlinear function approximators, and favours transfer learning [93, p.1726]. The Critic is implemented with a NN [93, p.1731]. Options (or Skills [96, p.9]) can be learnt without any specification of subgoals or pseudo-rewards [93, p.1731].

HRL - Modulated Locomotor Controllers - explores proprioception, the sense of body position or self-movement, and defines a cortical network and a spinal network, as a way to surpass failures of monolithic architectures [96, p.1]. Addresses the question on how to combine coherent exploration (with modules and hierarchies) with the flexibility of data-driven end-to-end FRL approaches [96, p.1]. This work is inspired by the division of labour in motor control done by biological brains [96, p.1]. It defines two levels of controllers, high and low, the first using a non-recurrent NN and the later using a recurrent NN [96, pp.2-3]. Low-level controllers produce pre-trained locomotor behaviour [96, p.3] (or skills [96, p.9]) and high-level controllers direct behavior to achieve the task goal [96, p.6]. Uses A3C combined with the previously defined concepts [96, p.4]. This method of finding options with sparse reward environments differs from the babbling method [96, p.9].

HRL - hierarchical-DQN (H-DQN) - addresses RL problems of very sparse and delayed feedback, by combining top-level (meta-controller) learning of policies based on intrinsic goals , with low-level (controller) learning of policies based on atomic actions to accomplish the given goals [97, p.1]. This framework merges hierarchical action-value functions, considered at several temporal scales, with motivated and goal-driven deep RL [97, p.1]. Sparse feedback is something that is not handled well by non-linear approximators based RL (FRL), so H-DQN adopts a tactic of starting by learning ways to accomplish intrinsically generated goals, and then learning an optimal policy to sequence them [97, p.2]. Uses stochastic gradient descent at different time scales [97, p.2] to learn its own hyperparameters [97, p.5]. Uses separate DQNs for the top-level and for the low-level controllers [97, p.4].

HRL - STRategic Attentive Writer (STRAW) - This model can either be used in RL or in Natural Language Processing (NLP) [100, p.1] because it has a general sequence prediction architecture [100, p.6]. Proposes a deep RNN setup that can learn macro-actions, and output a several step action plan, and once in while updates the plan with new observations, using rewards as the driver of the learning process [100, p.1]. This model may suffer from the inability to react to stochastic environments while it is executing a plan. But this focus on execution allows to control computational costs [100, p.2]. Uses differentiable attentive reading [100, p.3] and A3C [100, p.5].

HRL - Hierarchical Deep Reinforcement Learning Network (H-DRLN) - proposes an approach to high-dimensional and lifelong learning type of problems [101, p.1553]. The architecture is able to transfer and reuse knowledge, by learning skills (referred to as Deep Skill Networks) that are integrated in a hierarchical DRL Network [101, p.1553]. Skills are selected and used or reused to solve sub-problems of the main unsolvable problem [101, pp.1553-1554]. The system can decide to execute a skill or a primitive action (in this case it will execute for only one timestep) [101, p.1556]. H-DRLN proposes a mechanism called skill distillation (a variation of policy distillation) that allows to retain knowledge efficiently and scale the system to lifelong scenarios [101, p.1553].

HRL - Iterative Hierarchical Optimization for Misspecified Problems (IHOMP) - proposes an hierarchical approach, for continuous high-dimensional domains, that reduces the effects of problem misspecification, by using arbitrary or automatically learned (with Regularized Option Interruption (ROI)) partitions of the state-space that are the base for a inter-communicating mesh of options [103, pp.1-2]. Each option (or skill or macro-action) specializes due to being focused on one

partition [103, p.2] and are updated throughout the life of the agent [103, p.5]. Misspecification can occur, mainly, when “good” features to represent a state are not found [103, p.1].

FRL - IRL - Generative Adversarial Imitation Learning (GAIL) - proposes an imitation learning algorithm to extract a policy directly from data, like one would be able to obtain using Inverse Reinforcement Learning (IRL), but in as faster way [31, p.1]. This algorithm is prepared to work in large environments [31, p.6]. https://github.com/higgsfield/RL-Adventure-2/blob/master/8.gail.ipynb

FRL - Actor Critic with Experience Replay (ACER) - proposes an actor-critic architecture using DRL and Experience Replay for continuous and discrete control problems [32, pp.1-2]. Introduces the following mechanisms: stochastic dueling networks architectures and truncated importance sampling with bias correction [32, p.1]. Also proposes a new TRPO method [32, p.1]. ACER can be seen as on off-policy equivalent of A3C [32, p.2]. The new TRPO method is done by maintaining a network of running average policies calculated from previous policies, that are used as references to avoid having the updated policies to go too far away from the references [32, p.5]. Humanoid agents, having a higher dimensionality action space, significantly benefit from using truncation and bias correction [32, p.10]. ACER uses the Retrace algorithm [32, p.3].

FRL - Bootstrapped-DQN - proposes a method to learn faster while doing deep exploration with deep NN in a complex environment [40, p.1], with a low computational cost [40, p.2], which allows scalability [40, p.8]. Applies the statistical resampling principle of bootstrapping (see Glossary) [40, p.2]. Implements a Network with several heads (Neural Networks) that have input from a shared NN that performs the task of feature extraction from a frame [40, p.2].

FRL - Asynchronous Advantage Actor Critic (A3C) - proposes a method that implements asynchronous gradient descent, in a parallel actor-learner setup [12, p.1]. This method performs well in several types of optimal control problems, like motor control or navigating using visual input [12, p.1]. Presents an alternative to Experience Replay (because of its limitations with on-policy learning), by using parallelism/multithreading, having multiple agents learning on several instances of the environment [12, p.1] and with the possibility of using different exploration policies [12, p.3]. It can run efficiently on extremely less powerful hardware, than other methods while performing at the same level or better [12. p.1][12. p.7]. Multithreading also showed benefits in older methods like one-step Q-learning, one-step Sarsa and n-Step Q-learning [12, p.6].

FRL - Advantage Actor Critic (A2C) - It is the synchronous version of A3C [12].

FRL - Policy Gradient and Q-Learning (PGQL) - proposes a model free algorithm that combines off-policy Q-Learning on a reply buffer with on-policy Gradient optimization [95, p.1].

FRL - Actor-Mimic - proposes an algorithm focused on transfer learning by having an agent to perform several tasks at the same time [117, p.1].

MRL - RL2 - proposes an approach that comprises an RL algorithm to create another RL algorithm [129, p.9], represented by a RNN [129, p.1]. The weights in the RNN represent the learned algorithm, and this RNN gets the same inputs as the “normal” RL algorithms: actions, observations, rewards and stop conditions [129, p.1]. The RNN is able to maintain state across multiple episodes [129, p.1]. DR2 can be used in high-dimensional state and action spaces [129, p.1].

FRL - Normalized Advantage Functions (NAF) -it is a continuous variation of the Q-learning algorithm [76, p.1]. NAF explores the use of learned models to make model-free learning faster [76, p.1]

FRL - Retrace - an off-policy, sample efficient multi-step method algorithm with low variance. [110][32, p.3]..

Year of 2017

HRL - FeUdal Networks (FUNs) - proposes a method that works at two different levels, with different time resolutions: the manager, that selects goals and is motivated by an extrinsic reward, and the worker levels [92, p.1]. The worker creates primitive actions at each time step and is motivated by an intrinsic reward [92, p.1]. FUNs allow an extremely long timescale credit assignment (which is good for sparse reward problems) and memorisation [92, p.1]. This method uses a new type of RNN called a dilated LSTM, that enables the usage of a backpropagation algorithm with hundreds of steps [92, p.1]. The goals of the Manager are trained with an approximate transition policy gradient [92, p.2]. FUNs use A3C for optimisation [92, p.5]. This method is also good at transfer and multitask learning [92, p.8].

HRL - Abstract Markov Decision Processes (AMDP) - proposes a model based [102, p.2] method for fast planning in large state-action spaces where the number of objects induce combinatorial growth [102, p.1] and the environment is stochastic [102, p.2]. Divides goals into subgoals that recursively solves; and selects only the information of the state space that is needed to solve each decision [102, p.1]. Uses graph nodes to represent primitive actions or sub-problems [102, p.2].

FRL - 51-Atom Agent (C51) - explores the analysis of value distribution in approximate distributional RL, instead of using the expectation in the states values as Expectation RL does [20,p.1]. C51 algorithm provides a more meaningful state value analysis, because instead of calculating only one value per state, it adheres to the notion that stochasticity can produce very disparate state values, and therefore a distributional approach makes more sense [20]. https://github.com/flyyufelix/C51-DDQN-Keras

FRL - EXpert ITeration (EXIT) - this algorithm is motivated by human thought dual process theory [52, p.9] and uses tree search to assist the NN training [52, p.1]. This method implements apprentices (generalize policies with NN) and experts (explore with tree search), to iterate through training in an Imitation Learning style and self-play [52, pp.2-3].

FRL - Alpha-Zero - proposes a general purpose algorithm that starts without any knowledge of the environment, except for the available actions (rules), and moves forward with self-play [26, pp.1-2]. It uses NN to calculate action values and MCTS to simulate self-play [26, pp.2-3].

FRL - Actor Critic using Kronecker-factored Trust Region (ACKTR) - As the name says this algorithm uses an Actor-Critic architecture [34, p.1]. It can be used with continuous and discrete control policies, and employs Deep NN, plus a variation of TRPO with a new technique called Kronecker-Factored Approximated Curvature (K-FAC) [34, pp.1-2]. https://github.com/openai/baselines/tree/master/baselines/acktr

FRL - Quantile Regression Deep Q Learning (QR-DQN) - presents a technique for RL by using the distribution of the value function (following C51 work), instead of using an averaged value, to guide an agent in a stochastic environment [21, p.2892]. It focuses on minimizing the Wassertein similarity metric to compare the value function distribution to a parametrized quantile distribution [21, p.2895]. QR-DQN builds on top of DQN and a set of distributional math operations, to achieve a complete distributional RL algorithm [21, p.2896].
https://github.com/senya-ashukha/quantile-regression-dqn-pytorch

FRL - Imagination-Augmented Agents (I2A) - proposes a new architecture for RL that consists of a model-free agent control method, plus a model-based simulation or imagination method [23, p.1]. In this way, negative, irreversible consequences of trial-and-error actions in the real environment can be avoided, and the better scaling of the model-based part can be loaded with more work to do, getting better results without a worse computational cost of the model-free part of the model [23, p.1]. This architecture tackles the inefficiency of model-based methods in complex

environments, by using the imagination part, and still getting the benefits of this type of methods in a agent acting in a complex world (i.e. better generalization) [23, pp.1-2]. States that reward expectation is helpful but it can be discarded, and still get good performance [23, p.6].
https://github.com/higgsfield/Imagination-Augmented-Agents

FRL - Model-Based and Model-Free (MBMF) - presents an algorithm that combines model-based and model-free approaches [24, p.7579]. Model-free methods need a lot of samples to get good results, so this algorithm uses a model-based approach to get a better sample efficiency, and does it with a Deep NN, that allows extraction of dynamics rules [24, p.7579]. The model-based part output stands as an expert from whom the model-free part will learn [24, p.7582]

FRL - Proximal Policy Optimization (PPO) - proposes an algorithm that is as efficient and reliable as TRPO but, it is of simpler implementation [14, p.1] by making minor changes to a vanilla policy gradient implementation [14, p.8]. PPO can be used on continuous high-dimensional control problems [14, p.7]. PPO introduces a new family of policy gradient methods that alternate between optimizing a “surrogate” objective function by the use of stochastic gradient ascent and sampling data through interaction with the environment [14, p.1]. Normal policy gradient methods execute one gradient update per data sample, and PPO uses an objective function that allows multiple epochs of minibatch updates. [14, p.1]. https://github.com/higgsfield/RL-Adventure-2/blob/master/3.ppo.ipynb

FRL - Soft Q-Learning (SQL) - proposes an algorithm for continuous action and state spaces, with improved compositionality and exploration, allowing transfer learning from task to task [111, p.1]. It uses an energy based model represented with an energy function or a neural network as an energy function approximator [111, p.3]. https://github.com/haarnoja/softqlearning

FRL - Value Prediction Network (VPN) - proposes a method that combines the model-free and model-based approaches in the same neural network [122, p.1]. It explores conditional options when facing future values of the reward function instead of using the future observations or sensorial data [122, p.1]. VPN performs well in stochastic environments when compared to purely model-free or model-based algorithms [122, p.1]. The new neural network architecture combines a dynamics model of states abstraction (model-based part) with these abstractions being mapped to rewards (model-free part) [122, p.1]. VPN can be combined with other RL algorithms [122, p.4].

MRL - Model Agnostic Meta Learning (MAML) - proposes a model-agnostic method which means it can also be used in a variety of learning problems [126, p.1]. MAML looks for good generalization using a small amount of training data in a new task, and also to produce a model training that is easy to tune [126, p.1]. https://github.com/cbfinn/maml / https://github.com/cbfinn/maml_rl

HRL - Fine Grained Action Repetition (FiGAR) - proposes a framework that allows an agent to choose the timeframe and repetition of an action [120, p.1]. It can be applied in combination with existing RL algorithms like DDPG,.TRPO or A3C and improve policy optimization by working with macro-actions at different time frames [120, p.10]. FiGAR is not prepared to interrupt a macro-action, which in a stochastic environment makes it less efficient [120, p.10].

Year of 2018

FRL - Model-based Value Expansion (MVE) - it is a technique that takes advantage of the imagination mechanism like the I2A algorithm, and the creation of dynamics models [25, p.1], but puts some restrictions, in the horizon length, that ultimately allows improvement of the learning sample complexity [25, p.8]. MVE is a method that tries to combine the best of model-free and model based approaches [25, p.1], like MBMF does. MVE calculates the short term horizon by creating a dynamics model (model-based), and the long term horizon by using Q-learning (model-free) [25, p.1].

FRL - World Models - this method creates a virtual environment and dynamics model to train the RL Agent [10, p.2]. It does that from high dimensional image data [10, p.7], and sometimes

gets details that are not important, while missing other details that are relevant [10, p.8]. The Agent also needs to explore the real world so that the model can be perfected [10, p.8]. This approach differs from many other models that do their training by having the agent interacting with the real environment [10, p.1]. In order to avoid having the model exploiting the virtual models flaws (by generating an adversarial policy [10, p.6]), the training occurs with a noise-added version of that same virtual environment [10, p.2]. This training in the virtual environment may lead, in some cases, to inadequate policies that will fail in the real environment [10, p.6]. By having the virtual environment, the model is able to plan ahead, since it has access to a probability distribution of future events [10, p.4]. https://learningtopredict.github.io/ https://worldmodels.github.io

FRL - Twin Delayed DDPG (TD3) - proposes an algorithm (that extends DDPG [17, p.6]) with an actor-critic architecture [17, p.2] and uses two critic NN [17, p.6]. Explores a technique called target networks to limit errors with function approximators and stochastic optimization [17, p.8]. By using two critics and getting the minimum value between the two it can avoid an overestimation problem [17, p.1]. Aims at solving problems affecting continuous control actor-critic settings, like overestimation bias and accumulation of error [17, p.1]. It is assumed that even an overestimated value may be used as an upper boundary for the true value [17, p.1].
https://github.com/higgsfield/RL-Adventure-2/blob/master/6.td3.ipynb

FRL - Soft Actor-Critic (SAC) - proposes a sample efficient, stable, model-free off-policy algorithm with an actor critic architecture and Deep NN [18, p.1]. Explores the concept of maximum entropy to create robust policies in high-dimensional and continuous spaces [18, pp.1-2]. SAC performs at the same level as DDPG, PPO and TD3 in simple tasks, but performs better in more complex ones [18, p.7]. SAC creates stochastic policies while learning, that are converted into deterministic policies in the end (for performance reasons) [18, p.7]. https://github.com/haarnoja/sac

HRL - HIerarchical Reinforcement learning with Off-policy correction (HIRO) - explores a generally applicable method that is suitable for real world optimal control problems, by using an off-policy approach [94, p.1]. Proposes two levels of controllers, where the top level (responsible for planning) creates intrinsic rewards and goals, in the form of goal states, for lower level controllers (movement, object interaction, discrete decision-making) to try to meet, and both levels are trained [94, pp.1-2]. Having the two levels being trained creates a non-stationary problem for the policy of the higher level, that HIRO corrects by using historical data labeled with the higher level chosen action [94, p.2]. HIRO uses the raw form of state observations [94, p.2]. The intrinsic reward is viewed as the distance between the goal observation and the current observation [94, p.5]. HIRO is related to the FeUdal Networks algorithm [94, p.6].

HRL - Modulated Policy Hierarchies (MPH) - proposes a method to solve problems with sparse rewards, applying intrinsic motivation and modulation signals by communicating with bit-vectors [99, pp.1-2]. Instead of selecting among the available skills in an exclusive way, MPH, with bit vectors, is able to combine the available skills (implemented as NN) [99, pp.1-2]. Each level of the hierarchy is trained separately using PPO in order to avoid non-stationarity problems [99, p.1]. The lower level (worker) combines the modulated signals with the environment state to produce a final action [99, p.4]. MPH implements time scales (temporal abstraction) by activating higher level policies less frequently [99, p.4]. As intrinsic motivation, this method uses a curiosity-guided exploration bonus that has the effect of accelerating learning in a sparse reward environment [99, p.4].

HRL - Metalearning Shared Hierarchies (MLSH) - proposes a method that learns temporally extended primitives (options/skills) and policies from a multi-task setting [98, p.1]. From an observation, a master policy (that is associated with a master action) prescribes the use of a sub-policy that will execute an action [98, p.3]. MLSH uses NN in an arbitrary RL algorithm like A3C, PPO, TRPO or DQN to update the master policy [98, p.4]. In training the sub-policies, the master policy decision is treated as a part of the environment [98, p.4].

FRL - Asymmetric Self-Play - proposes a model-free method of unsupervised training for an agent to explore and learn without extrinsic rewards in a sparse reward environment [113, p.1]. An

agent is configured with two controllers, being one responsible for setting tasks and the other responsible for completing those tasks, and once the later is becoming more experienced, the former is less frequently used [113, p.1]. This method can be used in discrete and continuous settings [113, p.5].

FRL - Probabilistic Ensembles with Trajectory Sampling (PETS) - this algorithm explores uncertainty aware deep neural network dynamics models to tackle the problem of deficient asymptotic performance in model based architectures [114, pp.1-2]. Other algorithms use Bayesian nonparametric models instead [114, p.9]. Applies some other model-free techniques to a model-based setting (i.e. ensembling, output Gaussian distribution parameters, MPC) [114, p.9]. PETS does not use policy learning [114, p.9]. https://github.com/kchua/handful-of-trials

FRL - STochastic Ensemble Value Expansion (STEVE) - proposes an algorithm that mixes model-based and model-free approaches, in order to mitigate bias errors of the model based approach and to improve problems of sample inefficiency of model-free methods [115, p.1]. The environment model is only used once in a while with rollouts of different horizon sizes to avoid introducing much error inherent to model-based methods [115, p.1]. STEVE is an extension of MVE [115, p.2]. This algorithm uses uncertainty-awareness mechanisms [115].

MRL - Proximal Meta-Policy (ProMP) - proposes a meta-RL algorithm with improved credit assignment and improved meta-policy gradient estimation [128, p.1]. ProMP achieves an effective identification of tasks to be learned by optimizing the pre-update sampling distribution [128, p.9].

HRL - Hierarchical Self-play (HSP) - this method tackles complex tasks with sparse rewards [104, p.8]. It uses unsupervised asymmetric self-play (to explore decomposition of tasks) and a continuous sub-goal vector [104, pp.1-2]. The agent sets its own goals forced by adversarial rewards (according to what the environment allows), and then tries to achieve these goals respecting a time limit [104, p.1]. The low level policies have access to the current state and to the goal vector (an encoded target state) [104, pp.1-2]. The higher level policies are trained in a sparse reward logic [104, p.2]. HSP defines two levels of policies implemented as NN that, at first are prepared by exploring and creating skills (by having 2 lower level actors playing against each other) and that, later are trained (at the higher level) using external rewards [104, p.2]. HSP breaks episodes into smaller segments in order to avoid problems coming from higher complexity [104, p.4]. It has limitations in the self-play task, because some knowledge of the domain needs to be embedded, to be possible to recognize if this task has been completed successfully [104, p.8].

FRL - Model Ensemble Trust-Region Policy Optimization (ME-TRPO) - proposes a method to combine model-free and model based approaches, by using deep NN to both learn a model and learn policies [132, p.1]. Uses likelihood ratio derivatives instead of backpropagation to seek a more stable learning process [132, p.1]. https://github.com/thanard/me-trpo

Year of 2019

FRL - Stochastic Lower Bounds Optimization (SLBO) - proposes a model based algorithm that can be used in continuous control settings [116, p.10]. SLBO learns the models by using multi-step prediction loss [116, p.8]. Makes use of TRPO for policy optimization, introducing an entropy term to the objective function [116, p.9]. SLBO is an instantiation of a framework aimed at giving theoretical guarantees when designing and analyzing model based algorithms [116, p.1].

HRL- Probabilistic Embeddings for Actor-critic meta-RL - (PEARL) - proposes an actor-critic Meta-RL algorithm that aims at allowing an agent to develop new skills with a small amount of data [127, p.1]. PEARL is an off-policy method that disentangles task inference and control [127, p.1]. It uses online probabilistic filtering on task variables that are latent in order to understand how to solve a new task using a small sample [127, p.1]. Learning is not only made more efficient, but also exploration is made more efficient with this approach [127, p.1]. PEARL trains by learning latent representations of tasks and then adapts to test tasks from their inferred representations [125, p.1].

This algorithm focuses not on learning a policy for each task, but on to learn a policy for a group of tasks that share the same structure [127, p.1]. PEARL builds on top of SAC [127, p.5]. https://github.com/katerakelly/oyster

FRL - Model-Based Policy Optimization (MBPO) - proposes a procedure of using small model-generated rollouts coming from real data, as a better solution than having a big model of the environment [54, p.1]. MBPO combines the generalization strength of model-free algorithms with the speed in learning of model-based ones [54, p.2]. This approach allows it to be used in high-dimensional problems [54, p.2]. MBPO also works well with long horizon tasks [54, p.9].

FRL - Bootstrap Dual Policy Iteration (BDPI) - this is a model free, very sample-efficient algorithm for continuous state spaces and discrete action spaces [39, p.1]. BDPI uses a value based approach with an actor-critic architecture having several critics (implemented with a flavour of DQN) operating in off-policy [39, p.1]. The actor is trained using all the critics [39, p.2]. With BDPI, hyperparameters do not need a lot of tuning, since the algorithm is very robust concerning hyper parametrization [39, p.2]. BDPI uses an experience buffer [39, p.3]. https://github.com/vub-ai-lab/bdpi

HRL - Factorized Macro Action Reinforcement Learning (FaMARL) - proposes a method that allows macro actions to be applied to general RL algorithms (to reduce dimensionality of the action space), by using disentangled representation [41, p.1]. FaMARL focuses on the segmentation of a macro action from expert demonstrations [41, p.1]. Instead of using micro / primitive actions, macro actions latent space can be then used with any RL algorithm [41, p.3].

FRL - Deterministic Value Gradients (DVG) - proposes a method for infinite horizon problems [29, p.2216]. DVG is a model based algorithm that uses an actor-critic architecture [29, p.3320].

FRL - Deterministic Value-Policy Gradient (DVPG) - proposes a method that combines model free and model based approaches for infinite horizon problems [29, pp.3316-3317]. Uses Deterministic Value Gradients and Deterministic Policy Gradients [29,p.3317]. DVPG can either be used with imagination rollout and value expansion model-based techniques [29, p.3317]. This is a temporal difference method [29, p.3320]. DVPG uses an actor-critic architecture [29, p.3321].

FRL - Deep Soft Policy Gradient (DSPG) - proposes a maximum entropy RL model-free algorithm with an actor-critic architecture, and off-policy optimization [46, p.3425]. DSPG combines policy and value based methods [46, pp.3425-3426]. DSPG can be used in continuous control problems [46, p.3430].

HRL - Hierarchical Actor-Critic (HAC) - proposes an HRL method (named Hindsight action transitions) that solves the non-stationary instability problem when more than one level of policies are learnt at the same time, and it does this by having each layer trained considering the lower layer is stable and optimal with the use of a simulation transition function [80, pp.1-3]. HAC can be used in continuous state and action spaces with sparse rewards, and has successfully implemented a >2 layer architecture learning different levels policies in parallel [80, pp.1-2]. HAC can also be used in discrete problems [80, p.10]. Each layer has access to the external current state, but its goals are provided by the upper layer as a sub-goal [80, p.2].

HRL - Model Free HRL framework - proposes an HRL model free algorithm that focuses on unsupervised subgoal discovery, learning skills and the use of intrinsic motivation [105].

Year of 2020

FRL - Meta Q Learning (MQL) - proposes an off-policy algorithm taking context into account and meta-training policies [53, p.1]. Context stands as a meta-training technique [53, p.9]. MQL handles the hyper-parameter sensitivity problem of off-policy methods by adapting to the distribution shifting [53, p.9]. MQL reuses data from the replay buffer [53, p.10].

FRL - Advantage-weighted Behaviour Model (ABM) - proposes a method that allows off-policy RL in continuous control settings, without resorting to batch processing alone, which has worse results than on-policy processing [49, p.1]. ABM explores policies in a batch to create a weighted model, as a prior, that can be used to correct the policy being used, and in this way combine off-policy and on-policy strengths [49, p.2].

MRL - Model-based Adversarial Meta-Reinforcement Learning (AdMRL) - focuses on solving one of the problems of meta-RL that arises when the distribution of the tasks used to learn is very different from the tasks being executed [125, p.1]. Proposes a minimax formulation to tackle the gap between distributions, and optimize it by either learning the dynamics model on a fixed task or by looking for the adversarial task for the current model, in an alternated process [125, p.1].

Appendix D – Coding Environment

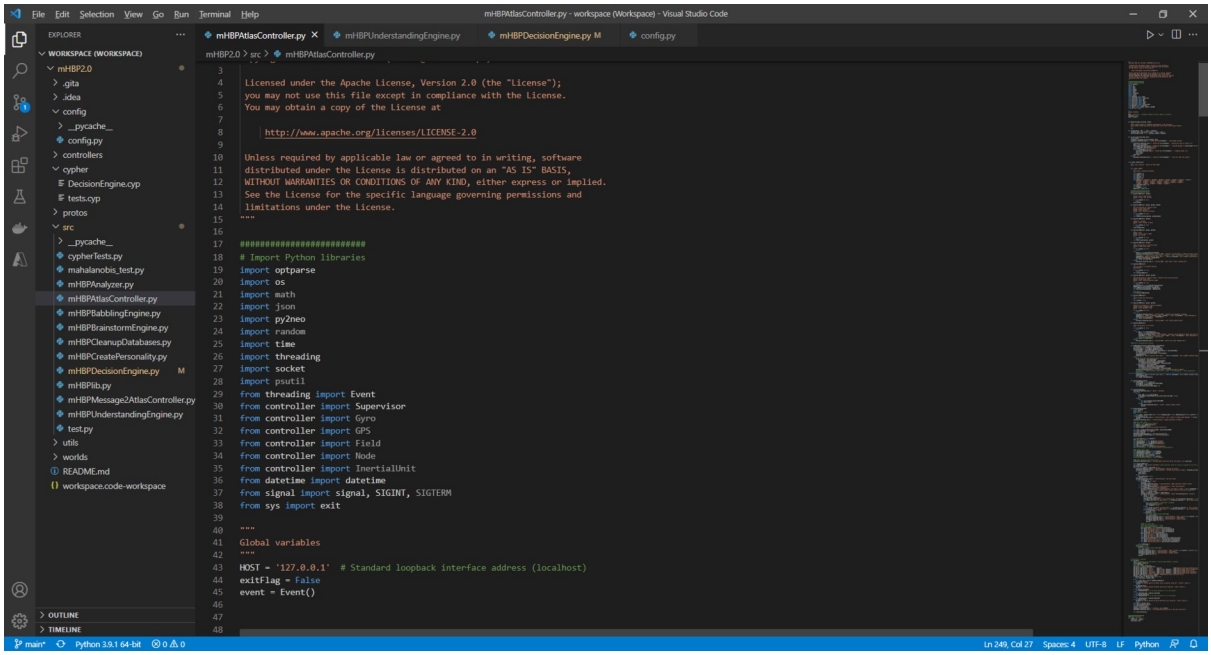


Figure D-1 – Visual Studio Code

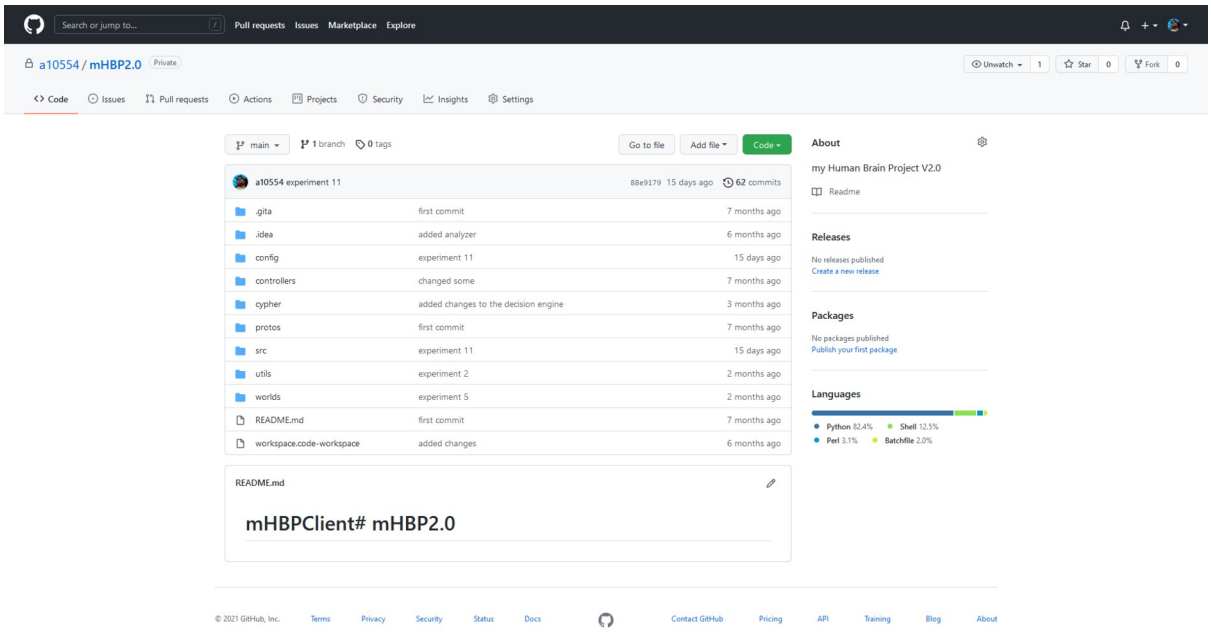


Figure D-2 – Github.

Annexes

Annex A - Physics Engines and other test Environments Screenshots

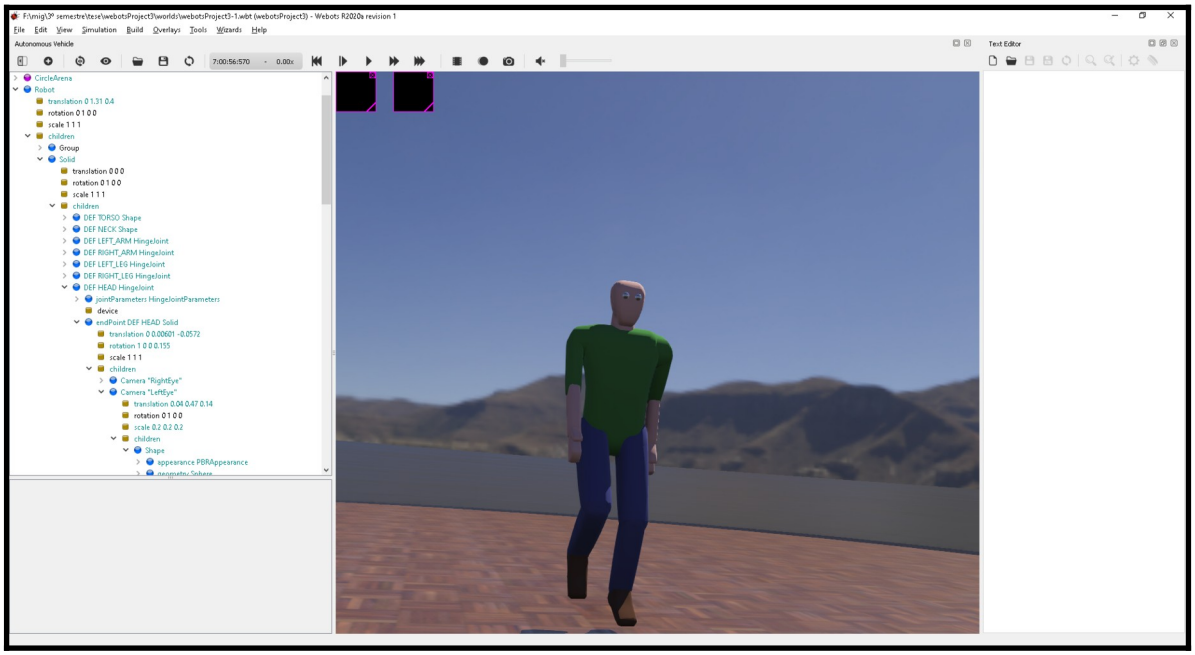


Figure A-1 *- Webots world simulation with a “pedestrian” model.*

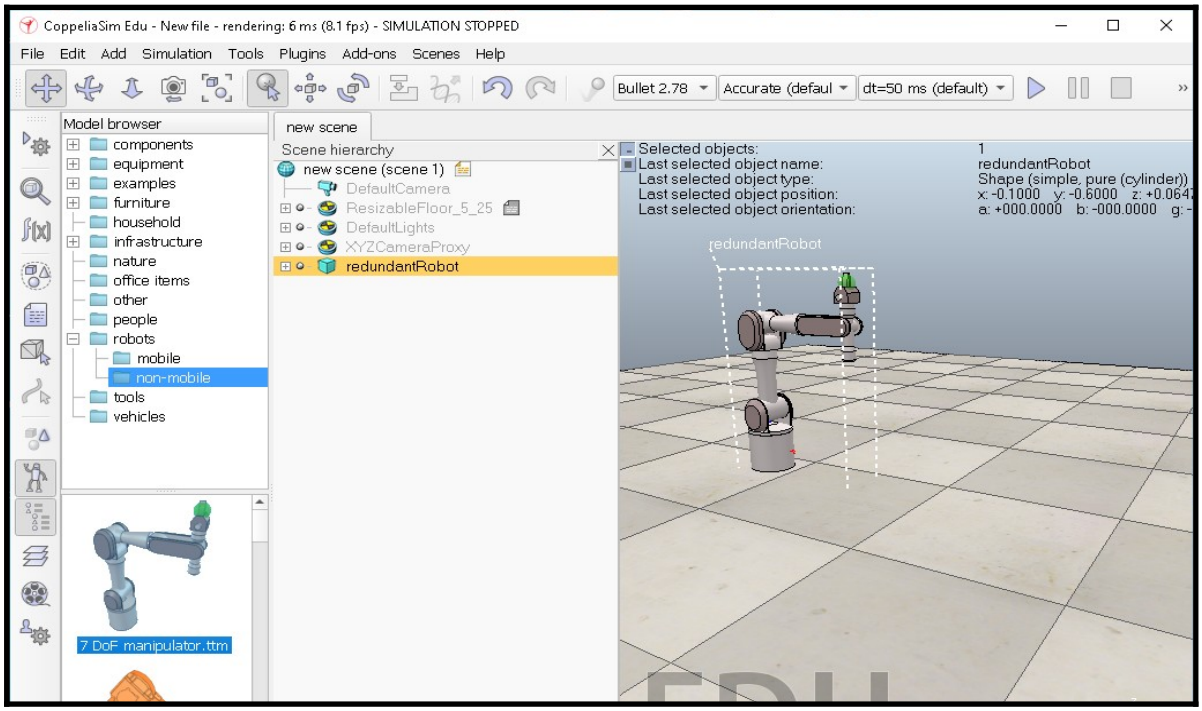


Figure A-2 *– Coppelia / Vrep IDE Screenshot.*

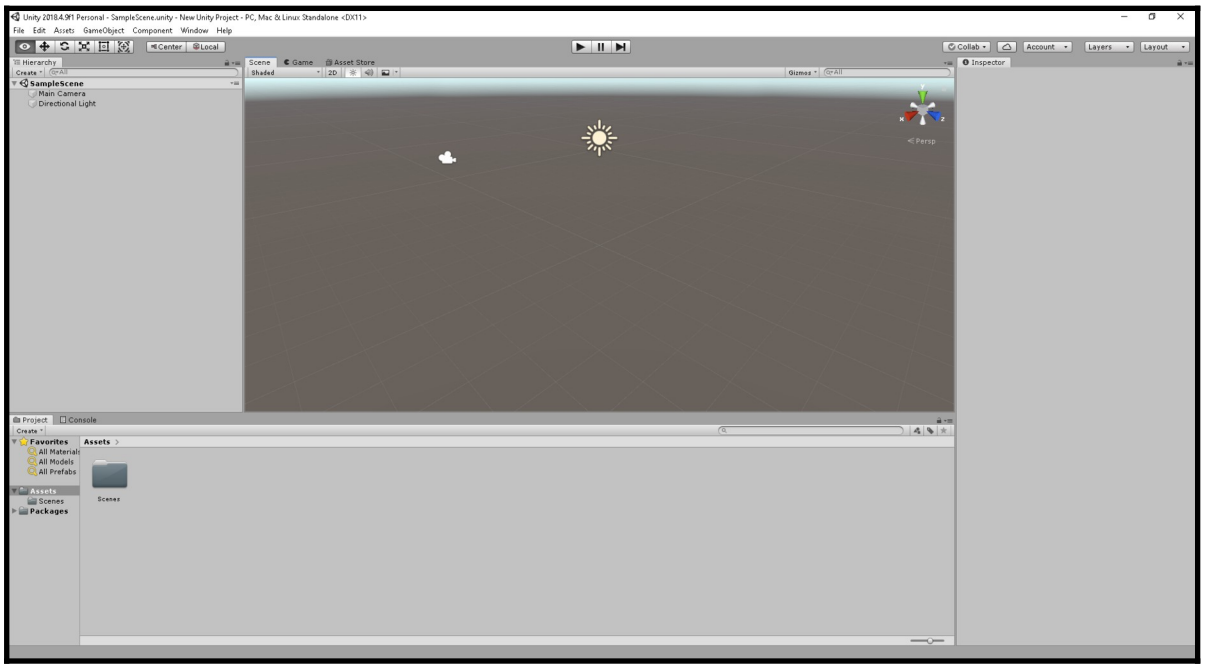


Figure A-3 *– Unity IDE Screenshot.*

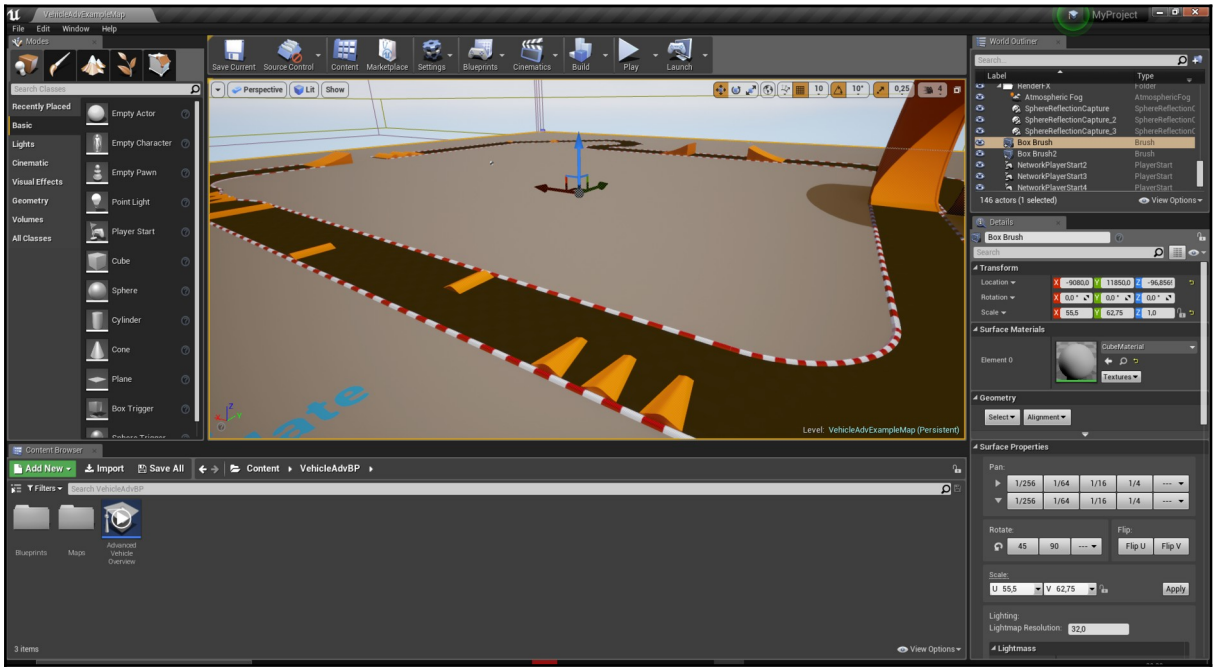


Figure A-4 *– Unreal Engine.*

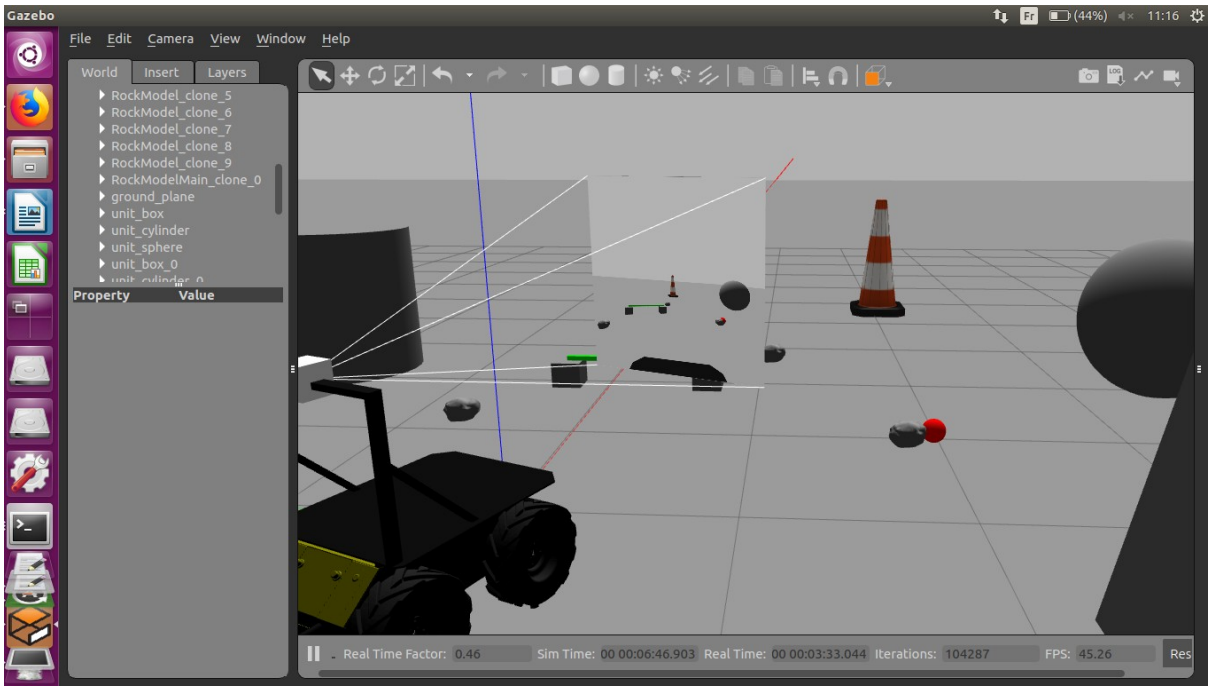


Figure A-5 *– gazebo IDE Screenshot.*

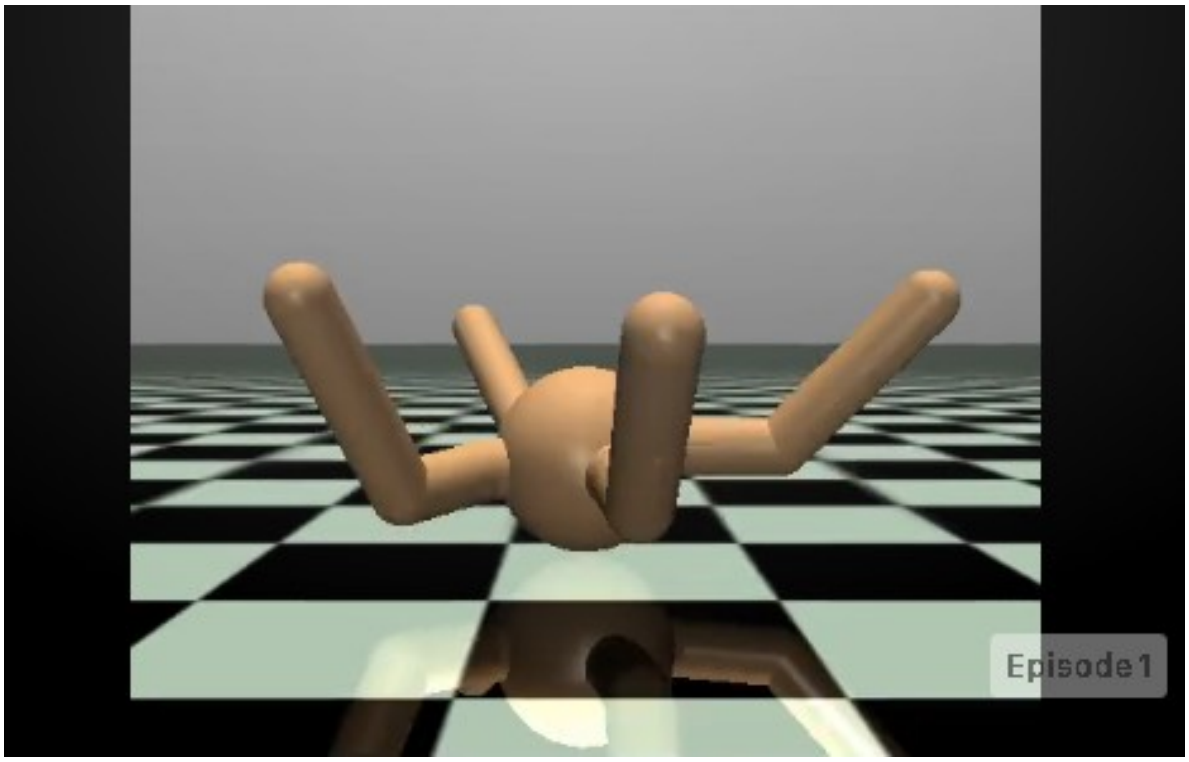


Figure A-6 *- OpenAi Gym Screenshot.*

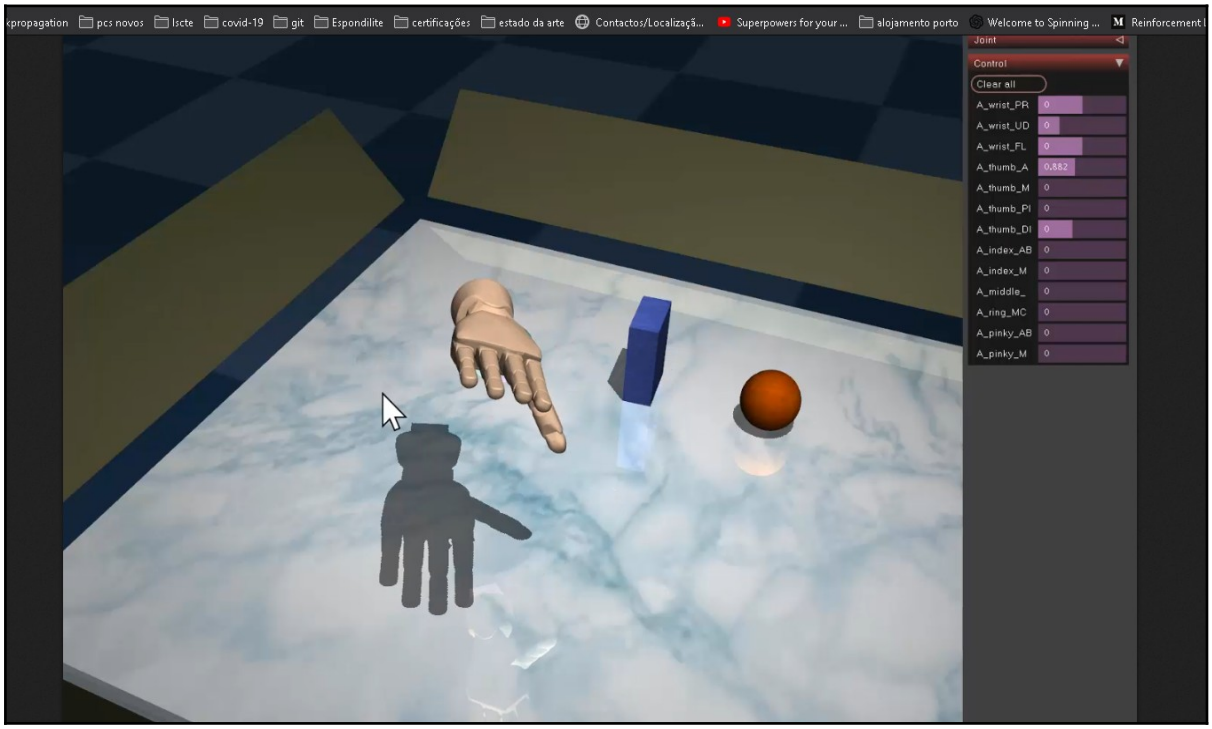


Figure A-7 *- MuJoCo Screenshot.*

Figure A-8 *- DeepMind Lab Screenshot.*

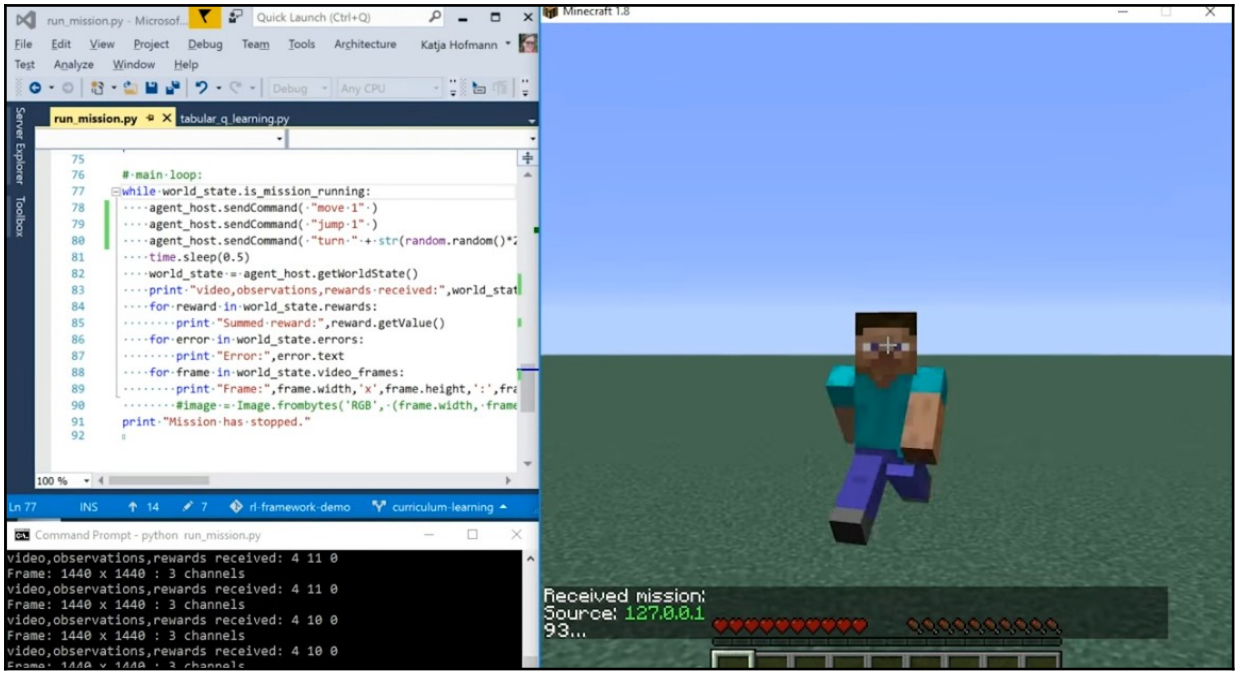


Figure A-9 *- Malmo Platform Screenshot.*

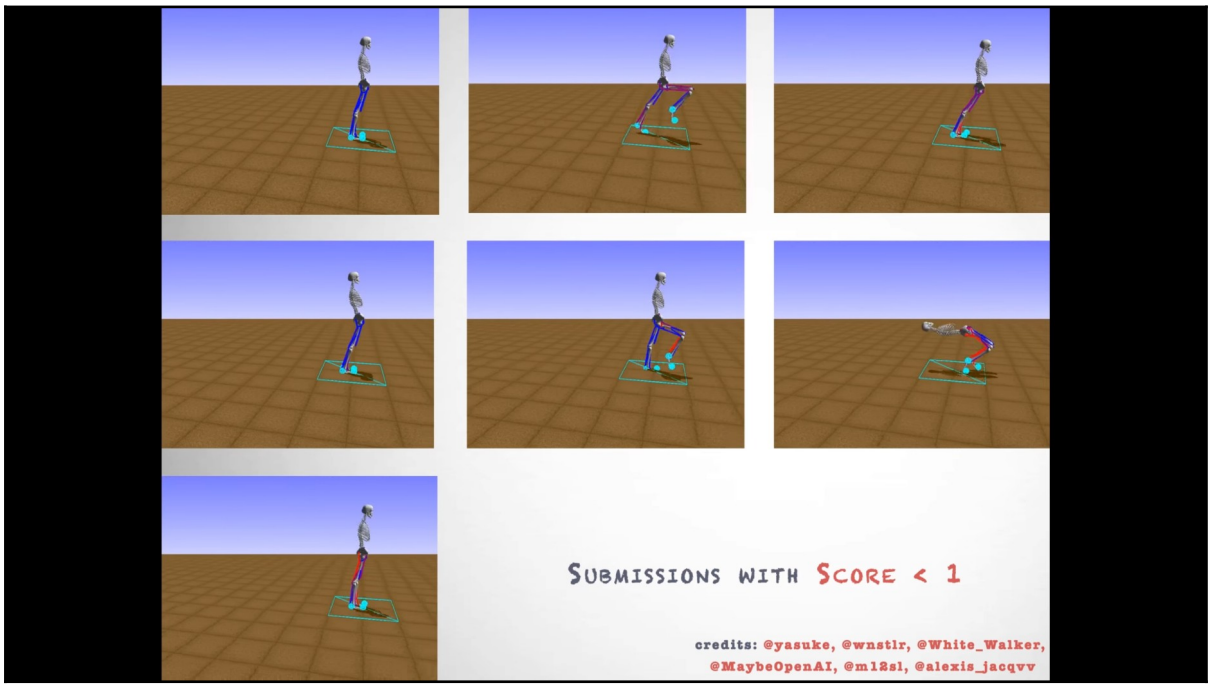


Figure A-10 *- OpenSim RL Screenshots.*

Figure A-11 *- Atari benchmarks (Bellemare et al., 2012), Arcade Learning Environment (ALE) (Bellemare et al., 2013), a benchmark of Atari 2600 arcade games.*

INSTITUTO UNIVERSITÁRIO DE LISBOA | My Human Brain Project (mHBP) | 2021 | **Author**: José Salvador