TECHNICAL UNIVERSITY OF CRETE
ELECTRICAL AND COMPUTER ENGINEERING

DIPLOMA THESIS

Deep Reinforcement Learning Exploiting a Mentor's Guidance

Author:
IASON CHRYSOMALLIS

Committee:
GEORGIOS CHALKIADAKIS, ASSOCIATE PROFESSOR - SUPERVISOR
AGGELOS BLETSAS, PROFESSOR
VASILIS SAMOLADAS, ASSOCIATE PROFESSOR

A thesis submitted in fulfillment of the requirements for the diploma of Electrical and Computer Engineering

September, 2021

Abstract

IASON CHRYSOMALLIS

Deep Reinforcement Learning Exploiting a Mentor's Guidance

Imitation is a popular technique of behavioral learning widely practiced in nature. The most famous applications involve animal babies imitating their parents, with imitation providing the stepping stone to walk their first steps in life survival. Additionally, imitation examples can be found in cross species instances, with most known samples the voice imitation of parrots or crow behavioral imitation.

The imitation learning paradigm has naturally been taken up in machine learning applications, implemented in supervised learning and in reinforcement learning, mostly with the use of explicit imitation, where the mentor agent attempts to explicitly teach learners. Implicit imitation, on the other hand, assumes that learning agents observe the state transitions of an agent they use as a mentor, and try to recreate them based on their own abilities and knowledge of their environment. Though it has also been employed with some success in the past, implicit imitation has only recently been utilized in conjunction with deep reinforcement learning, the current leading reinforcement learning paradigm.

In this thesis, we enhance the operation of implicit imitation by adding four state-of-the-art deep reinforcement learning algorithms, treated as "imitation optimization modules". These include Double Deep Q-network [Hasselt, Guez, and Silver, 2016], Prioritized Experience Replay [Schaul et al., 2016], Dueling Network Architecture [Wang et al., 2016] and Parameter Space Noise for Exploration [Plappert et al., 2018]. We modify these appropriately to better fit the implicit imitation learning paradigm. By enabling and disabling those methods we create diverse combinations of them; systematically test and compare the viability of each one of these combinations; and end up with a clear "winner": the combination of Double Deep Q-network, Prioritized Experience Replay and Dueling Network Architecture.

Περίληψη

Ιάσων Χρυσομάλλης

Βαθιά ύπο Καθοδήγηση Ενισχυτική Μάθηση

Η μίμηση αποτελεί μία τεχνική συμπεριφορικής εκμάθησης, ευρέως χρησιμοποιούμενη στην φύση. Στο ζωϊκό βασίλειο, για παράδειγμα, τα μωρά μιμούνται τους γονείς τους, και η μίμηση τα εφοδιάζει με τις κατάλληλες γνώσεις για να περπατήσουν στα πρώτα τους βήματα επιβίωσης. Παραδείγματα μίμησης παρατηρούνται και μεταξύ διαφορετικών ειδών, όπως στην φωνητική μίμηση των παπαγάλων ή στην συμπεριφορική μίμηση των κορακιών.

Η μίμηση, λοιπόν, δεν θα μπορούσε να μην συμπεριληφθεί σε εφαρμογές της μηχανικής μάθησης, όπου αλγόριθμοι των πεδίων επιτηρούμενης μάθησης και ενισχυτικής μάθησης εκμεταλεύονται την χρήση τεχνικών απευθείας, κυρίως, μίμησης, όπου ο πράκτορας που λειτουργεί ως 'μέντορας' προσπαθεί να 'διδάξει' απευθείας άλλους. Η μηχανική εκμάθηση μέσω έμμεσης μίμησης, από την άλλη, θεωρεί ότι οι πράκτορες-μιμητές απλά παρατηρούν τις αλλαγές καταστάσεων που προκύπτουν από την συμπεριφορά ενός πράκτορα που επιλέγουν ως μέντορα, και προσπαθούν να τις αναπαράγουν με βάση τις δικές τους δυνατότητες και γνώση του περιβάλλοντός τους. Αν και η έμμεση μίμηση έχει χρησιμοποιηθεί με ικανοποιητικά αποτελέσματα στο απώτερο παρελθόν, μόλις πρόσφατα έχει αξιοποιηθεί σε συνδυασμό με βαθιά ενισχυτική μάθηση, η οποία αποτελεί μια τρέχουσα τεχνολογία αιχμής στη μηχανική μάθηση.

Στην παρούσα διπλωματική εργασία, βελτιώνουμε περαιτέρω την διαδικασία της έμμεσης μηχανικής εκμάθησης ενσωματώνοντας τέσσερις σύγχρονους αλγόριθμους βαθειάς ενισχυτικής μάθησης, τους οποίους θεωρούμε και χρησιμοποιούμε ως δομικά στοιχεία βελτιστοποίησης της προσπάθειας μίμησης. Οι εν λόγω αλγόριθμοι είναι οι Double Deep Q-network [Hasselt, Guez, and Silver, 2016], Prioritized Experience Replay [Schaul et al., 2016], Dueling Network Architecture [Wang et al., 2016] και Parameter Space Noise for Exploration [Plappert et al., 2018]. Προσαρμόσαμε τη λειτουργία των αλγορίθμων ώστε να συνάδει με το μοντέλο της έμμεσης μίμησης. Ενεργοποιώντας και απενεργοποιώντας τις παραπάνω μεθόδους, δημιουργούμε ποικίλους συνδυασμούς αυτών, και δοκιμάζουμε μεθοδικά και συγκρίνουμε την βιωσιμότητα του κάθε ενός από αυτούς τους συνδυασμούς. Οι πειραματισμοί μας κατέληξαν στην ανάδειξη ενός ξεκάθαρου "νικητή": συγκεκριμένα, του συνδυασμού των Double Deep Q-network, Prioritized Experience Replay και Dueling Network Architecture.

Acknowledgements

I would like to deeply thank all of those that supported and helped make this thesis a reality.

First and foremost, I would like to express my gratitude towards my supervisor Georgios Chalkiadakis PhD for his continuous motivation, guidance and advice throughout the development of this work.

Also, I am thankful to my family and friends that were next to me with their constant encouragement and suggestions.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Deep Reinforcement Learning, a machine learning sub-category that combines reinforcement learning with neural networks, is gradually becoming a more reliable way to solve problems or play games and create intelligent agents that can outperform human experts in the field. In environments with continuous state space where tabular reinforcement learning fails to provide results, Deep Reinforcement Learning provides a model with the ability to solve such complex problems that were out of reach otherwise. Currently, the most used technique, Deep Q-network, has created a new benchmark to the latest state of the art, drawing the attention of research and inspiring a new wave of reinforcement learning improvements. However, the more complex the problem in need of solving the longer it takes for the training procedure to be completed.

The need for accelerated training arises. This can be achieved through imitation methods. The most common applications of imitation in machine learning take advantage of explicit imitation. According to explicit imitation, a highly trained agent -the mentor- provides all information available to it (includes states, actions taken and rewards received) to the observer agent -the trainee-, granting the opportunity to explicitly imitate its behaviour. Since complete transparency is not always available, another application of imitation learning is used, implicit imitation learning. With just the state transition observations of the mentor, the trainee can decrease its training time and reach the skill level of its mentor in less training steps than the mentor by filling the blanks of the information it needs, using its own heuristic approaches. This method, inspired by behavioral examples in nature, was thoroughly examined by [Price and Boutilier, 2003]. By creating a framework for imitation reinforcement learning, accelerated training was achieved. No information other than the state transitions is required, such as underlying code or model architecture, meaning it can even be combined with human agents provided that a perfectly observed environment can be used.

Implicit imitation learning was combined with deep reinforcement learning in [Papathanasiou, 2020]. Using neural networks for reinforcement learning together with mentor observations, complex continuous state space can be solved in reduced time when compared to state of the art deep reinforcement learning implementations. According to [Papathanasiou, 2020]'s methods, state similarities are firstly detected between the observer and the mentor, using the mentor state transition demonstrations. Afterwards, the observer can approximate the mentor's optimal action and use it to calculate error values based on newly introduced augmented loss functions. Finally, these values are used to enhance the deep reinforcement learning algorithm (Figure 1.1).

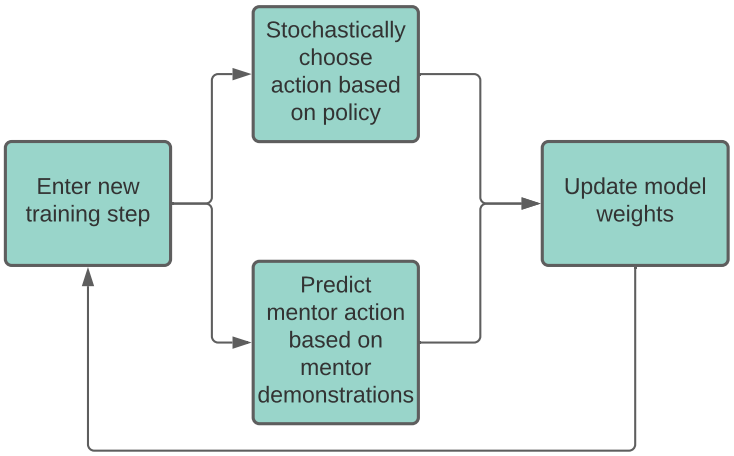

FIGURE 1.1: Observer simple Implicit Imitation depiction

Together with modern reinforcement learning algorithms that benefit from artificial neural networks outstanding results can be noted. We integrate four state of the art deep reinforcement learning algorithms in the imitation process, in order to optimize the imitation performance: Double Deep Q-network [Hasselt, Guez, and Silver, 2016], Prioritized Experience Replay [Schaul et al., 2016], Dueling Network Architecture [Wang et al., 2016] and Parameter Space Noise for Exploration [Plappert et al., 2018]. In this sense we consider these algorithms as imitation *optimization modules* and we will regularly refer to them as such in our work. Double Deep Q-network provides a double Deep Q-network estimator that overcomes value overestimation issues, Prioritized Experience Replay implements an improved buffer which takes into consideration rare state transitions with high error values, Dueling Architecture utilizes advantage learning by creating a reformed model architecture with no extra hidden layers and Parameter Space Noise for Exploration helps to progress the exploration stage by adding Gaussian noise to the network, ending up with a larger variety of state transitions explored, whilst staying within the intended chosen actions boundaries.

Method variance is added in our experiments by enabling and disabling our algorithmic modules, creating different combinations for testing that compete against each other (Figure 1.2). This way, we progressively build better methods which, through our systematic experimentation, end up with a clear winner. The combination of Double Deep Q-network, Prioritized Experience Replay and Dueling Network Architecture outperforms any other optimized model of our work.

1.1 Contributions

The initial work of adapting implicit imitation methods to the Deep Q-Network reinforcement learning is inspired by the work of [Price and Boutilier, 2003]. Following their steps, an implicit deep imitation learning application was later developed by [Papathanasiou, 2020], providing the fundamentals for an accelerated deep reinforcement learning model, by creating augmented loss functions that update neural network's weights in accordance with deep imitation procedures. As expected, the contribution of the four optimization algorithms already mentioned

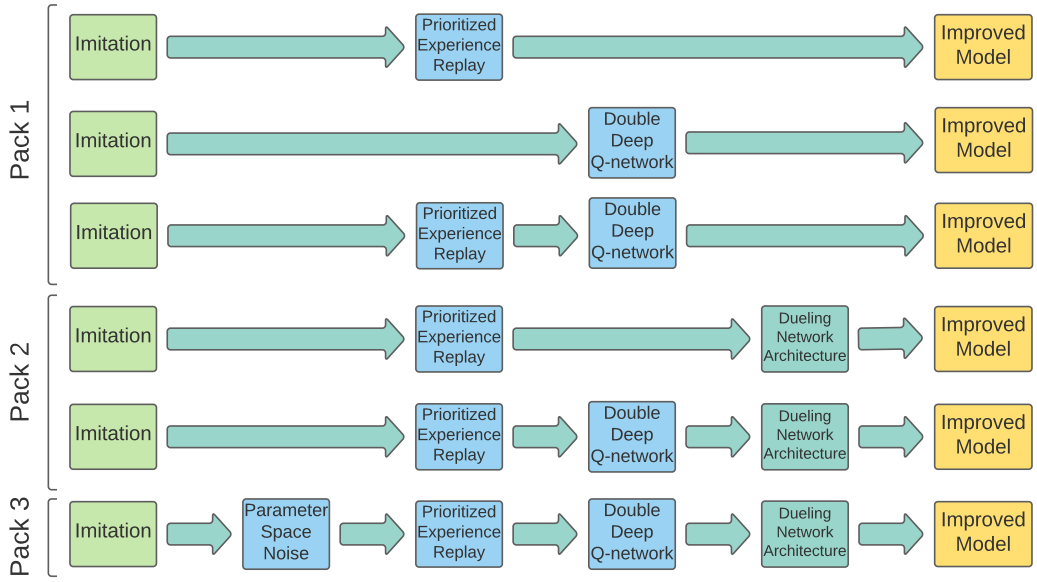

FIGURE 1.2: Module combinations. Imitation refers to the deep imitation algorithm developed in [Papathanasiou, 2020]. Since Double Deep Q-network and Prioritized Experience Replay directly affect the deep imitation method, all possible combinations including them are investigated (**Pack 1**). Dueling Network Architecture is added as a complementary optimization method, resulting to two more combinations (**Pack 2**). Finally, Parameter Space Noise for Exploration underperformed in all combinations, so only its most successful combination is inspected (**Pack 3**).

is to be included. We modify and adapt the first two existing algorithms [Hasselt, Guez, and Silver, 2016; Schaul et al., 2016] so as to make them better suited to our existing model. To be more specific, we modify the augmented loss functions for deep imitation introduced in [Papathanasiou, 2020] to benefit from the Double Deep Q-network algorithm [Hasselt, Guez, and Silver, 2016]. Also, changes are made to the prioritized buffer [Schaul et al., 2016] to better update its weights, based on the error values derived from these augmented loss functions. We, additionally, make use of non modified Dueling Network Architecture [Wang et al., 2016] and Parameter Space Noise for Exploration [Plappert et al., 2018] through adjustments of the neural network's architecture. We note that a plethora of different approaches and algorithms on imitation research that were helpful to conclude to this work's decisions and changes can be found in [Hussein et al., 2017].

1.2 Thesis Structure

This thesis follows a specific structure. In *Chapter 2* we provide the necessary background on reinforcement learning, imitation learning, various optimizing methods, neural networks and deep reinforcement learning. Moving on to *Chapter 3*, we can continue with the work this thesis was based on, [Papathanasiou, 2020]. On that note we can proceed to *Chapter 4* with our four optimization techniques: Double Q-learning, Prioritized Experience Replay, Dueling Network Architecture and Parameter Space Noise for Exploration. Experiments follow afterwards in *Chapter 5*

with the test results gathered and, finally, some suggestions in *Chapter 6* on future work are included.

Chapter 2

Background

In this chapter we provide some essential background on *reinforcement learning*, *neural networks*, *deep reinforcement learning*, *imitation reinforcement learning* and related work required for the explanation of our work.

2.1 Reinforcement Learning

Reinforcement learning is an area of *Machine learning* regarding maximizing given rewards to an agent by carefully choosing actions in an environment. Reinforcement learning can be explained by analyzing one of the famous mouse maze experiments [Tas, 2021] (Figure 2.1). Having a mouse-sized maze with only an entrance and no other exits, we place a massive amount of food at a specific spot far deep into the maze, whilst having small doses of peanut butter at random spots. We let the mouse enter the maze, navigate at its own pace and explore the labyrinth. Its goal is to satisfy its needs by eating the peanut butter on its way, but most importantly to bring the large quantity of food back to its nest, symbolized as the entrance. While at first it will be lost, after numerous trial and error attempts it will learn to guide itself from the entrance to the trophy and back, minimizing the distance required, thus bringing more food at faster timings. From this example we can extract interesting information that will help us correlate this real life experiment scenario with the fundamentals of reinforcement learning. At its core we have the

Problem: task in need of solving, bringing food back to the mouse's home.

Agent: active and intelligent autonomous entity, the mouse.

Environment: world within the agent interacts, the maze.

Actions: different ways of the agent to interact and cause changes to the environment, the movement of the mouse and the ability to pick food.

State: full set of information that depicts the current situation of the environment, the position of the mouse relative to the maze.

Reward: positive or negative value received after following a specific action at a certain state based on the agent's performance. A small positive reward is awarded to the mouse when it satisfies its need of eating small doses of peanut butter, while a greater reward is given when it brings food back to its home. Provided this is timed, the environment encourages the mouse to bring food back faster by following the shortest path discovered. Thus, it is considered higher reward for the mouse to use the optimal path. On the other hand, negative reward is the penalty of time wasted when reaching a dead end or an extensive path.

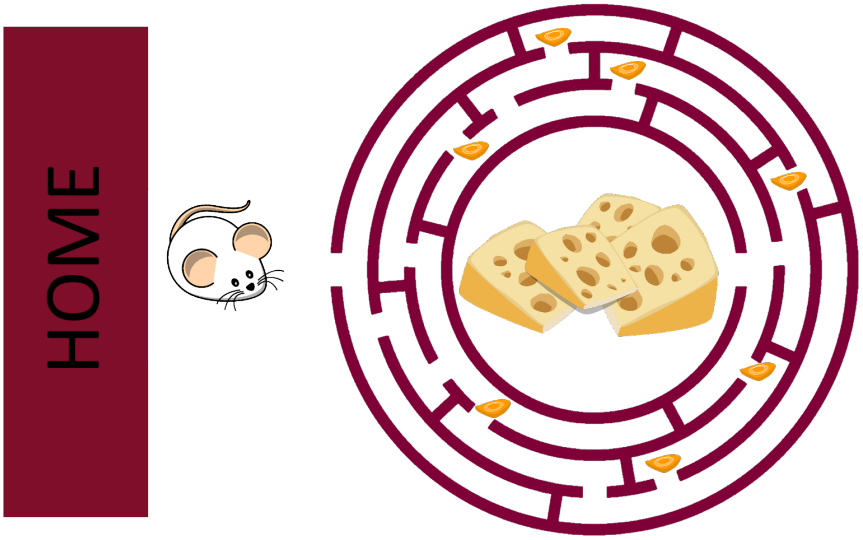

FIGURE 2.1: Mouse maze depiction

Training: procedure of consecutive actions and received rewards that will eventually change the behavior of the agent in order to maximize its performance, the trial and error phase of the mouse.

Exploration: trying random actions in hope of discovering new results with better rewards, the curiosity of the mouse to try new paths.

Exploitation: using the agent's past experience to take the currently most optimal action, the mouse's navigation based on its training.

Goal: ending training by solving the problem, the mouse has minimized the distance required to bring back food.

Policy: behavior of the agent from a certain state, the mouse's strategy of getting more food back.

Such learning environments can be encountered in reinforcement learning algorithms (Figure 2.2), varying from simple two dimensional mazes to complex, real-time, multi-thousand action games, requiring the observance of thousands of values at each step [OpenAI et al., 2019].

One of the the biggest challenges in reinforcement learning is the value selection of exploration and exploitation. While both are needed for the healthy training of an agent, these two concepts are complementary and careful balance should be used [Sutton and Barto, 1998]. Under this predicament $\epsilon - greedy\ exploration$ [François-Lavet et al., 2018] is the most frequently used method. According to it, we operate a hyperparameter ϵ with value that anneals at each time step slowly from almost zero value to one. The agent uses this term to decide its next action a

$$a = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \text{highest estimated value action}, & \text{with probability } 1 - \epsilon \end{cases} \tag{2.1}$$

FIGURE 2.2: **Left:** Simple maze environment. **Right:** Dota 2 video game complex environment

A representation of a reinforcement learning model is complete through the use of a Markov Decision Process (MDP) [Puterman, 1994]. MDP is a discrete-time stochastic control process, providing a structure for partly random environments with action depending changes (Figure 2.3). It is consisted of a tuple $\langle S, A, P, R \rangle$, where S is the state space, A is the action space, P is the stochastic function for state transitions and R is the reward function. Starting from a state $s \in S$, the agent executes an action $a \in A$, transitions to the next state $s' \in S$ based on the probabilistic function P and receives reward r based on the reward function R. Describing the reinforcement learning process as an MDP, we can already find similarities to the core terms of reinforcement learning mentioned earlier.

From the agent's viewpoint, at every possible training step, a policy $\pi : S \longrightarrow A$ is available. This describes the strategy of the agent on certain states. To be more specific, $\pi(s_t) = a_t$ denotes that experience of the learner has guided it to choose at state s_t as optimal action, the action a_t. When every action a deriving from the policy $\pi(s)$ results to the highest cumulative reward, then we consider that policy optimal π^*.

Under a policy π we can define the value function $V^\pi(s)$, as the expected cumulative reward, starting from state s and following the actions provided by the policy π [Sutton and Barto, 1998]. We declare as random variable R

$$R = \sum \gamma^t r_t, \tag{2.2}$$

where at the time step t, we receive reward r_t discounted by $\gamma \in [0, 1)$ factor, based on how immediate the reward is. Distant rewards are considered less important to determine the viability of a state. Having defined the R variable, the value function $V^\pi(s)$ is

$$V^\pi(s) = \mathbb{E}[R|\pi, s]. \tag{2.3}$$

As optimal value function $V^*(s)$ we identify the value function with policy that provides the higher cumulative reward $\max V^\pi(s)$.

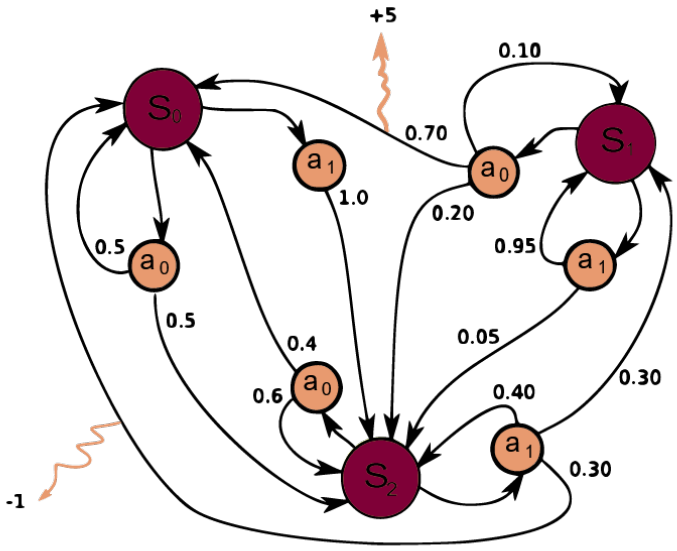

FIGURE 2.3: Markov Decision Process example

However, in order to create space for exploration, we need a way to break out from the actions deriving from the deterministic policy π. We declare as Q-value function [Watkins and Dayan, 1989]

$$Q(s,a) = \mathbb{E}[R|a,\pi,s], \tag{2.4}$$

where a is an action that is not necessarily following the policy π. After executing the action a at the initial state s, the policy determines the rest actions, enabling exploration to be effective. Again, optimal Q-value function Q^* is available when the optimal policy π^* is used.

Since each Q value includes the discounted future rewards, we can rewrite the Q-value function as a Bellman equation

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right), \tag{2.5}$$

where s' is the next state after the initial execution of action a and α is the learning factor, deciding the impact of each update in our Q values, $\alpha = 0$ means that no learning is taking place, while $\alpha = 1$ translates to training with only the latest reward evaluation. In this function we name as *target function* the term $r + \gamma \max_{a'} Q(s',a')$. It contains the maximized future value of our Q-value function, starting from the next state s'. Since we subtract the old value $Q(s,a)$ from it, it is clear that we try to minimize the distance between our Q-value function and our target function. This distance is called *temporal difference (TD)*.

2.1.1 Double Q learning

Previously, we showed that Q-learning tries to calculate the next state value by maximizing over the available actions. This estimation approximates the maximum expected value of the next state. However, during an experiment, numerous occasions of value overestimation can take place, resulting to a performance penalty that will either slow or confuse the training agent. Learning algorithms similar to Q-learning that use single estimator methods have been proven to present such biased estimates [Smith and Winkler, 2006]. In response to that, a double estimator method has been introduced, which, provided with independent and identically distributed samples, can produce unbiased results [Van Hasselt, 2010].

The corresponding integration of the double estimator method into the Q-learning algorithm requires the use of two Q functions instead of one. On every time step of the experiment only one of them is getting updated, selected randomly. Consider the two functions Q^A, Q^B. During the Q^A update, the next state action is predicted normally through the maximization over all the available actions, but the final next state value is calculated by using that action with the Q^B function. Respectively, the Q^B update is being completed the same way. At any given point, if a policy is to be extracted, both functions can be deemed reliable. Also, it is expected that both Q^A and Q^B will converge to the optimal value function Q^* at some point.

Algorithm 1: Double Q-learning

initialization of Q^A, Q^B
while *training* **do**
  Choose stochastically a based on either Q^A or Q^B
  Observe r, s'
  Choose to update either Q^A or Q^B
  if *update* Q^A **then**
    $a' = argmax_a Q^A(s', a)$
    $Q^A(s, a) \leftarrow Q^A(s', a) + \alpha(r + \gamma Q^B(s', a') - Q^A(s, a))$
  else
    $a' = argmax_a Q^B(s', a)$
    $Q^B(s, a) \leftarrow Q^B(s', a) + \alpha(r + \gamma Q^A(s', a') - Q^B(s, a))$
  end
  $s \leftarrow s'$
end

2.1.2 Advantage Learning

It is frequent when choosing an action based on a Q-value function policy to have multiple actions that end up to different states with small differences in Q values. This distinction, responsible for updating the model and converging to the optimal Q-value function, can be easily overlooked or get lost in the noise, resulting to unnecessary and extensive training iterations to reach optimal policy precision [Harmon and Harmon, 2000]. The problem is magnified in cases where a function approximator is used.

Advantage updating [Baird and III, 1993] and its revised version *Advantage learning* [Harmon and Iii, 1998], provide a value function which overcomes this issue in addition to being used for reinforcement learning in continuous time and continuous

state problems. For each combination of state and action (s, a), the advantage value is stored. This value expresses the value difference between executing an action and the current policy suggested action. When investigating the optimal advantage function $A^*(s, a)$, we will receive negative values for every sub-optimal action a and zero value if a is the optimal.

The optimal advantage $A^*(s, a)$ for state s and action a is defined [Harmon and Iii, 1998]

$$A^*(s, a) = V^*(s) + \frac{\langle r + \gamma^{\Delta t} V^*(s') \rangle - V^*(s)}{\Delta t K}, \tag{2.6}$$

where $\gamma^{\Delta t}$ is the discount factor, K is a time unit scaling factor and $\langle\rangle$ is the expected value of executing action a, receiving the reward r and transitioning to the next state s'. As previously explained, the second term is zero for the optimal action and negative for any sub-optimal action.

For the purpose of using the concept of advantage learning in our work, the need of an equation that involves the Q-value function arises. Since the value function based on a stochastic policy π can be defined as

$$V^\pi = \mathbb{E}_{a \sim \pi(s)}[Q^\pi(s, a)], \tag{2.7}$$

then we can define the advantage function under the same policy as [Wang et al., 2016]

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{2.8}$$

Both the expected value $\mathbb{E}_{a \sim \pi(s)}[A^\pi(s, a)]$ and $A(s, a^*)$, where a^* is the optimal action at state s, are equal to zero, while any other sub-optimal actions would result to negative values. To make it more clear, while both the value $V(s)$ and Q-value function $Q(s, a)$ take into consideration future rewards, the value function is strictly tied to the policy we deemed best, whereas the Q-value function can stochastically choose actions that may be out of policy. In that case, the advantage function can evaluate the viability of such policy aberration and can be used with the benefits previously mentioned.

2.2 Neural Networks and Deep Reinforcement Learning

Artificial Neural Networks are networks consisted of artificial neurons which resemble in structure - and named after - biological neural networks [Rumelhart and McClelland, 1987]. They are sectioned into three parts (Figure 2.4). The input layer, to which we feed the available observed information, the hidden layer(s), describing all the layers that try to analyze and process the input information to conclude to a result and, finally, the output layer, the layer depicting an evaluation of a decision it was tasked to make. In order to create different products for diverse input information, weights are added between the connected nodes or neurons, updated during training procedures and finalized to provide stable results. An agent or user can only interact with the input and output layers, hence the name of the hidden ones.

Great feats have been achieved through the carefully built and trained use of neural networks. Some examples are the every day use of e-mail spam filtering [Dada et al., 2019], image recognition [He et al., 2015], autonomous driving [Fan et al., 2018],

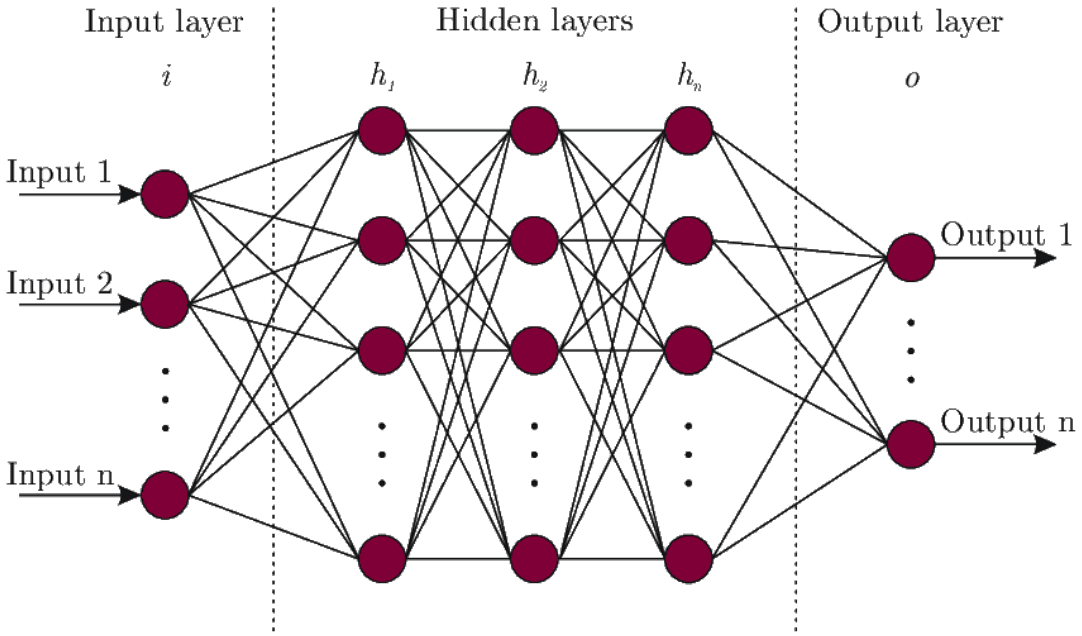

FIGURE 2.4: Neural Network example

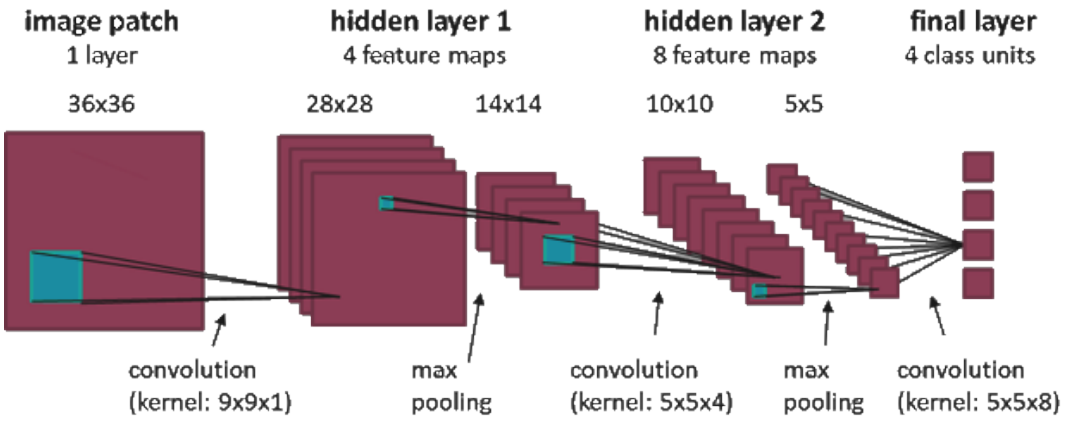

FIGURE 2.5: Convolution Neural Network example

medical advancements [Kononenko, 2001] and complex competitive game environments ranging from backgammon [Wiering, 2010], to chess and Go [Silver et al., 2018].

Convolutional Neural Networks are a variant of artificial neural networks [Lecun et al., 1998], used in deep learning. They are commonly applied in learning cases of image processing, meaning they receive images at their input layer. They are a special case of *multilayer perceptons*, single directional *feedforward*, fully connected between layers artificial networks. The visual analysis is completed by the convolution operation on each layer. Creating different activation maps on each layer, a process of pixel value multiplication by weights and summing takes place, resulting to extracting different feature information with each one of them. For example, the first hidden layer can obtain edges of the image, while the second layer corners, later producing actual object components that are critical for the identification and the decision of the agent.

An essential part of neural networks is the use of function approximation. Function approximation is a method of estimating a function that best describes a target function, by utilizing available demonstrations from the environment [Sutton and Barto, 1998]. There will always be small deviations between the final product of a approximated function and its target function, but we can control the magnitude of such deviations with thresholds. In supervised learning, using a dataset with inputs and outputs, the neural network model is the function approximator that gets fed with input data batches and the target function is the output examples. In reinforcement learning, function approximation is used to estimate the value function or, more commonly, the Q-value function.

Reinforcement learning, as a highly contested area of machine learning, for obvious reasons, exploited neural networks in order to carry out remarkable accomplishments, most notably achieving human level skill or higher on Atari 2600 games [Mnih et al., 2013].

Deep Q-Networks [Mnih et al., 2013], the most distinguished method of deep reinforcement learning, uses an implementation of Q-learning with neural networks (Figure 2.6). We have a Q-network, a neural network with weights θ used for function approximation, deciding the next action taken by the agent. The θ parameters is adjusted by training using as criterion the square divergence of the Q-value from the target value. As target function we declare

$$y = r + \gamma \max Q(s', a'; \theta^{-}), \tag{2.9}$$

where θ^{-} is the network's parameters from a past time step. Occasionally, we update the parameters' values with the most recent ones. This provides a sense of stability, while following the basics of the Q-learning algorithm.

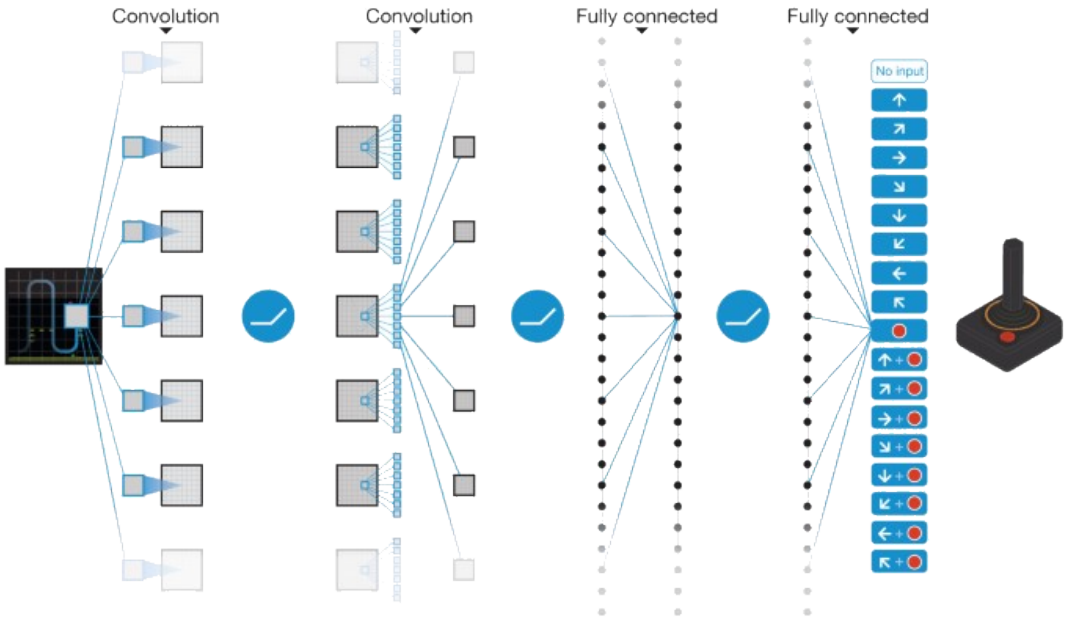

FIGURE 2.6: Deep Q-Network architecture

The loss function deriving from the squared distance is [Mnih et al., 2013]

$$L(\theta_t) = \mathbb{E}_{s,a,r,s'}[(r + \gamma \max_{a'}[Q(s', a', \theta_t^-)] - Q(s, a, \theta_t))^2], \tag{2.10}$$

Another important component is replay memory, used in experience replay method [Lin, 1992]. The replay memory is a data structure of size N that saves past agent experience tuples $\langle s, a, r, s' \rangle$. For every step the current tuple is stored. When the training procedure begins, the agent does not train itself based on this last tuple, but over a small minibatch taken from the replay memory data structure. The distribution function for the selection is uniform, leaving any unintended bias behind. This has strong effect in certain state chains that can misguide the training and pollute it with over-fitting data. By breaking the correlation between consecutive samples, the agent is more confident when it needs to face newly-encountered states.

2.3 Imitation Learning

The concept of transferring knowledge between two or more agents for the benefit of constant improvement is not a newly discovered category [Mataric, 1998]. Its most popular application, *imitation reinforcement learning*, is an area that has received attention during the past twenty-five years [Atkeson and Schaal, 1997]. The core idea behind imitation reinforcement learning includes two agents, the mentor and the observer. The observer is the agent under training, completely clueless about solving a problem, with unknown starting values to all predictions, trying to learn from scratch. The mentor is the agent that has already undergone training up to a satisfactory level, leaving it capable of guiding the observer through different ways [Price and Boutilier, 2003; Stadie, Abbeel, and Sutskever, 2019a]. The goal of imitation learning is to accelerate the learning procedure of the observer, by feeding information provided by the mentor. This insight transmission can be achieved explicitly, implicitly [Price and Boutilier, 2003] or by other means, such us third person inversely [Stadie, Abbeel, and Sutskever, 2019a]. Currently, the first two cases were found promising for further development.

Explicit imitation reinforcement learning [Price and Boutilier, 2003; Bakker and Kuniyoshi, 1996] poses that the observer agent is trying to replicate the behavior of the mentor by following step by step the mentor's actions. The mentor shares both its states, followed actions and rewards, providing enough knowledge for the observer agent to fully imitate its performance. Due to the influence the mentor has over the observer with the instructions provided, *explicit imitation* is categorized as supervised learning. However, in many cases, complete transparency of information is not available.

In *implicit imitation reinforcement learning* [Price and Boutilier, 2003], less insight of the mentor is accessible to the observer. The observer can note only the changes of the mentor's experiment environment, leaving to fill the blanks for actions taken. *Implicit imitation* tries to approximate the mentor's behavior by observing the state transitions and estimating the action needed for these alterations. While on rare occasions mentor actions are transferred, in implicit imitation the observer does not blindly follow the mentor's directions. This results to a more flexible and stable observer agent that hopefully, after its training, will be able to adapt and even outperform the mentor in non visited states.

While both *explicit* and *implicit imitation* are considered viable, we will focus on developing an agent with the use of the latter. In order to define correctly the mentor's shared knowledge, we, firstly, need to make some core assumptions [Price and Boutilier, 2003].

Observability: We assume that the observer has access to the mentor's MDP. In our case that includes the mentor's state transitions vector, but not its actions.

Analogy: We assume that both agents, mentor and observer, use the same local state spaces, otherwise the information received by the learner would be irrelevant and not of any use, thus failing to be helpful. We conclude that $S_m = S_o = S$, removing any distinctions between the state spaces.

Abilities: As in *Analogy*, here we also assume homomorphism between the abilities of the mentor and the observer agent, meaning that they both have the same or highly similar action spaces. If identical actions from common initial states would result to different states, this nullifies the imitation procedure. As homomorphism suggests, the action spaces are assumed equal $A_m = A_o = A$ and the state transitions follow the same MPD for each action $a \in A$. It should be noted that the action spaces can have a weak equality instead, but in this work no such case is studied.

Objectives: No assumptions are made for the reward functions used individually for the two agents. Even though one can infer that the closer they are, the easier it is for the observer to imitate the mentor, no explicit directions are being instructed by the mentor. However, due to the nature of our experiments, identical reward functions are being used for both the mentor and the observer agents.

2.4 OpenAI Gym

The environment of our testing stages, which is OpenAI Gym, includes a plethora of games we can test our algorithm on. OpenAI Gym is a toolkit designed to experiment on different problems with reinforcement learning [Brockman et al., 2016]. It is widely accepted as a robust and stable environment and for this reason it is greatly supported by the community, including complete compatibility with famous python machine learning libraries such as the ones we are using (Tensorflow etc). Combined with the latter, its easy accessibility, open source approach and dependency update makes it a perfect candidate for our work. OpenAI Gym provides a collection of environments that vary in difficulty, ranging from small-scale tasks to classic Atari games and 3D modeled robots. For our work, based on the requirements for each problem, plain classic control games are selected. Through its main interface, we can choose to emulate a game, feeding to it our agent's decisions on action selection, receiving an environment observation together with the reward and training our model in real time. At our agent's pace, we can automate the game reset and continue testing and training our agent's capabilities with a fresh start.

2.5 Related Work

A plethora of other works on imitation are to be noted to encourage improvements and future extensions.

[Hester et al., 2018] introduces an approach of implicit imitation by feeding mentor demonstrations to the observer, making a difference by using a pre-training supervised phase. During that phase mentor demonstrations are used is a supervised manner to imitate its behavior. Its main goal is to start the deep reinforcement learning procedure from a point that limit negative action choices and errors, making it a perfect candidate for real life environment problems.

A different imitation method that includes the influence of a guide is [Hester et al., 2017], taking advantage of human experience combined with a checkpoint system. At each training step batches are sampled from both the agent's and human's experience buffer, resulting to more guaranteed positive rewards. Also, by abusing set checkpoints they can choose to improve specific state transitions. When an episode with poor performance exhibited is terminated, a checkpoint is selected to resume the training procedure instead of the initial state. This discovers more efficiently the state transitions with the highest rewards before the exploration stage is finished.

[Le et al., 2018] introduces a hierarchical technique to the imitation learning application. A two-level hierarchy is used; the HI level is in charge of choosing subtasks and LO level is in charge of executing them. By using behavioural cloning [Ross, Gordon, and Bagnell, 2011], a hybrid setting is created with imitation learning from demonstrations on the HI level and reinforcement learning methods on the LO level.

A unique, interesting approach of imitation learning is [Stadie, Abbeel, and Sutskever, 2019b]. Instead of receiving mentor demonstrations from a first-person point of view, which are not obtainable in many cases, third-person demonstrations are used. Generative adversarial networks are utilized based on the work of [Ho and Ermon, 2016]. Pixel-level demonstrations are observed by the agent and in an unsupervised manner, implicit imitation is achieved. However, it was examined that as the camera angle difference between the agent and the mentor increases, lower performance is exhibited.

[Price and Boutilier, 2003]'s work on accelerating reinforcement learning through implicit imitation is an important related work, considering that [Papathanasiou, 2020]'s work was heavily influenced by it. It introduces a set of assumptions that simplifies and deconstruct the imitation reinforcement learning problem in hand. By introducing Stochastic games (multiagent environments generalization where not individual, but combined agent actions are responsible for the state transitions), observable state transitions can be extracted from each agent. By providing these demonstrations, they constantly improve and help each other through reinforcement learning. Their final goal is to find Nash or approximate equilibrium.

Finally, [Papathanasiou, 2020]'s work, as later explained in detail in section 3, includes methods that we based on to further develop and add optimization improvements. They introduce an implementation of implicit imitation in deep reinforcement learning, by providing mentor demonstrations. At each training step the agent, after finding a similar mentor state transition, tries to approximate the mentor's action on that step. By comparing its own error value with the predicted mentor's action error value and updating the weights according to a confidence testing application, accelerated deep reinforcement learning is achieved.

Chapter 3

Previous Work on Deep Imitation

In order to develop and test our work a major dependency is included. The work of [Papathanasiou, 2020] is the foundation of this project and it includes several techniques we built upon to provide our optimized version.

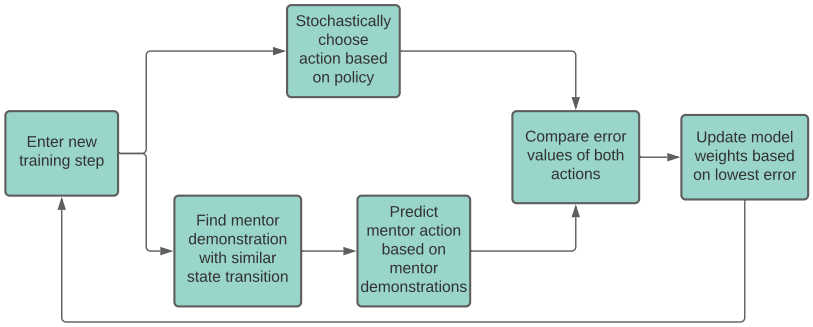

FIGURE 3.1: Observer Implicit Imitation depiction

Since our work is based on [Papathanasiou, 2020]'s work, the thesis explanation would be incomplete if we did not include their additions to the default deep Q-learning model. [Papathanasiou, 2020]'s work, heavily influenced by *Bob Price's* approach on the subject of imitation reinforcement learning [Price and Boutilier, 2003], adds heuristics to correctly adapt to the deep Q-network (Figure 3.1). In this section a description of each step is analyzed to cover the fundamentals, before we move on to our improvements over this model.

3.1 State Transition Extraction

Their first mechanism is the experience extraction from the mentor. The mentor is a normal Deep Q-network model. After training the mentor to its highest potential, they let it complete the experiment repeatedly whilst keeping a log file including every consecutive state transition.

N observations of the form

$$Observation\ Tuple\ 1,$$

$$Observation\ Tuple\ 2,$$

Observation Tuple 3,

. . .

Observation Tuple N.

are stored in a *csv* file, containing tuples of $\langle s_m, s'_m \rangle$, where s_m is the mentor state before it takes the unknown action a_m and s'_m is the state after the action execution.

Multiple mentors are being trained, saving a file from each. Afterwards, when the agent under training - the observer - initializes its parameters, it reads all files, gaining access to all recorded observation tuples and, finally, stores them in a list for future use.

Note that only the state values are being tracked, neither the mentor action a_m or the received reward r_m is known. The assumption used is that the environment rewards have equal values among observer and mentor $r_m = r_o$ states, if they are found to be similar. However, the specification of the mentor action a_m creates new challenges that will be explained in detail in the next subsections.

3.2 State Similarity

In order for the observer agent to make use of the mentor files it needs to correlate its encountered states with the mentor stored ones. In discrete state space environments this is not a problem, since the observer encounters identical states contained in the mentors' demonstrations. However, in continuous state space environments, one cannot expect to encounter the exact same state, due to the large number of dissimilar states (often converging to infinite). This calls for a heuristic that will associate states by similarity approximation.

Denoting as Φ the set of all possible state vectors, they calculate the state divergence $\Delta\phi$ between the observer's state vector $\phi_o \in \Phi$ and mentor's state vector $\phi_m \in \Phi$ using the percentage formula

$$\Delta\phi = \frac{\phi_o - \phi_m}{\phi_m} \cdot 100. \tag{3.1}$$

After enumerating the state difference, they judge if the states are similar by using a similarity threshold $\Delta\phi^*$

$$|\Delta\phi| < \Delta\phi^*. \tag{3.2}$$

If the absolute divergence value is lower than the threshold hyperparameter, they recognize the two states as similar.

For every visited observer state, they parse through the entire saved mentor state list and individually check for mentor states that are comparable. This way, for every training step t a completely new similarity list is being filled and a random mentor state entry is picked for the rest of this step. The random selection ensures that favoring of specific states is avoided, restraining from early overtraining problems.

The threshold hyperparameter $\Delta\phi^*$ value differs for various environments. It can be selected through experimentation, but one should keep in mind that the higher its value, the higher the chance of polluting the training procedure with negatively affecting mentor states.

3.3 Mentor Action Prediction

Under most circumstances the unknown mentor action a_m is the most rewarding action for the observer when it is on a similar mentor state. Now that they potentially have a mentor state s_m that is closely similar to the observer state s_o, a process of predicting the mentor action a_m engages.

When the similarity list is non-empty, after taking a step executing the observer action a_o selected based on the exploration policy, they compare the next states s'_o and s'_m for similarities. If the threshold is satisfied they conclude to assigning the observer action a_o to the predicted mentor action a_m. Otherwise, they duplicate the current environment and by brute-forcing all possible actions from the action space A, they try to find which action would lead to the next state being s'_m. In other words, they enable an emulation trying to find actions that will provide ending states similar to the mentor one. Finally, if the similarity list is empty, a random action is picked for the mentor action a_m reinforcing the stochastic part of the exploration policy.

Algorithm 2: Predicting the mentor action

```
if similar state s_m exists then
    if Δφ_{i+1,a_o} < Δφ* then
        a_m = a_o
    else
        Backup environment
        for a in A do
            Access backup point
            Emulate environment step with action a
            if Δφ_{i+1,a} < Δφ* then
                a_m = a
            end
        end
    end
else
    Assign random a_m from action space A
end
return a_m
```

3.4 Augmented Loss Functions

After specifying all parts of the mentor's tuple $\langle s_m, a_m, r_m, s'_m \rangle$ is known, the observer can identify the Q-value function from its viewpoint as

$$Q_m(s_m, a_m) = (1-a)Q_m(s_m, a_m) + a\left(r_m + \gamma \max\left[\max_a[Q_m(s'_m, a)], Q_m(s'_m, a_m)\right]\right), \tag{3.3}$$

where s_m is the similar state to the observer's encountered state s_o, a_m is the predicted mentor action, r_m is equal to the observer's reward r_o and s'_m is the next step's similar state of observer's s'_o.

As noticed, since the minimum value is equal to $Q_m(s'_m, a_m)$ the corresponding bellman equation for the observer itself is described as

$$Q_o(s_o, a_o) = (1-a)Q_o(s_o, a_o) + a\left(r_o + \gamma \max\left[\max_a[Q_o(s'_o, a)], Q_m(s'_m, a_m)\right]\right). \tag{3.4}$$

In order to carry out these changes to the DQN architecture they need to create new and updated loss functions. Based on the default loss function

$$L(\theta_t) = \mathbb{E}_{s,a,r,s'}[(r + \gamma \max_{a'}[Q(s', a', \theta_t^-)] - Q(s, a, \theta_t))^2], \tag{3.5}$$

they can create augmented loss functions for both the mentor and observer, from the observer's perspective. The observer augmented loss function can be constructed as

$$\begin{aligned} L(\theta_t)_o = \mathbb{E}_{s,a,r,s'}[(r_o + \gamma \max\{Q(s'_o, \operatorname*{argmax}_{a'} Q(s'_o, a'; \theta_t^-); \theta_t^-), Q(s'_o, a_m; \theta_t^-)\} \\ - Q(s_o, a_o; \theta_t))^2], \end{aligned} \tag{3.6}$$

while the mentor augmented loss function as

$$\begin{aligned} L(\theta_t)_m = \mathbb{E}_{s,a,r,s'}[(r_o + \gamma \max\{Q(s'_m, \operatorname*{argmax}_{a'} Q(s'_m, a'; \theta_t^-); \theta_t^-), Q(s'_m, a_m; \theta_t^-)\} \\ - Q(s_m, a_m; \theta_t))^2]. \end{aligned} \tag{3.7}$$

One can notice that the two terms within the max function $Q(s'_m, \text{argmax}_{a'} Q(s'_m, a'; \theta_t^-); \theta_t^-)$ and $Q(s'_m, a_m; \theta_t^-)$ are correlated, since the second term is strictly equal or lower than the first term. More specifically, since the mentor action a_m is a subset of the *argmax* function, the final Q-function value of the mentor action will be equal to the Q-function value of the *argmax* when $a' = a_m$ and lower otherwise. This distinction helped [Papathanasiou, 2020] create intuitively the bellman equations and loss functions, but it only provides unnecessary complexity onwards. However, even though

it should be removed from the augmented loss function (eq. 3.6, eq. 3.7), in this thesis it is decided to be kept within the augmented loss functions' formulas, because on later improvements over the algorithm, these functions will be partially changed, resulting to ending the property of the strictly equal or lower relationship.

3.5 Confidence Testing

Having two different loss functions creates the need for a decision making condition selecting the one minimizing the model. For each training step they store the values Q_o and Q_m for future use. Now, they can compare past Q-function values with present ones. This divergence is the deciding factor on what augmented loss function they use. More specifically, using square distance, two terms are calculated on each step

$$D_o = (Q_o - Q_{o,past})^2, \tag{3.8}$$

$$D_m = (Q_m - Q_{m,past})^2. \tag{3.9}$$

By comparing these two values, they pick the augmented loss function associated with the minimum distance. Since having close Q values during different states translates into a more stable model, it also follows that it converges to the optimal policy Q-function.

A distance estimator alternative is the absolute difference

$$D_o = |Q_o - Q_{o,past}|, \tag{3.10}$$

$$D_m = |Q_m - Q_{m,past}|. \tag{3.11}$$

However, they concluded to using the square difference, as it detects more effectively small differences in values.

Chapter 4

Enhancing Deep Imitation

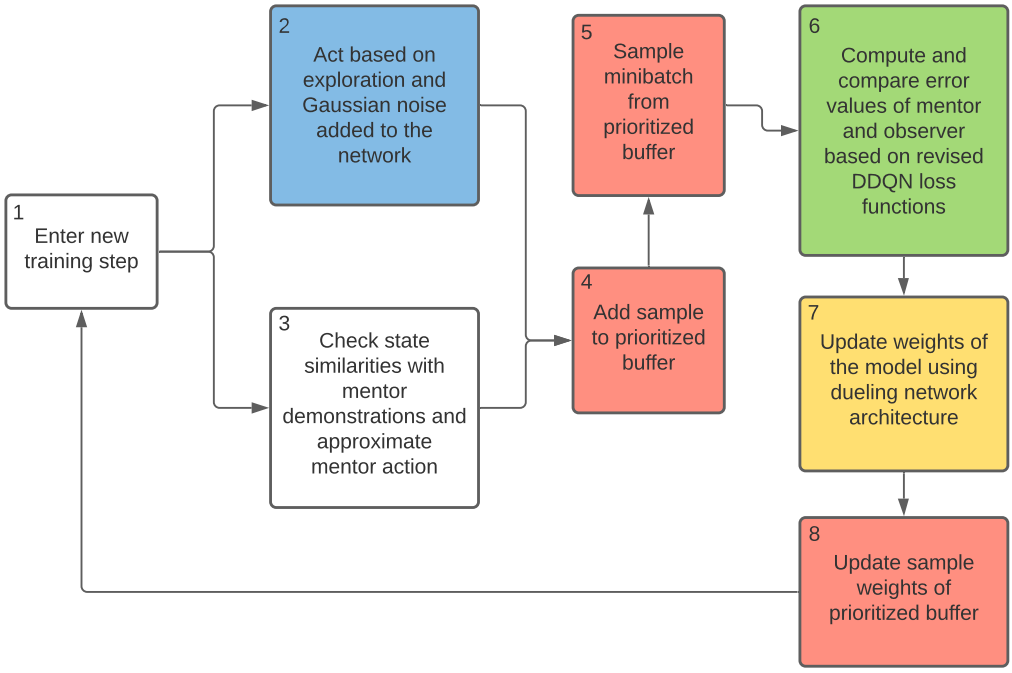

FIGURE 4.1: Training step flowchart. Step(s) labeled as: **6** is part of Double Deep Q-network, **4-5-8** are part of Prioritized Experience Replay, **7** is part of Dueling Network Architecture, **2** is part of Parameter Space Noise for Exploration

Even though [Papathanasiou, 2020]'s implementation of deep imitation provides superior results over the default deep network, we can expect to achieve even better results through further optimization. In this section we will explain the work and the main subject of this thesis, including four different optimization algorithmic changes that try to improve the existing deep imitation network. Our benchmark is the DQN agent with added imitation methods of [Papathanasiou, 2020]. We start with the addition of *Double Deep Q-learning* that removes value overestimation problems created by single DQN estimators. Afterwards, the use of *Prioritized Experience Buffer* is described, which changes the buffer sample selection by deterministically favoring the samples with the highest error values, ensuring no rare occasions of specific states will be neglected. Following that, the *Dueling Architecture* is analyzed improving the architecture of the model without adding extra hidden layers, whilst keeping

the benefits of advantage learning. Finally, *Parameter Space Noise Exploration* is introduced, advancing the exploration stage of the training by adding Gaussian noise to the network, resulting to sporadically acting slightly different, but not far from the stochastically chosen actions.

As far as implementation is concerned two of those optimization modules need to be changed in order to get integrated to the deep imitation work [Papathanasiou, 2020]. More specifically, to implement Double Deep Q-network, modifications are expected to the augmented loss functions (see eq. 3.7, 3.6). Moreover, in Prioritized Experience Buffer, since weight updating is directly dependent on each training step's estimated error values, i.e., the augmented loss functions, additional algorithmic steps are needed. However, as Dueling Network Architecture and Parameter Space Noise for Exploration make changes to the neural network, which do not explicitly interact with the deep imitation methods, no algorithmic adjustments are required for their implementation.

4.1 Deep Reinforcement Learning with Double Q-learning

As previously discussed, Q-learning suffers from overestimation problems, negatively affecting performance. Even though optimism can be used in favor of the explorer to bring desired results [L. P. Kaelbling, 1996], it is often not the case when the estimations are not distributed correctly and are not focused on specific states.

This issue is, also, transferred to the Deep Q-learning or DQN implementation [Hasselt, Guez, and Silver, 2016], making space for our first algorithmic improvement over the existing model. Following the original $Double\ Q-learning$ algorithm [Van Hasselt, 2010], a Deep Q-learning application of it, named Double Deep Q-learning or DDQN, was proposed and tested on Atari 2600 environments [Hasselt, Guez, and Silver, 2016]. Following the Double Q-learning algorithm, its application includes two different Q-networks taking turns on updating

$$y_t^{DDQNforA} = r_t + \gamma Q(s'_t, \underset{a}{\operatorname{argmax}}\, Q(s'_t, a; \theta_t^A); \theta_t^B). \tag{4.1}$$

In this case we have two models, model A and model B, built with the same initial architecture, but updated with different values. Notice that after deciding which action is currently the optimal by using model A - with set of weights θ_t^A - prediction, we use the predictions of model B's - with set of weights θ_t^B - to update our model A. Interchanging the terms θ_t^A and θ_t^B symmetrically updates the second model B.

However, such approach can prove to be excessive, complex and time consuming considering the fact that we would need to update two different set of weights. Since our architecture has already two different models available - the constantly updated Q-network and the occasionally updated target Q-network - we can replace the second Q function model with the target Q-network and avoid creating a completely new Q function model from scratch. We thus change the updating step of 4.1 to [Hasselt, Guez, and Silver, 2016]

$$y_t^{DDQN} = r_t + \gamma Q(s'_t, \underset{a}{\operatorname{argmax}}\, Q(s'_t, a; \theta_t); \theta_t^-). \tag{4.2}$$

As noted, after identifying the action with the currently higher Q function value using a set of weights θ_t, we choose to use the target network value with set of weights

θ_t^-, provided with that action, to update our current Q function model. Since the target model is a deep copy of our model which gets refreshed every N steps, it simulates the parallel update that takes place in normal Double Q-learning.

Taking into consideration these adjustments, our loss functions should change their forms to include them. We substitute the selection of maximum target Q-model value with the target Q-model value that uses the action providing the maximum Q function model value

$$L(\theta_t)_o = \mathbb{E}_{s,a,r,s'}[(r_o + \gamma \max\{Q(s'_o, \underset{a'}{\mathbf{argmax}}\, \boldsymbol{Q(s'_o, a'; \theta_t)}; \theta_t^-), Q(s'_o, a_m; \theta_t^-)\} - Q(s_o, a_o; \theta_t))^2], \tag{4.3}$$

$$L(\theta_t)_m = \mathbb{E}_{s,a,r,s'}[(r_o + \gamma \max\{Q(s'_m, \underset{a'}{\mathbf{argmax}}\, \boldsymbol{Q(s'_m, a'; \theta_t)}; \theta_t^-), Q(s'_m, a_m; \theta_t^-)\} - Q(s_m, a_m; \theta_t))^2]. \tag{4.4}$$

Note that in both loss functions the target Q-function value with the predicted mentor action a_m is not strictly lower than the first term included in the max function anymore. The chosen action a' is the currently best action for our Q-function, but not necessarily for our target Q-function as well, meaning that the mentor action a_m could be a better alternative.

4.2 Prioritized Experience Replay

The use of an experience replay method is characterized as vital for the smooth conduct of a deep reinforcement learning model. By training the model with simultaneous old and newly encountered transitions, we can stabilize the training procedure. However, a case can be made for favoring specific transitions by prioritizing them correctly. This may be required since certain transitions can be deemed unnecessary or not valuable enough to spare a training step on them. By prioritizing the samples we pick, we can accelerate our model's training and reach convergence earlier than expected.

Moving on to the implementation of such a method, an initial thought would propose to choose our minibatch of samples greedily. In that case we would prioritize our samples based on their TD-error. After parsing through a large amount of transitions and estimating their initial TD-error, which would also be their criterion for their spot in the replay memory, we would choose to replay a small amount of high priority transitions. High priority is associated with high TD-error value. That raises the question "what will happen to the transitions with low TD-error values?". Since these cases would not get investigated, it is highly likely that the model would suffer from missing essential training samples, resulting to never ending high TD-error transitions and possibly never visiting the key transitions of mediocre or low TD-error values.

A solution to this problem is a stochastic sampling approach. According to it, based on the magnitude of the error, each sample is assigned a probability. High TD-error

samples are more likely to be chosen for replay, while at the same time low TD-error samples have non-zero chances to be picked. This guarantees a smoother and more balanced selection of samples, favoring the problematic samples of high TD-error values. This method can be assigned somewhere between the questionable greedy approach and the default uniform random approach [Lin, 1992] used in deep reinforcement learning [Mnih et al., 2013].

Firstly, we need to specify the priority term. Since we base our priority of TD-error value, priority of transition i can be defined as [Schaul et al., 2016]

$$p_i = |\delta_i| + \epsilon, \tag{4.5}$$

where ϵ is a small constant that avoids zero value priority and guarantees positive probability. Now that we have our mean of prioritization we can move to the definition of the probability of sampling transition i [Schaul et al., 2016]

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{4.6}$$

where α denotes the priority highlight we choose to use. Higher value of α translates to magnification of the priority value, increasing the favoring of high TD-error value transitions. Note that when α has a zero value, the probability function is identical to the uniform one. The higher its value, the further away we are from the uniform distribution.

However, sampling based solely on the probability function we created would result to a terribly biased estimator leaning towards high priority samples. This means that our network will initially improve at high speed, but end up overfitting really fast. In order to reduce this bias the use of *importance sampling* is needed [Mahmood, Hasselt, and Sutton, 2014]. Our initial sample distribution is the probability function based on priority. By smoothing the values used to update our model's set of weights, we limit the bias created by this sampling method. Intuitively, we try to train on high priority samples while removing part of the value's magnitude, since there is a high chance we will encounter them again during a future sampling session. At the same time, when a low priority sample is encountered, we try to fully train on it as it is probable that we will never encounter it again. The weight function for transition i introduced for this purpose is defined as [Schaul et al., 2016]

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta. \tag{4.7}$$

The term $\frac{1}{P(i)}$ is mainly used to correct this bias, although it can end up with obscene values, increasing our gradient magnitudes. For this reason, we normalize it by multiplying it with $\frac{1}{N}$. Using this term avoids further adjustment for our hyperparameters, which is essential for this project, since we introduce numerous new hyperparameters that would need tweaking just to adjust to importance sampling. The β parameter is used to control the amount of prioritization we apply. Since, at earlier stages the process is considered unstable due to how quickly and with what magnitude the model's parameters change, the importance sampling bias correction is mainly needed when proceeding to the end of the training. In order to make full

use of this observation the β hyperparameter starts at a low value and linearly anneals to one. Finally, we downscale the weight value to range $(0, 1)$ by dividing with $\max_i w_i$.

Whenever we store a new sample, we assign it the maximum priority value currently available, providing high chances to be sampled. To avoid excessive computation effort we only calculate and update values of samples when they are picked by our distribution.

When requested for a minibatch sample, the replay memory initially samples the requested transitions based on the probability function, afterwards it calculates the importance weights as described above, then it computes the TD-error value using it to update the priority value of the transition and, lastly, it estimates the final weight change (algorithm 3).

Algorithm 3: Sampling with Prioritized Experience Replay and Deep Imitation

1 Store transition t with maximum priority $p_t = max_i p_i$
2 **for** $j = 1$ *to minibatch size* m **do**
3 Sample transition j based on distribution $P(j) = p_j^a / \sum_i p_i^a$
4 Calculate importance sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
5 Compute mentor error value δ_m using mentor augmented loss function $L(\theta_t)_m$
6 Compute observer error value δ_o using observer augmented loss function $L(\theta_t)_o$
7 Calculate square distance of confidence testing mentor and values D_o, D_m
8 Assign error value δ_j to δ_m or δ_o based on the confidence testing results
9 Update priority value $p_j = |\delta_j| + \epsilon$
10 Estimate model's weight change $\Delta_j = w_j \cdot \delta_j$
11 **end**
12 Optimize model's set of weights using Δ

It should be highlighted that with the implementation of deep imitation, extra steps are added in the process. To be more precise, the TD-error value δ_j, included in the original Prioritized Experience Replay algorithm [Schaul et al., 2016], is now dependent on our confidence testing procedure (see line 7). After we calculate the mentor and observer error values based on revised loss functions for deep imitation (see section 3.4), the confidence testing step takes place (see section 3.5). According to it we can conclude to a single error value (either mentor δ_m or observer δ_o, see lines 5, 6) that will be responsible to update the network's weights. It is tested that with each minibatch created for training, a mixed sample variety is observed. In earlier stages of the training procedure more mentor samples are included, since on average they provide more stable Q-value differences between state transitions in comparison to the random action state transitions taken by the observer. In later stages a balanced sampling variety is detected, since the observer is converging to the optimal policy. Considering that the mentor has already converged to a similar optimal policy, small differences in error values between mentor and observer are to be expected.

4.3 Dueling Network Architecture

Advantage learning, introduced in section 2.1.2, can yield good results if implemented correctly. However, the original algorithm application would require a complete make over of our architecture, negating previous work and the state-of-the-art research. As suggested, we have to involve the advantage learning algorithm with a Q-value function model.

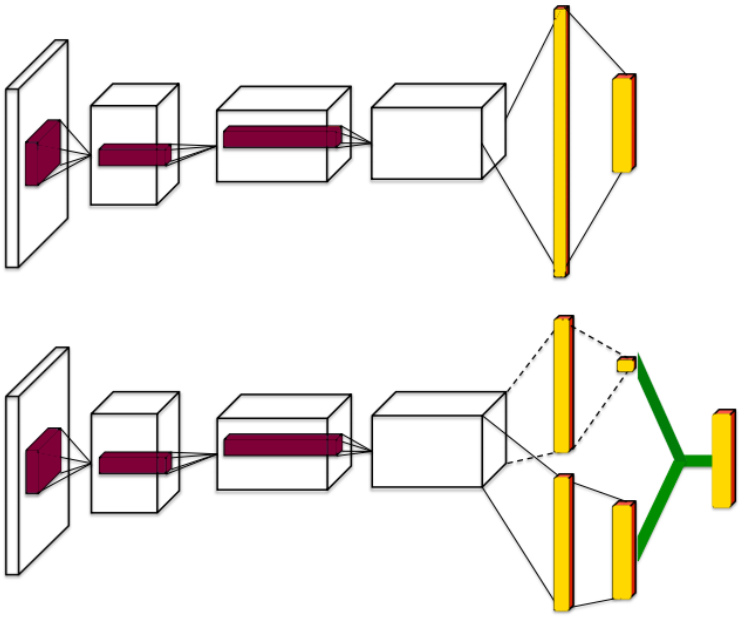

FIGURE 4.2: Comparison of the single stream Q-network (**top**) and the dueling Q-function network (**bottom**)

Following the work of [Wang et al., 2016], we can isolate Q^π from

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s), \tag{4.8}$$

to get

$$Q^\pi(s,a) = V^\pi(s) + A^\pi(s,a), \tag{4.9}$$

where for $V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}[Q^\pi(s,a)]$, we can conclude that $\mathbb{E}_{a\sim\pi(s)}[A^\pi(s,a)] = 0$. Also, when following a policy with optimal action $a^* = \text{argmax}_{a'}$, then $Q(s,a^*) = V(s)$, resulting to $A(s,a^*) = 0$.

Based on this equation, we can modify our current network model so that it calculates both the state-value $V(s)$ and the advantage function $A(s,a)$ to provide an outcome of Q-value $Q(s,a)$. With this method, we isolate the algorithmic changes inside one layer of our model, whilst including the advantage function and its benefits to our project. Instead of driving the hidden layers to our output layer that is our Q-values, we redirect the output to two parallel different models, $V(s;\theta,\beta)$ and $A(s,a;\theta,\alpha)$ [Wang et al., 2016]. The θ refers to the set of weights of our model up to that point, the β refers to the set of weights used for the value-state connected layer and the α to the set of weights used for the advantage connected layer. To receive

the desired results of Q-function all we have to do is add the value outputs of both layers. Note that the value-state function $V(s)$ output consists of single numeric value, whereas the advantage function $A(s,a)$ output is a $|D|$-dimensional vector, where D is the set of available actions. Adding these values together grants the desired $|D|$-dimensional Q-function value vector. A clear visualisation can be seen in (Figure 4.2). Our new statement would look like

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha). \tag{4.10}$$

Even though this is a logical final statement, a major flaw emerges when put to practice. According to our way of thinking the value-state function $V(s)$ and the advantage function $A(s,a)$ would provide accurate actual values. However, there is not a distinction between those two parallel models, meaning we cannot expect the overall model to guess and identify individually the correct values for each submodel. In order to accomplish that we have to guide the models with an additional step. With the optimal action $a^* = \text{argmax}_{a'} Q(s,a';\theta,\alpha,\beta) = \text{argmax}_{a'} A(s,a';\theta,\alpha)$ we can force the output $Q(s,a^*;\theta,\alpha,\beta) = V(s;\theta,\beta)$ by eliminating the advantage function estimator [Wang et al., 2016]

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'} A(s,a';\theta,\alpha) \right). \tag{4.11}$$

Here, for the action a^* the second term is equal to zero, hence the advantage function estimator elimination. With this special case as a guide, the model is trained correctly so it can evaluate correctly both the value-state function $V(s)$ and the advantage function $A(s,a)$.

Even though this formula is valid and can successfully train a model that benefits from the advantage function, an alternative, more stable one has been suggested and proved to be more effective [Wang et al., 2016]. Instead of subtracting the maximum value, we can subtract the average value. This will result to a small constant offset, but now the advantage has to catch up to the mean and not the optimal maximum value, which converges to a faster training

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|D|} \sum_{a'} A(s,a';\theta,\alpha) \right). \tag{4.12}$$

As previously mentioned, no changes are needed for the Dueling Network Architecture algorithm to work with the deep imitation methods [Papathanasiou, 2020]. The only existing common ground between deep imitation and Dueling Network Architecture is the chosen error value from the augmented loss functions. However, since the error value is just a variable input of the neural network, its structure and additional modifications, namely, Dueling Network Architecture are integrated with no adjustments.

At this point it should be highlighted that the addition of the dueling network architecture is an attempt to simulate and integrate the advantage function into a model. It does not follow any of the advantage's actual algorithmic steps, it only trains the model based on some ground rules that derive from equations involving the advantage function. The value identification for both value-state function $V(s)$ and

advantage function $A(s)$ is completed automatically through the network's back propagation with no algorithmic additions.

4.4 Parameter Space Noise for Exploration

The final optimization step introduces changes to the exploration stage of the agent. Multiple studies have tried to improve the exploration through numerous methods [Rückstieß, Felder, and Schmidhuber, 2008; Salimans et al., 2017] with positive results. However, the need for an exploration advancement that is not on-policy exclusive and includes temporal information management was recently satisfied with very good results [Plappert et al., 2018]. Its main premise is to enrich the exploration phase of the training by adding Gaussian noise directly to the parameters of the neural network. Through this addition a bigger variety of consecutive transitions will be encountered, ending up with a more fulfilling exploration development.

Denoting with θ the variables of our network, at the start of every episode we add Gaussian noise to create the perturbed network with variables [Plappert et al., 2018]

$$\tilde{\theta} = \theta + \mathcal{N}(0, \sigma^2), \tag{4.13}$$

where $\mathcal{N}$ is the Gaussian noise distribution, with mean $\mu = 0$ and variance σ^2. The mean is equal to zero for obvious reasons, we do not want to permanently offset the parameters on average. We symbolize as $\tilde{\pi} = \pi_{\tilde{\theta}}$ the policy deriving from the perturbed network and as $\pi = \pi_\theta$ the original policy.

The first challenge comes with the value selection of the variance σ. Using a constant value η is not a viable option considering that the noise will not be able to keep up with the changes made over the network. Instead of a constant, the use of a scale for σ is suggested. That way, after providing an initial value, we can adapt the Gaussian noise addition to the dynamic network. Initializing the variance σ with a really small value, we can continue to update that value at every episode by multiplying or dividing with a constant [Plappert et al., 2018; Ranganathan, 2004]

$$\sigma_{t+1} = \begin{cases} \kappa \cdot \sigma_t, & \text{if } d(\pi, \tilde{\pi}) \leq \zeta \\ \frac{1}{2\kappa} \cdot \sigma_t, & \text{otherwise.} \end{cases} \tag{4.14}$$

Originally, the division suggested by the author was with κ, but through experimentation we produced superior results with 2κ. The $d(\cdot, \cdot)$ function represents the difference between the two networks, the original and the perturbed, while ζ is a threshold factor that judges whether the noise exceeds the desired boundaries. When the difference between the models excels ζ we try limiting the noise by dividing the variance by the 2κ factor. On the alternative case we reinforce the noise by multiplying with the κ value.

The next goal is to define the difference function $d(\cdot, \cdot)$. Trying to make a numerical representation of the dissimilarities between the two networks Q and $\tilde{Q}$ by directly subtracting its values is too unstable to rely on. Small changes could still lead to the same policies, whereas large value gaps could trick the algorithm and exceed the threshold ζ when such magnification would not be relevant. The need for a normalized formulation of the networks arises. Consequently, one can design a probabilistic

representation of the original and perturbed networks by administering the softmax function over the output of the networks

$$P_i(s) = \frac{e^{Q_i(s)}}{\sum_j e^{Q_j(s)}}, \tag{4.15}$$

$$\tilde{P}_i(s) = \frac{e^{\tilde{Q}_i(s)}}{\sum_j e^{\tilde{Q}_j(s)}}. \tag{4.16}$$

where $Q_i(s)$ is the Q value of action i with input state s. The final probabilistic functions $P(s)$ and $\tilde{P}(s)$ are vectors of dimension $|A|$, where $|A|$ denotes the number of actions available. Since both functions are probabilistic, they individually both sum to one. Now that we have a normalized formulation of the networks, we can use Kullback-Leiber divergence [Kullback, 1959] to measure the differences between them

$$D_{KL}(P||\tilde{P}) = \sum_s P(s) \cdot \log\left(\frac{P(s)}{\tilde{P}(s)}\right), \tag{4.17}$$

granting us the final definition of the difference function $d(\cdot,\cdot)$ [Plappert et al., 2018]

$$d(\pi, \tilde{\pi}) = D_{KL}(P||\tilde{P}). \tag{4.18}$$

Now that we can enumerate the distance between the two networks, our final objective is to specify the noise threshold ζ. In order to avoid the use of another hyperparameter, we can associate this term with exploration-greedy action space noise. According to pure-greedy policy $\pi_g(s,a)$ with

$$\pi_g(s,a) = \begin{cases} 1, & \text{if } a = \text{argmax}_{a'}\, Q(s,a') \\ 0, & \text{otherwise,} \end{cases} \tag{4.19}$$

and exploration-greedy policy $\pi_\epsilon(s,a)$ with

$$\pi_\epsilon(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|}, & \text{if } a = \text{argmax}_{a'}\, Q(s,a') \\ \frac{\epsilon}{|A|}, & \text{otherwise,} \end{cases} \tag{4.20}$$

where ϵ is the exploration value of our current training step, one can use Kullback-Leiber divergence to create a representation of the action space dissimilarities between the two policies

$$D(\pi_g||\pi_\epsilon) = -\log\left(1 - \epsilon + \frac{\epsilon}{|A|}\right). \tag{4.21}$$

Correlating this diversity to the outcome of the difference function $d(\pi, \tilde{\pi})$ 4.18, we can use the action space noise divergence to define the parameter space noise threshold ζ term [Plappert et al., 2018]

$$\zeta = -\log\left(1 - \epsilon + \frac{\epsilon}{|A|}\right). \tag{4.22}$$

Note that for this optimization improvement to be performed, when we deterministically predict an action, this action is input by the perturbed $\tilde{Q}$-network (see algorithm 4). Also, as seen in Figure 4.1, the predicted mentor action procedure is unrelated to this step, since we do not receive such actions from our network. This means that the inclusion of Parameter Space Noise for Exploration in our work does not directly affect the procedures of deep imitation, but, nevertheless, further optimizes the model.

Algorithm 4: Choosing an action with parameter space noise [Plappert et al., 2018]

while *training* **do**
  Calculate $\zeta = -\log\left(1 - \epsilon + \frac{\epsilon}{|A|}\right)$
  Receive prediction from the unmodified Q-network $q_{pred} = Q(s)$
  Receive prediction from the perturbed Q-network $\tilde{q}_{pred} = \tilde{Q}(s)$
  if *start of episode* **then**
    for *each variable in* $\tilde{Q}$*-network* **do**
      Add Gaussian noise $variable = variable + \mathcal{N}(0, \sigma^2)$
    end
  end
  Calculate the Kullback-Leiber divergence $KL = d(\pi, \tilde{\pi})$
  if $KL < \zeta$ **then**
    Update $\sigma = \kappa\sigma$
  else
    Update $\sigma = \frac{1}{2\kappa}\sigma$
  end
  Predict deterministic action a_d based on $\tilde{q}_{pred}$
  Choose random action a_r
  return action a based on exploration ϵ, either a_d or a_r
end

Chapter 5

Experimental Evaluation

In this section we test our optimization improvements over selected environments. We will start by describing the games' functionalities and how our agent can interact with them. Afterwards, for each game separately, we will move on to the results evaluation. We remind that our mentor is a normal Deep Q-network application and our benchmark learning curve is the Deep Q-network with imitation methods applied, named as vanilla trainee in our experiments. A variety of different combinations was tested (Figure 5.1) ending up with three sets of outcomes that will be explained. It should be noted that since Dueling Network Architecture is used as a complementary method, providing small but stable improvements in our results, it is not tested on its own. We examine combinations that encourage the enabling and disabling of our Double Deep Q-network and Prioritized Experience Replay optimization modules.

With that in mind, in our first setting the mentor and our benchmark agent are included together with the Prioritized Experience Replay and Prioritized Experience Replay with Dueling Network Architecture agent variants. As expected, in this set, the Prioritized Experience Replay with Dueling Network Architecture combination stands on top on both environments. In our next setting, we enable Double Deep Q-network creating three more combinations: Double Deep Q-network, Double Deep Q-network with Prioritized Experience Replay, Double Deep Q-network with Prioritized Experience Replay and Dueling Network Architecture. They are compared to the highest performance agent from the previous set, the Prioritized Experience Replay with Dueling Network Architecture agent, resulting to our overall winner being the combination of Double Deep Q-network with Prioritized Experience Replay and Dueling Network Architecture (see highlighted combination in Figure 5.1). Finally, due to poor performance exhibited by the addition of the Parameter Space Noise for Exploration module, in our third setting we compare the mentioned winner agent with the Parameter Space Noise for Exploration, Double Deep Q-network, Prioritized Experience Replay, Dueling Network Architecture agent and comment on the reasons we suspect Parameter Space Noise for Exploration underperformed.

5.1 Description of Experimental Settings

Starting with a basic and simple game environment, our first selected environment from OpenAI Gym is 2D-maze-v0 (Figure 5.2). The agent, depicted as a blue ball, tries to find its way from the entry point (blue square) to the goal (red square). Main objective is to find the shortest path. Actions include the four direction keys *go up*, *go down*, *go left*, *go right* and the observation space is just the coordinates of the agent in

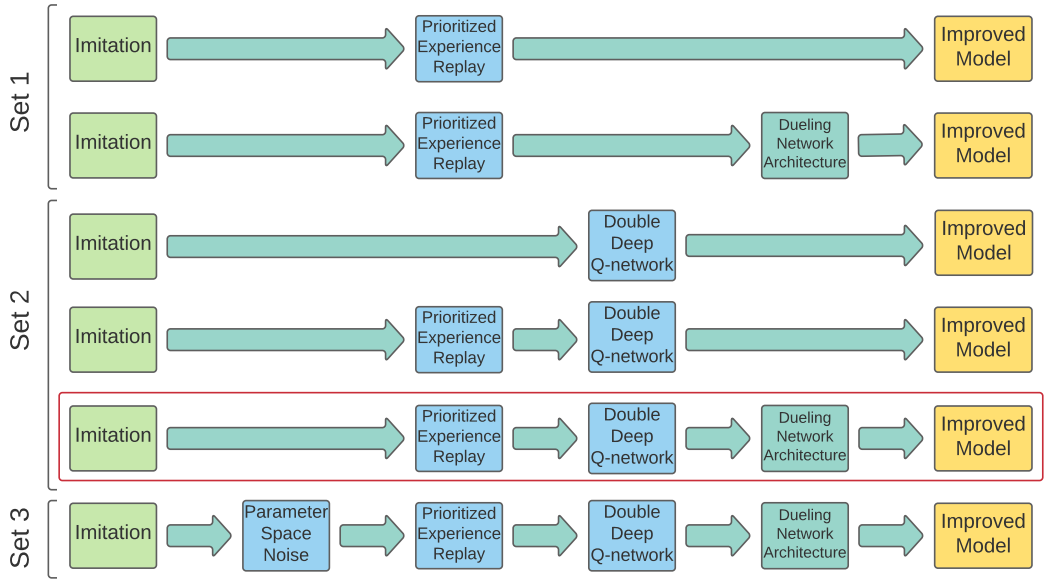

FIGURE 5.1: Module combination depiction with highlighted winner. Imitation refers to the algorithm developed in [Papathanasiou, 2020]. Module combinations are separated to sets based on our analyzed experiment evaluation.

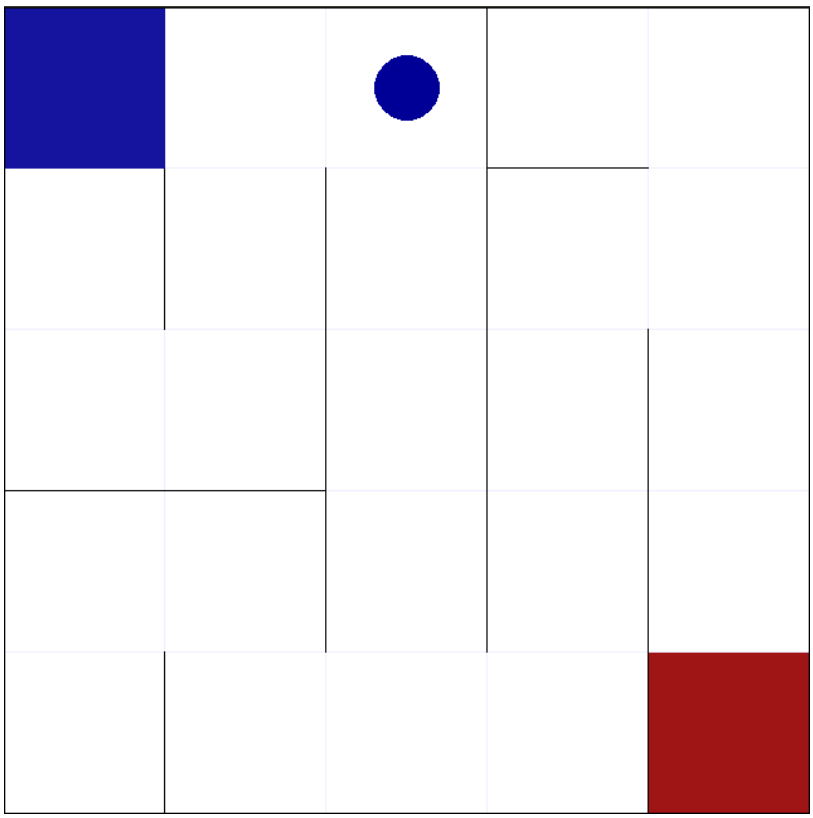

FIGURE 5.2: 2D-maze-v0 environment depiction

the maze. For each time step a reward of $-\frac{0.1}{total\ number\ of\ cells}$ is awarded, with a bounty of 1 if the agent reaches the goal. There is no time restriction, but considering that

the agent gets punished the longer it takes to fulfill its trip, certain limits must be placed to consider an agent perfect. Since our agent performed on $3x3$ mazes, it was calculated that an accumulated reward of at least 0.9111 is needed to solve the maze problem on a training step.

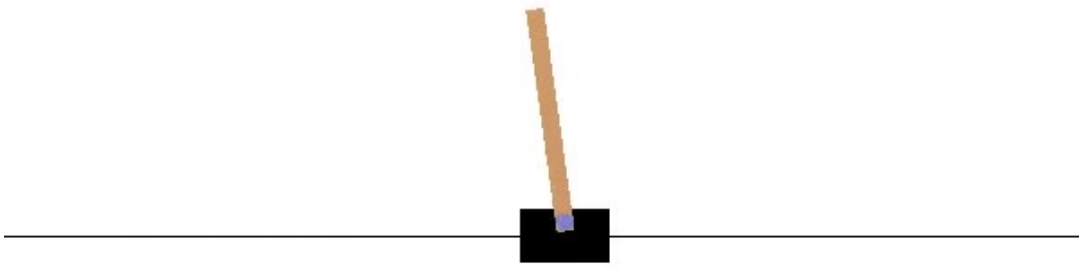

FIGURE 5.3: Cartpole-v1 environment depiction

Our second selected game environment for OpenAI Gym is Cartpole-v1 (Figure 5.3). It is a famous problem [Barto, Sutton, and Anderson, 1983] and it includes a cart with a pole attached to it, moving horizontally on a platform. Due to the nature of the pole and the emulation of gravity, the goal of this game is to balance the pole on the cart for a specific amount of time steps. Actions include only *go left* and *go right* and observations forwarded to the agent's network include cart position, cart velocity, pole angle and pole velocity. For each time step with pole balanced on top of the cart, the agent is rewarded with 1. Each training step is stopped at 200 time steps, creating strict boundaries for a perfect agent at 200 accumulated reward. Upon failure, the environment is reset and a new training step begins, starting from a central position.

At each depiction we present accumulated rewards per training step for over 550 training steps, averaged from 15 untrained agents. It was decided to use training steps over time steps as x axis, since most training steps vary on the skill of the agent and upon failure it resets, ending up with different amount of time steps for each progression level. This provides a faulty sense of improvement over time.

5.2 2D Maze

5.2.1 Prioritized Experience Replay and Dueling Architecture

The first plot (Figure 5.4) includes the learning curves of the mentor, the vanilla trainee, the trainee with Prioritized Experience Replay and the final trainee with

added Dueling Architecture. It is obvious that the original trainee outperforms the mentor with outstanding improvement. Due to the nature of the problem, it is highlighted a great improvement from the mentor to the trainee. Since the states are low in number and discrete, the mentor demonstrations cover all possible combinations of state transitions. That way the trainee can easily adapt to the mentor's guidance, hence the notably earlier convergence to the optimal policy. Moving on, the first stages of exploration are also getting higher total rewards with the help of Prioritized Experience Replay, yielding a more desirable outcome. At the end, using Dueling architecture generates stable and slightly better developments.

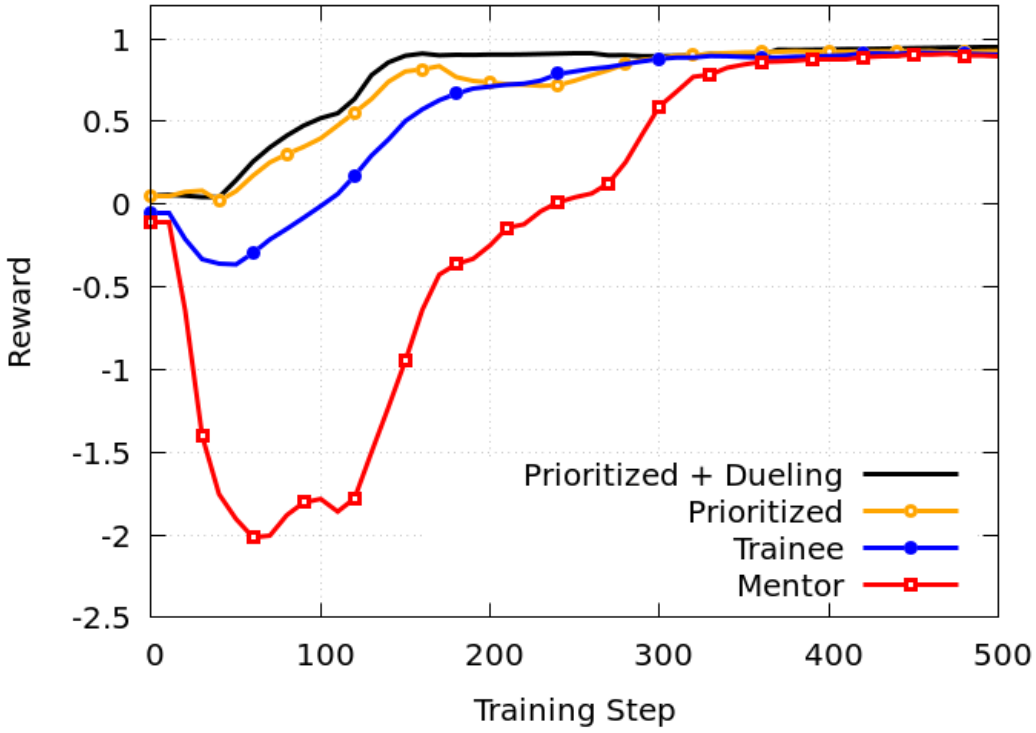

FIGURE 5.4: 2D Maze Prioritized Experience Replay and Dueling Architecture evaluation

5.2.2 Addition of Double Deep Q-Network

We try to further upgrade our plots by enabling Double Deep Q-Network (Figure 5.5). This figure compares the trainee of Prioritized Experience Replay and Dueling Architecture with Double Deep Q-Network and its variants. Even though the plain Double Deep Q-Network trainee is close to our previous best one, it is clear how advanced it gets when combined with the use of Prioritized Experience Replay and Dueling Architecture. As highlighted (Figure 5.1), the most skilled agent in this problem turns out to be the agent with active modules: Double Deep Q-network, Prioritized Experience Replay and Dueling Network Architecture.

Unfortunately, since the exploration stage window of this problem is really small there is no benefit in enabling Parameter Noise Exploration. That is why comments on Parameter Noise Exploration are entered only in the second experiment, cartpole.

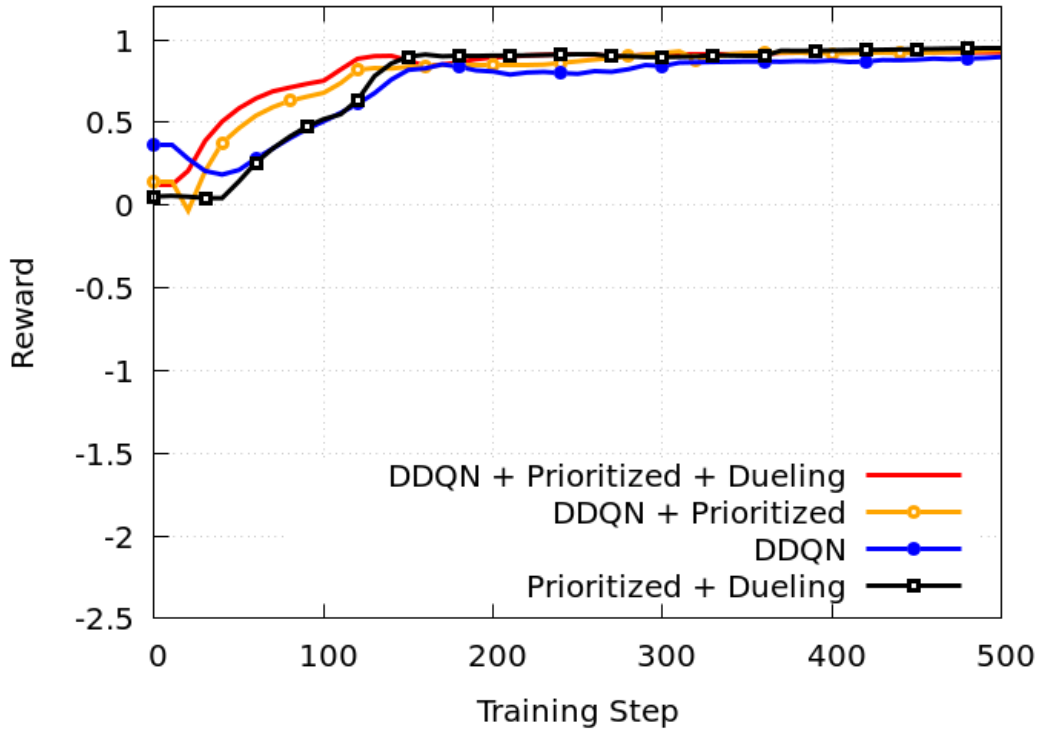

FIGURE 5.5: 2D Maze Addition of Double Deep Q-Network

5.3 Cartpole

5.3.1 Prioritized Experience Replay and Dueling Architecture

Our results are depicted in Figure 5.6. We include the mentor's average reward, the vanilla trainee agent, the trainee agent with improved modeling of Prioritized Experience Replay and another trainee with a Dueling Architecture on top of that. As expected, the trainee is a faster solver than its mentor and that is improved further with the addition of Prioritized Experience Replay. Even though all the agents start off slowly, after the first exploration stage is finished the similarity comparison starts to favor the trainee. The Prioritized Experience Replay's advancement of sample selection provides a clear overall improvement over the vanilla trainee. Finally, with the help of advantage learning provided by Dueling Architecture we reach earlier optimal position. The mentor is converging at around 750 training steps, but due to how exponentially slower the training would be to reach that training step with all the algorithmic combinations, we decided to cut it on plot at 550 training steps.

5.3.2 Addition of Double Deep Q-Network

Next on, we extend our list of observable outcomes by adding Double Deep Q-Network (Figure 5.7). We compare the previously best combination with Double Deep Q-Network and its variants of Prioritized Experience Replay and Dueling Architecture. It is then compared with the Prioritized Experience Replay plus Dueling Architecture agent we mentioned in the previous section to highlight the gain that Double Deep Q-Network provides. This section makes obvious how much Deep Q-Network (both mentor and trainee) suffer from overestimation, a problem resolved

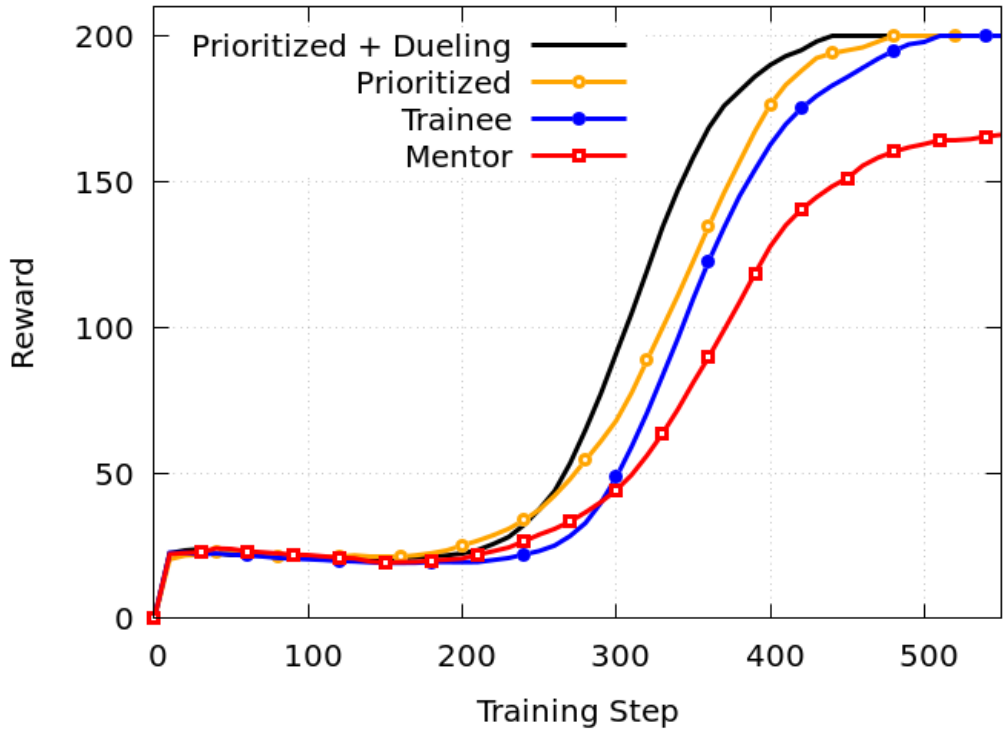

FIGURE 5.6: Cartpole Prioritized Experience Replay and Dueling Architecture evaluation

by Double Deep Q-Networks. Even though the progress changes are limited, it is definite that with each layer of improvement, superior results are returned. Once again, the most skilled agent of this game environment is the combination of Double Deep Q-network, Prioritized Experience Replay and Dueling Network Architecture.

5.3.3 Parameter Space Noise for Exploration

Our final method not included so far in the observations is Parameter Space Noise for Exploration. As one can notice (Figure 5.8), the final variation of the Parameter Noise agent under-performs, ending up with slower problem solving skills than our most successful agent so far, i.e., the one employing Double Deep Q-network, Prioritized Experience Replay and Dueling Network Architecture (Figure 5.1).

However, a remarkably steeper learning curve is noted. More specifically the Parameter Noise agent improves from 30 total reward to 180 total reward in only 110 training steps, while our skilled agent reaches this performance improvement in over 160 training steps. This happens due to the fact that the Parameter Noise provided an enriched exploration phase to the agent, covering more variety of transition samples and supplying the agent with more useful information for its training, hence the steeper curve. Although, possibly due to the nature and simplicity of our experiment, this exploration stage takes longer than expected. In more complex settings and environments we can reasonably expect to outperform our highest score agent.

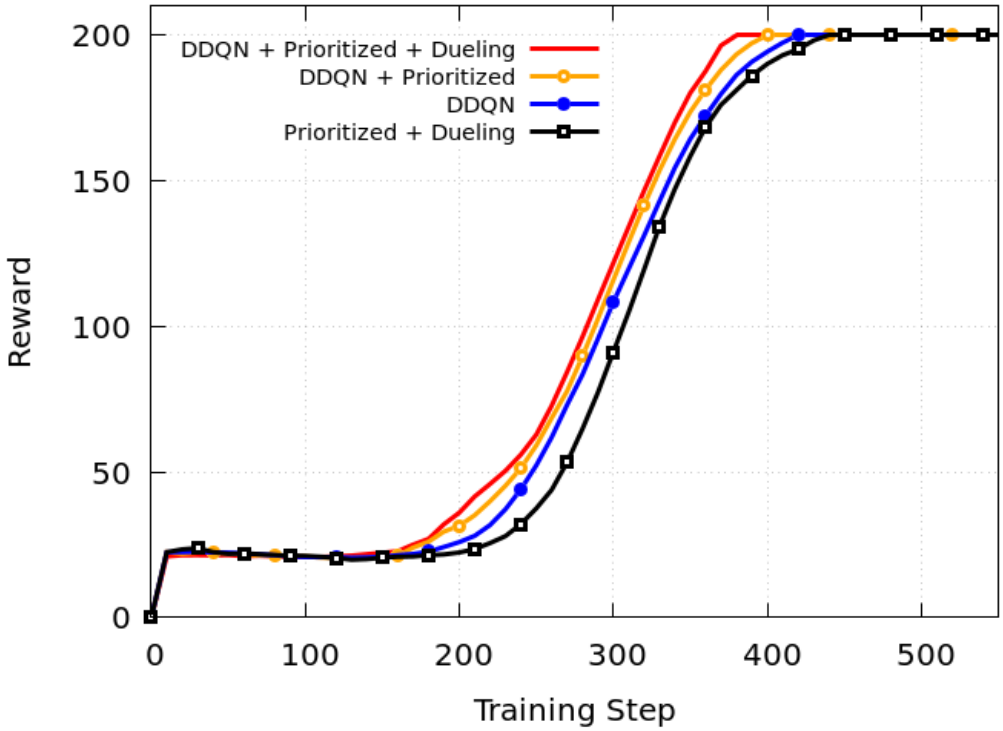

FIGURE 5.7: Cartpole Addition of Double Deep Q-Network

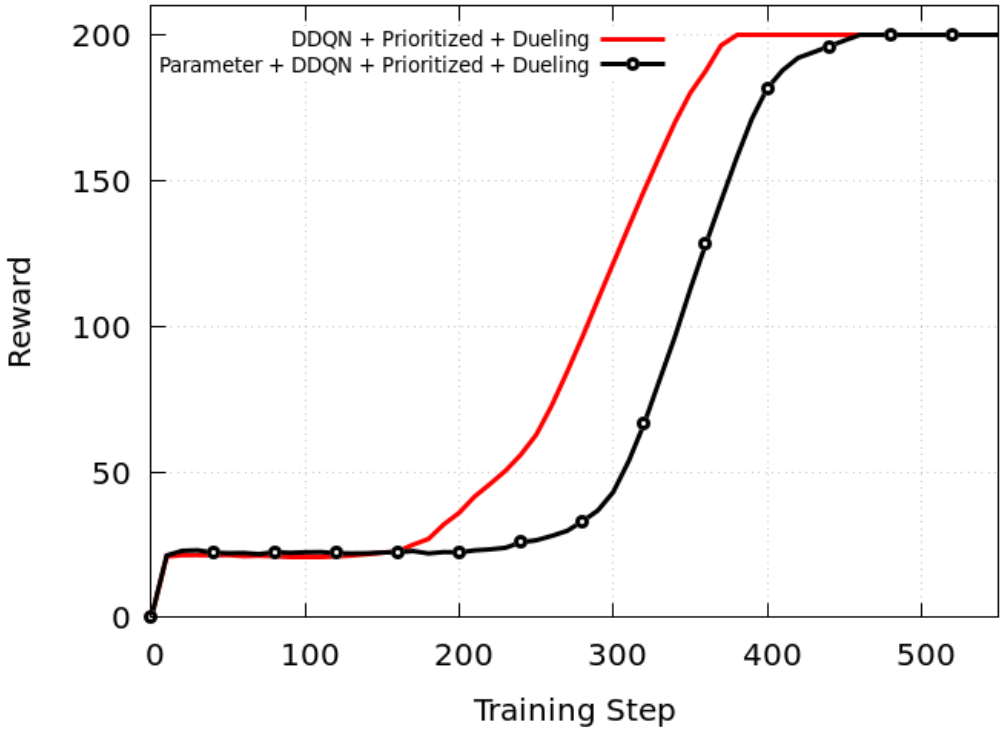

FIGURE 5.8: Cartpole Addition of Parameter Noise

Chapter 6

Conclusion and Future Work

Imitation is a behavioural model firstly observed in nature with numerous varying applications in machine learning. Explicit and implicit imitation in reinforcement learning was in depth analyzed in [Price and Boutilier, 2003]'s work. Implicit imitation deep reinforcement learning used as an acceleration of the training operation is an implementation that was developed by [Papathanasiou, 2020], providing ground for further optimization. In our work we improve their algorithm by applying several optimization techniques: Double Deep Q-network [Hasselt, Guez, and Silver, 2016], Prioritized Experience Replay [Schaul et al., 2016], Dueling Network Architecture [Wang et al., 2016] and Parameter Space Noise for Exploration [Plappert et al., 2018]. We created multiple combinations by enabling and disabling modules in our experiments, with our highest skilled agent being the combination of Double Deep Q-network, Prioritized Experience Replay and Dueling Network Architecture.

However, there is always room for improvement, giving space for future work. One major issue with this experiment is computation power. The neural networks' needs require high quality hardware to reduce the running time of each execution down to minutes. Unfortunately, no such parts were available in this thesis, resulting to multi-hour experiments. Ideally, a high end computer would improve the experience and the research total time, while providing the opportunity to test hyperparameter values and even suboptimal parameterizations for more network behavior information.

Also, another important step that needs to be noted is package and library dependencies. Due to the incredibly fast development of deep learning, these dependencies are being updated constantly, removing specific functions that are deemed obsolete and changing the content and algorithms to match those used in state-of-the-art. These small changes can easily break projects like this thesis, meaning that extra time is required to rewrite specific parts of our algorithm.

Finally, more optimization can be added by playing around with the neural network's layers. This suggestion, mainly due to the hardware restrictions mentioned earlier, was not implemented so far in our project, but surely can provide greater results. It is crucial to highlight that we do not suggest the model itself should be changed as in dueling architecture optimization, but the hidden layers' options specifically.

Appendix A

Hyperparameters

Throughout this thesis the use of hyperparameters has been mentioned multiple times. Their final values were decided after experimentation.

Hyperparameter	**Description**	**Value**
ϵ	exploration (maze)	$1 \rightarrow 0.1$ over 1000 steps
ϵ	exploration (cartpole)	$1 \rightarrow 0.1$ over 10000 steps
a	learning factor	0.00005
γ	discount factor	0.99
N	replay memory size	50000
b	batch sample size	32
M	mentor observation files	4
$\Delta\phi$	similarity threshold (maze)	1%
$\Delta\phi$	similarity threshold (cartpole)	20%
α	priority highlight	0.6
β	priority control	$0.4 \rightarrow 1$ over 10000 steps
κ	noise variance constant	1.01
g	training step reward goal (maze)	0.9111
g	training step reward goal (cartpole)	200

Bibliography

Atkeson, C. G. and S. Schaal (1997). "Robot learning from demonstration". In: *Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97)*. clmc. Morgan Kaufmann, pp. 12–20. URL: http://www-clmc.usc.edu/publications/A/atkeson-ICML1997.pdf.

Baird, Leemon C. and III (1993). *Advantage Updating*.

Bakker, Paul and Yasuo Kuniyoshi (May 1996). "Robot See, Robot Do : An Overview of Robot Imitation". In: *AISB96 Workshop on Learning in Robots and Animals*.

Barto, A., R. Sutton, and C. Anderson (1983). "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13, pp. 834–846.

Brockman, Greg et al. (2016). *OpenAI Gym*. arXiv: 1606.01540 [cs.LG].

Dada, Emmanuel Gbenga et al. (2019). "Machine learning for email spam filtering: review, approaches and open research problems". In: *Heliyon* 5.6, e01802. ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2019.e01802. URL: https://www.sciencedirect.com/science/article/pii/S2405844018353404.

Fan, Haoyang et al. (2018). *Baidu Apollo EM Motion Planner*. arXiv: 1807.08048 [cs.RO].

François-Lavet, Vincent et al. (2018). "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends® in Machine Learning* 11.3-4, 219–354. ISSN: 1935-8245. DOI: 10.1561/2200000071. URL: http://dx.doi.org/10.1561/2200000071.

Harmon, Mance and Stephanie Harmon (July 2000). *Reinforcement Learning: A Tutorial*.

Harmon, Mance and Leemon Iii (Sept. 1998). "Residual Advantage Learning Applied to a Differential Game". In: DOI: 10.1109/ICNN.1996.548913.

Hasselt, Hado van, Arthur Guez, and David Silver (2016). "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2094–2100.

He, Kaiming et al. (2015). *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385 [cs.CV].

Hester, Todd et al. (2017). *Deep Q-learning from Demonstrations*. arXiv: 1704.03732 [cs.AI].

Hester, Todd et al. (2018). "Deep Q-learning from Demonstrations". In: *Annual Meeting of the Association for the Advancement of Artificial Intelligence (AAAI)*. New Orleans (USA).

Ho, Jonathan and Stefano Ermon (2016). *Generative Adversarial Imitation Learning*. arXiv: 1606.03476 [cs.LG].

Hussein, Ahmed et al. (Apr. 2017). "Imitation Learning: A Survey of Learning Methods". In: *ACM Comput. Surv.* 50.2. ISSN: 0360-0300. DOI: 10.1145/3054912. URL: https://doi.org/10.1145/3054912.

Kononenko, Igor (2001). "Machine learning for medical diagnosis: history, state of the art and perspective". In: *Artificial Intelligence in Medicine* 23.1, pp. 89–109. ISSN: 0933-3657. DOI: https://doi.org/10.1016/S0933-3657(01)00077-X. URL: https://www.sciencedirect.com/science/article/pii/S093336570100077X.

Kullback, S. (1959). *Information Theory and Statistics*.

L. P. Kaelbling M. L. Littman, A. W. Moore (May 1996). *Reinforcement Learning: A Survey*.

Le, Hoang M. et al. (2018). *Hierarchical Imitation and Reinforcement Learning*. arXiv: 1803.00590 [cs.LG].

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

Lin, Long-Ji (May 1992). "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Mach. Learn.* 8.3–4, 293–321. ISSN: 0885-6125. DOI: 10.1007/BF00992699. URL: https://doi.org/10.1007/BF00992699.

Mahmood, A. Rupam, Hado P van Hasselt, and Richard S Sutton (2014). "Weighted importance sampling for off-policy learning with linear function approximation". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2014/file/be53ee61104935234b174e62a07e53cf-Paper.pdf.

Mataric, Maja J. (1998). "Using communication to reduce locality in distributed multiagent learning". In: *Journal of Experimental & Theoretical Artificial Intelligence* 10.3, pp. 357–369. DOI: 10.1080/095281398146806. eprint: https://doi.org/10.1080/095281398146806. URL: https://doi.org/10.1080/095281398146806.

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

OpenAI et al. (2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv: 1912.06680 [cs.LG].

Papathanasiou, Theodoros (2020). "Accelerating Deep Reinforcement Learning via Imitation". In: *Senior Undergraduate Diploma Thesis, Technical University of Crete*.

Plappert, Matthias et al. (2018). *Parameter Space Noise for Exploration*. arXiv: 1706.01905 [cs.LG].

Price, B. and C. Boutilier (Dec. 2003). "Accelerating Reinforcement Learning through Implicit Imitation". In: *Journal of Artificial Intelligence Research* 19, 569–629. ISSN: 1076-9757. DOI: 10.1613/jair.898. URL: http://dx.doi.org/10.1613/jair.898.

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley and Sons, Inc. ISBN: 0471619779.

Ranganathan, Ananth Th (2004). *The Levenberg-Marquardt Algorithm*.

Ross, Stephane, Geoffrey J. Gordon, and J. Andrew Bagnell (2011). *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. arXiv: 1011.0686 [cs.LG].

Rückstieß, Thomas, Martin Felder, and Jürgen Schmidhuber (2008). "State-Dependent Exploration for Policy Gradient Methods". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 234–249. ISBN: 978-3-540-87481-2.

Rumelhart, David E. and James L. McClelland (1987). "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pp. 318–362.

Salimans, Tim et al. (2017). *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. arXiv: 1703.03864 [stat.ML].

Schaul, Tom et al. (2016). *Prioritized Experience Replay*. arXiv: 1511.05952 [cs.LG].

Silver, David et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144. ISSN: 0036-8075. DOI: 10.1126/science.aar6404.

Smith, James E. and Robert L. Winkler (2006). "The Optimizer's Curse: Skepticism and Postdecision Surprise in Decision Analysis". In: *Management Science* 52, 311–322.

Stadie, Bradly C., Pieter Abbeel, and Ilya Sutskever (2019a). *Third-Person Imitation Learning*. arXiv: 1703.01703 [cs.LG].

— (2019b). *Third-Person Imitation Learning*. arXiv: 1703.01703 [cs.LG].

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction*. Second. The MIT Press.

Tas, Seyma (2021). *Reinforcement Learning For Mice*. URL: https://towardsdatascience.com/reinforcement-learning-3f87a0290ba2.

Van Hasselt, Hado (Jan. 2010). "Double Q-learning." In: pp. 2613–2621.

Wang, Ziyu et al. (2016). "Dueling Network Architectures for Deep Reinforcement Learning". In: arXiv: 1511.06581 [cs.LG].

Watkins, Christopher J. C. H. and Peter Dayan (May 1989). "Q-learning". In: *Machine Learning* 8.3, pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698.

Wiering, Marco A. (2010). "Self-play and Using an Expert to Learn to Play Backgammon with Temporal Difference Learning". In: *Journal of Intelligent Learning Systems and Applications*, pp. 55–66.