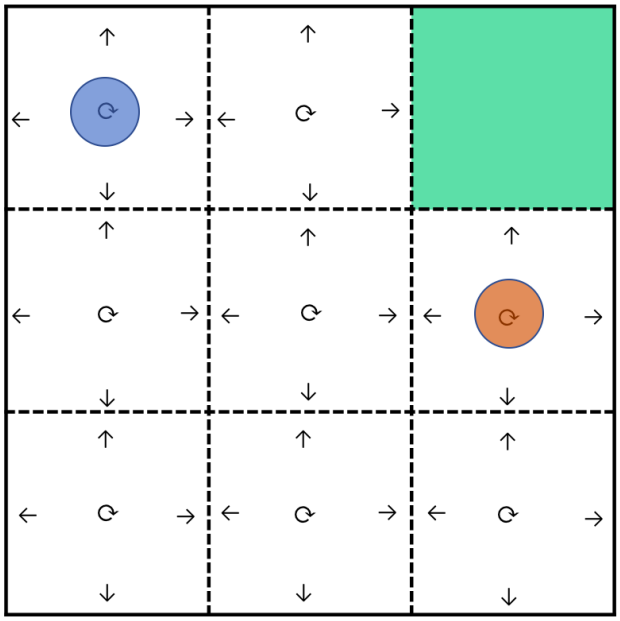

A Comparative Study of Reinforcement Learning Algorithms in a Multi-Agent System

Developing different cooperation strategies with Q-learning and DQN

Master's Thesis in Engineering Mathematics and Computational Science

INGJERD MYHRE

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:EEX30

A Comparative Study of Reinforcement Learning Algorithms in a Multi-Agent System

Developing different cooperation strategies with Q-learning and DQN

INGJERD MYHRE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

A Comparative Study of Reinforcement Learning Algorithms in a Multi-Agent System
Developing different cooperation strategies with Q-learning and DQN
INGJERD MYHRE

Supervisor: Bile Peng, Department of Electrical Engineering
Supervisor and Examiner: Henk Wymeersch, Department of Electrical Engineering

Master's Thesis 2019:EEX30
Department of Electrical Engineering
Division of Communication Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Two agents in a grid world environment.

Typeset in LaTeX
Gothenburg, Sweden 2019

A Comparative Study of Different Multi-Agent Reinforcement Learning Algorithms
Developing different cooperation strategies with Q-learning and DQN.
INGJERD MYHRE
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Machines are an invaluable tool for humans regarding efficiency in many areas. Machines can perform a lot of tasks if explicitly told what to do, i.e., *rule-based control*. In recent years the research and use of *machine learning* have increased. A machine that has the ability to learn what to do, instead of being explicitly told what to do, can help improve efficiency even further. This thesis is looking at a branch of machine learning, namely reinforcement learning.

The work in this thesis is an implementation of two agents cooperatively learning to interact with an environment using the reinforcement learning algorithms Q-learning and DQN, i.e., implementation of a multi-agent reinforcement learning problem. The result in this thesis shows that the agents are able to learn to interact with the environment using both Q-learning and DQN, and the ability for both Q-learning and DQN to benefit from different cooperation strategies are shown. To gain robustness of the methods the agents have been trying to learn both what to communicate and how to interpret a message simultaneously. With Q-learning, the agents were able to do this, but using DQN the agents failed. The DQN is, on the other hand, showed to gain robustness by the ability to interact with slightly changed environments.

Keywords: Artifical intelligence, Reinforcement learning, Q-learning, Deep Q-network, Multi-agent system, Communication

Acknowledgments

I would like to thank Henk Wymeersch both for the opportunity to write this thesis (the work has opened up my eyes for this exciting field of machine learning) and also for the guidance towards improved quality of the work. I would also like to express my gratitude to Bile Peng for supervising me throughout the work of this thesis, for encouraging me and always being available to answer my questions.

Ingjerd Myhre, Gothenburg, June 2019

Contents

1 Introduction

1.1 Background

Today, the role of automatization and digitalization are increasing. Letting computers perform tasks earlier performed by humans is believed to be more cost effective [1] and can achieve better performance [2]. However, a benefit of humans versus computers is the intelligence of humans [3]. Having the ability to adapt to small modifications and stochastic perturbations, and make intelligent decisions based on these, is an advantage for the human. It is therefore desirable to try to make machines learn how to behave intelligent. In this thesis, intelligence is measured as an agent's ability to achieve goals in an environment.

1.2 Motivation

In rule-based control [4], we have a set of input and rules, and we want to produce the output given these. But these rule-based controls are not able to adapt to complicated problems and their complexity grows drastically as the problem scales up. In machine learning, we are trying to let the machine learn the rules themselves in a data-driven manner. Machine learning can be divided into three subcategories [5]:

- Supervised learning - with a set of inputs with belonging targets, the machine will learn how to produce the target by reducing the difference between its produced output and the belonging target. Examples of supervised learning are regression, classifications.
- Unsupervised learning - with only a set of input, the machine learns the structure of the data without targets. Examples of unsupervised learning are clustering of data, discovery of structure.
- Reinforcement learning - The focus of this thesis. In reinforcement learning there are no input-output pairs as clear as in the supervised learning case, but still with some feedback. Given some input, the machine tries to interact with an environment and receives a reward for this. This reward is a feedback, not as clear compared to the desired output in supervised learning, but it is a feedback of good or bad interaction. Examples of reinforcement learning are Robotic navigation, real-time decisions.

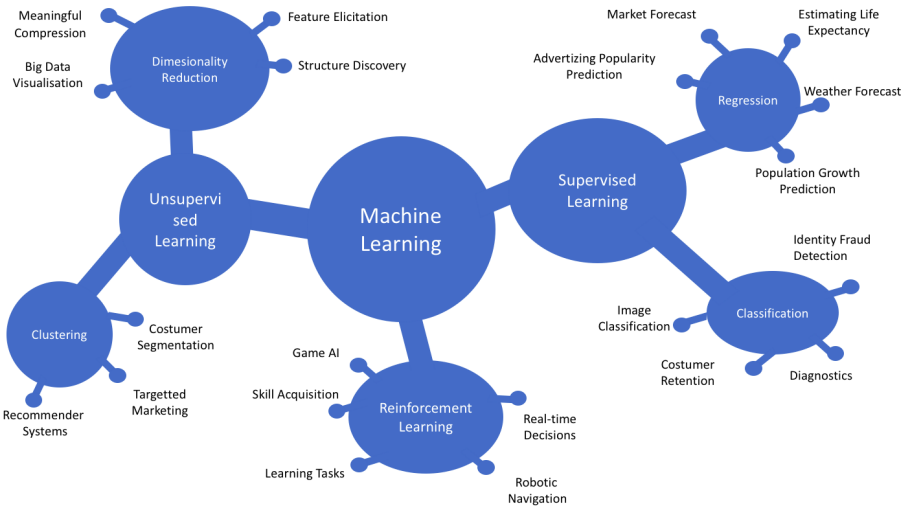

Figure 1.1: Machine Learning branches.

Reinforcement learning is a method aiming to optimize long term rewards possibly at the cost of short term rewards. Without any prior knowledge of the environment, i.e., *model free* reinforcement learning, the machine shall learn how to accomplish tasks. By interacting with the environment, the machine should learn, by trial and error, a policy for taking actions to maximize the long term reward. A tutorial for reinforcement learning can be found in [6].

As mentioned in the introduction, machines might even perform tasks even better than humans. Their computational capacity makes them able to explore more options than humans. Machines are also not biased by previously learned knowledge and rules, and will not be limited by this. Reinforcement learning is aiming to outperform human minds, and a known example of this is given in [2]. Here the machine has learned to play the game of *Go*, a well-known board game from China. In 2016 this machine beat one of the highest-ranked Go players in the world in 4 out of 5 games.

In this work, a comparative study of multi-agent reinforcement learning (MARL) algorithms will be carried out. The methods to be investigated are tabular Q-learning [7] and Deep Q-network (DQN) [8]. First, tabular Q-learning for multi-agent systems are applied for a set of problems, then the same problems are solved using DQN. The set of problems to be investigated are different cooperation strategies applied for two agents learning to cooperate in an environment. The implementations will be done using Python and TensorFlow and the problem will be based on a grid world environment with a goal state.

The difference between single agent reinforcement learning and MARL is that in a single-agent case, we can assume that the environment is stationary [6], while in the MARL case two agents are acting in the same environment, which causes major difficulties of non-stationarity of the environment [9]. One agent's reward does not only depend on its own action and state, but also on the other agent's

action and state. This non-stationary causes violation of the assumptions needed for convergence of the single-agent methods [6]. However, the effect is not only negative when having multi-agent systems. This opens up to investigate methods for cooperation between the agents [9]. If a task is too difficult for one single agent to perform, it might be useful to introduce another agent so that the agents can solve the task together in a cooperative manner.

1.3 Problem Formulation

Reinforcement learning is commonly developed in game theory [8], due to the good match with the setup of such games and the reinforcement learning theory. In this work, a simple game environment will be implemented.

The problem to be investigated in this thesis is the following: In a 3×3 grid world (see Figure 1.2), two agents shall learn to reach a goal state simultaneously. To be able to do this the agents need to learn to find their way to the goal state. By luck, they might reach the goal simultaneously, but if they shall learn to reach the goal simultaneously more often than pure luck they would need to learn to cooperate somehow.

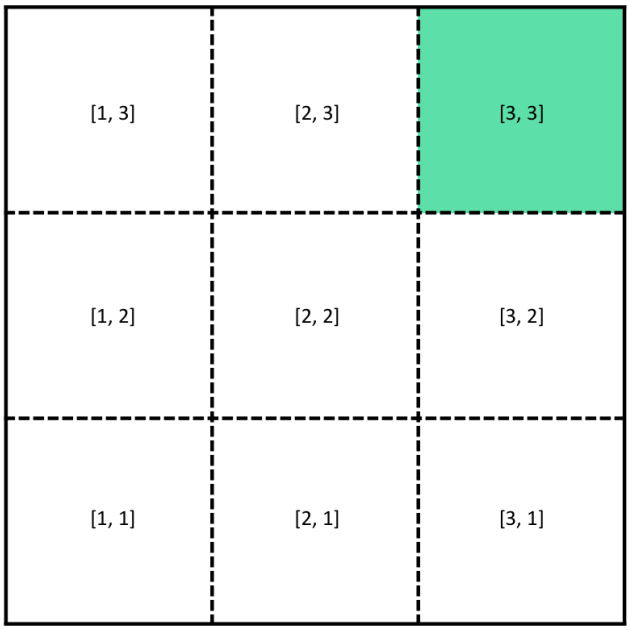

Figure 1.2: The environment for the thesis, a 3×3 grid world.

1.3.1 Challenges and limitations

A challenge in this thesis is the computational complexity, which increases with the size of the environment. Hence, the environment in this thesis is quite simple.

1.4 Aims

The aims of this thesis is the following:

- Implement cooperation between the agents to reach a goal simultaneously
- Compare different cooperation strategies:
  - Independent reinforcement learning
  - Joint reinforcement learning
  - Message passing
- Investigate whether the algorithms can be generalized to be adaptive to other environments

1.5 Outline of the thesis

In Chapter 1 the thesis has been introduced and motivated. In Chapter 2 the theoretical aspects will be described. In Chapter 3 the problem will be formulated and the methods will be described. In Chapter 4 the results, based on programming work, will be presented. Finally, in Chapter 5, the thesis will be summarized.

2
Theory

In this chapter, the theoretical concepts will be introduced. First, the concept behind reinforcement learning, including a Markov decision process, is described. Then a value for how good a state-action pair is, as well as an optimal solution, is defined. Then Q-learning and DQN are described. In the section of DQN, the concepts used for the neural network are also briefly introduced. Finally, the theoretical concepts of MARL are described.

2.1 Reinforcement Learning

Introductions to reinforcement learning can be found in [10]. In reinforcement learning the following interaction between an agent and the environment exists: An agent takes an action a according to its observed environment state s, the environment feeds back the reward r and a new state s', according to s and a. Then the agent takes a new action corresponding to the new state, and the environment feeds back the reward r and the next state s', according to the current s and a. This is repeated until a predefined termination of the interaction process.

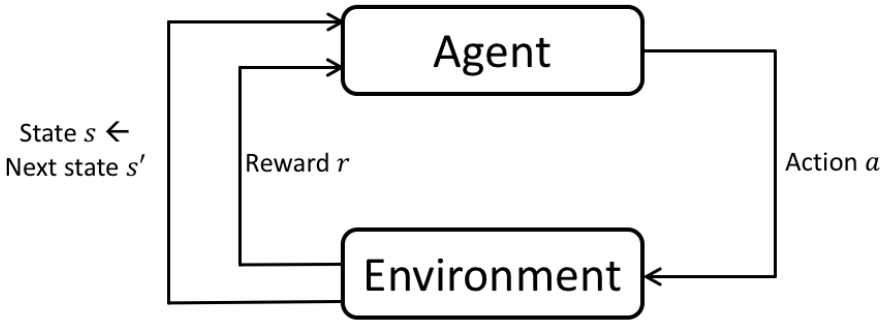

Figure 2.1: Interaction between an agent and the environment.

2.1.1 Markov Decision Process

The decision-making process in reinforcement learning is a Markov Decision Process (MDP), which is based on the Markov Property, where the next state only depends on the immediate current state and chosen action, not the previous states or actions, see Figure 2.2. The reader might be familiar with a Markov chain where a single action is possible in every state, in an MDP multiple actions are possible in a given

state (but only one is to be chosen). In an MDP the following concepts need to be defined:

$\mathcal{S}$ - set of possible states $s \in \mathcal{S}$,
$\mathcal{A}$ - set of possible actions $a \in \mathcal{A}$
$\mathcal{R}$ - reward function $\mathcal{R} : \mathcal{S} \cdot \mathcal{A} \to \mathbb{R}$, a distribution of the reward given state-action pairs.

$\mathcal{T}$ - transition function $\mathcal{T} : \mathcal{S} \cdot \mathcal{A} \to \mathcal{S}$, distribution over the next state given a state-action pair:
π - policy function $\pi : \mathcal{S} \to \mathcal{A}$, specifies what action to be taken in a state.
γ - discount factor $[0, 1)$, the importance of the future reward compared to the immediate reward.

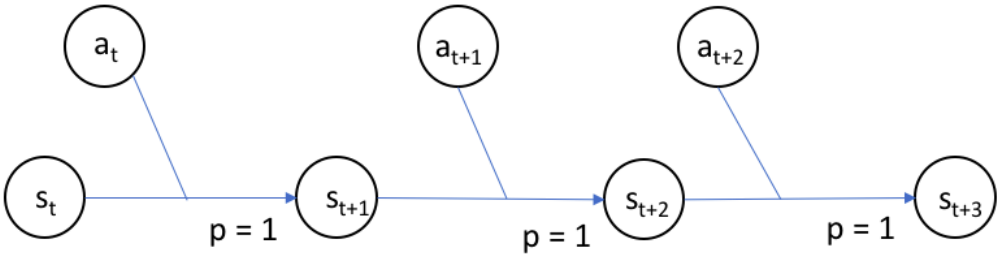

Figure 2.2: A Markov Decision Process.

The decision making process in reinforcement learning is as follows: At each time step, the agent is in some state $s \in \mathcal{S}$ and chooses one available action $a \in \mathcal{A}$. The environment returns the next state s' and corresponding reward r. The transition from s to s' depends on the transition function. The objective is to find an optimal policy π^* that maximizes the expected reward.

$$\max_{\pi} \mathbf{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right] \tag{2.1}$$

where γ is the discount factor $[0, 1)$.

2.1.2 Policy

A policy π is a function that maps the state to an action. Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, ...$ A policy π is called an ϵ-greedy policy if it in state s by probability ϵ assigns a random action $a \in \mathcal{A}$ instead of the optimal action in state s, where $0 \leq \epsilon \leq 1$.

2.1.3 Value function

To evaluate how good a state s is, a value function is defined. The value function is the expected cumulative reward from following the policy π from state s:

$$V^{\pi}(s) = \mathbf{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right] \tag{2.2}$$

2.1.4 Q-value function

To evaluate how good a state-action pair is, a Q-value function is introduced. It is defined as the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s, a) = \mathbf{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right] \tag{2.3}$$

2.1.5 Bellman Equation

The goal of reinforcement learning is to make optimal decisions that lead to maximal sum of expected rewards. The optimal value function is defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in S \tag{2.4}$$

The optimal Q-value function is defined as

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.5}$$

The policy π that satisfies (2.4) and (2.5) is defined as the optimal policy π^*. For an optimal policy the following relationship holds,

$$V^*(s) = \max_{a \in \mathcal{A}(s)} Q^*(s, a) \tag{2.6}$$

where $\mathcal{A}(s)$ is the actions available at state s. Hence (2.4) and (2.5) gives:

$$\begin{aligned} V^*(s) &= \max_a \mathbb{E}\left[\sum_{t\geq 0} \gamma^t | s_0 = s, a_0 = a, \pi^*\right] \\ &= \max_a \sum_{s'} p(s'|s, a)\left[R(s, a) + \gamma V^*(s')\right] \end{aligned} \tag{2.7}$$

(2.7) is known as the Bellman equation. If the transition function p and the reward function R is known then (2.7) and (2.6) gives:

$$Q(s, a) = r + \mathbf{E}\left[\gamma \max_{a'} Q(s', a')\right] \tag{2.8}$$

This is known as the Q-value of taking action a in state s. (2.8) can be solved using *Dynamic programming* methods such as the *Q-learning*.

2.2 Q-learning

Q-learning is a *Temporal Difference* algorithm that builds on the idea of calculating a temporal error, i.e. the difference between the new and the old estimate of the value function.

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{2.9}$$

where α is the learning rate, r is the reward received at current time step, s is the current state, a is the current chosen action and s' is the next state, resulting from taking action a in state s. The Q-learning algorithm is a tabular method that stores the Q-values for action-state pairs in a so-called *Q-table*. By interfering with the environment the Q-table is updated successively by trial and error. In a state s the action is chosen with respect to an ϵ-greedy policy and the Q-table is updated by the following:

$$Q(s,a)_{new} = \begin{cases} (1-\alpha)Q(s,a)_{old} + \alpha r & \text{if } s' \in \mathcal{S}_{goal} \\ (1-\alpha)Q(s,a)_{old} + \alpha(r + \gamma \max_{a'} Q(s',a')) & \text{else} \end{cases} \tag{2.10}$$

where $\alpha \in (0,1]$ and $\gamma \in [0,1)$.
In [7] it is proven that the Q-learning algorithm converges to an optimal policy if all state-action pairs, represented discretely, are visited repeatedly.

2.3 Deep Q-network

Q-learning generates a table of state-action pairs. When the state space increases or the state space is continuous, this table will take up a lot of memory. To try to overcome this, a so-called Deep Q-network (DQN) algorithm is used. It builds on the Q-learning algorithm but uses artificial neural networks to produce the Q-values. In Q-learning there is a table storing the Q-values $Q(s,a)$. To get the Q-values for all actions in state s, we can go to row s in the table and read out the columns. In DQN the Q-values are approximated by an artificial neural network. The input to the neural network is state s, and the network should output the Q-values for all actions in this state.

To be able to update the quality of the Q-values, the neural network needs a target value to compare the output with. This target is calculated as the following:

$$Q^{target} = \begin{cases} r & \text{if } s' \in \mathcal{S}_{goal} \\ r + \gamma \max_{a'} Q(s',a')) & \text{else} \end{cases} \tag{2.11}$$

where r is the reward received at current time step t, s is the current state, a is the current chosen action, s' is the next state and a' is the action in the next state s'.

The DQN uses artificial neural networks to approximate the Q-values. An artificial neural network aims to approximate a certain target given some inputs. The theory behind artificial neural network is inspired by the the nervous system of humans [11], even though the nervous system of humans are still unknown.

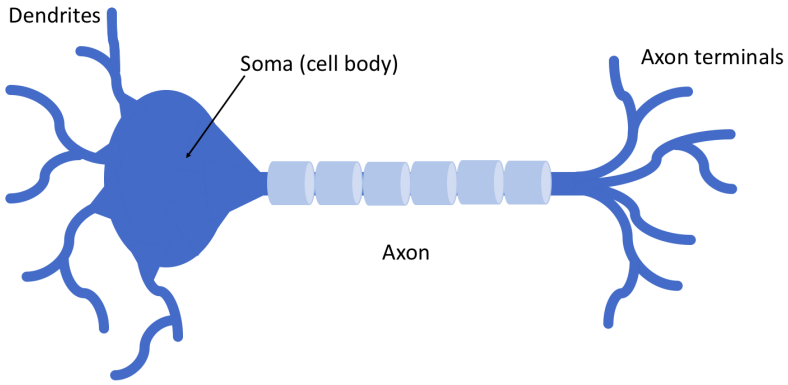

Figure 2.3: Illustration of a neuron.

In artificial neural network inputs and hidden layers aim to work as the dendrites and cell body in the neural network respectively, and outputs that works as the axon terminals.

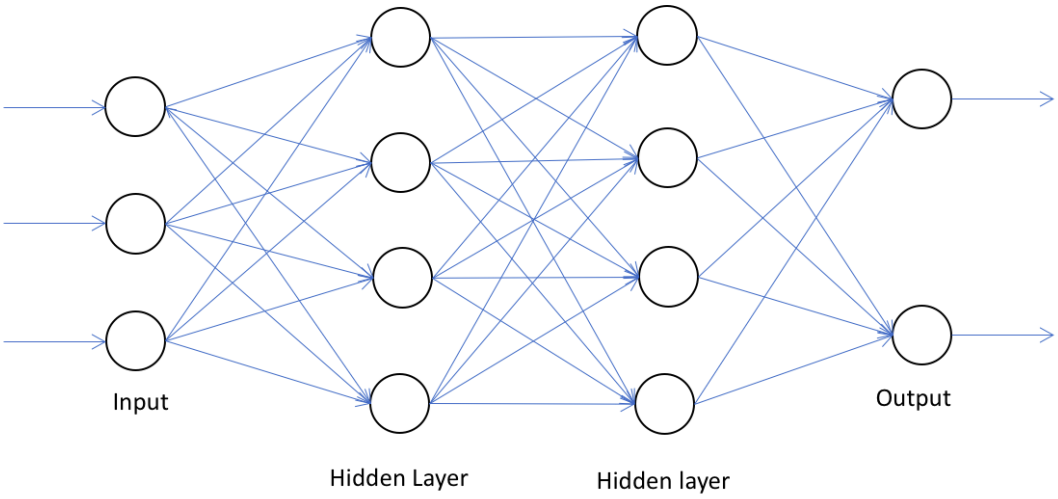

Figure 2.4: Structure of an artificial neural network.

Inside each of the neurons, represented by circles in Figure 2.4 the following calculations are done:

$$\boldsymbol{y} = f(\boldsymbol{x} \cdot \boldsymbol{\omega} + \boldsymbol{b}) \tag{2.12}$$

Where $\boldsymbol{y}$ is the output vector of the neural network, f is an activation function, $\boldsymbol{x}$ is the vector of the inputs, $\boldsymbol{\omega}$ is the matrix of all the weights and $\boldsymbol{b}$ is the vector of the biases.

The inputs are fed forward through the neural network and the neural network is trained by back propagating the errors. The errors are the mean squared error, (2.13), of the actual output compared to the desired output i.e. (2.11)

$$MSE(\theta) = \frac{1}{n}\sum_{i=1}^{n}(Q_i^{target} - Q_i^{output}(\theta))^2 \tag{2.13}$$

2.3.1 Activation function

The activation function of a neural network should act like a linear function, but should in fact be a non-linear function. The Rectified Linear Unit (ReLU) function is such a function. It is given by the following equation:

$$f(x) = \begin{cases} 0, \text{if } x < 0 \\ x, \text{if } x \geq 0 \end{cases} \tag{2.14}$$

This works as a threshold for the input

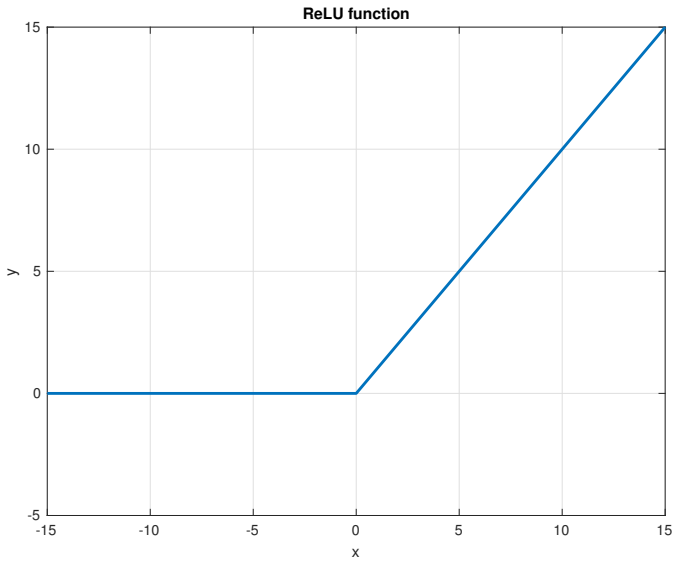

Figure 2.5: Plot of activation function ReLU.

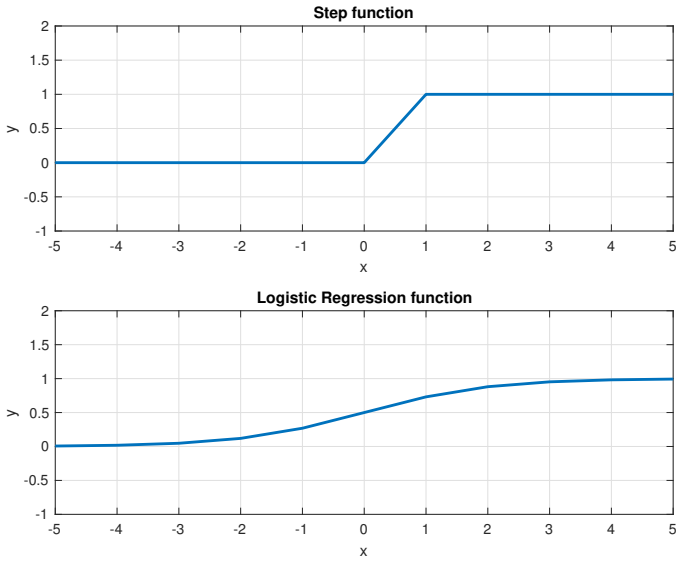

Figure 2.6: Plot of examples of other activation functions. Step function: the output is zero until the input exceeds a certain value, here 0.5, then the output will be 1. Logistic Regression function: the output is the following function of x: $y = \frac{1}{1+e^{-x}}$.

As seen in Figure 2.5, the input is only limited to be greater than or equal to zero and has no upper limit, this is an advantage with ReLU. For the step function and logistic regression function, as seen in Figure 2.6 the output is limited between 0 and 1. The stepfunction is good for classification problems, and logistic regression function is useful when the output are approximating a probability.

2.3.2 Number of hidden layers

A network consisting of three layers, the layer from the input, one hidden layer and the layer from the output are considered as a dense neural network. Adding additional layers results in a so-called deep neural network [12]. There is no direct guidelines on how many layers a neural network should contain. This is a matter of trial and error. There is still a matter of research how many layers one should have in a neural network, so it is a matter of trial and error.

2.3.3 Number of hidden neurons

The number of hidden layers and the number of neurons in each layer is also still a case of trial and error. Some guidelines exists and from [12] there are stated some rules of thumb:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.

- The number of hidden neurons should be less than twice the size of the input layer.

2.3.4 Optimization method

The update of the parameters θ in the neural network is done using a stochastic optimization method [13]. This method is a gradient descent method, but it uses random samples to evaluate the gradients. The method used in this thesis is the so-called "Adam" method, derived from "adaptive moment estimation" [14]. An advantage with this method is that is uses adaptive learning rate to update the parameters and it also uses momentum i.e. an average of past gradients when compute the new gradients.

2.3.5 Learning rate

The learning rate of a neural network is the step size towards the gradient. How certain you are about the target value and how fast the cost function changes, i.e. in which range the gradient is valid should determine the learning rate. A high learning rate corresponds to a higher certainty, and a lower learning rate should be used when not so certain about the target, or if the target will vary. If agents are to learn to cooperate they should not be so sure about the target value, because that depends on the other agents intelligence, which should be trusted with care. In a stationary environment the learning rate can be slightly higher, because a state-action pair always results in the same reward. In a non-stationary environment, such as in the MARL problem, the learning rate should be smaller, because the reward for a state-action pair depends on the other agents behaviour. The learning rate is also a matter of trial and error.

2.4 Multi-Agent Reinforcement Learning

When a single agent is acting in the environment without disturbances, the environment is stationary. When two agents interact in the same environment the environment is no longer stationary. A state-action pair might lead to different rewards at different time-steps depending on the other agent's behaviour.
An overview of MARL can be found in [15]. In the multi-agent case the MDP is slightly changed, and can be viewed in [16]. The following yields for each agent $i = 1, 2$:

- $\mathcal{S}_i$ - set of possible states $s_i \in \mathcal{S}_i$,
- $\mathcal{A}_i$ - set of possible actions $a_i \in \mathcal{A}_i$
- $\mathcal{M}_i$ - set of possible messages $m_i \in \mathcal{M}_i$
- $\mathcal{R}_i$ - reward function $\mathcal{R}_i : \mathcal{S}_i \cdot \mathcal{A}_i \rightarrow \mathcal{R}_i$, distribution of the reward given state-action pairs.
- $\mathcal{T}_i$ - transition function $\mathcal{T}_i : \mathcal{S}_i \cdot \mathcal{A}_i \rightarrow \mathcal{S}_i$, distribution over next state given state-action pair.
- π_i - policy function $\mathcal{S}_i \rightarrow \mathcal{A}_i$, specifies what action to be taken in a state.

- γ - discount factor $[0, 1)$, the importance of the future reward compared to the immediate reward.

MARL problems can be solved using ordinary reinforcement learning algorithms [17] such as Q-learning and DQN. Depending on how the problem is defined, e.g., if the reward function is dependent on all agents state-action-pairs, the solution can be optimized by introducing cooperation between the agents [17]. Examples of cooperation strategies are: shared policy, joint learning and communication between the agents. Cooperation strategies used in this thesis are described in Section 3.3.

3
Methods

In this chapter the problem is formulated in detail, the implementation is described and finally, the cooperation strategies are presented.

3.1 Problem formulation

The problem is an extension of the problem in [18]. The agents are interacting in a 3×3 grid world, see Figure 3.1. The agents will interact with this environment by moving in the grid and by reinforcement learning learn to reach the goal state after being randomly initiated at a starting position.

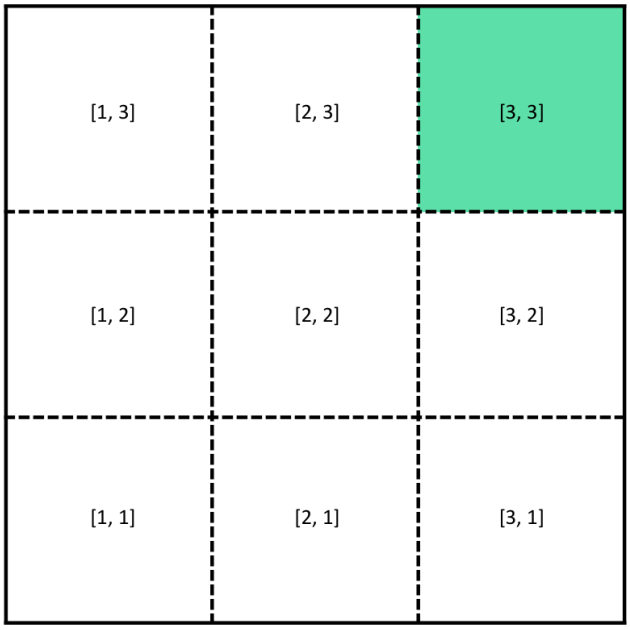

Figure 3.1: Visualization of the grid world. The colored square is the goal state. The states are labeled with coordinates.

The state space is discrete, and is a grid world with coordinates shown in Figure 3.1.

$$\mathcal{S} = \{[1,1],[1,2],[1,3],[2,1],[2,2],[2,3],[3,1],[3,2],[3,3]\} \tag{3.1}$$

The goal state to be reached is:

$$\mathcal{S}_{goal} = \{[3,3]\} \tag{3.2}$$

The action space is the following:

$$\begin{aligned}\mathcal{A} &= \{up, down, left, right, standing\ still\} \\ &= \{[0,1],[0,-1],[-1,0],[1,0],[0,0]\}\end{aligned} \tag{3.3}$$

If the action chosen results in the agent moving outside the grid, the agent will receive the penalty for moving a step, see the reward function below, but will remain in the same position. The MDP is deterministic, meaning that taking an action a in state s will always lead to state s'. The message space is:

$$\mathcal{M} = \{0, 1\}$$

The reward function is defined in ((3.4)). The reward is assumed to only depend on the environment state s'.

$$\mathcal{R}(s,a) = \begin{cases} 9 & \text{if } s_1 \text{ and } s_2 \in \mathcal{S}_{goal} \\ 4 & \text{if } s_1 \text{ or } s_2 \in \mathcal{S}_{goal} \\ 0 & \text{if } a = \text{standing still} \\ -1 & \text{else} \end{cases} \tag{3.4}$$

where s is the current state, a is the current chosen action and s' is the resulting next state.
Finally, the discounting factor γ and the learning rate α are set as the following:

$$\gamma = 0.9$$
$$\alpha = 0.1$$

A visualization of the Q-values is shown in Figure 3.2. Appendix A shows the evolution of Q-values.

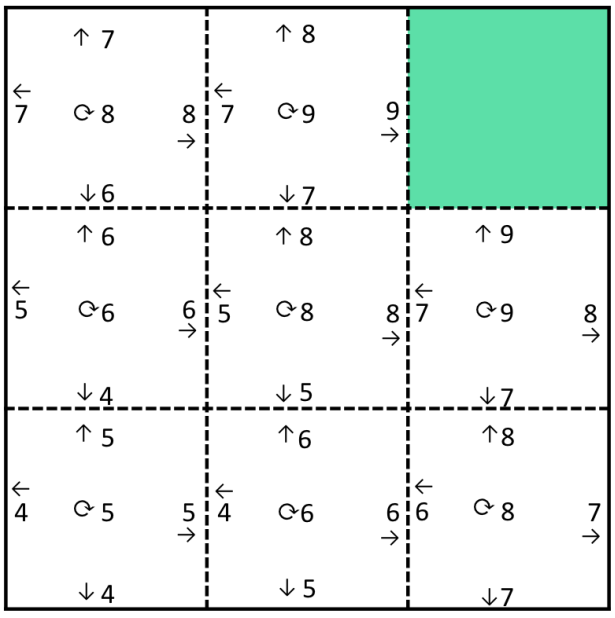

Figure 3.2: Visualization of the Q-values for each action in each state. The numbers are the Q-values of taking the action of the corresponding arrow. When the reward function is like (3.4), $\gamma = 1$ and $\alpha = 1$.

Two agents interact in the same environment. The agents can occupy the same state at the same time step.

3.2 Implementation

Both Q-learning and DQN are implemented in Python. The algorithm is as follows:

```
Initialize Q-table(s) / Q-network
for training episode do
    Initialize state s
    while not terminated do
        Choose action a according to an ϵ - greedy based on state s observed
         from the environment
        Feed a to environment
        Get r and s' from environment
        Update Q-table/Q-network
    end
end
```

Algorithm 1: Q-learning/DQN

An episode terminates if $s' \in \mathcal{S}_{goal}$ or if the agents have taken 20 actions in the current episode.

The Q-table and Q-network are updated according to (2.10) and (2.11) respectively.

To speed up the learning process and reduce the correlation, all the state-action-next state-reward groups are stored in a replay memory [19], and samples are drawn from this in each step. This replay memory is overridden when it exceeds 100 groups.

3.3 Cooperation strategies

Cooperation and coordination have been investigated in [9] and [17]. In this thesis the following cooperation strategies will be implemented: independent interaction, joint interaction, and communication, as well as shared policy. Communication have been investigated in [18] and [20], however in this thesis the actual communication is instantaneous and noiseless.

3.3.1 Independent Q-learning

The agents do not communicate and they interact with the environment independently. However, they share the same policy and therefore they also share Q-table/Q-network. The size of the state space is here 9, and the size of the actions space is 5, hence the number of elements in the Q-table is 45.

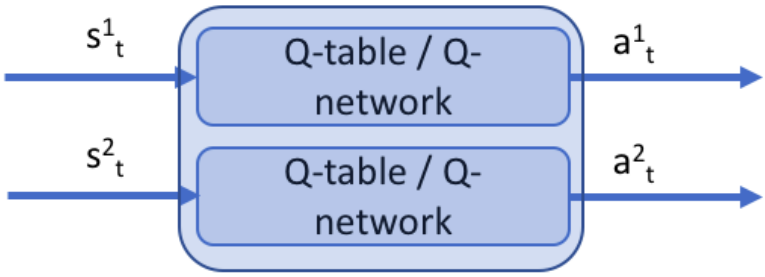

Figure 3.3: Input and output of Q-table/Q-network in the independent case. The outer blue square illustrates shared policy. The subscript index is the time step and the superscript is the index of an agent.

3.3.2 Joint Q-learning

The two agents are now jointly interacting with the environment. The Q-table contains state-action pairs, where the state is both of the agent's states and actions combined. The state and actions are now the combined state and action of both agents, the size of the state space is therefore 9^2, and action space is of size 5^2. Hence, the number of elements in the Q-table is now $81 \cdot 25 = 2025$. The benefit of this method is that the environment is now stationary and the information about the environment is complete. The input to the Q-network is both the agent's states represented by coordinates and the output is the Q-values for a combination of both agent's actions.

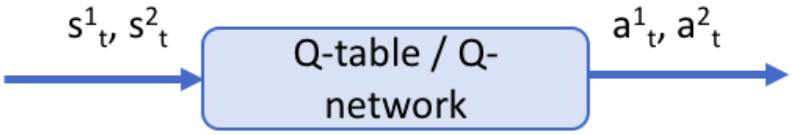

Figure 3.4: Input and output of Q-table/Q-network in the joint case. The outer blue square illustrates shared policy The subscript index is the time step and the superscript is the index of an agent.

3.3.3 Communicating state

Here, the agents communicate their state to each other. This situation will be called a fully observable case. Now the agents know both their own and the other agent's state. Hence, the state is now, as in the independent case the combined states and the state space is therefore of size 81 also here. The action space is not changed compared to the independent case, hence of size 5. The number of elements in the Q-table is here $81 \cdot 5 = 405$ Here, the agents also share policy and therefore also Q-table/Q-network.

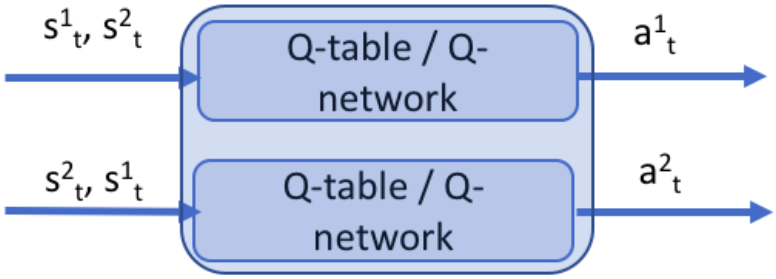

Figure 3.5: Input and output of Q-table/Q-network in the case where the agents communicate their state. The outer blue square illustrates shared policy. The subscript index is the time step and the superscript is the index of an agent.

3.3.4 Communicating message of one bit

Here the agents are communicating a message whether they are one step from the goal or not. The message is set explicitly to either 0 or 1, representing one step away from the goal or not respectively. The state space is of size $9 \cdot 2 = 18$ and the action space is of size 5, hence the size of the Q-table is $18 \cdot 5 = 90$

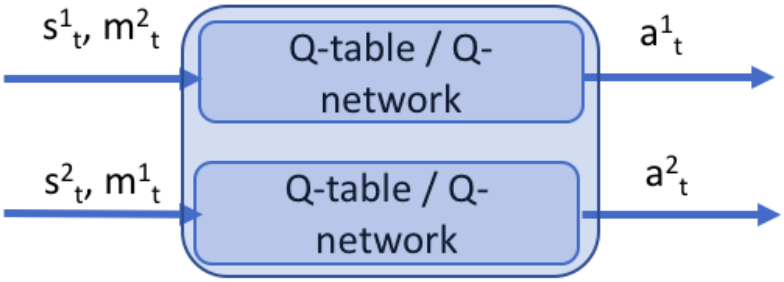

Figure 3.6: Input and output of Q-table/Q-network in the case where the agents communicate a message. The outer blue square illustrates shared policy. The subscript index is the time step and the superscript is the index of an agent.

3.3.5 Learning to communicate

Previously the communication between the agents is set explicitly. For the agents to be independent of such external help, they should learn by themselves what to communicate. Given its own state, the agent should decide whether to send a 0 or a 1. In addition to a Q-table/Q-network as in the case when communicating one bit, the agents have a separate Q-table/Q-network to learn what to communicate. The input to this is the agent's own state, and the output is the message to be sent, either a 0 or a 1. The Q-values are set as in (2.10). In this case, the agents do no longer share policy, due to the risk of confusing each other, and hence they have their own Q-tables/Q-networks. The agents are training separately, switching every 30 training episodes.

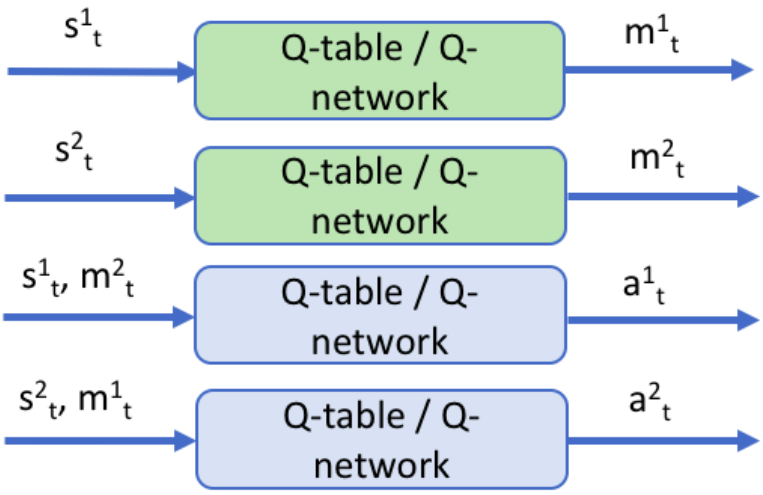

Figure 3.7: Input and output of Q-tables/Q-networks in the case where the agents learn what to communicate. The subscript index is the time step and the superscript is the index of an agent.

3.3.6 Exploration and exploitation

As mentioned, the Q-values are improved successively as they are visited by the agents. The agents must explore the environment to visit the states often enough. However, they also need to exploit the ”good” actions enough to converge to the correct Q-values, hence an issue of balancing exploration and exploitation rises. To much exploration might take a longer time to converge, but to little might result in local optimums. When should the agents be satisfied with its current optimal solution, and when should the agents keep searching for better solutions? This is a case of trial and error, depending on the size and complexity of the problem.

The actions are chosen according to an ϵ-greedy policy. With a perfectly converged model, the agents should always choose the action with the highest Q-value. For the Q-learning algorithm to converge it is important that all states are visited successively. In this thesis ϵ is set to 1 in the beginning and then decreased by a small factor, ϵ_{decay} after each training episode, see 3.5.

$$\epsilon_t = \epsilon^t_{decay} \tag{3.5}$$

In Figure 3.8 the change of ϵ with respect to the training episode can be seen.

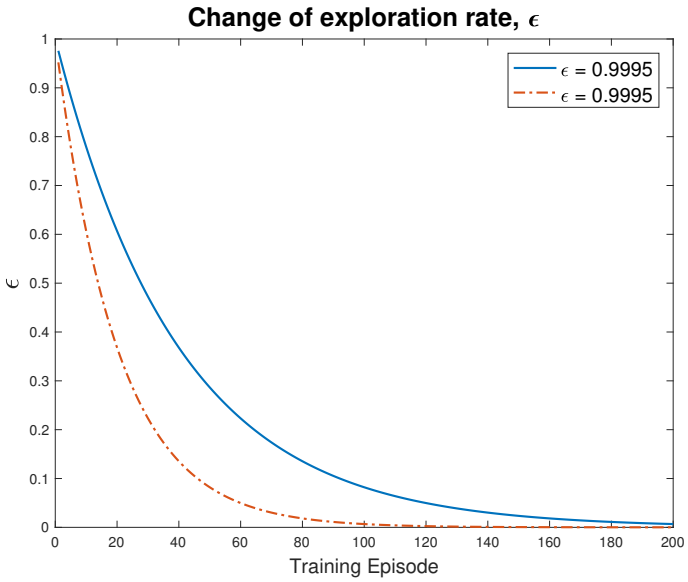

Figure 3.8: Decrease of ϵ with different ϵ-decay.

4
Results

4.1 Performance metrics

The methods will be compared by the two agents' ability to reach the goal simultaneously. When this happens the biggest reward is issued for both of the agents, see reward function in (3.4), hence this should reflect the achievement of the largest long term reward.

4.2 Validation of the models

For each 100 training episode, a validation of the model is done. The validation is as follows: 1000 validation rollouts are run, and the average number of common goals out of 100 is counted and stored. The validation run is similar to the training run, except here the agents always chooses the maximum action, i.e. $\epsilon = 0$ and no updates of Q-tables/neural network are done. As mentioned in the theory chapter, there are some variables that can only be tuned empirically. Hence, the time until convergence could be optimized further, but the results are exact enough for a comparison.

4.3 Parameters

In Table 4.1 and Table 4.2 the parameters and sizes of Q-table and Q-network respectively are shown.

Q-learning	ϵ_{decay}	# elements in Q-table
Independent	0.999	45
Joint	0.999	2025
Fully observable	0.9999	405
Message	0.999	90
Learn message environment	0.9995	90
Learn message communication	0.9995	36

Table 4.1: Parameters and size of Q-table for each cooperation strategy

DQN	ϵ_{decay}	Learning rate	# layers in Q-network	# neurons in each layer
Independent	0.999	1e-2	3	8 - 8 - 8 - 8
Joint	0.9993	1e-3	3	12 - 30 - 30
Fully observable	0.9999	1e-2	3	10 - 10 - 10
Message	0.999	1e-2	3	8 - 8 - 8
Learn message environment	0.99999	1e-5	4	8 - 8 - 8 - 8
Learn message communication	0.99999	1e-5	4	5 - 5 - 5 - 5

Table 4.2: Parameters and size of Q-network for each cooperation strategy

4.4 Results

4.4.1 Q-learning

In Figure 4.1 the results of the Q-learning algorithm is shown.

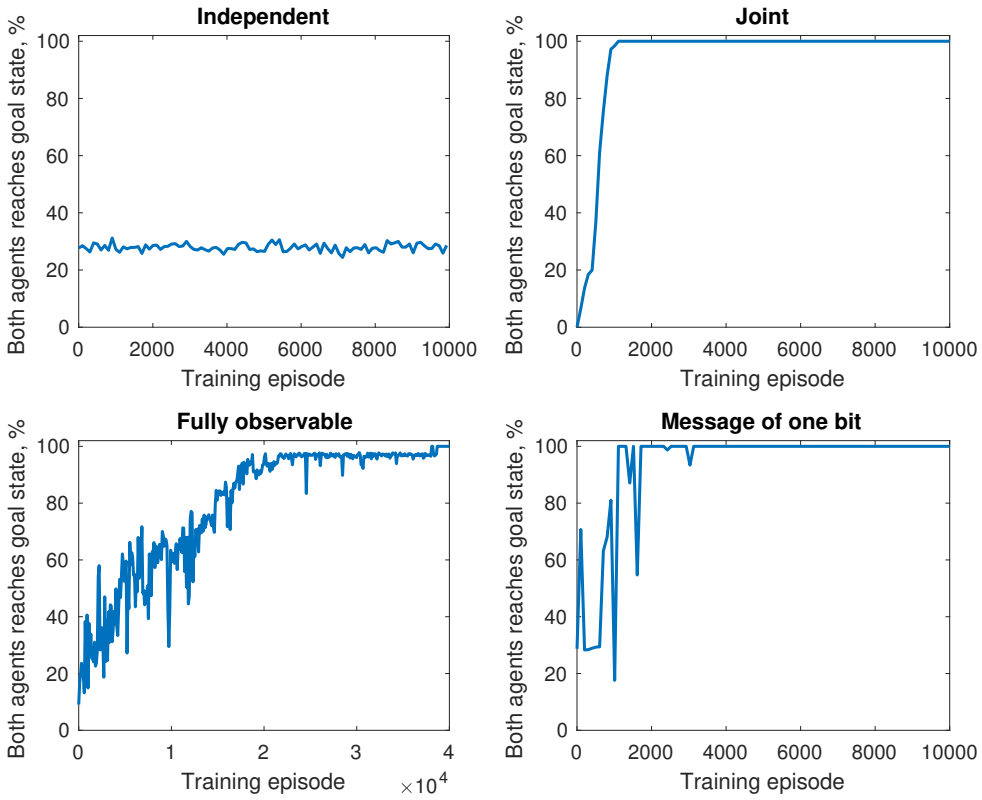

Figure 4.1: Performance of Q-learning.

In the independent case, there is no cooperation as expected due to the incomplete information in the state, other than shared policy. In a 3×3 grid with 8 possible

initial states, it is approximately 30 % chance that the agents will be initialized equally close to the goal. Taking the maximum actions to the goal from here results in the agents reaching the goal at the same time step. The joint control of the agents behaves as a single agent in a bigger environment and reaches a 100 % performance fast. In the fully observable case, they are able to reach close to 100% as well. In the fully observable case they dependent on both agents actions to be correct at all states, hence the curve is more oscillated. In the set message case the agents learn a bit faster to cooperate than in the fully observable case, this is because the information about the other agent's state are simpler, either a 0 or a 1, than in the fully observable case where the information about the other agent's state is the state represented by coordinates.

4.4.2 DQN

In Figure 4.2 the results of the DQN algorithm is shown.

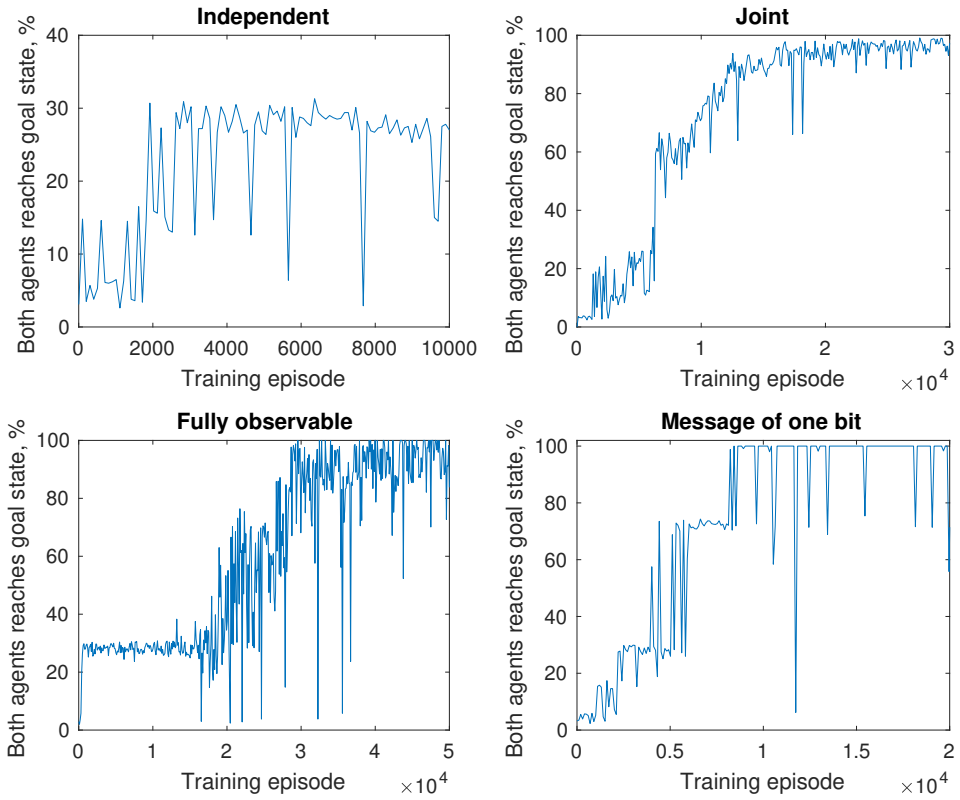

Figure 4.2: Performance of DQN.

The performance of the DQN is similar to the Q-learning, only now the agents need more training to perform equally well. Also, the methods are a bit more uncertain (oscillating) because the artificial neural network is an approximation, rather than an explicitly updated value as in the Q-learning.

4.4.3 Learning to communicate

The problem of learning to communicate introduces two operations to be learned simultaneously.

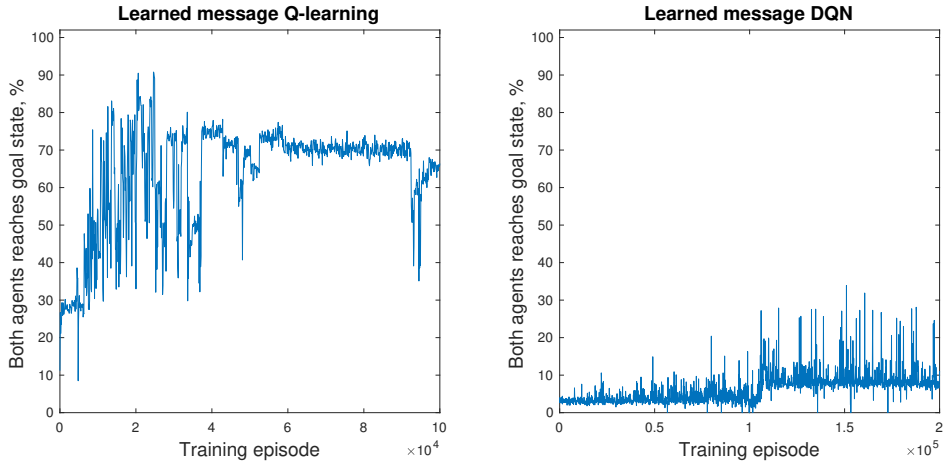

Figure 4.3: Learning to communicate. The left plot is when using Q-learning, and the right plot is when using DQN.

As seen in Figure 4.3 the agents are able to learn, using Q-learning, to send and interpret a message which enables cooperation. In the DQN, on the other hand, they are not able to learn anything useful in a reasonable time. The problem of learning both movement and communication simultaneously is a huge challenge for the DQN. There are both uncertainties from the other agent's behaviour, as in the other multi-agent cases, and the other agent's message, hence the learning rate of the neural networks must be so small that they need extensively much training.

4.4.4 Performance on bigger environments

The Q-learning method needs to have a table the size of the number of states times the number of actions. Hence, if the agents are being introduced to a new environment they will need to store Q-values for all the new states. The Q-network in the DQN method can take as input the coordinates for the new states and still produce nice results, hence it only needs to train on the smaller network.

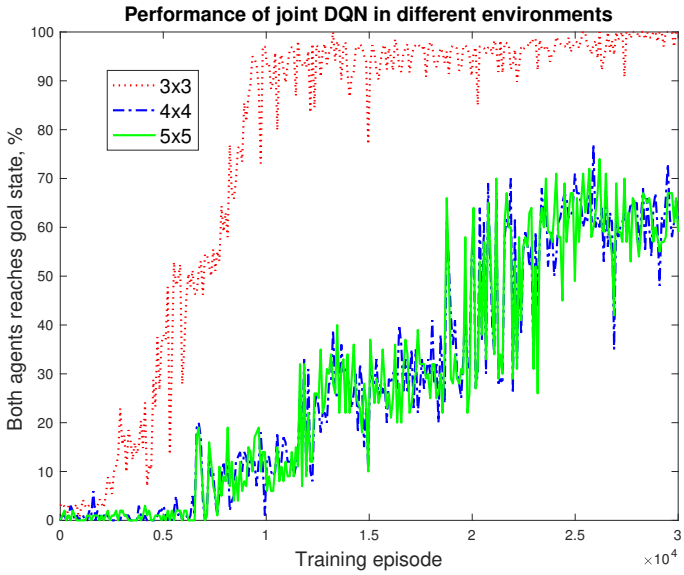

Figure 4.4: Performance on bigger environments for the DQN method. The model is trained on a 3×3 environment but validated also for 4×4 and 5×5 environment.

In Figure 4.4 it can be seen that the trained model also shows results of quite succeeded cooperation between the agents.
In comparison to the tabular case, a model has been trained on the three environments separately and validated as earlier.

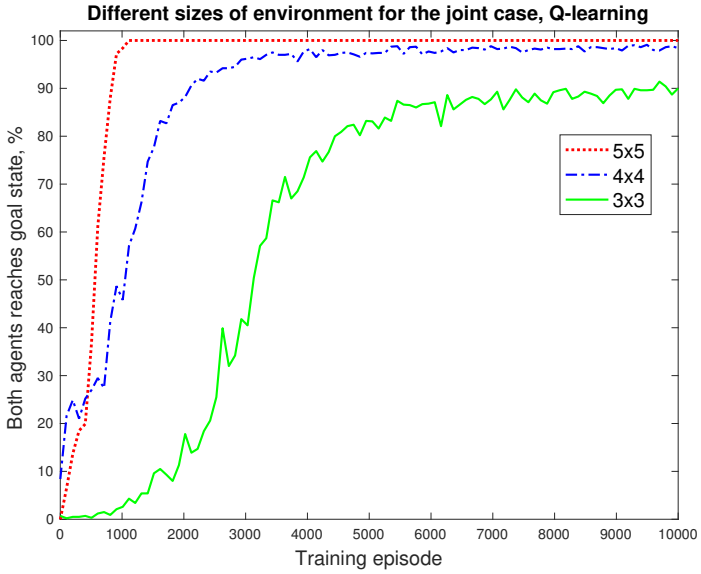

Figure 4.5: Performance on bigger environments for the Q-learning method. ϵ_{decay} is the same in all three cases.

In Figure 4.5 it can be seen that the convergence to a good result is quite fast also for the bigger environments. The lower converged value for the 4×4 and 5×5

is due to the exploration rate, ϵ_{decay}, being the same in all three cases. Below, experimentation with the exploration rate, ϵ_{decay}, can be seen.

4.4.5 Experimenting with exploration rate

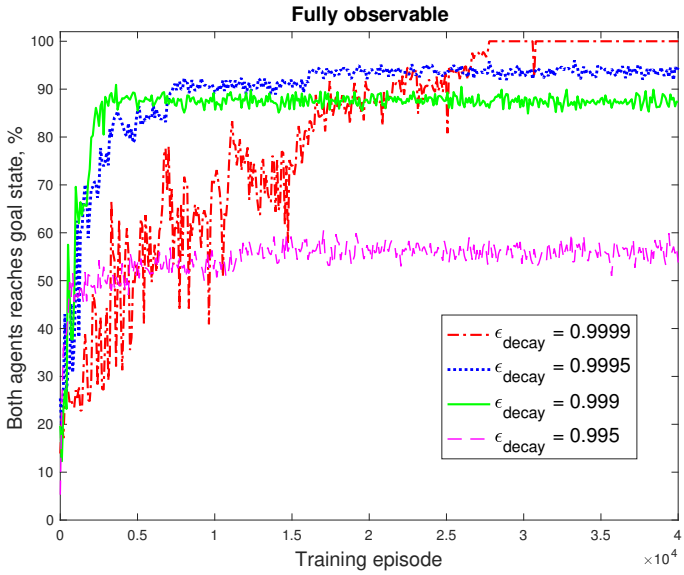

Figure 4.6: Result for different exploration rates for fully observable Q-learning.

In Figure 4.6 the result of different exploration rate, ϵ, is shown. It can be seen that the lower the exploration rate, the faster the convergence. However, the agents are stuck in local optimums for the lower exploration rates. Hence, if interested in perfect behaviour one should increase the exploration rate, but for quick alright results, the exploration rate can be decreased in favour of exploitation.
In the table below, the comparison of the computational time for Q-learning and DQN can be viewed.

Method	Q-learning	DQN
Independent	4.9068 s	13.3938 s
Joint	4.4785 s	9.6212 s
Fully observable	4.5026 s	21.1810 s
Message of one bit	4.4569 s	18.7594
Learn message	4.7463 s	21.1104 s

Table 4.3: Computational time for 100 training episodes

Table 4.3 shows that the computational time is much larger for the DQN then Q-learning. Keeping in mind that the DQN models needed extensively more training episodes to perform on the level as Q-learning, Q-learning might be preferred over DQN for such small problems as in this thesis. However, the number of elements in a Q-table is increasing fast when the environments complexity increases.

5
Discussion

5.1 Conclusion

In this thesis agents' ability to cooperate has been investigated. It is shown that there are ways for the agents to cooperate to reach a common goal. For the agents to be totally independent of external help, with predefined communication, they have been trying to learn how to send and interpret useful messages. The result in this thesis shows that with Q-learning, the agents have been able to learn to send and interpret messages to be able to cooperate. When implementing this using DQN, the learning rate of the Q-network had to be so small that the agents were not able to learn anything in a reasonable time. Due to this, it is tempting to conclude that the Q-learning method is to be preferred in such a small environment as a 3x3 grid world. However, when introduced to bigger environments it is shown that as the size of the environment increases, DQN might have an advantage compared to Q-learning due to the ability to perform well in a bigger environment even if trained on a smaller environment. This shows that the DQN algorithm is robust to small changes. It might be that the environment includes more disturbances other than one other agent, hence it is an advantage to be able to perform well on something one might not have specifically trained on.

5.2 Future work

It is briefly shown in this thesis that DQN can be trained on a smaller environment and applied to bigger environments with success. It would be interesting to take this idea further. To divide a complex environment into smaller pieces, and see if there is an advantage of training on a smaller network and then train the model further in an extended environment.
It would also be interesting to optimize the DQN algorithm and the Q-network further to be able to make the agents learn what to communicate also with DQN. This will make this algorithm robust.

5.3 Ethical aspects

Machine learning is a branch of artificial intelligence and there are several ethical aspects regarding this. Some of them are listed here.

- **Elimination of jobs**: There are discussions about how much one should let, more or less intelligent, machines take over human jobs. It can improve efficiency in many situations, but society might need these types of jobs to employ people that might not are able to perform other jobs. In [1] it is estimated that 7-9 % of jobs in Norway are "lost to automation" over the period 2009-2014. However, automation is also opening up for new jobs. Developers are needed, but also jobs where digital expertise are not demanded such as delivery jobs due to increased e-retail. According to [1] 1.2 % new jobs were "created" over five years.
- **Military aspect**s: Lethal autonomous weapons. Weapons that are able to interact with the environment and be able to make decisions about shooting at the enemy. There are some more or less automated weapons today, and scientists predict that this will develop in the direction of machine learning. The United Nations are working on an ethical basis of human control, but there will be difficulties making sure this complies [21].
- **Fake news**: There will be possible to generate fake information that is trustworthy [22] which might lead to difficulties with telling fake news from real news. However, AI might also help us detect fake news [23].

Regardless of the ethical aspects, the field of machine learning will continue to develop. Hence, it is important to contribute to an ethical evolution of this field.

Bibliography

[1] S. Fölster, *"Norway's new jobs in the wake of the digital revolution"*(2018)

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis *"Mastering the Game of Go without Human Knowledge"* (2017)

[3] J. Rodriguez-Ramos *"Brains vs. Computers"* (2018) Available at: https://becominghuman.ai/brains-vs-computers-f769548010f1

[4] D. Parkes *"Machine Learning vs Rules Systems"* (2017) Available at: https://deparkes.co.uk/2017/11/24/machine-learning-vs-rules-systems/

[5] O. Simeone*"A Very Brief Introduction to Machine Learning"* (2018) With Applications to Communication Systems

[6] M.E. Harmon, S. S. Harmon *"Reinforcement Learning: A Tutorial"* (2000)

[7] P. Dayan, C. Watkins,*"Q-learning"* (1992)

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, *"Playing Atari with Deep Reinforcement Learning"* (2013)

[9] L. Matignon, G. J. Laurent, N. Le Fort-Piat *"Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems"* (2012)

[10] R. Sutton, *"Reinforcement Learning: An Introduction"* (1998)

[11] A. Jain, (2016) https://www.analyticsvidhya.com/blog/2016/03/introduction-deep-learning-fundamentals-neural-networks/ Accessed: 22.03.2019

[12] J. Heaton, (2017) https://www.heatonresearch.com/2017/06/01/hidden-layers.html Accessed: 16.04.2019

[13] S. Ruder, *"An overview of gradient descent optimization algorithms"* (2017)

[14] D. P. Kingma, J. L. Ba, *"Adam: A Method for Stochastic Optimization"* (2017)

[15] L. Busoniu, B. De Schutter, R. Babuska *"Multi-agent Reinforcement Learning: An Overview"* (2010)

[16] E. Yang and D. Gu *"Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey"* (2014)

[17] C. Guestrin, M. Lagoudakis, R.Parr *"Coordinated Reinforcement Learning"* (2002)

[18] A. Mostaani, S. Osvaldo, *"On Learning How to Communicate Over Noisy Channels for Collaborative Tasks"* (2018)

[19] R. Liu, J.Zou*"The Effects of Memory Replay in Reinforcement Learning"* (2017)

[20] J. N. Foerster, N. de Freitas, Y. M. Assael1, S. Whiteson, *"Learning to Communicate with Deep Multi-Agent Reinforcement Learning"* (2016)

[21] International Committee of the Red Cross *"Ethics and autonomous weapon systems: An ethical basis for human control?"* (2018)

[22] W. Knight *"An AI that writes convincing prose risks mass-producing fake news"* (2019) Available at: https://www.technologyreview.com/s/612960/an-ai-tool-auto-generates-fake-news-bogus-tweets-and-plenty-of-gibberish/

[23] K. Hao, *"An AI for generating fake news could also help detect it"* (2019) Available at: https://www.technologyreview.com/s/613111/an-ai-for-generating-fake-news-could-also-help-detect-it/

A

Appendix A

An example of how the Q-values are updated for each episode of interaction with the environment. An episode starts with the agents being initialized at random starting positions, and terminates when one or both of the agents reaches the goal state. The Q-values are updated according to (2.10). Figure A.1 and Figure A.2 shows the update of the Q-values for two episodes.

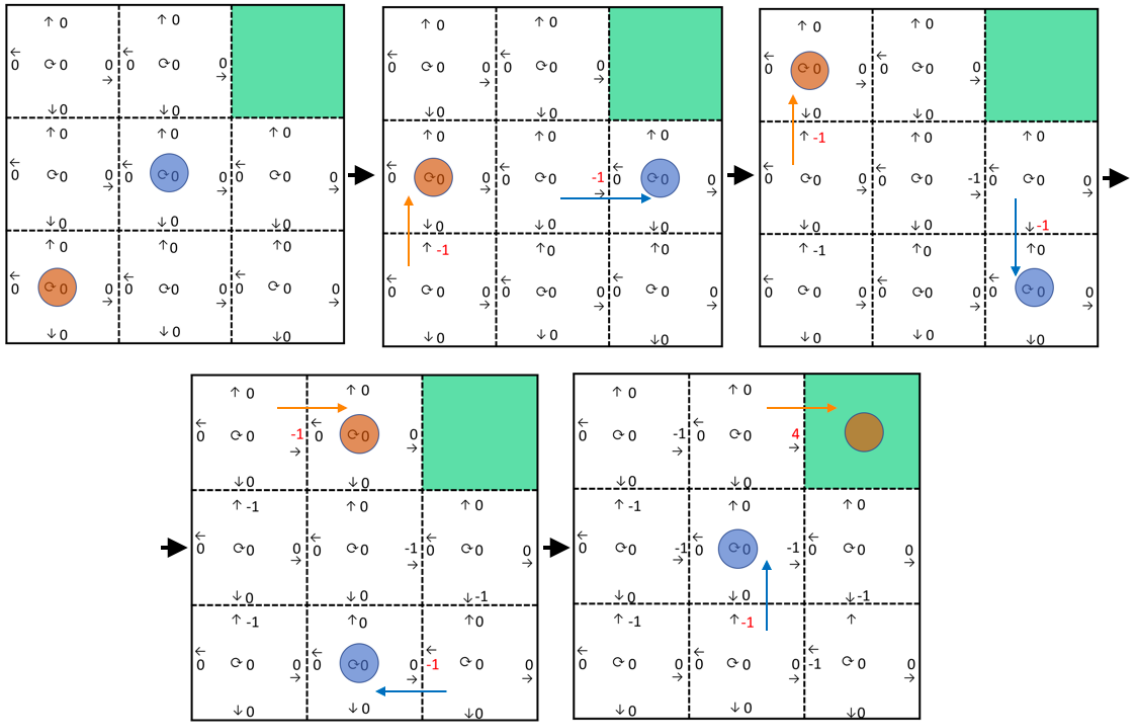

Figure A.1: Update of Q-values in first episode

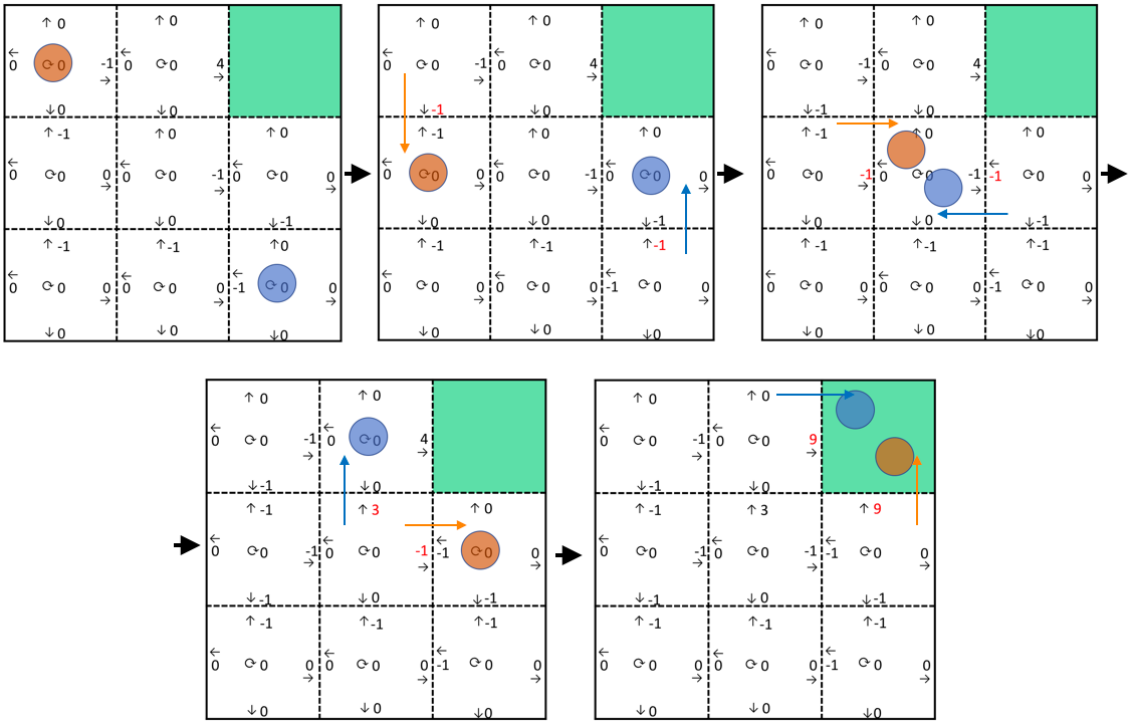

Figure A.2: Update of Q-values in second episode