Master in Intelligent Interactive Systems

Universitat Pompeu Fabra

Learning Non-Linear Payoff Transformations in Multi-Agent Systems

Emma Fraxanet

Supervisor: Vicenç Gómez

September 2021

Master in Intelligent Interactive Systems

Universitat Pompeu Fabra

Learning Non-Linear Payoff Transformations in Multi-Agent Systems

Emma Fraxanet

Supervisor: Vicenç Gómez

September 2021

Contents

Acknowledgement

I would like to express my sincere gratitude to:

- My supervisor Vicenç Gómez, with whom I will have the pleasure to continue working during the new stage of my research career.

- My family and friends for the unconditional support and encouragement, especially when the effects of this past COVID-19 pandemic year have been hard to navigate. On another note, I also want to make evident that without the financial support of my parents and my uncle I might not have had the privileges and opportunities to continue my interest and disposition towards an academic career, and for that I want to thank you, since it is one of my identity and fulfillment pillars.

Abstract

The use of Deep Reinforcement Learning methodologies has been successful in recent years in cooperative multi-agent systems. However, this success has been mostly empirical and there is a lack of theoretical understanding and solid description of the learning process of those algorithms. The discussion of whether the limitations of these algorithms can be tackled with tuning and optimization or, contrarily, are constrained by their own definition in these models can also easily be put forward. In this work, we propose a theoretical formulation to reproduce one of the claimed limitations of Value Decomposition Networks (VDN), when compared to its improved related model QMIX, regarding their representational capacity. Both of these algorithms follow the centralized-learning-decentralized-execution fashion. For this purpose, we scale down the dimensions of the system to bypass the need for deep learning structures and work with a toy model two-step game and a series of one-shot games that are randomly generated to produce non-linear payoff growth. Despite their simplicity, these settings capture multi-agent challenges such as the scalability problem and the non-unique learning goals. Based on our analytical description, we are also able to formulate a possible alternative solution to this limitation through the use of simple non-linear transformations of the payoff, which sets a possible direction of future work regarding larger scale systems.

Keywords: Reinforcement Learning; Multi-Agent Learning; Action-Value representation; One-shot games

Chapter 1

Introduction

Machine Learning algorithms are being used nowadays in a broad range of daily applications. A specific branch of Machine Learning is Reinforcement Learning (RL), which is based on the training of an agent to make proper sequential decisions in a given environment in the presence of uncertainty. This has applications in real-time strategy games, robotic control, autonomous driving and many other fields. One of the most known advances was the computer success in the game of Go [1]. These kind of settings usually call for the need of more than one agent, giving birth to the multi-agent RL (MARL) settings. Multi-agent settings can also be applied in other areas such as sensor and communication networks, social sciences, and finances. Modelling these systems faces additional challenges given their complexity, large dimensions and variability in communication structures.

Currently, there is a large interest and work focused on addressing these challenges, and the improvement of single-agent RL, such as the use of Deep Learning, is also boosting the research in multi-agent systems. In this context, most of the progress in Multi-agent Deep RL is driven by empirical success, and it is often difficult to disentangle which design principles determine when one algorithm is superior to another one.

This thesis is pointed at theoretically analysing with simple models the representational complexity allowed by two of the state-of-the-art algorithms in fully coop-

erative MARL. Through the use of toy models, a one-step sequential decision task is analyzed and an alternative way to learn an optimal policy based on learning non-linear monotonic rewards is proposed.

In this first chapter, the needed theoretical basis and formulation is presented in order to allow the understanding of the used methods throughout the thesis. In the second part, the detailed formulation of the problem and settings is presented, and a brief intuition of the proposed solution is introduced. Finally the results and proposed solutions are displayed and described in the third chapter, and properly discussed and summarized in Chapter 4.

1.1 Motivation and objectives

What this thesis tries to tackle is a theoretical analysis of a simple question: How to represent and learn non-linear growing reward functions in multi-agent reinforcement learning settings. We could find similar shaped payoffs in non-factored games such as the Climb Game [2], in which, having each agent three possible actions, there is a specific action that leads to the highest reward only if all agents coordinate to choose it. If only some of the agents choose such action there is no reward, and the choice of the other two actions provide lower rewards but don't require coordination. If the reward for the optimal action is much larger than the other sub-optimal choices, systems will tend to underestimate the best obtainable reward.

This motivation originates in a claim made in the *Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning* paper [3], by Rashid,T. *et al*, where their algorithm's representation complexity is declared to be the reason their algorithm (QMIX) performs better for non-linear growing payoffs than the *Value-Decomposition Networks*(VDN) algorithm. This is also supported by the use of a toy model two-step game and an experiment with random matrices, which will also be presented and analysed in this thesis.

At the same time, both of these algorithms make use of *Deep learning* to represent the values functions, which makes difficult to understand and compare both methods

theoretically. A use of such entangled structures leads to less understanding of the model and to a higher need for computational power. In this thesis a simpler alternative for the claimed improvement in complexity representation is also developed, though only at its first stages and in the form of a method proposal.

Therefore, the motivation is to analyse this claim and propose a theoretical formulation that is able to explain it, as well as proposing a new method to deal with non-linear monotonic payoff behaviours. This is done with the use of simple non-linear transformations of the payoff that aims to feed the learning system a more linear payoff, and then recover the actual shape again once it has been learned.

This motivation shares similarities to the paper *Analysing factorizations of action-value networks for cooperative multi-agent reinforcement learning*, by Castellini, J. [2] that appeared at the time when this thesis was in its last stages. Their work uses one-shot game settings to understand the representational capacity of different factored structures of the Q-value function. The format of their results and overall objectives is fairly similar to this work, and provides a very instructing and comprehensible reading. While this project tackles the specific problem of non-linear payoffs, their work is more extensive in terms of the analysed games and uses deep learning algorithms.

It is worth to note that other methodologies were considered at the start of the thesis. For example, working with different Loss functions such as the logistic Bellman error [4] in dynamic programming, in order to avoid local minima when optimizing expected loss. This line of work could still be explored but it was found to extend the project too much beyond its temporal restraints.

1.2 Reinforcement Learning and MDPs

Reinforcement learning is based on sequential decision-making problems in a setting where an agent interacts with an uncertain environment and needs to learn by interacting with such environment how to improve its performance.

In episodic RL, an agent at state $s_t \in \mathcal{S}$ at time t takes an action a_t in $\mathcal{A}(s_t)$ and

obtains a reward for this execution $R(s_t, a_t, s_{t+1}) \in \mathbb{R}$ before transferring to a new state $s_{t+1} \in \mathcal{S}$ according to the state transition probability distribution $p(s_{t+1}|s_t, a_t)$. The sets $\mathcal{S}$ and $\mathcal{A}(s_t)$ correspond to the state space and the space of allowed actions in a given state, respectively. This process is repeated T steps until an episode ends.

A general mathematical framework for formalizing this setting is the infinite-horizon discounted Markov Decision Process (MDP) [5], in the case the agent can fully observe the state. In case the agent can only access a few states of the system, hence is subjected to partial observability, a POMDP model would represent the problem better [6]. There are also other formulations for MDPs: finite-horizon episodes, average reward, etc. For simplicity, we consider the standard infinite-horizon, discounted MDP.

The goal in this setting is to obtain a mapping from the state space $\mathcal{S}$ to the action space $\mathcal{A}$ (referred as a policy $\pi : \mathcal{S} \to \mathcal{A}$) so that the expected accumulated discounted reward

$$\mathbb{E}\left[\sum_{t\geq 0} \gamma^t R\left(s_t, a_t, s_{t+1}\right) \mid a_t \sim \pi\left(\cdot \mid s_t\right), s_0\right], \tag{1.1}$$

is maximized. The discount factor $\gamma \in [0, 1]$ ensures that closer rewards (in time) are more relevant distant ones, and that the sum in (1.1) remains finite.

According to this goal, one can define both the Value function and the action-Value function, or Q-value function for each state s and action a under a specific policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t, s_{t+1}\right) \mid a_t \sim \pi\left(. \mid s_t\right), s_0 = s\right] \tag{1.2}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t, s_{t+1}\right) \mid a_t \sim \pi\left(. \mid s_t\right), s_0 = s, a_0 = a\right] \tag{1.3}$$

The functions linked to the optimal policy π^*, obtained by maximizing over the actions, are the optimal state-value function and optimal Q-value function.

Assured by the Markov property, where it is stated that the probability of a future

state only depends on the immediate past state and not the whole previous sequence, one can obtain the optimal policy by means of learning the optimal Q-value function through dynamic programming approaches. These approaches are strongly based on the Bellman equations formulation:

$$V^{\pi}(s_t) = \sum_{a \in \mathcal{A}(s_t)} \pi(a \mid s_t) \sum_{s' \in \mathcal{S}} p(s' \mid s_t, a) \left[r(s_t, a) + \gamma V^{\pi}(s') \right] \tag{1.4}$$

And the optimal versions, by maximizing over the actions, take the form of the Bellman Optimality equations:

$$V^{\pi^*}(s_t) = \max_a \sum_{s'} p(s' \mid s_t, a) \left[r(s_t, a) + \gamma V^{\pi^*}(s') \right] \tag{1.5}$$

$$Q^{\pi^*}(s_t, a_t) = \sum_{\mathbf{c}'} p(s' \mid s_t, a_t) \left[r(s_t, a_t) + \gamma \max_{a'} Q^{\pi^*}(s', a') \right] \tag{1.6}$$

These are all for the case where the environment is known i.e. we know the transition probability distribution $p(s_{t+1}|s_t, a_t)$ and the reward for all states and actions $r(a, s)$. However, in real life, this is not true. Reinforcement Learning is based on finding the optimal policy in these unknown environments. The RL algorithms usually work by sampling tuples of state, action and reward, while learning the policy.

These algorithms can be value-based, their aim is to estimate the state-action value function correctly, or policy based methods, which work on the policy directly by parametrizing and optimizing it. A common practise, since the scale of the system tends to increase and the state and action spaces reach very large dimensions, is to use function approximation [7]. This approach basically tries to learn parameters of a function by supervised learning in order to decrease the dimensions of the problem. Then, the function that needs to be maximized is:

$$J(\theta) = \mathbb{E}_{a \sim \pi_\theta(.|s), s \sim \rho_{\pi_\theta}} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t; \theta), \tag{1.7}$$

Where the expectation is taken for the actions and distribution of occupied states.

Some of the most common value-based methods are *Q-learning* [8], *SARSA* and *Monte Carlo Tree Search* (MCTS). Only *Q-learning* will be explained for its relevance for the upcoming sections. In the same way, no policy-based methods will be presented.

Q-learning is a method of asynchronous dynamic programming that is closely related to the method of temporal differences [8]. It is off-policy, which means it uses different policies to evaluate and generate new samples. It is based on performing updates (1.8) on the Q-value function when receiving a new sample of transition from a state s to the next one s' with an action a and a reward r. With $\alpha > 0$ as the learning rate, one can see within finite action and state spaces that Q-learning can converge towards the optimal Q-value function.

$$\hat{Q}(s,a) \leftarrow (1-\alpha)\hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}\left(s', a'\right) \right] \tag{1.8}$$

1.3 MARL

Multi-agent RL is also based on sequential decision-making problems but in this case it has to capture the interaction and decisions of multiple agents. The fact that different agents work in the same environment directly affects the state evolution and individual rewards, which will then depend on the combination of actions taken by such agents. The optimal solution will also depend on whether the agents seek to maximize their own individual reward or try to work together. In this line a way to classify multi-agent settings can be *fully cooperative*, where there is a general reward for all agents or a team-average reward, *competitive setting*, which is modelled as zero-sum Markov games [5], or *mixed setting*, where agents only seek their own benefit. For the description of the project only fully cooperative settings will be considered.

Under other perspectives, however, there are more ways to classify a multi-agent

setting. For example, there are different information structures that describe the level of information the agents have on the environment or about other agent's states or actions. Different cases would be: *centralized settings*, where a central controller can use the information of the agents at all states and design the optimal policies, *decentralized settings*, where there is no information exchange between agents and no central controller, or a mix between both, *decentralized setting with networked agents*, where there is some communication between agents but no central controller... Regarding this classification, a split between the information state during learning and execution is also possible: in the models described in this project, we will work with *centralized-learning-decentralized-execution*.

The tuple $G = \langle S, A, P, r, Z, O, n, \gamma \rangle$ can fully describe a *decentralized partially observable Markov decision process* (Dec-POMDP) suitable for our multi-agent fully cooperative setting. The notation used is selected from [3]. Where $s \in S$ describes the full state of the environment, $\vec{a} \in A$ describes the joint-action vector formed by the individual action choices of each agent $i \in I \equiv \{1, ...n\}$, the action taken by agent i is then a^i. P is the environment state transition function, the reward function $r(s, \vec{a})$ describes the reward shared by all agents and γ is a discount factor as in the single-agent case. $O(s, i) : S \times I \rightarrow Z$ is the observation function that defines the space Z of individual observations $z \in Z$.

Actions are decided and executed simultaneously for all agents. Therefore new observations are perceived at the same time as well for all agents, as the environment has changed according to the transition of states. Each agent has a history that takes into account previous actions and transition states, τ^i. When working with centralized learning and decentralized execution, the stochastic policy of each agent $\pi^i(\vec{a}^i \mid \tau^i)$ can only access their own history, but when learning the model can also use the global state as well as all agent's histories. The joint policy is defined by the global Q-value function $Q^\pi(s_t, \vec{a}_t) = \mathbb{E}_{s_{t+1:\infty}, \vec{a}_{t+1:\infty}} [\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t, \vec{a}_t]$.

1.3.1 Challenges

MARL faces several problems in its theoretical definitions, besides the challenges single-agent systems might already suffer from. In [5] a summary and detailed explanation of these challenges is proposed in order to point at possible improvements for developing new models. We revise these challenges in order to identify the obstacles the models we will go through might face.

- *Non-unique learning goal*: It refers to the unclarity of what the learning goal in that system is. The goal is usually the convergence to Nash equilibrium, which is characterized by an equilibrium point where none of the agents have the need to change their state. In the systems we analyse (VDN and QMIX) the main goal is to maximize the joint reward, and therefore the global optimum defines a Nash equilibrium.

- *Non stationarity*: Since the learning is simultaneous, the variation of the environment at each step of the policy leads to the observation of the environment from each agent to be non-stationary. Therefore, the Marokivan property does not apply anymore and convergence is not established like in single agent anymore. A solution would be that the agents assume a stationary environment nevertheless, which is called an *Independent Learner*, but this leads to problems in converging. In this work we are not affected by this drawback since we focus on single and two step games, as it will be seen in Chapter 2.

- *Scalability*: This challenge refers to the combinatorial nature of joint action space, which grows exponentially in dimension when adding agents to the setting. That is why some models use factorized structures, like the ones covered in the next section. This challenge is also addressed with the use of theories for deep multi-agent RL, which at the same time uses function approximation.

- *Different information structures*: As mentioned, several information structures can be used when building a model, that is who knows what in learning and execution. The fact that some structures consider partial observation worsens the aforemen-

tioned problem of non-stationarity. In this work we are not affected by this drawback since we focus on single and two step games and their learning, as it will be seen in Chapter 2.

1.3.2 Deep Q-Learning

Deep-Q-Learning [9] is a method that uses deep neural network parameters to describe the Q-value function. It is based in the same idea as Q-learning, that is why it is named after it. In *Deep Q-Networks* (DQNs) θ is used to parametrize the Q-value function and is learned through minimizing the *TD error* (1.9). A *replay memory* buffer is used to store tuples of state, action, reward and transition state, and batches of b transitions from this buffer are used to learn the parameter θ. The use of a buffer helps randomize the data. A *target network*, with parameters θ^-, is used to only adjust the Q-value function towards target values that are periodically updated, since this can help reduce correlations.

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(r + \gamma \max_{u'} Q\left(s', u'; \theta^-\right) - Q(s, a; \theta) \right)^2 \right] \tag{1.9}$$

For agents that have partial observations, it is interesting to use their action-observation history. *Deep Recurrent Q-Networks* (DRQN) consider the prior knowledge [10], and can be used with that purpose. The architecture is similar to DQNs but replacing only its first fully connected layer with a recurrent *Long Short-Term Memory*(LSTM) layer of the same size.

1.3.3 State-of-the-art relevant models

The two relevant models for this project are Value Decomposition Networks (VDN) and QMIX. Both consider a fully cooperative setting with centralized learning and decentralized execution, and also use Deep Q-learning structures. The notation introduced in section 1.3 is used. The two models differ in how the global Q-value function is defined regarding the individual Q-value functions.

In VDN the joint Q-value is the sum of each agent's Q-value function, and therefore can be additively decomposed. These individual Q-value functions are conditioned only on the specific agent's observation history. The obtained policy is decentralized, since each agent chooses the action greedily with respect to its own Q-value function.

$$Q_{tot}(\boldsymbol{\tau}, \vec{a}) = \sum_{i=1}^{n} Q_i\left(\tau^i, a^i; \theta^i\right) \tag{1.10}$$

As mentioned in [11]: "We learn Q_i by backpropagating gradients from the Q-learning rule using the joint reward through the summation, i.e. Q_i is learned implicitly rather than from any reward specific to agent i, and we do not impose constraints that the Q_i are q-functions for any specific reward". Therefore, the optimal linear value decomposition is learned from the team reward signal. Q_i are represented with Deep Neural Networks.

QMIX [12] has a similar structure to VDN but instead of using the full factorization of VDN a monotonicity constraint is applied to assure consistency between the decentralized policies and the centralized policy based on the optimal joint Q-value function (1.11).

$$\underset{\vec{a}}{\text{argmax}} Q_{tot}(\boldsymbol{\tau}, \vec{a}) = \begin{pmatrix} \text{argmax}_{a^1} Q_1\left(\tau^1, a^1\right) \\ \vdots \\ \text{argmax}_{a^n} Q_n\left(\tau^n, a^n\right) \end{pmatrix} \tag{1.11}$$

Even if both models can satisfy (1.11) QMIX has a larger representational complexity, since it can represent combinations of individual Q-value functions that are montonic but non-linear. Monotonicity is ensured with:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i \in I \tag{1.12}$$

The structure of QMIX is the following:

- *Agent Networks*: DRQNs that represent individual Q-functions. The input is the current individual observation and last action at each time step and output is the Q_i.

- *Mixing network*: Feed Forward Neural Network. Individual Q_i as input and output is the non-linear mixing of those.

- *Hypernetworks*: They provide the weights for the mixing networks and ensure positive weights (monotonicity). The system's state s is used as input in the hypernetworks because the Q-value function can depend on it in a non-monotonic way.

See Fig.1 to see the overall structure and combination of the different parts and Fig.2 for the pseudocode of the implementation of QMIX. This examples will be recovered for a modification proposal in the discussion.

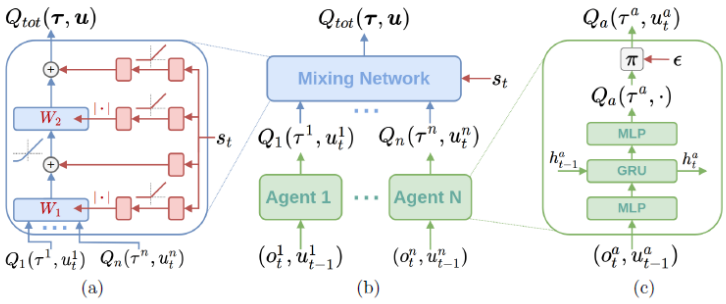

Figure 1: Structure of QMIX (from [3]). (a) Mixing network structure, in red are the hypernetworks. (b) General structure. (c) Agent network structure.

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($step$)
      $u_t^a = \begin{cases} \text{argmax}_{u_t^a} Q(\tau_t^a, u_t^a) & \text{with probability } 1-\epsilon \\ randint(1, |U|) & \text{with probability } \epsilon \end{cases}$
    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* $(Q_1(\tau_t^1, u_t^1), \ldots, Q_n(\tau_t^n, u_t^n);$ *Hypernetwork*$(s_t; \theta))$
      Calculate target Q_{tot} using *Mixing-network* with *Hypernetwork*$(s_t; \theta^-))$
    end for
    $\Delta Q_{tot} = y^{tot} - Q_{tot}$ // Eq (6)
    $\Delta\theta = \nabla_\theta (\Delta Q_{tot})^2$
    $\theta = \theta - \alpha \Delta\theta$
  end if
  if update-interval steps have passed **then**
    $\theta^- = \theta$
  end if
end while

Figure 2: Pseudocode for QMIX implementation (from [3]).

Chapter 2

Monotonic versus additive Value Function Factorisation

The aim of this chapter is to analyze the differences between using monotonic (possibly non-linear) value function factorizations, as in QMIX [3], and using additive (linear) value function factorizations, as in VDN [11].

Our approach is bottom-up: we consider absence of estimation errors produced by sampling and neural networks. We are interested in:

1. Analyzing, using the experimental setup similar to the one of [3], the difference between using a QMIX decomposition and a linear one, such as in VDN.

2. Introducing the idea of using an explicit parametrized model of the payoff function, as a non-linear transformation of the observed payoff. We analyze what are the benefits of such an approach compared to the two previous methods, that do not learn the reward transformation, in the same experimental setting.

2.1 Two-Step Game

This experiment is used in [3] to illustrate how the increased representational capacity of QMIX provides benefits when compared to VDN, in the presence of bootstrapping, i.e., updating Q-values based on other Q-values estimates.

We first analyze this problem in the absence of neural networks, that is, using a tabular representation and a linear parametrization of the Q-values.

The structure of the two-step cooperative matrix game for two agents is as follows: in the initial state, Agent 1 can choose between two actions (A or B) and the action choice of Agent 2 has no effect. If Agent 1 chooses A, the environment transitions to State 2A with probability one, and if it chooses action B, it transitions with probability 1 to State 2B. In State 2A and State 2B both agent's choices are relevant. The payoff at the subsequent state is determined by the matrices described in Figure 3.

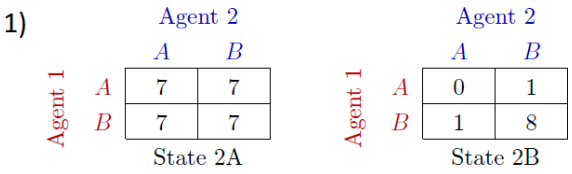

Figure 3: Two-step game theoretical tables. Figure taken from [3].

Figure 4 shows the results of QMIX and VDN. We observe that, contrary to QMIX, VDN fails to learn correctly the (non-linear) payoff in State 2B, and this error backpropagates, leading to an incorrect initial decision (Agent 1 taking action A), since the value of State 1 is computed bootstrapping from the State 2B estimate.

2.1.1 Analytical approach

We start by considering an analytical framework for how would a linear factorization of the value function approximate the maximum payoff of the two-step game assuming that the payoff function is known and without approximations and in the

2)

(a) VDN

	State 1		State 2A		State 2B	
	A	*B*	*A*	*B*	*A*	*B*
A	6.94	6.94	6.99	7.02	-1.87	2.31
B	6.35	6.36	6.99	7.02	2.33	6.51

(b) QMIX

	A	*B*	*A*	*B*	*A*	*B*
A	6.93	6.93	7.00	7.00	0.00	1.00
B	7.92	7.92	7.00	7.00	1.00	8.00

Figure 4: Results for VDN and QMIX. Figure taken from [3].

absence of other sources of approximation errors. For that we define the matrix J, that contains the one-hot encoding $\vec{a} \in \{0,1\}^{m \times n}$ of each possible joint action. In this example of two players and two actions, the joint action corresponds to vectors of size 4

$$J = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \tag{2.1}$$

where in the first row both agents would proceed with the first available actions, in the second row the first agent would execute the first action and the second agent would execute the second available action, and so on.

We can then express the estimated $\hat{Q}_{tot}$ for every state as a linear combination of the entries in J and associated $m \times n$ parameters, e.g., $\omega = (a_{11} \quad a_{12} \quad a_{21} \quad a_{22})^\top$. The objective is to solve the linear system $J \cdot \omega = b$ (one for each state) where b is a vector containing the actual payoff entries for each joint action $\vec{a}$.

The optimal solution is given by

$$\hat{b} = J \cdot \omega^* \tag{2.2}$$

where $\omega^* = J^\dagger \cdot b$ are the learned local Q-values that are combined for each state and $\dagger$ denotes pseudo-inverse. The "best" performance of VDN would be given by the above estimator at each state.

The corresponding solutions for State 2A and State 2B are:

$$\hat{b}(2A) = (7 \quad 7 \quad 7 \quad 7)^\top \tag{2.3}$$

$$\omega^*(2A) = (3.5 \quad 3.5 \quad 3.5 \quad 3.5)^\top \tag{2.4}$$

$$\hat{b}(2B) = (-1.50 \quad 2.50 \quad 2.50 \quad 6.50)^\top \tag{2.5}$$

$$\omega^*(2B) = (-0.75 \quad 3.50 \quad -0.75 \quad 3.50)^\top \tag{2.6}$$

.

Note the similarity between the solution in Eqs. (2.3), (2.5) and the VDN solution of Figure 4.

To work through the process and see the backpropagation, we use the aforementioned Bellman equations. Since we are working with an additive formulation for the Q-value functions, the Q-value function of a certain joint state Q_{tot} is the sum of the respective agents' individual Q-functions $Q_i, i \in \{1, 2\}$. We first get the joint Q-value function for State $2A$, which is terminal, and thus does not need a recursion:

$$\hat{Q}^*_{tot}(2A, \vec{a}) = Q^*_1(2A, \vec{a}_1) + Q^*_2(2A, \vec{a}_2) = \hat{b}(2A), \tag{2.7}$$

where $\vec{a}_i$ indicates the corresponding one-hot encoding of action of agent i. The linear estimate of the (optimal) value function is then provided by the best rewarding joint action

$$\hat{V}^*(2A) = \max_{\vec{a}} \hat{Q}^*_{tot}(2A, \vec{a}) = 7. \tag{2.8}$$

Similarly, for state $2B$:

$$\hat{Q}^*_{tot}(2B, \vec{a}) = \hat{Q}^*_1(2B, \vec{a}) + \hat{Q}^*_2(2B, \vec{a}) = \hat{b}(2B) \tag{2.9}$$

$$\hat{V}^*(2B) = \max_{\vec{a}} \hat{Q}^*_{tot}(2B, \vec{a}) = 6.5. \tag{2.10}$$

The value function for the initial state I (State 1) is calculated recursively

$$\hat{V}^*(I) = \max\{0 + \gamma\hat{V}^*(2A), 0 + \gamma\hat{V}^*(2B)\} = 7\gamma, \tag{2.11}$$

which leads to the suboptimal solution (action A for agent 1).

2.1.2 Modifying the Payoff

We have shown that using a linear decomposition of the payoff can lead to problems, once the approximated estimates are used for bootstrapping the values.

We now propose an alternative way which will keep the additive decomposition of the value function, but will assume that the observed payoff is a modified version of an *true*, unobserved, payoff. We show that a simple transformation can recover the optimal solution. In a sense, instead of increasing the representational capacity of the value function estimate, we will assume a (simple) observation model of the payoff function.

Let's assume that the observed payoff b_o is, for example, quadratic with respect to an unobserved payoff, that is

$$b_o = b^2. \tag{2.12}$$

We can then compute ω^* and $\hat{b}$ according to Equation (2.2), but replacing the state payoffs by taking the inverse transformation, i.e., for each state:

$$\omega_o^* = J^\dagger \cdot \sqrt{b_o}. \tag{2.13}$$

The resulting parameters, after recovering the real value by applying the inverse transformation, correspond to the actual payoffs:

$$\begin{aligned} \hat{b_o}(2A) &= (7 \quad 7 \quad 7 \quad 7)^\top \\ \hat{b_o}(2B) &= (0 \quad 1 \quad 1 \quad 8)^\top. \end{aligned} \tag{2.14}$$

and the Bellman equations then lead to the optimal solution:

$$\hat{Q}^*_{tot}(2A, \vec{a}) = Q^*_1(2A, \vec{a}_1) + Q^*_2(2A, \vec{a}_2) = \hat{b_o}(2A) \tag{2.15}$$
$$\hat{V}^*(2A) = \max_{\vec{a}} \hat{Q}^*_{tot}(2A, \vec{a}) = 7.$$
$$\hat{Q}^*_{tot}(2B, \vec{a}) = \hat{Q}^*_1(2B, \vec{a}) + \hat{Q}^*_2(2B, \vec{a}) = \hat{b_o}(2B) \tag{2.16}$$
$$\hat{V}^*(2B) = \max_{\vec{a}} \hat{Q}^*_{tot}(2B, \vec{a}) = 8$$
$$\hat{V}^*(I) = \max\{0 + \gamma\hat{V}^*(2A), 0 + \gamma\hat{V}^*(2B)\} = 8\gamma. \tag{2.17}$$

Figure 5 shows the resulting values comparing the linear approximation and the nonlinear transformation. Clearly, the choice of payoff transformation specifically works for this example. In the next chapter we will analyze more general ways of modeling transformations.

(a)

	State 1		State 2A		State 2B	
	A	*B*	*A*	*B*	*A*	*B*
A	6.93	6.93	7.00	7.00	-1.50	2.50
B	6.44	6.44	7.00	7.00	2.50	6.50

(b)

	A	*B*	*A*	*B*	*A*	*B*
A	6.86	6.86	7.00	7.00	0.00	1.00
B	7.84	7.84	7.00	7.00	1.00	8.00

Figure 5: Two-step game obtained results with our analytical approach for the given payoff (a) and using a quadratic transformation (b).

2.2 Random matrices

We now consider another experiment from [3] using random matrix games. We reproduce the results for the Value Decomposition Network (VDN) and analyze a possible improvement using a non-linear transformation. Contrary to the previous task, this is a single-step game, and we analyze the capacity of this framework to assimilate non-linear payoffs comparing the obtained maximum payoff to the real one. As such, this setup studies the produced error but does not really study the backpropagation of it.

The idea is that the estimated value should not fall far from the real one to be able to still choose the best option in a situation where other steps follow this choice. As we have seen, the maximum value's error is the only value that is involved in the backpropagation, and therefore needs to be minimum.

To analyze how QMIX outperforms VDN, in [3] they create a payoff vector sampling values uniformly from $[0, 10)$ and set one of the values to the maximum value of 10. This is done with full exploration and for $\{2, 3, 4\}$ agents and $\{2, 3, 4\}$ actions. In their results, QMIX clearly outperforms VDN, which generates very poor results (see Fig. 6).

We take a different approach. As before, instead of running VDN with its default configuration, as in [3], we compute the approximated linear estimation of the payoffs assuming the exact payoff known. This approximation can be seen as the best possible estimator one could obtain using linear decomposition, and ignores other possible errors due to learning, exploration, or sampling.

Our intuition was that using the analytical approach without a transformation we would reach similar results to VDN in Fig. 6. However, this is not the case, as it will be proven in the results (Fig.13) the analytical approach obtains a better overall performance, even if not the best. Note that VDN has an extremely poor performance in Fig.6. We thus conclude that the reported difference in performance between QMIX and VDN cannot directly be attributed to the more general representation capacity of QMIX, but to other factors involved in the learning algorithm.

We also need to take into account that VDN is composed of a whole different structure and, even if considering additive state-action value functions, its use of different networks in learning could be creating this underperformance for non-linear as well as linear payoffs.

Another relevant aspect to analyze is how the methods scale as a function of the number of agents and actions. Both QMIX and VDN suffer from a decrease in performance for higher dimensions of the payoff, as shown in Fig. 6.

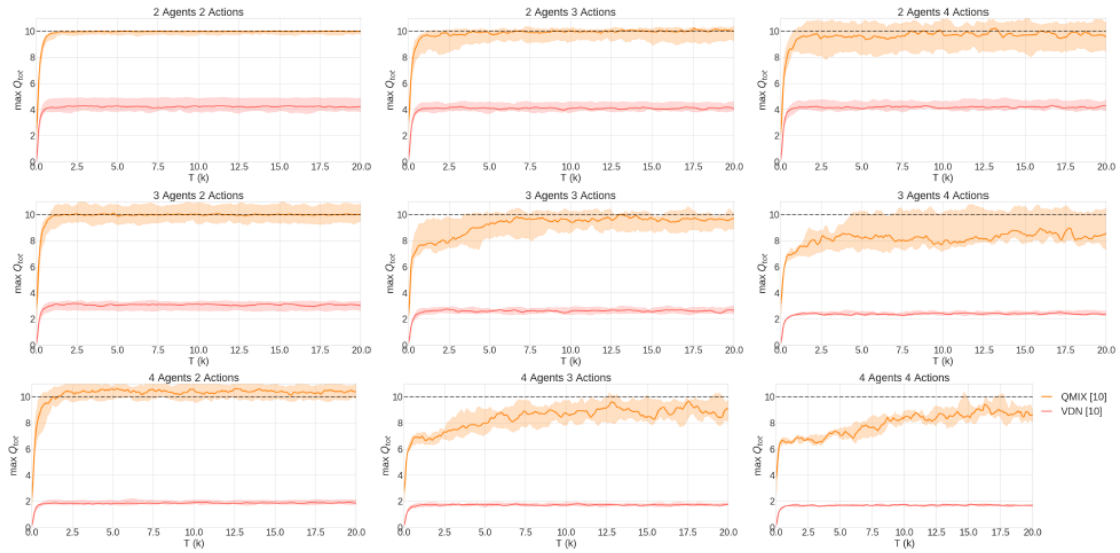

Figure 6: Median $\max_{\vec{a}} Q_{tot}(s, \vec{a})$ for $n = \{2, 3, 4\}$ agents with $m = \{2, 3, 4\}$ choices of actions accross 10 runs for VDN and QMIX. The dashed line at 10 indicates the right value. Figure taken from [3].

In our case, we notice that sampling payoff values according to a uniform distribution induces a tendency for a more "linear" behaviour in the payoffs for higher dimensions. This simply has to do with the dimensions of the payoff function, which increase at the same time our maximum payoff value stays at 10. Therefore, we obtain a higher performance for the linear solution with higher dimensions just because the payoff shape is adapting to linear shapes better learned by it (Fig.7). This again brings the question of: why does VDN downperform with such examples in Fig.6? If higher dimensions in this model generate more linear payoff distributions, we would expect the learning of VDN to be better for higher number of agents and actions.

We can thus conclude that the experimental setup considered in [3] can be improved by considering different ways of generating the payoff functions that are not biased towards favoring the linear decomposition (corresponding to VDN) as the number of dimensions increases.

For that purpose, we consider the following strategies to generate random payoff matrices:

1. Random Payoff for logarithmic distribution (randlog): In this case the payoff

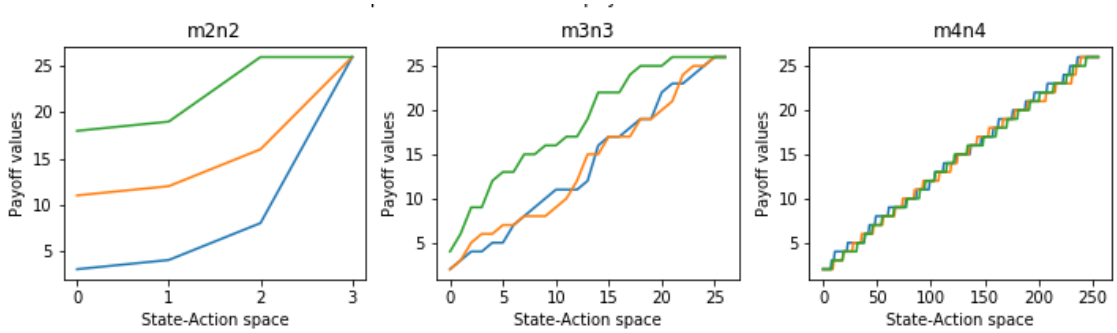

Figure 7: Examples of payoff instances sampling payoffs uniformly for increasing values of m and n. The state-action space is modelled into a vector and ordered to capture the tendency of the payoff. The payoff vector looks more linear for increasing dimensionality.

values are drawn from a log series distribution with specified shape parameter (in this case $p = 0.8$) and inverted. See Fig.8 for examples.

2. Uniformly sampled Payoff with variable maximum value (uni_var): In this case the payoff values are sampled from a uniform distribution but the range of values grows with the payoff dimension. This is done with the intention of compensating the increase of linear behaviour. However, we still find a tendency towards linear shapes as seen in Fig.8.

3. Random Payoff for Laplace distribution (Laplace): In this case the payoff values are drawn from a Laplace distribution of mean $\mu = 0$ and scale $\lambda = 1$. Then a shift is done in order to obtain non-negative values. This method is used to test on non-linear payoff vectors that have a different shape than the randlog method.

One thing to notice is that the maximum value of these payoff instances will not be necessarily fixed as in the example above (which was 10), but can vary. To work with this what we do is work with error percentages (how large is the error compared to the full range of the payoff), where we assume that a system of a given dimension will have variations in the payoffs larger than a system with smaller dimensions (or that two choices with very similar values will lead to similar total rewards).

Our approach will consist on: generating random payoff instances within one of the

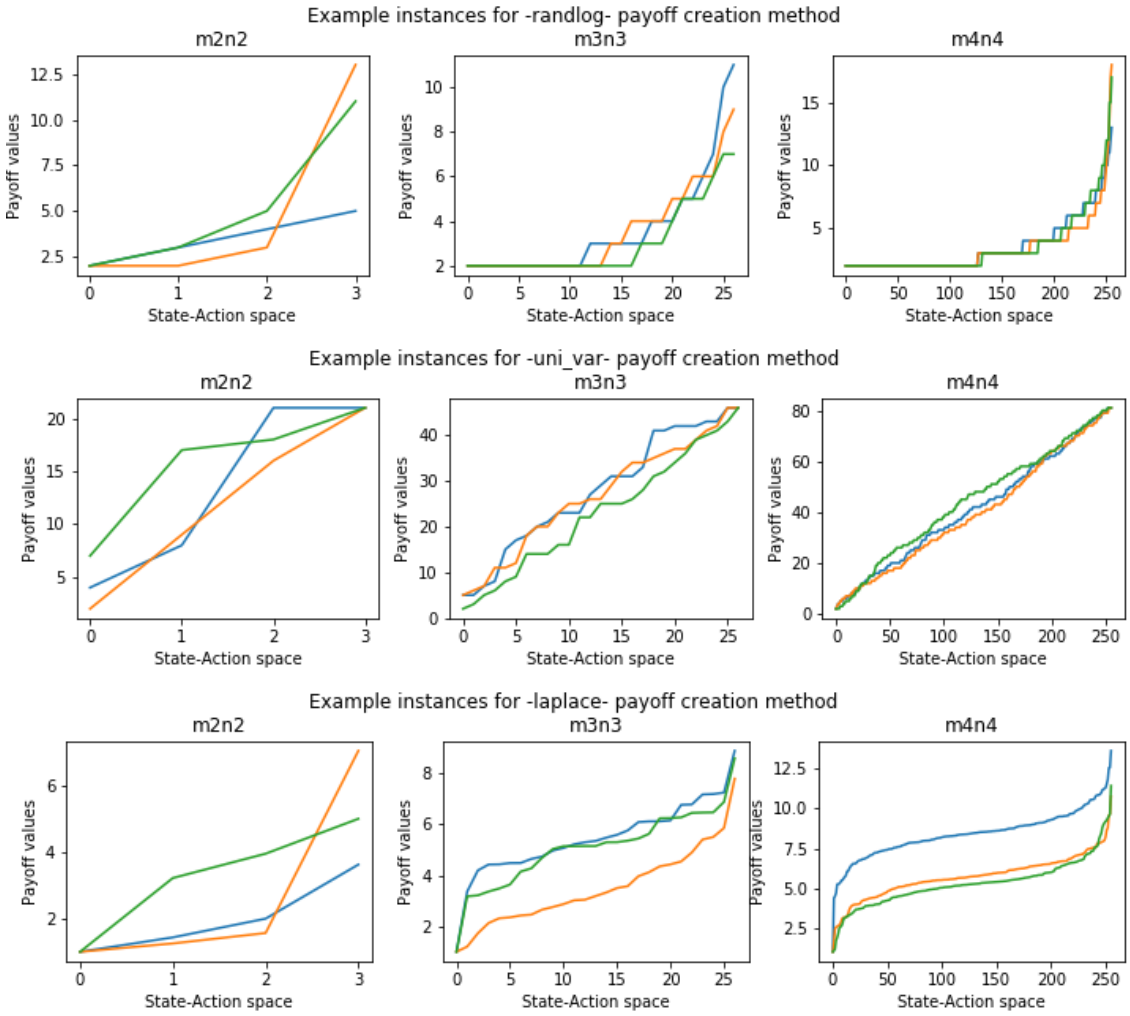

Figure 8: Examples of payoff instances for the three payoff creation methods. In descending order: randlog, uni_var and laplace.

presented creating models and learning the state-action value functions Q_{tot} with and without the help of non-linear transformations. Then, comparing the error of the learned maximum payoffs with respect to the real value to see which learning was better.

Chapter 3

Nonlinear Payoff Transformations

In this chapter we present and analyze in more detail the idea of learning parametrized reward models. First, we describe three different transformations. Then, we consider the learning setting, where neither the payoff and the transformation is known in advance. We then presents some results using these three transformations in one-step tasks composed of random matrices generated according different strategies.

3.1 Modeling Payoff Transformations

Globally, we want to show that by learning an appropriate non-linear transformation of the payoff, a linear decomposition of the Q function is sufficient for accurate policy estimation, i.e., we do not need to consider non-linear monotonic functions.

Let n, m be the number of actions and agents, respectively, which will be used consistently during the description of the methods. Since we are working with single step instances (with a common initial state and several terminal states), we consider the reward function to depend only on the action chosen: $r(s, \vec{a}) = r(\vec{a})$.

For simplicity, we assume the following non-linear transformations that map the well-behaved payoff, that we assume can be represented by a linear decomposition, to our observed payoff r_o (non-linear):

Logarithmic transformation: to recover the linear, well behaved, payoff we need to apply a logarithmic transformation:

$$r_o(\vec{a}) = \exp(r(\vec{a})) + c_{log}, \tag{3.1}$$

where c_{log} is a parameter.

Inverse hyperbolic sine transformation:

$$r_o(\vec{a}) = \sinh(r(\vec{a})) + c_{sinh}, \tag{3.2}$$

where c_{sinh} is a parameter.

Exponential transformation:

$$r_o(\vec{a}) = \log(r(\vec{a})) + c_{exp}, \tag{3.3}$$

where c_{exp} is a parameter.

Our approach considers these three transformations simultaneously, we can cover differently shaped reward functions. Some of this non-linear transformations do not behave well with negative values or 0, and therefore for some instances they can lead to singularities. A proper preparation of the reward data can also help in avoiding such singularities, but the use of the three transformation models is a good way to recover a good result even if one of them fails.

3.2 Learning a Model of Transformed Payoffs

The simplest way to learn the is using gradient descent (GD). We consider both batch GD or Stochastic Gradient Descent (SGD). This depends on whether we can access all instances simultaneously, on the scale of the problem if we need faster

convergence, amongst other reasons. The following descriptions of the method are formulated for SGD. However, in our experiments, we have worked with both batch GD and SGD, and found similar results.

Let $\boldsymbol{\omega}$ be the parameter vector containing the linear weights and the parameters of the reward transformation, when applicable, $\boldsymbol{\omega} = (w_1, w_2, \ldots, w_{n\times m}, c)$. We want to learn the optimal $\boldsymbol{\omega}$ using gradient descent from a batch of N i.i.d. random samples $\{(\vec{a}^{(i)}, r_o^{(i)})\}_{i=1}^N$.

3.2.1 Single-step Bellman Squared Error with *Log* Transformation

The empirical Bellman error for sample i is defined as $BE^{(i)}(\boldsymbol{\omega}) = Q_{\boldsymbol{\omega}}(\vec{a}^{(i)}) - r_c^{(i)}$.

The Bellman squared error becomes

$$\mathcal{L}(\boldsymbol{\omega}) = \frac{1}{2}\sum_{i=1}^{N}\left(BE^{(i)}(\boldsymbol{\omega})\right)^2, \tag{3.4}$$

$$= \frac{1}{2}\sum_{i=1}^{N}\left(Q_{\boldsymbol{\omega}}(\vec{a}^{(i)}) - r_c^{(i)}\right)^2, \tag{3.5}$$

where $r_c^{(i)} = \log(r_o^{(i)} - c)$ transforms our observed reward in the sample $r_o^{(i)}$ to the well-behaved, unknown reward. Note that $Q_{\boldsymbol{\omega}}(\vec{a})$ can be computed efficiently for one joint action $\vec{a}$, as $Q_{\boldsymbol{\omega}}(\vec{a}) = Q_{\boldsymbol{\omega},\mathbf{1}}(a_1) + Q_{\boldsymbol{\omega},\mathbf{2}}(a_2) + \ldots + Q_{\boldsymbol{\omega},\boldsymbol{m}}(a_m)$.

Denote the $a_1^i, a_2^i, \ldots a_m^i$ the action indices of i-th sample for each agent indexing the weight vector $\boldsymbol{w}$. The parameter updates for the i-th sample are

$$w_{a_j^i} \leftarrow w_{a_j^i} - \eta\left(\sum_{k=1}^{m} w_{a_k^i} - r_c^{(i)}\right), \qquad \forall j = 1, \ldots, m, \tag{3.6}$$

$$c^i \leftarrow c^i - \eta\left(\sum_{k=1}^{m} w_{a_k^i} - r_c^{(i)}\right)\frac{1}{r_o^{(i)} - c^i}, \tag{3.7}$$

where η is the learning rate, e.g., $\eta = 0.01$.

3.2.2 Single-step Bellman squared error with *Arcsinh* transformation

In this case $r_c^{(i)} = arcsinh(r_o^{(i)} - c)$ transforms our observed reward in the sample $r_o^{(i)}$ to the well-behaved, unknown reward.

The corresponding parameter updates are:

$$w_{a_j^i} \leftarrow w_{a_j^i} - \eta \left(\sum_{k=1}^{m} w_{a_k^i} - r_c^{(i)} \right), \qquad \forall j = 1, \ldots, m, \tag{3.8}$$

$$c^i \leftarrow c^i - \eta \left(\sum_{k=1}^{m} w_{a_k^i} - r_c^{(i)} \right) \frac{1}{\sqrt{1 + (r_o^{(i)} - c^i)^2}}. \tag{3.9}$$

3.2.3 Single-step Bellman squared error with *Exp* transformation

In this case $r_c^{(i)} = \exp(r_o^{(i)} - c)$ transforms our observed reward in the sample $r_o^{(i)}$ to the well-behaved, unknown reward.

The corresponding parameter updates are:

$$w_{a_j^i} \leftarrow w_{a_j^i} - \eta \left(\sum_{k=1}^{m} w_{a_k^i} - r_c^{(i)} \right), \qquad \forall j = 1, \ldots, m, \tag{3.10}$$

$$c^i \leftarrow c^i - \eta \left(\sum_{k=1}^{m} w_{a_k^i} - r_c^{(i)} \right) (-1) e^{(r_o^{(i)} - c)}. \tag{3.11}$$

3.3 Results for Fixed Dimensionality

We first consider fixed values of $dim = 9$, for $m = 2$ and $n = 3$. By performing gradient descent using $N = 100$ different generated payoff instances, the error between the learned maximum payoff value ($\max_{\vec{a}} Q_{tot}(s, \vec{a})$) and the real value is obtained. We evaluate both the error of the maximum value and the ranking of the joint actions according to their corresponding value estimates.

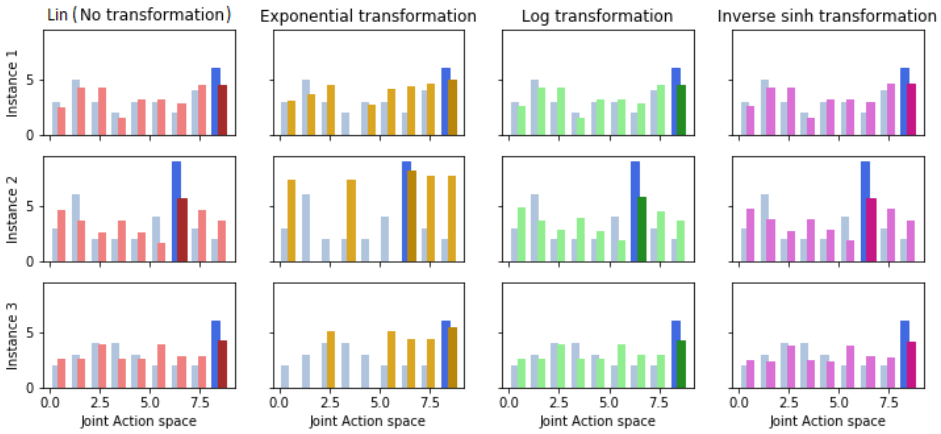

Figure 9: Results on three instances (rows) using a logarithmic distribution to generate the payoffs (*Randlog*). The x-axis indicates the joint actions indices and the y-axis shows the corresponding learned Q-values. The original payoff is colored in blue (dark blue for the optimal joint action). Methods in columns: no transformation (in red), exponential transformation (in orange), logarithmic transformation (in green), and inverse sinh transformation (in purple). The missing marks correspond to numerical errors. The exponential transformation, despite having some numerical errors, approximates best the maximum payoff value.

Fig. 9 shows the obtained results for each method together with a baseline that does not consider a transformation in the payoff. In this case, the ranking of the Q-values given the joint action vectors is learned efficiently for most values. However, it often underestimates the value of the maximum payoff. The solutions provided by this method are consistent and numerically stable, which is not always the case as it will be explained for the other methods.

Results using non-linear transformation:

When applying a transformation, the behaviour is fairly different for the three transformations that have been considered. In Fig. 9 We observe that the exponential transformation, despite having some numerical errors, approximates best the maximum payoff value (see the caption for details).

Fig.10 shows the results for $N = 100$ instances and different generation models displayed as a percentage error for each case: without applying a transformation to the original payoff (*Lin*), applying an exponential transformation (*Exp*), an inverse

hyperbolic sine transformation (*Arcsinh*) and a logarithmic transformation (*Log*). With these results, choosing the lowest error value (*bestmethod*), it is possible to see if the use of non-linear transformations helps improve the error without any transformation(*Lin*).

The logarithmic transformation fails to both learn the ranking properly and reach an accurate value for the maximum payoff value (see Fig.9). At the same time, it also behaves with growing instability for the Laplace distribution generated payoffs (see Fig.10). The *Log* transformation also tends to fall into singularities and returns considerably inefficient results for higher dimensions. Therefore this transformation does not lead to satisfactory results.

The exponential transformation in this dimensions is the optimal one in terms of reaching the maximum value accurately. However, for the Laplace distribution generated payoffs it is matched with the inverse hyperbolic sine transformation. This can be explained by the fact that the latter type of payoffs have a shape that can easily be related to a $f(x) = sinh(x)$ function, and therefore the inverse function can transform the payoff to a linear shape. It is interesting to note that an *Arcsinh* transformation usually leads to slightly improved or identical solutions than with no transformation, while an *Exp* transformation usually reduces the error considerably. Using an *Arcsinh* transformation does not lead to notably improved results compared to the no transformation case and therefore is ruled out of the optimal solutions.

Despite the fact that the most successful transformation is using an exponential function to reshape the payoff growth into a more linear behaviour, this transformation sometimes fails to converge and leads to singularities. It also sometimes fails to learn the lowest payoff values, which does not have repercussions in case of focusing on the optimal policy, but is worthy to take into account, as it might become a problem within the learning stage. In some specific situations, this irregular behaviour can lead to inaccurate ranking of the rewards for non-optimal joint-action vectors. Since the solution without any transformation behaves well under all conditions and learns all payoff values, but fails in reaching the correct value for the maximum pay-

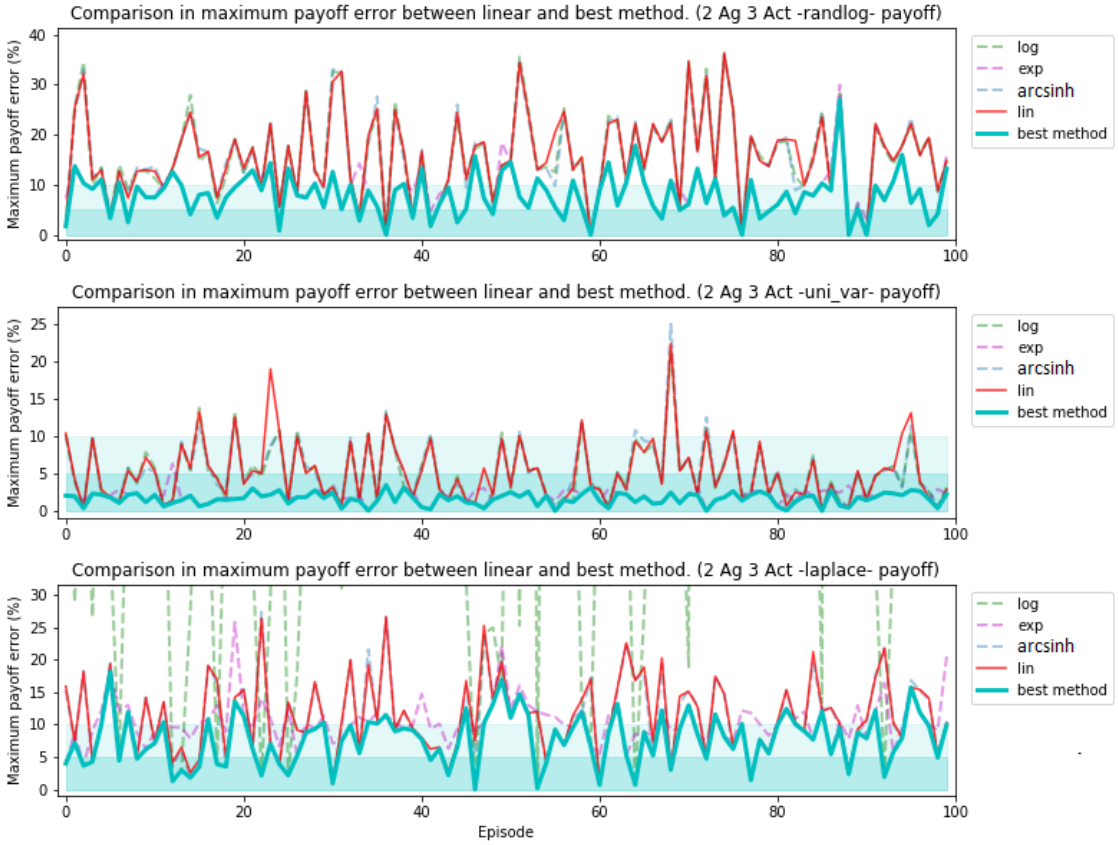

Figure 10: Relative error between the maximum payoff value and $max_{\vec{a}} Q_{tot}(s, \vec{a})$ using different tranformations in $N = 100$ episodes and fixed dimensions. The three figures correspond to the three different payoff generation methods. In red (Lin) the error obtained with no transformation and in blue the optimal value out of all the used methods (Lin,Log, Exp and $Arcsinh$). In dashed lines there is the solution obtained using each transformation (Log, Exp and $Arcsinh$).

off, a combination of both solutions would lead to an optimal situation: By using the well behaved solution, the determination of the joint-action vector that corresponds to the maximum reward is possible, while the learned value at that specific joint-action vector will be given by the learned maximum payoff using the exponential transformation. Then, one of the solutions will help by pointing out the optimal combinations of actions while the other solution will help define the reward more accurately, so that a back-propagation of errors is less possible.

In the following sections, for each combination of m, number of agents, and n, number of actions per agent, an average percentage error of the $N = 100$ instances is performed. This allows to see a change in behaviour for a growing scale of the system and get an idea of possible scalability issues.

3.3.1 Results for *Randlog* payoff

For the payoff generated through samples of a logarithmic distribution (*randlog*), the error percentage is improved substantially when using non-linear transformations on the payoff, as presented in Fig.11. For this payoff behaviour, using the exponential function to transform the payoff seems to be the ideal transformation (Fig.12). However, as mentioned, it leads to singularities for some instances. Even if this failures only happen less than 10% of the times, it can still create problems. This also seems to grow with dimensions, so it can become a real struggle for larger scale settings. For this type of payoffs, using an exponential transformation can be helpful in terms of estimating the maximum value accurately but this use has to be relegated to a support role since using just the solution with the transformation can lead to singularities and bad ranking, which can be critical for decision-making.

3.3.2 Results for *Uni_var* payoff

The solutions given uniform distribution payoffs, even when using the variable maximum value to improve the variability in the shape of the payoff, reveal that for higher dimensions the use of transformations in the payoff are not needed since the error becomes the same as for the no-transformation case (Fig. 13). For these in-

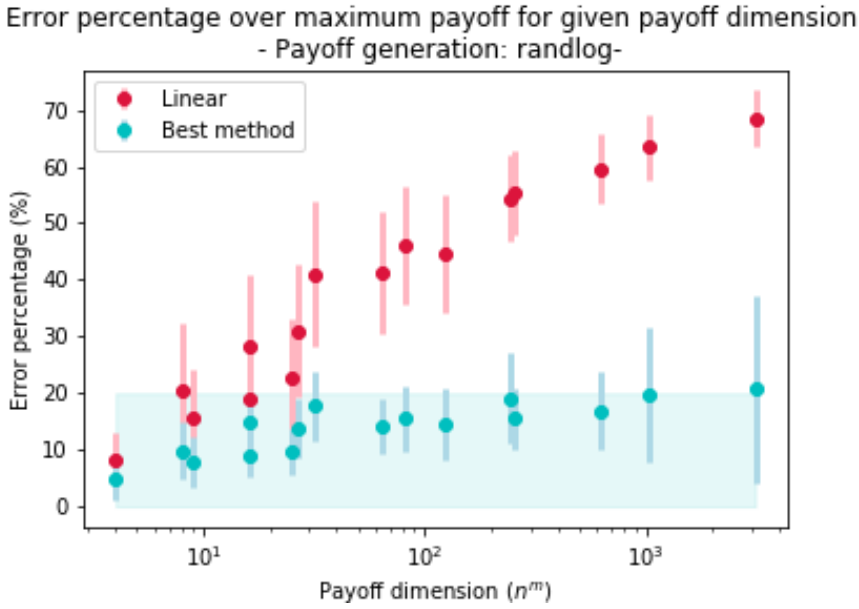

Figure 11: Average error percentage of the maximum learned payoff $\max_{\vec{a}} Q_{tot}(s, \vec{a})$ by the linear decomposition (Linear, no use of transformation) and the best method within the previous and the other methods that use transformations. The average is done amongst $N = 100$ instances generated from a logarithmic distribution. The variance is presented with vertical error bars. Note the error for the best method stays within the 20%.

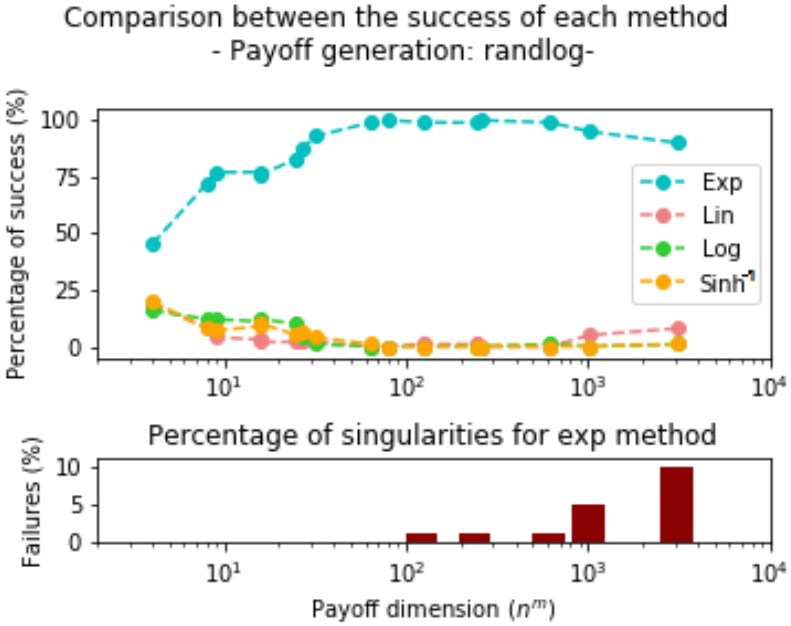

Figure 12: Percentage of success for each method, where *Lin* is without the use of any transformation and *Exp, Arcsinh and Log* correspond to the presented transformations. In the bar chart below, a percentage of cases in which the *Exp* transformation fails to deliver a solution.

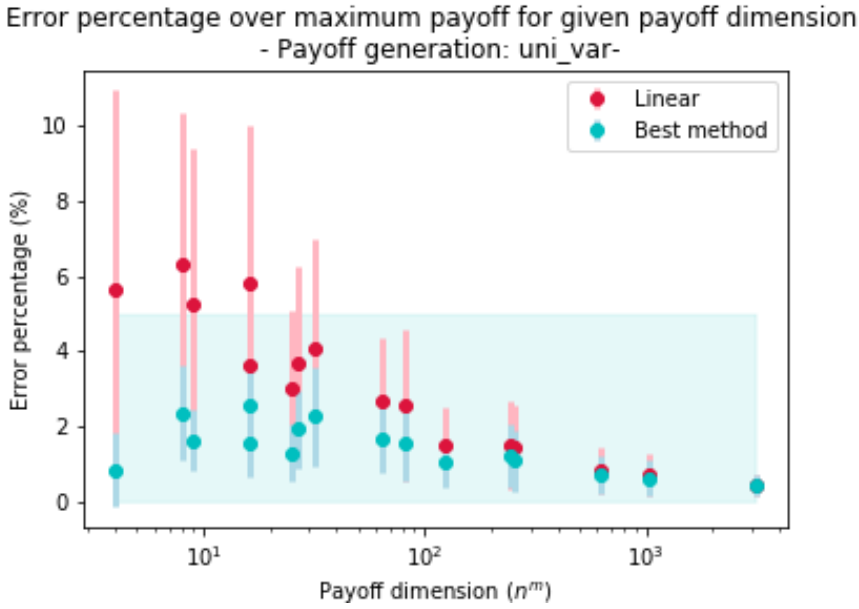

Figure 13: Average error percentage of the maximum learned payoff $\max_{\vec{a}} Q_{tot}(s, \vec{a})$ by the Linear decomposition (no use of transformation) and the best method within the previous and the other methods that use transformations. The average is done amongst $N = 100$ instances generated from a variable uniform distribution. The variance is presented with vertical error bars. Note that both results stay within less than a 5% error for larger dimensions.

stances, performing no transformation becomes the best method, since if error is the same the no-transformation (*Lin*, in Fig.14) is more efficient. Note that in this case the exponential transformation of the payoff leads to less singularities.

3.3.3 Results for *Laplace* payoff

For the solutions obtained from the Laplace distribution generated payoffs the improvement is more subtle than for the *randlog* payoffs, but it still has a different tendency and thus seems to stay within lower error values for larger dimensions (Fig.15). Regarding the best transformation, using an exponential function seems to work properly for larger dimensions, while the inverse hyperbolic sine works for smaller settings. The singularities for the *Exp* transformation method also stay within minimums. Therefore, this proves that for payoffs that have a non-linear shape different from a exponential growth, the *Exp* transformation can still be useful in capturing the non-linear behaviour.

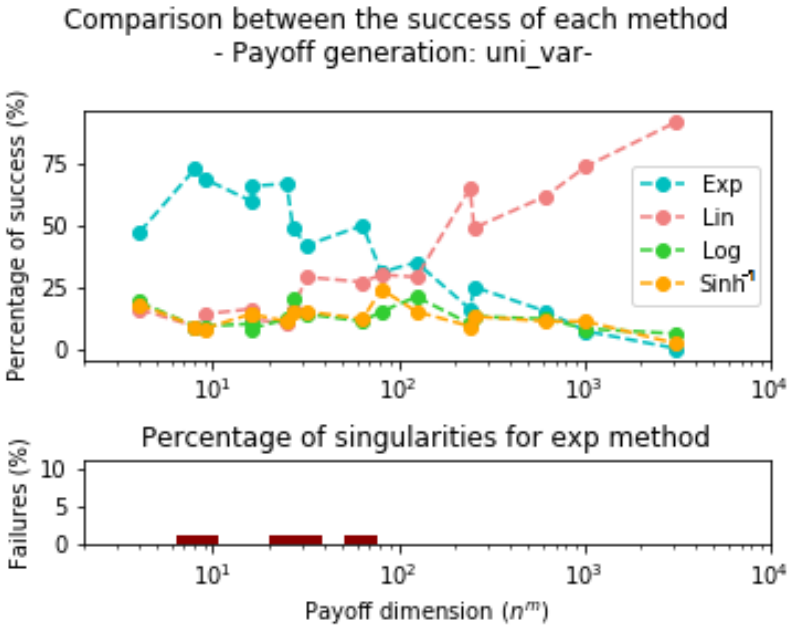

Figure 14: Percentage of success for each method, where *Lin* is without the use of any transformation and *Exp, Arcsinh and Log* correspond to the presented transformations. In the bar chart below, a percentage of cases in which the *Exp* transformation fails to deliver a solution.

In summary of the obtained results, the only transformation that was found to be useful in reducing the maximum payoff value error was the *Exp* transformation. The transformation was not needed for uniformly generated payoffs, where not applying a transformation already obtained low error values, but it still delivered good results. The scalability of this application could be threatened by the possible issues with singularities.

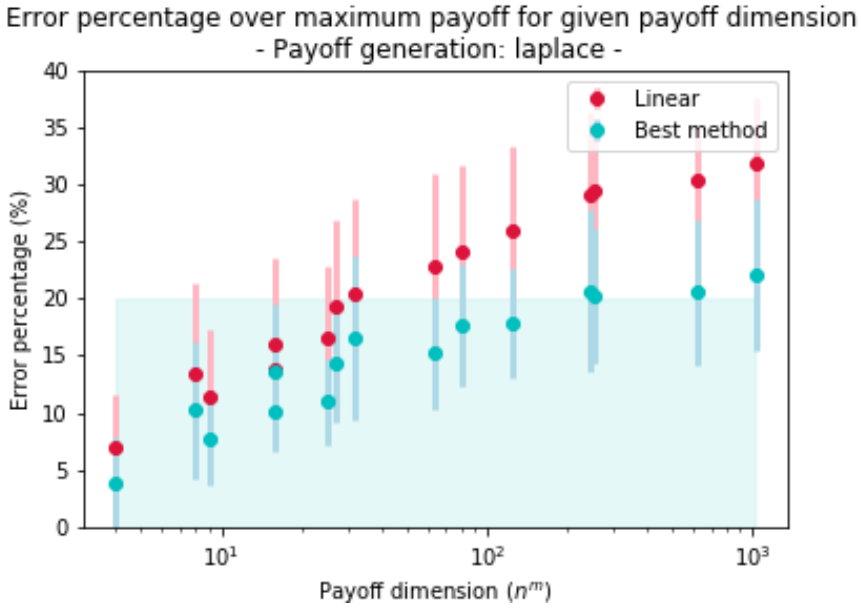

Figure 15: Average error percentage of the maximum learned payoff $\max_{\vec{a}} Q_{tot}(s, \vec{a})$ by the Linear decomposition (no transformation) and the best method within the previous and the other methods that use transformations. Results averaged over $N = 100$ instances generated from a Laplace distribution (vertical bars denote variance). Note the error for the best method is mostly below 20%.

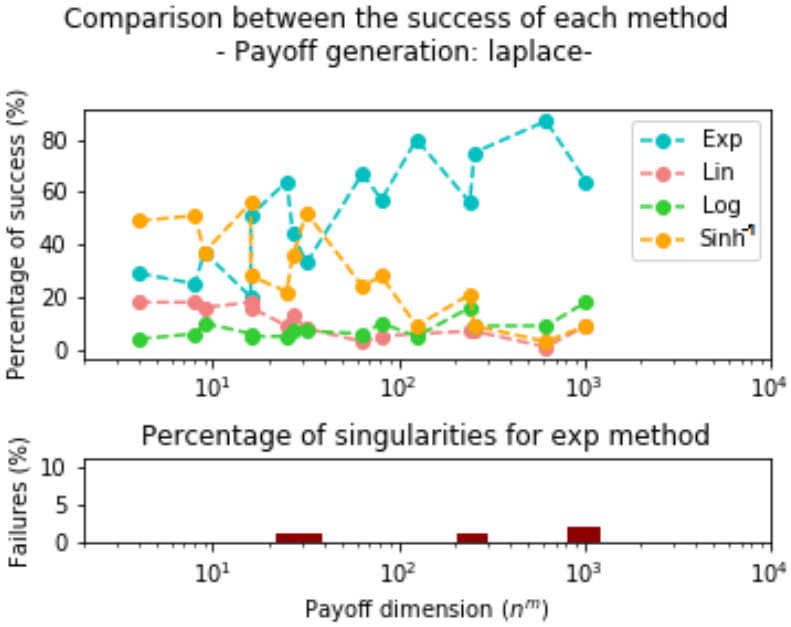

Figure 16: Percentage of success for each method, where *Lin* is without the use of any transformation and *Exp, Arcsinh and Log* correspond to the presented transformations. In the bar chart below, a percentage of cases in which the *Exp* transformation fails to deliver a solution.

Chapter 4

Discussion

In this thesis, we have properly described how value decomposition affects the learning of differently shaped payoffs, and how this can become a limitation for structures such as VDN. We also studied how a non-linear transformation of the payoff can become a possible solution to these limitations. Our results show that an exponential transformation can lead to satisfactory learning the accurate value. However, the use of non-linear functions can lead to singularities and is less efficient. It can also become a problem when using exploration methods such as an ϵ-greedy choice of actions when learning.

The fact that VDN performed poorly in the random matrix setting proposed by [3] might not be directly linked to its representational capacity, since the designed experiment would, in fact, provide more linear systems for higher dimensions, as seen in Section 2. If the value decomposition algorithm they use in this experiment continues to perform poorly or worsens its performance for larger systems, this could be caused by the tuning and optimization of this algorithm, and not its theoretical limitations. However, it is true that the theoretical description of both methods, VDN and QMIX, accounts for a difference in representational complexity, and that, as we proved in our analysis, simple linear mixing of individual Q-values can lead to underestimating the true maximum payoff in non-linear payoff growth.

Based on our conclusions, we now present a possible adaptation of QMIX that ad-

ditionally estimates a non-linear model of the payoffs that could be used to improve the performance. The adaptation is based in QMIX because, based on the state-of-the-art results, its structure has a better performance. The detailed analysis and evaluation of such adaptation is left for future work.

4.0.1 Proposed Structure to Work with Larger Systems

The proposed structure considers not only the maximum paying joint action, but also the ranking of actions according to the value estimate. Additionally, we use our results on scalability presented in the previous chapter.

The following structure is mainly inspired in the QMIX structure (refer to Fig.1 and Fig.2 to see the original work), as it was also used and adapted in [2] for value decomposition networks. The most notable change is the substitution of the mixing network and hypernetworks for a simple linear mixing. Our contribution in this case is the addition of a parallel process that helps adjusting the magnitude of the estimated maximum payoff, in order to avoid the discussed backpropagation errors.

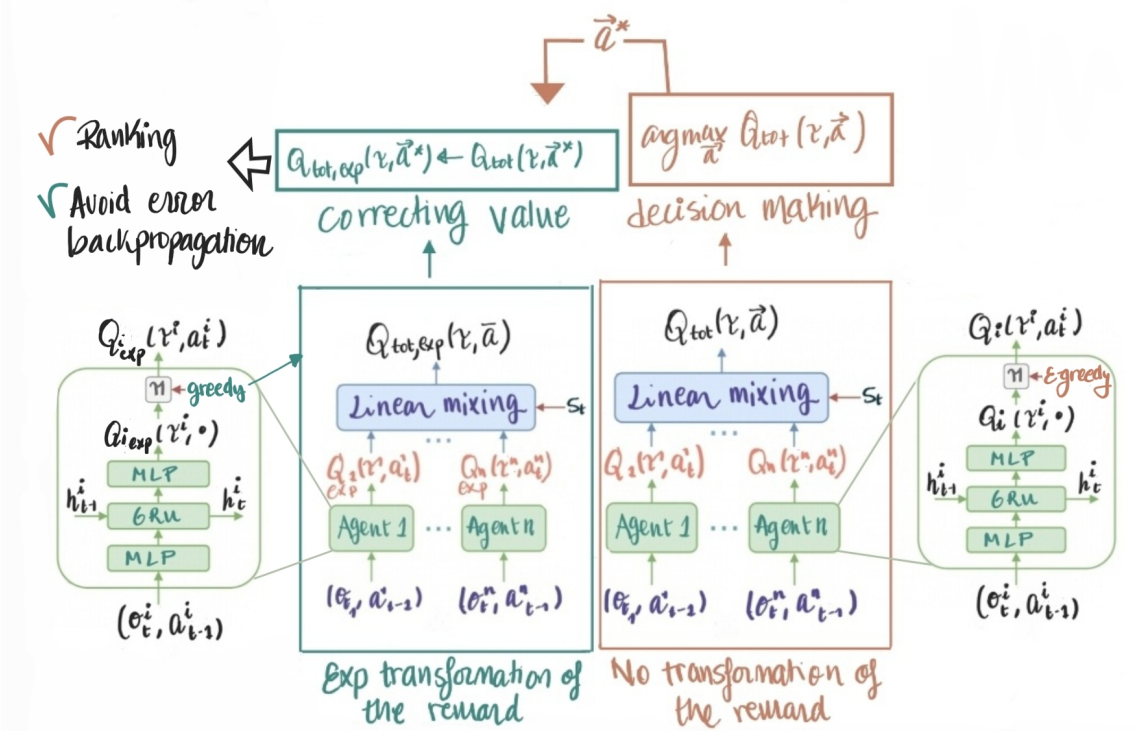

Figure 17: Adapted QMIX [3] structure.

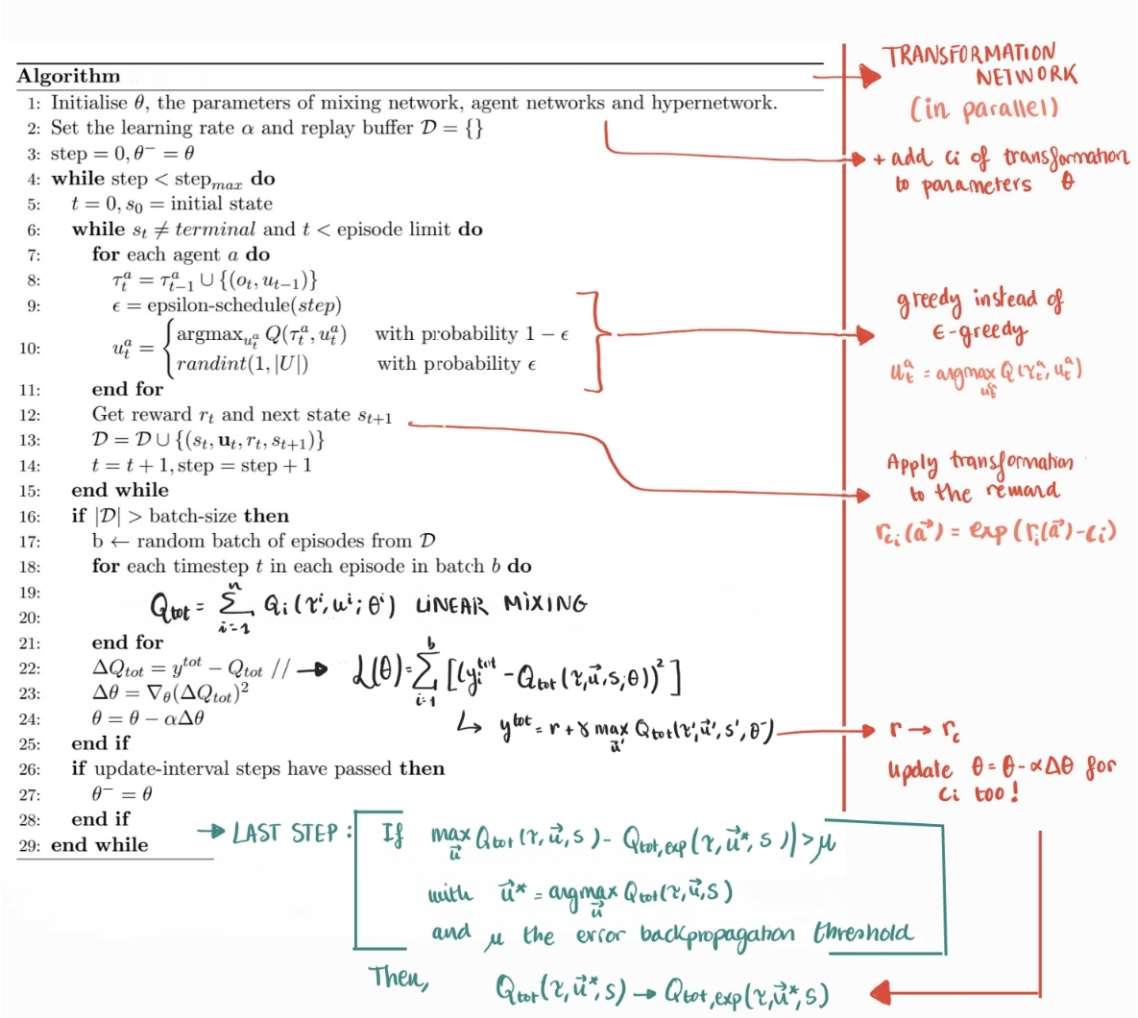

Figure 18: Adaptation of QMIX [3] pseudocode.

This proposed structure would also work with DRQNs within the agent networks and would be suitable for partially observable settings as long as the environment state is available for the mixing and Q-learning update stage. This is important because payoffs are defined for general states and the specific constant c_i defined for the parametrization of the payoff is linked to each of these joint states.

Then, the structure (Fig.17) uses two parallel structures: one to learn the ranking of joint actions properly, that does not use any transformation of the payoff and leads to optimal decision making, and one that uses the transformation and re-adjusts the value of the maximum payoff if needed. Note that the need for a good ranking does not imply that the algorithm will explore all joint action possibilities (which would

be intractable for scalability reasons) but that, in case of exploring, the ranking needs to be accurate to not lead to bad decision-making. We can assume the re-adjusting is needed if the learned values of the maximum payoffs differ more than a set threshold, which will depend on the system we work with. See Fig.18 for the description of this threshold.

A fair doubt would be: why not using only the learned Q-values with the transformation? But as commented in the previous section these results can be unstable for the Q-values corresponding to non-optimal joint actions and therefore if we use methods like an ϵ-greedy policy, the learning would likely be problematic. That is also why for the agent networks we use a greedy policy in the case of applying the transformation to the payoff. This, we assume, could provide support to the main algorithm regarding the issues with backpropagation.

Note that this is only a proposal and could generate new or unexpected issues in a further design, or could definitely be optimized and tuned to work better. For example, finding other types of transformations that can aid in the re-adjusting or provide possible resolutions to the instability when using the exponential transformation would be good directions for future work.

4.1 Conclusions

In this thesis a proper theoretical model and description of the representational capacity of an additive value function in cooperative MARL was presented. The model was strongly based on the work of [3] and part of their experiments was reproduced and discussed. This was done by scaling down the properties and characteristics of their description to reduced scale systems, in order to escape the need for deep learning structures.

Parting from this model, some possible solutions to the limitation of these description of the Q-value functions for non-linear payoff growth were presented. Finally, a possible adaptation of the current state-of-the-art algorithms was proposed, with its viability left for future work.

Some of the issues that could be also studied and considered in the future are: the scalability of this proposed solution and the improvement in the singularities obtained when applying non-linear transformations.

List of Figures

Bibliography

[1] Silver, D. *et al.* Mastering the game of go with deep neural networks and tree search. *nature* **529**, 484–489 (2016).

[2] Castellini, J., Oliehoek, F. A., Savani, R. & Whiteson, S. Analysing factorizations of action-value networks for cooperative multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems* **35**, 1–53 (2021).

[3] Rashid, T. *et al.* Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research* **21**, 178–1 (2020).

[4] Serrano, J. B., Curi, S., Krause, A. & Neu, G. Logistic Q-learning. In *International Conference on Artificial Intelligence and Statistics*, 3610–3618 (PMLR, 2021).

[5] Zhang, K., Yang, Z. & Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control* 321–384 (2021).

[6] Monahan, G. E. State of the art — a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management science* **28**, 1–16 (1982).

[7] OroojlooyJadid, A. & Hajinezhad, D. A review of cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1908.03963* (2019).

[8] Watkins, C. J. & Dayan, P. Q-learning. *Machine learning* **8**, 279–292 (1992).

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

[10] Hausknecht, M. & Stone, P. Deep recurrent Q-learning for partially observable MDPs. In *AAAI fall symposium series* (2015).

[11] Sunehag, P. *et al.* Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, 2085–2087 (2018).

[12] Rashid, T. *et al.* Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 4295–4304 (PMLR, 2018).