Master in Intelligent Interactive Systems
Universitat Pompeu Fabra

Empirical analysis of exploration strategies in QMIX

Arnau Colom

Supervisor: Vicenç Gómez

September 2021

Master in Intelligent Interactive Systems
Universitat Pompeu Fabra

Empirical analysis of exploration strategies in QMIX

Arnau Colom

Supervisor: Vicenç Gómez

September 2021

Contents

Dedication

I would like to dedicate this work to my parents and myself.

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Vicenç Gomez, for his support and guidance during the development of the project and for his patience. I would also like to thank my family who has been really supportive during this last year and for give me the opportunity to study in Barcelona. And finally express my appreciation to my friends, which have made this last year more bearable.

Abstract

In real world scenarios, to solve a gran majority of problems, there is the necessity for different agents to cooperate under the condition of local observations. Fortunately, in the recent years, significant advances in Multi-Agent Reinforcement Learning have been done regarding this matter. To tackle this kind of problems, a lot of approaches are based on the on Centralized Training with Decentralized Execution, which allow the agents to be trained in a simulated environment where they can have access to the global information to later solve the problem relying only on local observations. Some popular methods are Value-Decomposition Networks (VDN) and QMIX. They undertake the problem by computing the joint action-value function Q_{tot} as a combination of the individual action-value functions Q_a, that only condition on individual action-observation histories.

Specifically, this work focuses on QMIX, which has been gaining a lot of popularity in the last year due to its capacity to compute a richer representation of the joint action-value function than VDN by combining the individual Q-values in a non-linear approach. However, despite the fact that there have been a lot of improvements on QMIX, there have been small advances on how different exploration techniques could boost the learning in this context. In this work, by performing an experimental evaluation, its shown how some exploration methods outperform the $\epsilon - greedy$ approach used on the original implementation of QMIX on different cases.

Keywords: Multi-Agent Learning; Multi-Agent Coordination; QMIX; Exploration

Chapter 1

Introduction

Reinforcement Learning (RL) is an area of Machine Learning that has gained and is still gaining a lot of popularity due to its potential applications in real case scenarios, e.g self driving cars or robotics [1][2]. This thesis focuses on one of the many disciplines of RL, Multi-Agent Reinforcement Learning (MARL). In MARL multiple agents interact inside a common RL environment to collaborate, coordinate, compete, or collectively learn to accomplish a specific task [3]. In particular, the work will aim its attention at the *cooperative* Multi-Agent Reinforcement Learning domain.

There are diverse strategies to tackle the problem of cooperative multi-agent systems depending on the learning and execution of the method [3]. Specifically the thesis will aim its attention to the QMIX algorithm [4], which is characterized by a centralized learning and decentralized execution, and will test how different exploration techniques can affect the learning of the agents.

1.1 Motivation

In RL, given a state on the environment, the agent needs to face the following question: "Do I take the action that I know that can lead me to a known good result or do I search for better alternatives by exploring the environment?". This is what is

called the exploration-exploitation trade-off [5].

The thesis will focus especially on the idea of exploration, where the agent does not take the action that is optimal but gamble on the fact that may be better alternatives to explore that can lead to better results. The main objective of exploration is to avoid that the agent gets stuck with sub-optimal actions. The idea here is, instead of explore randomly the different actions, apply some heuristics to select more accurately the actions that can make more impact.

1.2 Objectives

The purpose of this work is to compare different exploration techniques through the QMIX algorithm [4], specifically on the The Star-Craft Multi-Agent Challenge (SMAC) environment [6]. The ideal goal would be to find if other exploration techniques, rather than epsilon-greedy exploration, can be useful to improve the learning performance in some scenarios.

1.3 Structure of the Report

First, there will be an introduction to the basic background related to RL, and how to translate it to a MARL environment. After this brief explanation to RL will be explained how the QMIX algorithm works and what are its advantages with respect to other techniques. Then there will be a presentation to the SMAC environment, where it will be detailed how it works and how has been useful for researchers in the field of *collaborative* MARL. Secondly, a description will be made regarding the different exploration techniques that have been used during the project. Finally the results will be exposed with a discussion of their effectiveness on the context exposed.

Chapter 2

Background

2.1 Reinforcement Learning

As mentioned in Chapter 1, RL is an area of Machine Learning that aims to teach an agent to complete a specific task. The agent is placed in an environment and need to interact with it, taking different actions, to maximize a reward in order to learn how to solve the problem that is facing.

The way an agent knows about the environment is through trial and error, resembling the human way of learning [7]. Through this iterative process the aim of the agent is to update its policy $\pi(u \mid s)$, its is behavior function, that maps which action $u \in U$ needs to be taken when being on state $s \in S$. An important thing to mention is that an RL problem can be modeled as a Markov Decision Process (MDP)[8]. The advantage of being modeled as an MDP is that the sequence of states share the Markov property, which states that next state s_{t+1} only depends on the current state s_t and action taken.

During training, the agent a faces a state $s \in S$, where he can take an action $u \in U$. After taking the action u the environment will change according to the state transition function $P(s' \mid s, u)$, that describes the probability of entering the next state s' given the current state s and having taken action u. After an action has been taken, the agent will receive a reward $r(s, a)$. This reward will normally be

negative if the outcome is not desired and positive if the next state helps to get the agent closer to the goal.

To quantify how good a policy π is, starting from a state s_t, the State-Value Function $V^{\pi}(s_t)$ is evaluated. In essence it measures how favorable it is for the agent to be in a specific state. Analogous to the State-Value Function is the Action-Value Function $Q^{\pi}(s_t, u)$, a function that measures how beneficial is for the agent to take the action u while being in state s_t under the policy π. Both concepts rely on the discounted future rewards that can be expected and are described as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s\right] \tag{2.1}$$

$$Q^{\pi}(s, u) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, U_t = u\right], \tag{2.2}$$

where γ is the discount rate, $0 \leq \gamma \leq 1$, and it is used to give more importance to the near rewards rather that the ones that are far away. The discounted reward is mainly used when no terminal states are defined or for infinitely repeated games.

An agent will iteratively explore an environment to find the best policy π, while trying to approximate the same policy or the State-Value Function and Action-Value Function. This work focuses on the learning of the agents through the evaluation of the action-value function or Q-function [9].

2.2 Cooperative Multi-Agent Reinforcement Learning and their different approaches

In a cooperative multi-agent scenario, multiple agents interact with the environment and with each other in other to achieve a common goal. In this context there are different ways to tackle the learning, that is, in a centralized or decentralized way. Usually, on MARL, partial observability or communication constraints necessitate the learning of decentralised policies. This type of policies only condition on the local action/observation of each agent which has the property of attenuate the prob-

lem of exponentially growing action spaces due to large number of agents. Another advantage of decentralized policies is that they can be learnt in a centralised fashion, being able to access additional state information, while having a decentralized execution [10].

One of the challenges of this paradigm of centralized training and decentralized execution is the representation of the action-value function. One solution to the problem would be represent an independent action-value function for each agent Q_a, this approach is called *independent Q-learning*[11]. However, it makes difficult the representation of the interaction of the different agents and may not converge. On the other side, the global action-value function can be learn fully centralized as Q_{tot} and learn the decentralise policies as an actor-critic framework, such as COMA [12]. Unfortunately, the training of the centralized critic became inefficient when a large number of agents are involved.

Lying between these two extremes is the value decomposition networks (VDN) [13], where a centralized action value function Q_{tot} can be learnt as a sum of individual action-value functions Q_a. This approach makes possible to have a decentralized policy by selecting the action greedily with respect to the individual Q_a. From this approach appear QMIX [4], a similar method that tackles the problem on a similar manner but is able to represent a more richer representation of the Q_{tot} by combining non-linearly the individual Q_a.

2.3 Dec-POMDP

A fully cooperative multi-agent sequential decision-making task can be described as a decentralised partially observable Markov decision process (Dec-POMDP) [14] consisting of a tuple $G = \langle S, A, U, P, r, Z, O, n, \gamma \rangle$. The state $s \in S$ describes the true state of the environment. At each time step, given a state s, each agent $a \in A \equiv \{1, \ldots, n\}$ chooses an action $u^a \in U$, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$. This causes a transition on the environment according to the state transition function $P(s' \mid s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$. Here all agents share the same reward function

$r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$ and $\gamma \in [0, 1)$ is the discount factor.

It also consider a partially observable scenario in which each agent draws individual observations $z \in Z$ according to observation function $O(s, a) : S \times A \to Z$. Each agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which it conditions a stochastic policy $\pi^a\left(u^a \mid \tau^a\right) : T \times U \to [0, 1]$. The joint policy π has a joint action-value function: $Q^\pi\left(s_t, \mathbf{u}_t\right) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}\left[R_t \mid s_t, \mathbf{u}_t\right]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return.

In this work, since the training is centralized and the execution is decentralised, the learning algorithm has access to all local action-observation histories τ and global state s, but each agent's learnt policy will condition only on its own action-observation history τ^a.

2.4 Deep $Q-$Learning and Deep Recurrent Q-learning

Normally the Q-function is hard to learn when there are a large number of states and actions. The task of evaluating the Action-value function for those situations can become easier using deep neural networks. *Deep Q-Networks*(DQN's)[15] make use deep neural networks and experience replay to learn. This experience replay is described as a tuple that stores $\langle s, u, r, s' \rangle$, where s' is the next state after taking the action u in state s and receiving the reward r. The neural network is parameterised by θ, which is learnt by sampling batches on the replay memory and minimising the squared TD[1] error.

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{\mathrm{DQN}} - Q(s, u; \theta) \right)^2 \right], \tag{2.3}$$

where the term $y^{\mathrm{DQN}} = r + \gamma \max_{u'} Q\left(s', u'; \theta^-\right)$ and θ^- are the parameters of a target network that are periodically copied from θ. If the learning environment rely on partially observable settings, instead of using the experience replay tuple, agents can benefit from the entire action-observation history. This variation is known as

[1] Temporal Difference

Deep Recurrent Q-Learning (DRQN) [16].

2.5 Value Decomposition Networks

In a MARL environment, VDNs compute the joint action-value function Q_{tot} as the sum of individual value functions $Q_a(\tau^a, u^a; \theta))$, which have previously been computed using DRQNs.

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q_i\left(\tau^i, u^i; \theta^i\right) \tag{2.4}$$

where $\boldsymbol{\tau} \in \mathbf{T} \equiv \mathcal{T}^n$ is a joint action-observation history and $\mathbf{u}$ is a joint action. During training, the loss function that is minimized is the same as described in Eq. (2.3) but replacing Q by Q_{tot}. This method makes possible the evaluation of the decentralised policy applying a greedy action selection for each Q_a

2.6 QMIX

From VDN arises QMIX [4], an approach that is also situated between the extremes of independent Q-learning and full centralised learning. The main idea of this method comes from the fact that the full factorisation of VDN is not unique in order to establish consistency between the deterministic greedy centralised policies and the deterministic greedy decentralised policies based on the optimal joint action-value function. The consistency is well represented on the following equality:

$$\underset{\mathbf{u}}{\text{argmax}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \text{argmax}_{u^1} Q_1\left(\tau^1, u^1\right) \\ \vdots \\ \text{argmax}_{u^n} Q_n\left(\tau^n, u^n\right) \end{pmatrix}. \tag{2.5}$$

Here the authors of [4] made the observation that (2.5) can be satisfied as long as the factorisation is described monotonically. In this sense the monotonicity is a constrain on the relation of Q_{tot} and Q_a described as:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a \in A \tag{2.6}$$

Keeping that in mind, the authors state the following theorem:

Theorem 2.1. *If* $\forall a \in A \equiv \{1, 2, \ldots, n\}, \frac{\partial Q_{tot}}{\partial Q_a} \geq 0$ *then*

$$\underset{\mathbf{u}}{\text{argmax}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \text{argmax}_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \text{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}$$

It is shown in [4] that 2.6 is sufficient to satisfy 2.5. This relationship between Q_{tot} and Q_a is illustrated by figure 1. On the image can be appreciated how the monotonicity influence the decentralization of the argmax.

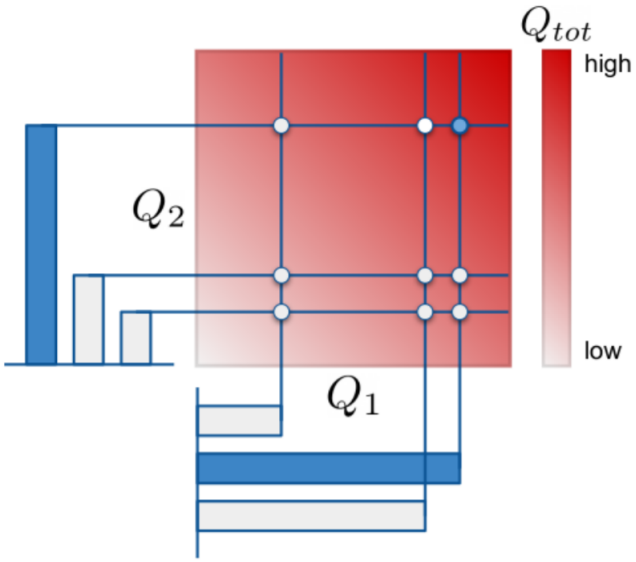


Figure 1: The monotonic function takes as input Q_1 and Q_2 to compute Q_{tot}. The maximum value of the individual action-values Q_a is illustrated as the blue bars, which also relate to the argmax of the Q_{tot}. Figure taken from QMIX [4].

2.6.1 QMIX Architecture

QMIX is composed by three different networks: the agent networks, a mixing network and a set of hypernetworks [17]. For each agent a, there is one agent network represented as a DRQN, that outputs its individual value function $Q_a(\tau^a, u^a)$. The

mixing network is described as a feed-forward neural network and takes as inputs the outputs of the agent networks to mix them monotonically. The weights of each layer of the mixing network are produced by separate hypernetworks. These hypernetworks take as input the state s and make sure that the weights that produce are non-negative. The relationship between each network and its composition is represented on the following image:

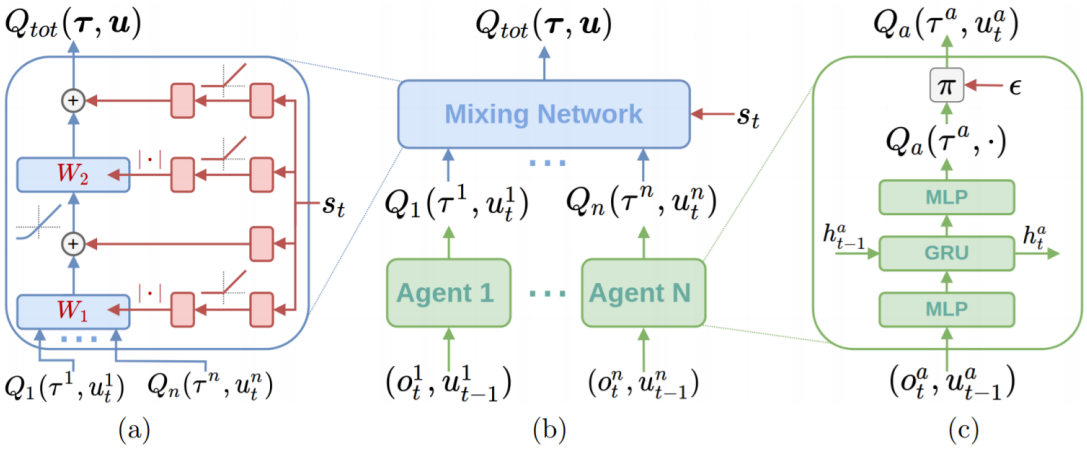


Figure 2: Architecture of QMIX. (a)The Mixing network. (b)Overall architecture of QMIX. (c) Agent network. Figure taken from QMIX [4].

The complete implementation of QMIX in pseudo-code is in A.2. In the algorithm, the $\epsilon-$ greedy action selection (line 10) and the choice of gather roll-outs from an entire episode before executing a single gradient descent step(line 17) is not a requirement for QMIX. However, the action selection need to based on Q-values.

Chapter 3

Methods

In this chapter will be described the environment on which the QMIX algorithm and the different exploration techniques have been tested. Also there will be an explanation of the different methods of exploration that have been evaluated during the project.

3.1 The StarCraft Multi-Agent Challenge

The project has been developed on a framework based on the the StarCraft Multi-Agent Challenge, or SMAC for short [6], called PyMarl [18]. The framework contains the implementation of diverse MARL algorithms such as VDN, COMA, IQL, QTRAN [19] and QMIX. The SMAC environment is based on the popular real-time strategy (RTS) game Star-Craft II, where different opponents fight against each other to get resources, create armies and build infrastructures to defeat the opponent.

Inside the same game, it can be identified two different components: Macro-management and Micro-management. Macro-management refers to the control and creation of high-level strategies, such as the economy and construction of the different buildings. Micro-management, on the contrary, focuses on the manipulation of individual units. To provide a decentralised micro-management scenario, SMAC makes use of

the Blizzard's StarCraft II Machine Learning API [1] and DeepMind's PySC2 [2].

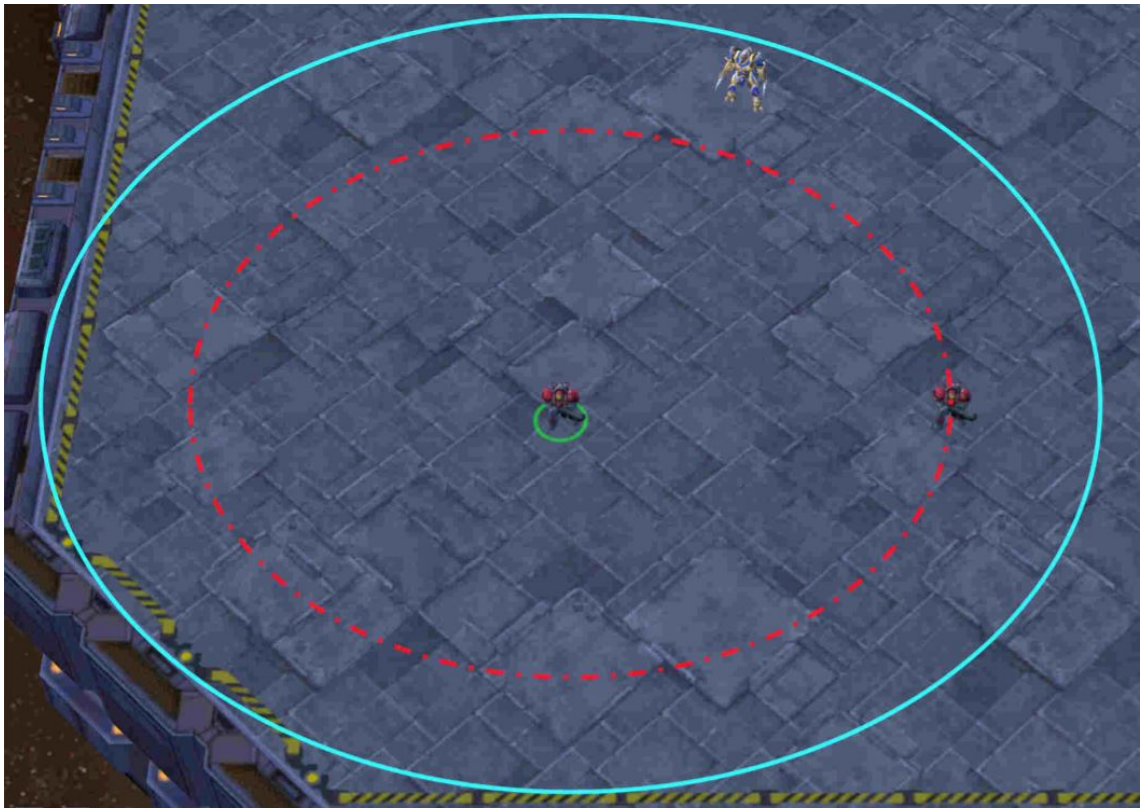


Figure 3: The cyan and red circles respectively border the sight and shooting range of the agent.

The different exploration approaches are tested in regards of the micromanagement component, where each unit is controlled by an independent agent that conditions only on local observations restricted to a limited field of view centred on that unit. Groups of these agents are trained to solve combat scenarios where they fight against an opposing army under the centralised control of the game's built-in scripted AI.

Each agent has its own local observation space that is delimited by the sight range described by the cyan circle illustrated in Figure 3. The partial observability of the agent is conditioned on the allied and enemies units that are inside of the circle, which means that he cannot distinguish between the ones that are outside the circle from the ones that are dead. For each unit, be it ally or enemy, inside the agent range, can be known its `distance, relative x, relative y, health, shield`

[1] Code is available at `https://github.com/Blizzard/s2client-proto`
[2] Code and paper is available at `https://github.com/oxwhirl/pymarl`

and `unit type`. On top of that, it can also know the history of the last action taken by the ally units. And last but not least, it will have a small intuition of the terrain features such as walkability and height.

Given a certain state, all the ally units that are not dead need to select an action either to explore the environment or to get the maximum reward possible. Each agent can select from a set of allowed discrete actions that are `move(north, south, west, east)`, `attack(enemy:ID)`, `stop` and `no-op`, the last being the action of being death.

An important piece of information that is also available in SMAC during the centralised training is the information of **all** the units on the game with respect to the center of the map. Additionally it introduces the `energy` of the Medivas, the `cooldown` of the ally units and the complete set of actions taken by the ally units.

All the characteristics described during this section makes SMAC a powerful tool for researchers thanks to the advantage of possessing important features that are in common with the real world, which are: partial observably, large number of agents, diversity, long-term planning, high-dimensional observation spaces, large per-agents action space, coordinated teamwork and stochasticity.

3.2 Exploration Strategies

As mention on section 1.1, in the RL context there is the critical topic of exploitation versus exploration. On this section the different exploration methods that have been evaluated during the project will be described.

3.2.1 Epsilon Greedy

This method is one of the most common, simple and effective ones used for exploration in RL. In the Epsilon Greedy approach, in state s, the agents select an action randomly with probability ϵ and the best know action a with probability $1 - \epsilon$. The

idea is represented in Eq. 3.1.

$$u_t^a = \begin{cases} \text{argmax}_{u_t^a} Q\left(\tau_t^a, u_t^a\right) & \text{with probability } 1-\epsilon \\ \text{randint}(1, |U|) & \text{with probability } \epsilon. \end{cases} \tag{3.1}$$

Usually the probability ϵ is decreasing over time. This is done so that at the beginning of the training the agent can take purely aleatory actions to explore the environment and have an approximation of the action-value function. As the exploration goes by, the value of epsilon became almost 0, forcing the agent choose almost all the time the best action.

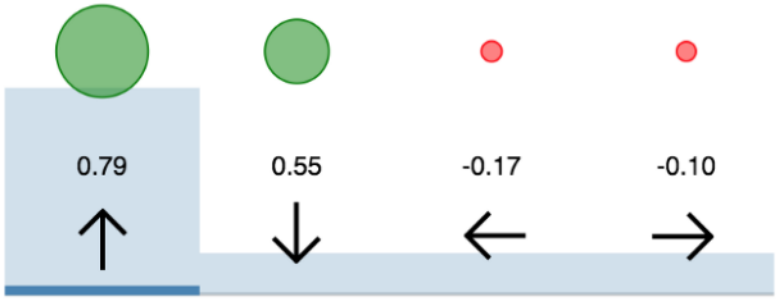


Figure 4: Toy example of the ϵ-greedy criteria to chose an action (`up, down, left, right`). The circles and the value inside the bars represent the value of the Q-function and the height of the blue bars represent the probability of choosing each action. In this example ϵ is set to a small number, which means that the probability of choosing the first action is greater than choosing the other actions. As can be seen, the other actions the same probability of being selected. [3]

3.2.2 Boltzmann

In Boltzmann exploration, the criteria for exploration is a weighted set probabilities computed from the action-value function. To map the outputs of the agent network to a probability distribution, the softmax function is used. For the agent action u_j at a time-step t:

$$P(u_j) = \frac{\exp\left(Q\left(\tau^a, u_j^a\right)/\rho\right)}{\sum_{i=1}^{|u^a|} \exp\left(Q\left(\tau^a, u_i^a\right)/\rho\right)} \tag{3.2}$$

[3] http://awjuliani.github.io/exploration/index.html

To assure that the agents are able to explore sufficiently the function includes the parameter ρ, which is used to tune how much stochastic will be the resulting probability distribution. Usually this parameter is set to a large number, for instance 1000.0, at the beginning of training and during the execution, its value will decrease to almost zero, so that it will select the action greedily.

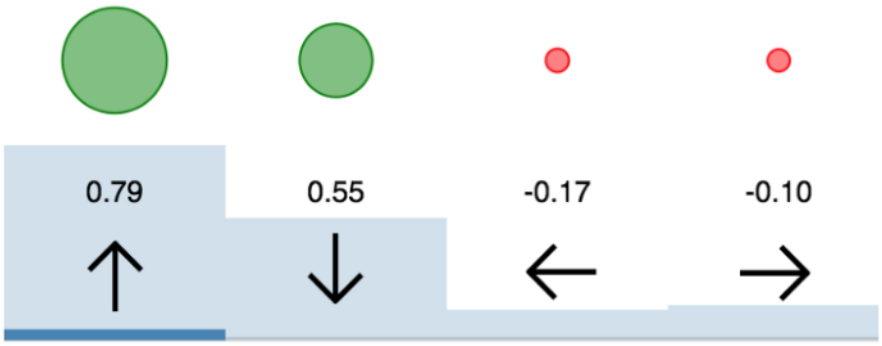


Figure 5: Toy example of the Boltzmann exploration criteria to chose an action(`up, down, left, right`). As mention on Figure 4 the circles and the value inside the bars represent the value of the Q-function and the height of the bars represent the probability of the action of being chosen. In this case, compared to the ϵ- greedy exploration, each action has its own probability weighted based on the Q-values. [4]

3.2.3 UCB

One of the problems of the previous methods is the randomness of the selection. It may be possible to select an action that has proven to be unfavorable. To avoid such inefficient exploration, one alternative approach is the Upper Confidence Bounds (UCB). UCB comes from the idea of exploring the actions that are known to be uncertain using the upper-bound function $\hat{U}_t(u)$ [20].

$$u_t^{UCB} = \text{argmax}_{u \in \mathcal{U}} \hat{Q}_t(u) + \hat{U}_t(u) \tag{3.3}$$

Given the equation 3.3 the action-value function can be interpreted as the idea of exploitation, while the upper-bound function the quantification of how good would be to explore the environment.

[4] http://awjuliani.github.io/exploration/index.html

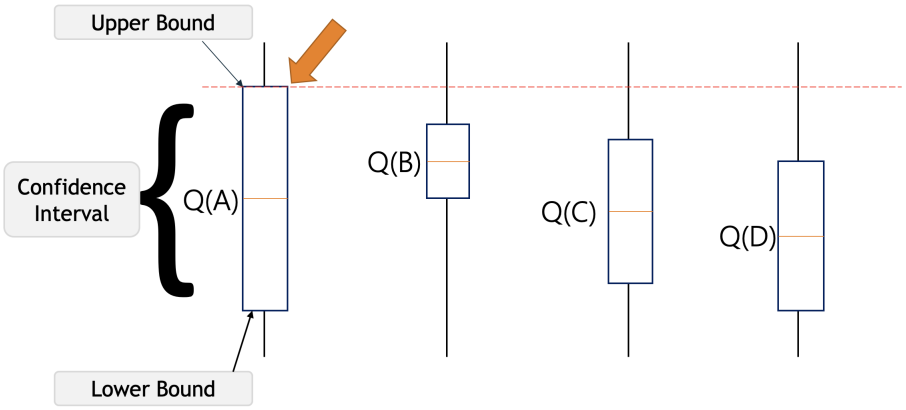


Figure 6: Visual Representation of the Upper bound intuition

Hoeffding's Inequality and UCB1

Let $X_1, \ldots, X_t$ be i.i.d. (independent and identically distributed) random variables and they are all bounded by the interval $[0, 1]$. The sample mean is $\bar{X}_t = \frac{1}{t} \sum_{\tau=1}^{t} X_\tau$. Then for a value c > 0, is:

$$\mathbb{P}\left[\mathbb{E}[X] > \bar{X}_t + c\right] \leq e^{-2tc^2} \tag{3.4}$$

This is called the Hoeffding's Inequality [21] and can be translated to a RL context in the following way:

$$\mathbb{P}\left[Q(u) > \hat{Q}_t(u) + U_t(u)\right] \leq e^{-2tU_t(u)^2} \tag{3.5}$$

Where $\hat{Q}_t(u)$ represents the the sample mean and $Q(u)$ the true mean. One heuristic is to reduce the threshold $e^{-2tU_t(a)^2}$ in time, as we want to make more confident bound estimation with more rewards observed. This is called the UCB1 exploration and can be expresses as 3.6.

$$U_t(u) = \sqrt{\frac{2\log t}{N_t(u)}} \text{ and } u_t^{UCB1} = \arg\max_{u \in \mathcal{U}} Q(u) + \sqrt{\frac{2\log t}{N_t(u)}} \tag{3.6}$$

$N_t(u)$ represents the number of times an action u has been selected, being described as $N_t(u) = \sum_{\tau=1}^{t} 1\left[u_\tau = u\right]$. Here, at the beginning of the training the agent will be

more susceptible to take actions that have not been chosen before. As long as the training progress, the agent will select the actions in a more greedy approach on the action-value function.

3.2.4 Entropy Regularization

Another way to tackle the problem of exploration is to perform the exploration directly on the learning of the action-value function through the loss function. Given the objective used in QMIX [4] is

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{tot} - Q_{tot}(\tau, \mathbf{u}, s; \theta) \right)^2 \right], \tag{3.7}$$

where b is the batch size of transitions sampled from the replay buffer, the DQN taget is given by $y^{tot} = r + \gamma \max_{\mathbf{u}'} Q_{tot}(\tau', \mathbf{u}', s'; \theta^-)$, and θ^- are the parameters of a target network, as in DQN.

The cost of (3.7) can be penalized using the following objective:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{tot} - Q_{tot}(\tau, \mathbf{u}, s; \theta) \right)^2 \right] - \beta H(\theta), \tag{3.8}$$

$$H(\theta) = -\frac{1}{n} \sum_{a=1}^{n} \sum_{u^a} Q_a(\tau^a, u^a; \theta) \log Q_a(\tau^a, u^a; \theta), \tag{3.9}$$

where $H(\theta)$ is the mean entropy averaged over the individual Q_a's.

Here the main idea is to penalize the learning if the entropy of the action-value function is too low. This component is regulated using by parameter β. At the beginning of the training, β will by to a initial value that will be decreasing during the execution. By forcing the entropy to have more importance at the beginning of the learning and performing a greedy selection based on the action-value function while exploring, it is avoid a locally optimal solution and the agent will be incentivized to try different actions. As long as β is decreasing over time, there will be a roughly good estimation of $Q_a(\tau^a, u^a; \theta)$ and the greedy selection will be more accurate.

The problem of using directly this approach is that the Q-values can be negative,

making the logarithm undefined. For this reason, the soft-max function is applied to map the Q-values and assure that their range is between 0.0 and 1.0, similarly used in Boltzmann (3.2.2).

Gradient

Since the exploration using entropy regularization takes place during the training phase, that is, when the loss is computed, it is necessary to compute the derivative to back-propagate the loss correctly.
The gradient of Equation (3.8) is [5]

$$\Delta\theta = \nabla_\theta(\Delta Q_{tot})^2 + \beta\frac{1}{n}\sum_{a=1}^{n}\sum_{u_a}\log Q_a\left(\tau^a, u^a; \theta\right). \tag{3.10}$$

But, as mentioned on 3.2.4, on the Q-function is applied the soft-max operation to transform their values in order to make possible to compute the entropy. This change makes the derivative of the entropy change a little bit. To find it is necessary to use the chain rule, it states that: given a function $h(x) = f(S(x))$, its derivative is $h'(x) = f'(S(x)) \cdot S'(x))$.

In this case we have that $f(S(x)) = S(x) \cdot log(S(x))$ where $S(x) = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}}$.

Then the first step is to compute $f'(g(x))$, that will be $f'(S(x)) = \log(S(x))$. The next step is to compute the derivative of the soft-max function. By definition the derivative of the output element S_i with respect to input element D_j is defined as:

$$D_j S_i = \frac{\partial S_i}{\partial a_j} = S_i\left(\delta_{ij} - S_j\right) \quad \text{,where } \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{3.11}$$

In this case in particular is only computed the derivative of the output of the $i-Th$ element of the soft-max with respect to its $j-Th$ input element, where $i = j$.

Taking that into account and simplifying $Q_a\left(\tau^a, u^a; \theta\right))$ as Q_a the gradient is de-

[5] $\frac{\partial}{\partial p} p \log p = \log p$

scribed as:

$$\Delta\theta = \nabla_\theta(\Delta Q_{tot})^2 + \beta\frac{1}{n}\sum_{a=1}^{n}\sum_{u_a}\log(S(Q_a))S(Q_a)(1-S(Q_a)) \tag{3.12}$$

,

Applying [6] and [7] the gradient becomes:

$$\Delta\theta = \nabla_\theta(\Delta Q_{tot})^2 + \beta\frac{1}{n}\sum_{a=1}^{n}\sum_{u_a}(Q_a - \log(\sum_{u_a} Q_a)S(Q_a)(1-S(Q_a)) \tag{3.13}$$

[6] $\log_c\left(\frac{a}{b}\right) = \log_c(a) - \log_c(b)$
[7] $\log_a\left(x^b\right) = b \cdot \log_a(x)$

Chapter 4

Results

On this chapter, first there will be a brief explanation of the setting that have been used to run the experiments, then a small description of the maps that have been tested during the project. Next will be described how each graph have been computed. Finally it will be shown the results of the different maps using the techniques detailed on chapter 3.

4.1 Exploration Settings

As mentioned on 3.1, SMAC is based on the game StarCraft 2. To be able to have a fair comparison, it needs to be tested concretely on the `SC2.4.6.2.69232` version instead of the new `SC2.4.10`. Also, the allies need to be training against the built-in AI set to difficulty 7. Regarding the characteristics of the hyper-parameters used for training, the authors use the values shown on the following table:

Hyperparameter	**Value**
Discount factor	0.99
Replay buffer size	5000 episodes
Batch size	32 episodes
Learning rate	5×10^4
Optimizer	RMSprop
Double DQN update	True

Table 1: SMAC parameters used for training

Its important to point that, even that these parameters have been used to have a comparison over the results presented on [4], they are far of being optimized. On [22] they re-tune the parameters of SMAC to accomplish better results.

4.1.1 Exploration parameters and variations

Here there will be described how the different exploration approaches have been configured and tested during the project. The majority of the methods have been tested with analogous parameters to ϵ-greedy exploration, to be able to compare the results fairly.

Epsilon greedy

For the ϵ-greedy exploration, the parameter ϵ is annealed linearly from 1.0 to 0.05 over 50k time steps and is kept constant for the rest of the learning. As showed on the following figure:

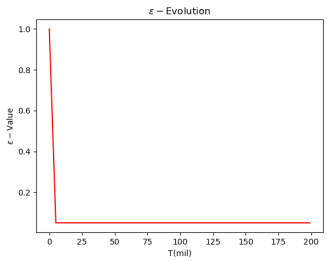


Figure 7: Anneal of the parameter ϵ during training

Boltzmann

Similarly to $\epsilon-$greedy, the value of τ is annealed linearly from 1000.0 to 0.05 over the first 50k time steps and is kept constant as well for the rest of the training. This values have been selected to be as analogous as possible with the values used for the ϵ- greedy exploration. The initial value need to be extremely high to assure a complete random selection at the beginning of the training.

Entropy Regularization

To change the performance of the entropy regularization, there are defined two parameters, β and σ. The first parameter β will describe how important is to the learning be conditioned to the entropy of the $Q-$values. The value of β will behave the same as ϵ in epsilon greedy exploration. Additionally σ is used to adjust the stochasticity of the soft-max mapping done before computing the entropy. There are three variations regarding how σ is computed. The first variation maintains σ constant $\sigma = 1$. On the second one $\sigma = \beta$. On the third one the soft-max is computed as:

$$S(Q_a\left(\tau^a, u_j^a; \theta\right)) = \frac{e^{Q_a\left(\tau^a, u_j^a; \theta\right)\cdot\frac{\beta}{\sigma}}}{\sum_{u^a} e^{Q_a(\tau^a, u^a; \theta)\cdot\frac{\beta}{\sigma}}} \tag{4.1}$$

Where β follows the same transformation as ϵ on the epsilon greedy approach and σ decay lineally from 1000 to 1.0 on the first $50k$ time-steps. Here the new factor will be called $\nu = \frac{\beta}{\sigma}$. Note that the evolution of β is the same for all variations, on the second and the third one only influence on the stochasticity of the soft-max mapping. On A.1 is detailed the behaviour and the selection of the parameters.

4.2 Maps description

On the SMAC environment the are different maps to test MARL algorithms. Since the aim of the project is to make a fair comparison with the results presented on [4] the maps that have been tested are the ones on which they analyse their results: `3s5z, 2c_vs_64zg,`and `MMM2.` Additionally the map `corridor` has been tested as well since is the most difficult one.

SMAC 3s5z: In this symmetric scenario, each team controls three Stalkers and five Zerglings for a total of eight agents.

SMAC 2c_vs_64z: In this asymmetric scenario, the ally team controls two Colossi that will battle against 64 Zealots.

SMAC MMM2: In this symmetric scenario, each team controls seven Marines, two Marauders, and one Medivac unit. The medivac unit assists other team members by healing them instead of inflicting damage to the enemy team.

SMAC corridor: In this asymmetric scenario, agents control six Zealots fighting an enemy team of 24 Zerglings controlled by the game. This tasks requires agents to make effective use of terrain features and employ certain game-specific tricks to win.

In table 2 there are simplified the main characteristics of each map.

Name	Ally Units	Enemy Units	Type	Difficulty
3s5z	3 Stalkers 5 Zealots	3 Stalkers 5 Zealots	Homogeneous, Symmetric	Easy
2c vs 64z	2 Colossi	64 Zealots	Heterogeneous, Asymmetric, Large action space	Hard
MMM2	1 Medivac 2 Marauders 7 Marines	1 Medivac 2 Marauders 7 Marines	Homogeneous, Asymmetric, Macro tactic	Super Hard
corridor	6 Zealots	24 Zerglins	Heterogeneous, Asymmetric, Large action space	Super Hard

Table 2: Starcraft Maps Challenges

4.3 Evaluation Metrics

To have a equitable comparison with the results presented by [4] their same settings have been used. The training is paused after every 10000 timesteps and 32 test episodes are tested using a decentralized greedy selection over the action-value function. Each experiment is run independently 5 times with the plots showing the median performance as well as the 25-75 % percentiles. Also mention that the results have been smoothed.

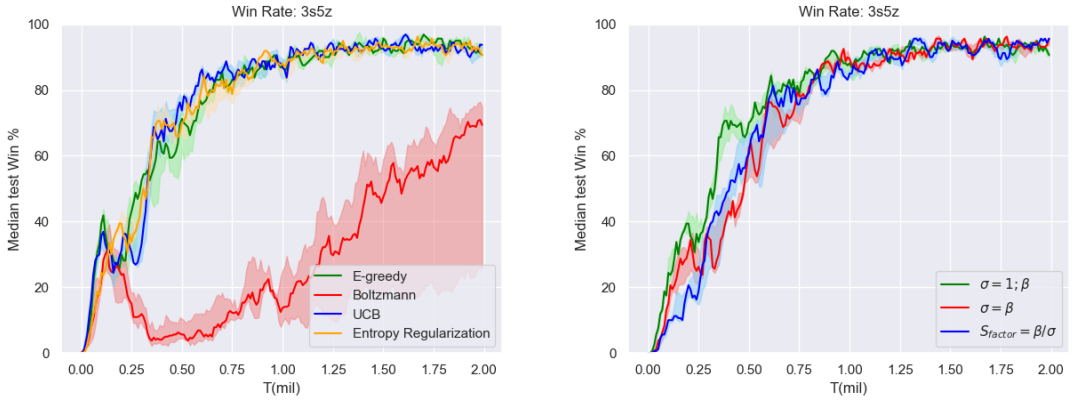
The hardware that has been used is composed by a GPU Quadro RTX 8000 and a CPU Intel(R) Xeon(R) Gold 5218R. Running different experiments independently

in parallel takes between 15-25 hours.

4.4 SMAC Results

On this section there will be displayed how the different exploration approach with their variation affect the learning in the different maps. For the plots where all the different methods are shown, the entropy regularization use the first variation described on 4.1.1, that is, applying the normal soft-max operation and reducing β as the epsilon-greedy exploration. Need to mention that the results of this work focus on the velocity of the learning rather than on the longer results, that is why all the experiments have been tested only in two million time-steps. This is done basically to have a fair comparison with the results presented on [4].

4.4.1 Easy map: 3s5z

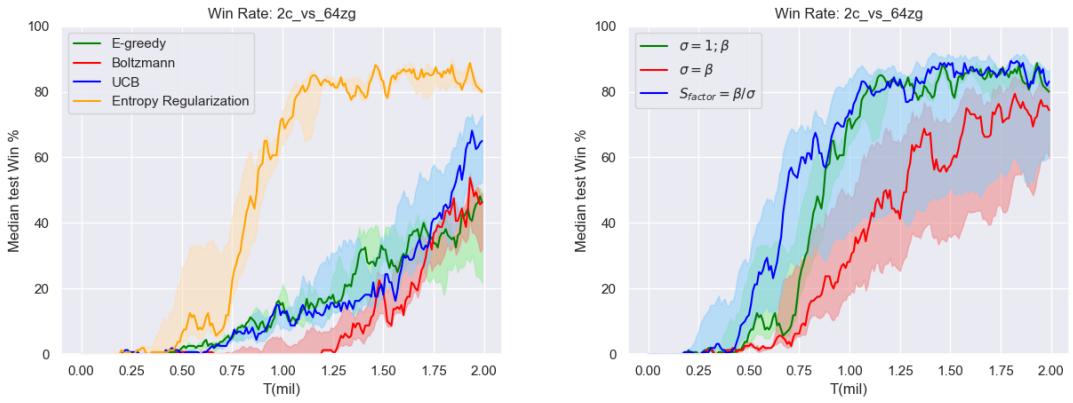


(a) All the techniques results (b) All variations entropy regularization

Figure 8: Median Win Rate of the different techniques on the 3s5z map

In this easy scenario, almost all the methods performs similarly (Figures 8a and 8b). Exceptionally, the use of the Boltzmann approach seems to works differently, delaying the learning. This may be caused by the fact that the finishing value of ρ is too small and the action selection is almost greedy, which may difficult the learning.

4.4.2 Hard map: 2c_vs_64zg



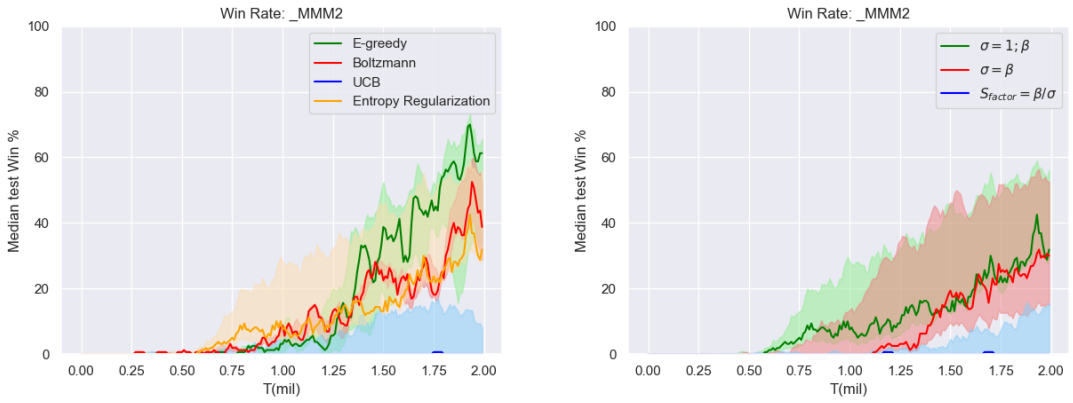
(a) All the techniques results (b) All variations entropy regularization

Figure 9: Median Win Rate of the different techniques on the 2c_vs_64zg map

In this hard map, the exploration that rely on a random selection criteria of the actions ($\epsilon-$greedy and Boltzmann) seems to perform worst than the other techniques (Figure 9a) . The UCB1 approach outperforms a bit the random selection of the actions. Since UCB1 makes sure to try each action, in this case, the two agents can explore better their large action-space.

Regarding to the entropy regularization, it seems to outperform all previous methods in this case. Given this large action space it may be easy to get stuck in local minima. Since the entropy regularization approach works pretty well to avoiding this problem, it may help the agents choose faster the more optimal actions. Respecting the variations of the entropy regularization (Figure 9b), the second variation seems to have a delayed learning, maybe caused by the fact that the loss is bigger than the other variations, putting off the learning. On the contrary the third variation act the opposite way, during training the entropy is smaller, being sufficient in this case to learn faster.

4.4.3 Super hard map: MMM2



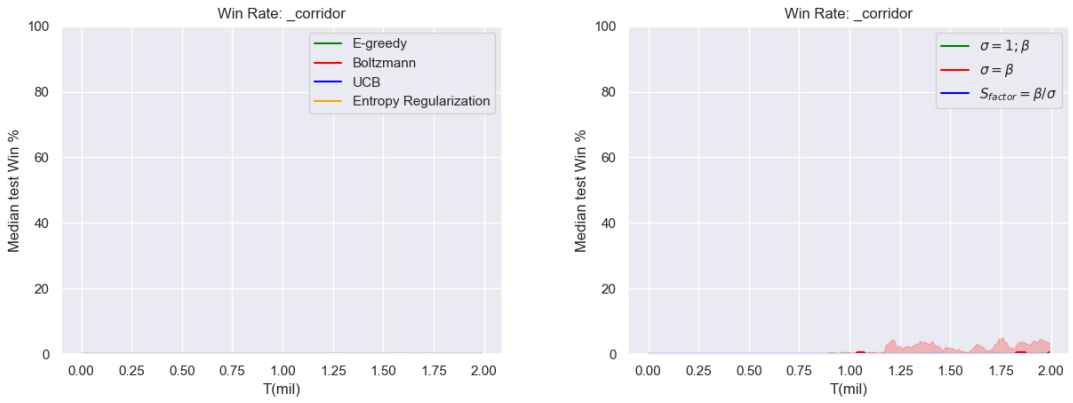
(a) All the techniques results (b) All variations entropy regularization

Figure 10: Median Win Rate of the different techniques on the MMM2 map

In this super hard scenario the UCB1 approach perform really poorly (Figure 10a. Since in this map the agent needs to perform more complex tactics and UCB1 chose actions based on how many times an action have been selected, it may not have the opportunity to explore more sophisticated behaviour. On the contrary, all the other approaches seem to work similarly. The Boltzmann and ϵ-greedy exploration behave very much alike, the random exploration seems to give to the agent enough experience at the beginning of the training to find a good policy.

Regarding the entropy regularization, by encouraging the agent to not get stuck in known actions, also performs pretty well. However in Figure 10b can be seen how the third variation perform pretty poorly. As mention on 4.4.2, this variation is the one that makes the entropy smaller, difficulting the learning.

4.4.4 SUPER hard map: corridor



(a) All the techniques results (b) All variations entropy regularization

Figure 11: Median Win Rate of the different techniques on the corridor map

Corridor is considered the most difficult map. It can be appreciate by the Figure 11a that almost none of the exploration techniques are able to learn even a bit in this environment. This is related to the fact that for this map the agent needs to learn how to use the terrain features correctly and employ certain game-specific techniques. The fact the second variation of the entropy regularization (Figure 11b) is able to learn a little bit can be explained by the fact that at the beginning, since σ behaves equally as β, it forces the entropy to decrease over time, which means that the loss will increase. Since at the same time the contribution of the entropy will be decreased by β, the influence of the entropy in the loss during the first steps will be almost constant, forcing the agent to explore more options.

Chapter 5

Discussion

In this work different exploration methods have been tested over the original QMIX implementation. It has been seen how the majority of the approaches appear to behave similarly in all the maps, but exceptionally the Entropy Regularization seems to outperform the other methods on the 2c_vs_64zg map (Figure 9). As mentioned in 4.4.2 a possible explanation could be that the entropy regularization exploration approach provides to the loss a parameter that teaches the network to avoid very confident predictions by taking actions more unpredictably and helping to avoid getting stuck in local minima. This explanation makes sense for this map, since the action space is very large, and it may by easy to learn a policy locally optimal. In corridor (Figure 11), the map that is being considered the most difficult one, the entropy regularization seems to have some effect over the learning as well, possibly caused as well by the large action space.

UCB1 seems to perform poorly on the super hard scenarios. This may be caused because of the macro-management component. Since UCB1 force the agent to try all the actions, may take a while for the agent to learn the policy for these situations and stochasticity may help to learn faster. Finally the ϵ-greedy and the Boltzmann exploration seems to behave similarly to all the maps except for the easy one, which Boltzmann seems to learn pretty slowly.

5.1 Future Work

Through this thesis there have been tested different Exploration Techniques on the QMIX algorithm. However, it could be interesting to try other and more complex methods such as Noise-based Exploration or Maximization for Long-term Entropy [23]. Also, given the nature of QMIX, could be interesting to explore the idea of some exploration approach that could take advantage of the richer representation of the Q_{tot} that QMIX offer. Finally, it would be interesting to try exploration approaches more centered on the multi-agent context rather than the individual agent paradigm. Within these methods, we can find a way of regularizing the anneal of the parameters independently for each agent or to apply variance on the UCB.

List of Figures

List of Tables

Bibliography

[1] Cao, Y., Yu, W., Ren, W. & Chen, G. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics* **9**, 427–438 (2013).

[2] Hüttenrauch, M., Šošić, A. & Neumann, G. Guided deep reinforcement learning for swarm systems (2017). `1709.06011`.

[3] Canese, L. *et al.* Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences* **11** (2021).

[4] Rashid, T. *et al.* Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research* **21**, 1–51 (2020).

[5] Coggan, M. Exploration and exploitation in reinforcement learning (2004).

[6] Samvelyan, M. *et al.* The StarCraft Multi-Agent Challenge. *CoRR* **abs/1902.04043** (2019).

[7] Silver, D. Deep reinforcement learning. *Deepmind* (2016).

[8] Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (John Wiley & Sons, Inc., USA, 1994), 1st edn.

[9] Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* (The MIT Press, 2018), second edn.

[10] Oliehoek, F. A., Spaan, M. T. J. & Vlassis, N. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research* **32**, 289–353 (2008).

[11] Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, 330–337 (Morgan Kaufmann, 1993).

[12] Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N. & Whiteson, S. Counterfactual multi-agent policy gradients. *CoRR* **abs/1705.08926** (2017).

[13] Sunehag, P. *et al.* Value-decomposition networks for cooperative multi-agent learning (2017). 1706.05296.

[14] Oliehoek, F. A. & Amato, C. A concise introduction to decentralized pomdps 14–17 (2016).

[15] Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).

[16] Hausknecht, M. & Stone, P. Deep recurrent q-learning for partially observable mdps (2017). 1507.06527.

[17] Ha, D., Dai, A. & Le, Q. V. Hypernetworks (2016). 1609.09106.

[18] Samvelyan, M. *et al.* The StarCraft Multi-Agent Challenge. *CoRR* **abs/1902.04043** (2019).

[19] Son, K., Kim, D., Kang, W. J., Hostallero, D. E. & Yi, Y. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning (2019). 1905.05408.

[20] Weng, L. The multi-armed bandit problem and its solutions. *lilianweng.github.io/lil-log* (2018).

[21] Duchi, J. C. Cs 229 supplemental lecture notes hoeffding ' s inequality (2016).

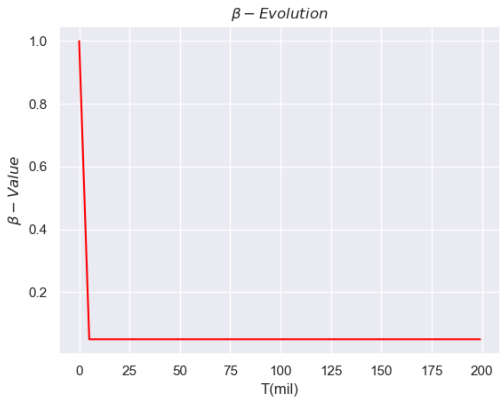
[22] Hu, J., Jiang, S., Harding, S. A., Wu, H. & wei Liao, S. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning (2021). 2102.03479.

[23] Schulman, J., Chen, X. & Abbeel, P. Equivalence between policy gradients and soft q-learning (2018). 1704.06440.

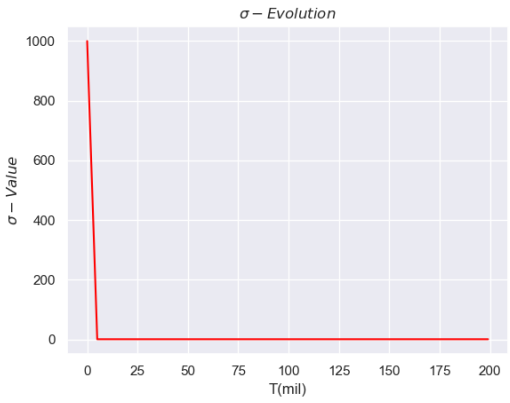
Appendix A

First Appendix

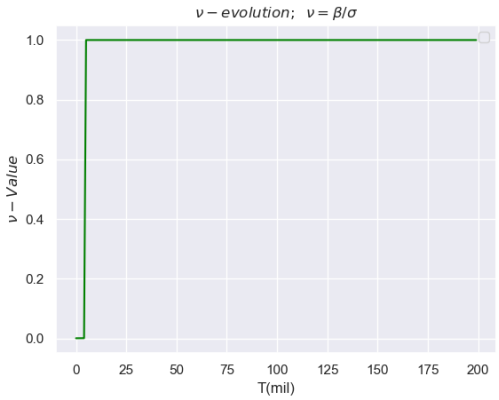
A.1 Entropy Regularization Exploration parameters plots



(a) Evolution of β for all the variations.



(b) Evolution of σ on the third variation



(c) Evolution of ν on the third variation

Figure 12: Visual representation of the main changes on the Entropy Regularization. For figures (a) and (b) they decay lineally on the first 50k steps. On figure (c) the value of ν change abruptly on the 50k step form 0 to 1.

A.1.1 Explanation of the different variations for entropy regularization

- On the first variation the stochasticity of the soft-max mapping only depend on the raw $Q-$values.
- On the second variation, σ follows the same evolution as β. This cause the output of the soft-max to be less stochastic over time, which means that the entropy will be decreasing its value over the first $50k$. In consequence the loss will increase.
- On the third variation, on the first $50k$ time-steps, the entropy will be as small as possible, which will not make almost any change to the loss, making it as large as possible. As long as the learning goes by on the first $50k$ the output of the soft-max will be more stochastic and that will force for the entropy to be larger, making the loss smaller.

A.2 QMIX Algorithm

Algorithm 1 QMIX

Initialise θ, the parameters of mixing network, agent networks and hypernetwork.
Set the learning rate α and replay buffer $\mathcal{D} = \{\}$
step $= 0, \theta^- = \theta$
while *step* $<$ *step*$_{max}$ **do**
    $t = 0, s_0 =$ initial state
    while $s_t \neq$ terminal and $t <$ episode limit **do**
        for each agent a **do**
            $\tau_t^a = \tau_{t-1}^a \cup \{(o_t, u_{t-1})\}$
            $\epsilon =$ epsilon-schedule(step)
            $u_t^a = \begin{cases} \text{argmax}_{u_t^a} Q\left(\tau_t^a, u_t^a\right) & \text{with probability } 1-\epsilon \\ \text{randint}(1, |U|) & \text{with probability } \epsilon \end{cases}$
        end for
        Get reward r_t and next state s_{t+1}
        $\mathcal{D} = \mathcal{D} \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1})\}$
        $t = t + 1$, step = step $+1$
    end while
    if $|\mathcal{D}| >$ batch-size **then**
        b $\leftarrow$ random batch of episodes from $\mathcal{D}$
        for each timestep t in each episode in batch b **do**
            Q_{tot} = Mixing-network $\left(Q_1\left(\tau_t^1, u_t^1\right), \ldots, Q_n\left(\tau_t^n, u_t^n\right)\right.$; Hypernetwork $\left.(s_t; \theta)\right)$
            Calculate target Q_{tot} using Mixing-network with Hypernetwork $(s_t; \theta^-))$
        end for
        $\Delta Q_{\text{tot}} = y^{\text{tot}} - Q_{\text{tot}}$ (2.3)
        $\Delta\theta = \nabla_\theta \left(\Delta Q_{\text{tot}}\right)^2$
        $\theta = \theta - \alpha \Delta\theta$
    end if
    if update-interval steps have passed **then**
        $\theta^- = \theta$
    end if
end while

A.3 Other Results

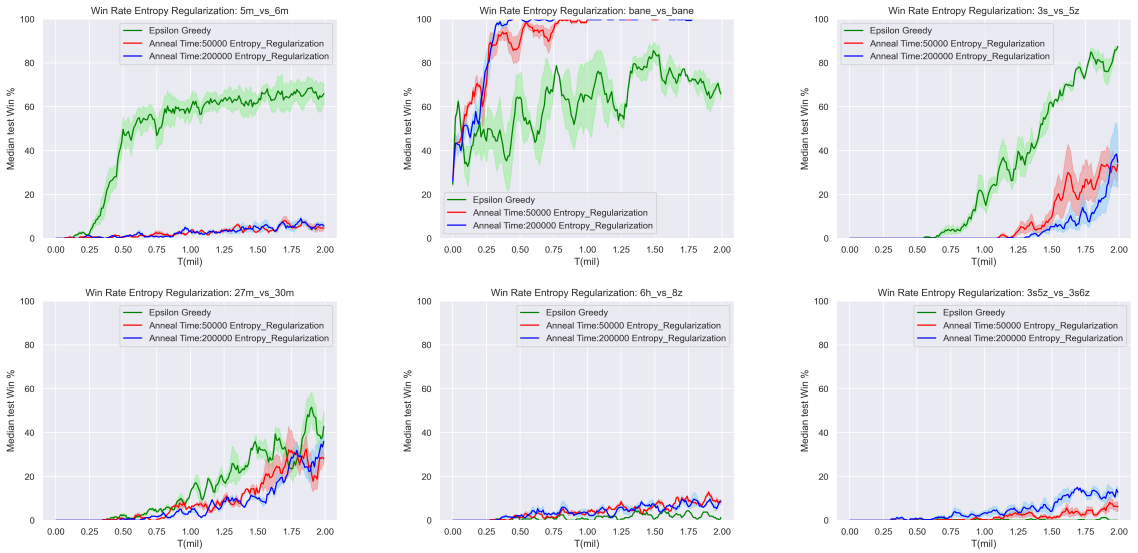


Figure 13: Entropy regularization Variation 1 and $\epsilon-$greedy exploration for different hard and super hard maps. Its seen that on the bane_vs_bane map how the entropy regularization outperform the learning of $\epsilon-$greedy. Also for the super hard maps 3s5z_vs_3s6z and 6h_vs_8z seems to work better. For the rest of the maps it perform relatively poorly compared to the ϵ-greedy exploration.