DTU Electrical Engineering
Department of Electrical Engineering

Deep Reinforcement Learning for Bidding Strategies in Peer-to-peer Energy Markets

Master Thesis

Deep Reinforcement Learning for Bidding Strategies in Peer-to-peer Energy Markets

Master Thesis
August, 2020

By
Lucas Beltram

Copyright:	Reproduction of this publication in whole or in part must include the customary bibliographic citation, including author attribution, report title, etc.
Cover photo:	Colourbox
Published by:	DTU, Department of Electrical Engineering, Brovej, Building 118, 2800 Kgs. Lyngby Denmark www.elektro.dtu.dk

Approval

This thesis has been prepared over six months at the Centre for Electric Power and Energy, Department of Electrical Engineering at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Science in Electrical Engineering.

Lucas Beltram - s182360

[illegible]

Signature

August 3, 2020

Date

Abstract

The current energy transition experienced globally, with increasing deployment of distributed energy resources and advances in information and communication technologies, brings the possibility of rethinking how individuals trade electricity. However, to seize the opportunities brought by the upcoming paradigm, the framework of existing markets must be restructured.

A peer-to-peer market structure, in which consumers and communities directly trade electricity with each other, is capable of taking full advantage of the opportunities brought by the upcoming paradigm of active consumers. The advantages of such a market go beyond enabling direct access to individuals: it allows electricity to be product-differentiated according to non-price attributes so that market participants can express their product preferences. For instance, an individual may prefer buying electricity generated from wind energy over a regular retailer.

This work formulates the decision-making problem of an individual trading electricity in a continuous double auction-based market and investigates the suitability of applying Reinforcement Learning in the context of trading activity automation. Specifically, we modeled the electricity market deployed in the Svalin community as part of the Energy Collective project and investigated the applicability of the Deep Q-Network algorithm.

An investigation of the discount factors showed that, for the problem formulated in this work and the assumptions made to the model of the environment, the performance of Reinforcement Learning agent trained with the Deep Q-Network algorithm is maximized by not discounting the future rewards. In this case, the individual (market participant) sacrifices lower immediate rewards to get higher rewards in the future. The results also indicate that the discount factor impacts the way an agent explores the environment's state space.

The suitability of the Deep Q-Network algorithm for different problem's complexity was investigated by training agents for models of the environment considering additional dimensions. We analyzed the performance of Reinforcement Learning agents interacting with environments considering 3 and 8 products available in the market, for an episode with 36 and 288 time steps. The results showed that, for the problem formulated and assumptions made, the complexity is more sensible to the number of time steps than the number of products available.

The electricity market's product differentiation capability was explored to introduce the possibility of expressing the agent's product preferences according to non-price attributes. By introducing a vector of preference factors to the reward function, agents learned policies that reflected their product's preferences, without impacting the performances obtained.

Acknowledgements

First, I want to thank my supervisor Pierre Pinson for allowing me to work with such an exciting topic and for all his effort in the Renewables in Electricity Markets course, which sparked my interest in the subject of this thesis.

I also want to thank my other supervisor Tiago Sousa for the constant support and guidance during the development of the project. The combination of his good mood and the constructive discussions during our weekly meetings was essential to keep me working during tough days.

Furthermore, I want to thank another supervisor, Morten Herget Christensen, for encouraging me to work with Reinforcement Learning and tranquilizing my anxieties while working under uncertainty. His mentoring along my entire journey at DTU – during a special course and master thesis – pushed me forward to achieve higher results.

I was fortunate to have a forth brilliant person guiding me during the development of the project. I want to thank Helge Stefan Esch for all his advice and discussions during these six months. The energy and enthusiasm in his actions to help me to succeed in the project are inspiring and worthy of mention.

Finally, I want to express my deepest gratitude for all the love and support I have always received from my family. Mãe e pai, todas as conquistas e os sonhos que estou realizando são pagos com o preço de estar longe de vocês. Sinto falta de vocês todos os dias.

Um menino caminha e caminhando chega no muro
E ali logo em frente, a esperar pela gente, o futuro está.

Toquinho & Vinícius de Moraes

Nomenclature

Abbreviations

ANN	Artificial Neural Network
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Networks
EV	Electrical Vehicle
ICT	Information and Communication Technology
MDP	Markov Decision Process
P2P	Peer-to-peer
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent

Symbols

α	Learning rate
β	Fixed quantity in the orders placed by the agent
χ_i	Preference factor for product i
$\dot{\chi}$	Vector of preference factors
ϵ	Probability of taking a random action
γ	Discount factor
$\hat{p}_{t,i}$	Normalized price of product i in time step t
$\hat{Q}$	Target network
$\mathcal{A}$	Set of possible actions in the environment
$\mathcal{S}$	Set of possible states in the environment
$\mathcal{T}$	Single transition of the MDP

μ_t	Progress of the episode
ω_t	Ratio between current holdings and agent's flexible demand
π	Policy
π^*	Optimal policy
ψ	Mean reversion speed
σ	Parameter that represents volatility of prices
θ	Set of weights and biases of the ANN
Υ	Long run average followed by the stochastic process
$\boldsymbol{\hat{p}^{min}}$	vector with the K-lowest normalized prices observed in a episode
a_t	Action taken by the agent in time step t
B	Size of the experience replay buffer
c	Brownian scalling factor
D_{flex}	Flexible demand
dt	Time step size
dW	Gaussian noise
E_{size}	Size of the experience replay buffer
$H_{i,t}$	Agent's holdings of product i
H_T	Agent's total holdings in the end of episode
I_i	Preference intensity for product i
K	Number of buy orders that need to be placed to meet agent's flexible demand
L	Loss function
N	Number of availble products in the market
$p_{t,i}$	Price of product i in time step t
p_{trend}	Price trend followed by products
Q	ANN that approximates the Q-value function

Q_*	Optimal action value function / Optimal Q-value function
Q_π	Action value function / Q-value function
R_t	Return in time step t
R_{max}	Maximum return that could be obtained in a episode
r_{t+1}	Reward received for action taken in time step t
s_t	State observed by the agent in time step t
T	Number of time steps of an episode
t	Time step index
T_{sync}	Number of time steps to synchronize target network
V_*	Optimal state value function
V_π	State value function
W_{flex}	Flexibility window
x	Stochastic variable that represents the price

List of Figures

List of Tables

Contents

1 Introduction

1.1 Motivation

The energy transition towards a more environmentally friendly and sustainable energy system has reached a stage where renewable energy capacities deployed in certain countries and regions of the world are dominating the generation mix [1]. Recent advances in Information and Communication Technology (ICT) support this energy transition that brings the deployment of distributed energy resources such as solar photovoltaic panels, small wind turbines, and storage devices (e.g., electric vehicles and household batteries).

Besides, recent advances in computing power and novel technologies such as blockchain (open distributed ledgers of transactions) are making people rethink how electricity is managed [2]. The increasing engagement of social commitment of individuals towards better energy usage (those called energy citizens) supports this transition and technology-based innovation [3]. Consequently, the management and operation of power systems are gradually moving away from a centralized structure, becoming more dynamic, digitized, and decentralized.

Unfortunately, electricity markets have not yet evolved to seize the opportunities brought by this new scenario [4]. However, this transition is favoring the emergence of the so-called consumer-centric electricity markets [1]. Consumer-centric electricity markets rely on peer-to-peer (P2P) and community-based structures [5, 6], which are conceptually inspired by existing platforms of the sharing economy such as AirBnB and Uber. In these structures, all peers (or agents) cooperate with what they have available for commons-based producing, trading, or distributing a good or service [7]. For electricity markets, this increases awareness, transparency and empowerment of consumers [8], allowing direct access for all individuals.

This market structure also enables electricity product differentiation according to non-price attributes [9]. In this case, peers can express and act according to their preferences. For instance, the market can have separate products for electricity generated from renewable sources or fossil fuels, and the market participant can decide which product to trade. Another example of product differentiation is to distinguish them according to electricity produced locally or remotely.

Sousa et al. [7] reviewed P2P and community-based electricity markets and provided a comprehensive understanding of three relevant consumer-centric electricity markets: full P2P market, community-based market, and hybrid P2P market. The authors identified two main challenges to be researched in these market typologies: First, the problem quickly becomes intractable when the problem is scaled to a bigger context. Second, the state of available information might diverge amongst peers trying to reach a consensus (asynchronicity).

Driven to overcome the main issues identified for these frameworks, Esch et al. [4] proposed an approach to future electricity markets that allows for asynchronicity and has good scaling properties. Inspired by the stock exchange, the authors proposed a continuous double auction-based market and studied online matching mechanisms that allow for continuous trading of futures contracts of electricity exchange. The framework is a virtual trading platform where buyers and sellers can continuously trade different products (e.g., green electricity, local electricity) – thus allowing electricity product differentiation.

A challenge that arises with the framework proposed by Esch et al. [4] is the number of trading options and the amount of trading activity needed to perform well in this market. According to the authors, the characteristic of being a continuous market results in a level of trading activity that goes beyond a task that could be executed manually by a human. Therefore, an autonomous agent is needed to act in favor of market participants.

Considering this challenge, Esch [10] formulated the problem of automating the agents' strategic trading activity needed for operating in the continuous double auction-based electricity market. Following the framework from [11], the author evaluated the following mathematical methods used for optimizing the decision-making process in the context of the particular problem: direct lookahead policies, value function approximation, policy function approximations, and cost function approximation. Esch identified that value function based methods – specifically Reinforcement Learning (RL) – are suitable for the specific application.

Esch [10] applied three different RL algorithms – tabular Q-Learning, Deep Q-Networks (DQN), and Deep Deterministic Policy Gradient (DDPG) – to solve the decision-making problem of an agent interacting with an environment that represents a simplified version of the continuous double auction-based electricity market. The environment investigated by Esch was a model that reduced the market to a single trading session as an episode, state-space represented by the equilibrium price (modeled as a stochastic process), and action space defined by the order size. The results showed that all three RL algorithms were able to learn reasonably simple policies completely model-free, merely relying on interactions with the environment.

1.2 Contribution

This project extends the work of [10], investigating the application of RL algorithms to learn policies in an environment that represents a more realistic continuous double auction-based electricity market. Specifically, we modeled an environment to represent the electricity market deployed in the Svalin community as part of the Energy Collective [12] project.

Svalin is a co-housing community located in Denmark, where the electricity market proposed in [4] was implemented. The market participants include 20 houses with solar photovoltaic panels, one common house with a heat pump, and one controllable electric vehicle charging station.

The objectives of this project included the following:

- Model the sequential decision-making task of an agent trading in Svalin's community-based electricity market as a Markov Decision Process (MDP).

- Implement a custom environment using the OpenAI Gym toolkit that simulates the Svalin's community-based electricity market.

- Apply RL algorithms for learning the policy for an agent trading electricity in the environment implemented.

- Investigate the suitability of RL algorithms to different problem's complexity and assumptions.

- Explore different rewards structures in order to account for product preferences and agents' goals.

1.3 Thesis Structure

Chapter 2 elaborates on the framework of the P2P electricity market used as the environment that the RL agent interacts with. A detailed explanation of the mechanisms involved with the market operation is provided, so the reader has a clearer understanding of the environment's rules.

After explaining the P2P electricity market structure, Chapter 3 introduces the mathematical framework used to solve the decision-making problem of an agent interacting with the electricity market explained previously. Each building block of the framework is explained, and the algorithm used to solve the problem in this project is described: *Deep Q-Network* (DQN).

In Chapter 4, the decision-making problem of the agent is formulated, and the rules of the agent-environment interaction are provided. Moreover, the assumptions that were made to the electricity market model are provided.

The *DQN algorithm* algorithm is then applied to the decision-making problem, and three analyses are made: First, we investigate the impact of changing the discount factor applied to the rewards received by the RL agent – the results are provided in Chapter 5. Next, the discount factor is fixed to the most effective value, and the impact of increasing the dimension of the environment's state space on the problem's complexity is investigated. The results for the curse of dimensionality analysis are described in Chapter 6. Last, Chapter 7 exploits the fact that electricity is a differentiable product in the proposed market framework, and modified the reward function used in the agent-environment interaction to account for the agent's product preferences based on non-price attributes.

Finally, Chapter 8 discusses the challenges and learnings of the project, provides a conclusion to the project, and describes potential future topics to be investigated within the same scope.

2 Electricity Market Framework

This chapter elaborates on the framework of the electricity market studied in this project, providing a detailed description of the market mechanisms, so the reader has a better understanding of the rules for the environment that the agent interacts with.

We describe who the market participants are, the product traded, what is a trade, a trading session, an order, how the matching mechanism works, and how the market is settled. This project uses the same definitions of [10].

The market framework follows the structure proposed in [4] and, briefly describing it, consists of a double auction market with multiple products, where participants can continuously place orders that are immediately processed by a matching algorithm.

2.1 Market Participants

The proposed continuous double auction market relies on a centralized matching engine that creates bilateral contracts between peers who buy and sell a specific product. For electricity markets, market participants are either anyone who is interested in selling (in case of a producer) or buying (in case of a consumer) electrical energy. When a participant has the combined capability to produce and consume electricity, it is usually referred to as a prosumer.

In Figure 2.1, the market structure is illustrated. It consists of the market participants (or peers) interacting with the centralized matching engine.

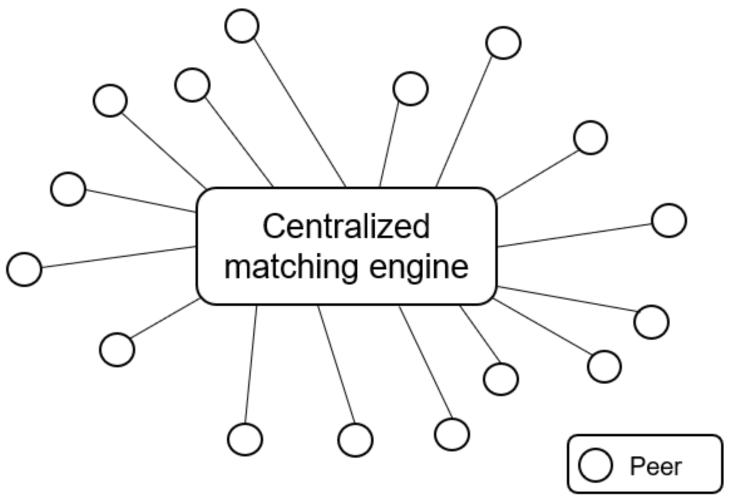

Figure 2.1: Illustration of the electricity market structure that consists of market participants (or peers) interacting with the centralized matching engine.

2.2 Product

In an electricity market, the product being traded between peers is the electrical energy to be produced or consumed in a specific time interval in the future, also called the time of delivery.

Considering that, for the same time of delivery, there could be multiple products with different characteristics available (e.g., generated from a local producer, generated from a renewable energy source, generated from fossil fuels), this mechanism allows the product to be differentiated according to non-price attributes. Product differentiation allows market participants to express and act according to their product preferences.

2.3 Trade

In the context of this work, a trade is an economic transaction, including a buyer and a seller, that consists of a legal commitment to consume or produce a certain quantity of electricity. The peers involved in the transaction sign a bilateral futures contract with a legal agreement to buy or sell a particular commodity asset at a predetermined price at a specified time in the future.

2.4 Trading Session

Each product being traded in the market has a particular trading session associated with it. The trading session has an opening and closing time (gate closure time) and refers to the time interval throughout which peers are allowed to place orders. When multiple products are being traded in the market, there are multiple trading sessions open at the same time.

Throughout the mutual trading sessions, the products' prices vary according to market participants' behavior, resulting in market dynamics with stochastic characteristics. The quantity available for each product is also dependant on the behavior of the participants.

Figure 2.2 illustrates trading sessions of 6 different products available in the electricity market. Note the independent price dynamics represented for the trading session of each product. Also, the products illustrated in Figure 2.2 are differentiated according to their time of delivery (for instance, Product 0/Product 2) and non-price attributes like green electricity or not (for instance, Product 0/Product 1).

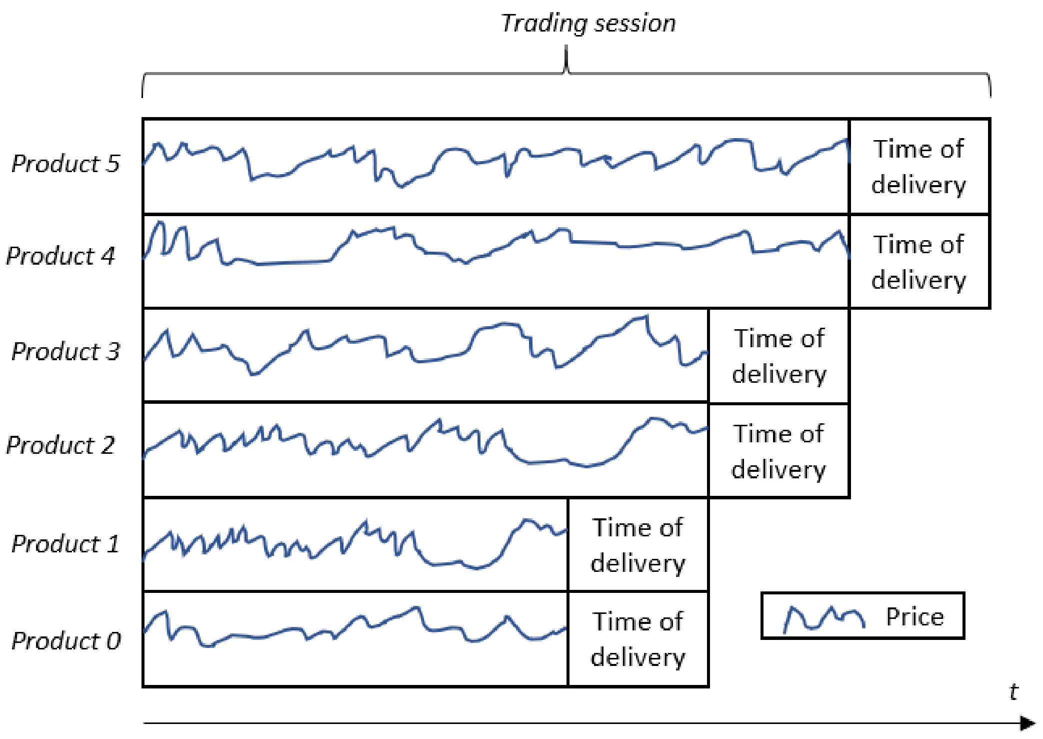

Figure 2.2: Illustration of trading sessions of 5 different products available in the electricity market.

2.5 Orders

An order is a mechanism used by market participants to buy/sell products, and it is defined by a quantity, a limit price, and a product identifier. The quantity refers to the amount of electricity the agent wants to buy/sell. The limit price is the price that the agent is willing to pay for the product or sell it. The product identifier signalizes to the market which product the agent wants to trade.

Orders meant to buy electricity are referred to as *bids* and orders meant to sell electricity as *asks*. When a new order arrives, it is processed against standing orders by the matching algorithm, and, if no trade is made, it moves to the group of standing orders. Generally, an incoming order that is not matched instantaneously is considered a *liquidity maker*, while an incoming order that is matched instantaneously is considered a *liquidity taker*.

2.6 Order Book

The collection of all standing orders in the trading session of a specific product is called an order book. The order book is publicly visible to other market participants, and it contains a list of available bids and asks. The available quantity at each price level can fully describe an order book. The difference between the highest bid price and the lowest ask price is referred to as the market spread and the middle point as the price equilibrium.

In Figure 2.3, we illustrate the order book of a specific product available in the market, containing the cumulative volume of electricity of standing bids and standing asks for each price level. In this example, the price equilibrium is equal to 0.22 [EUR/kWh].

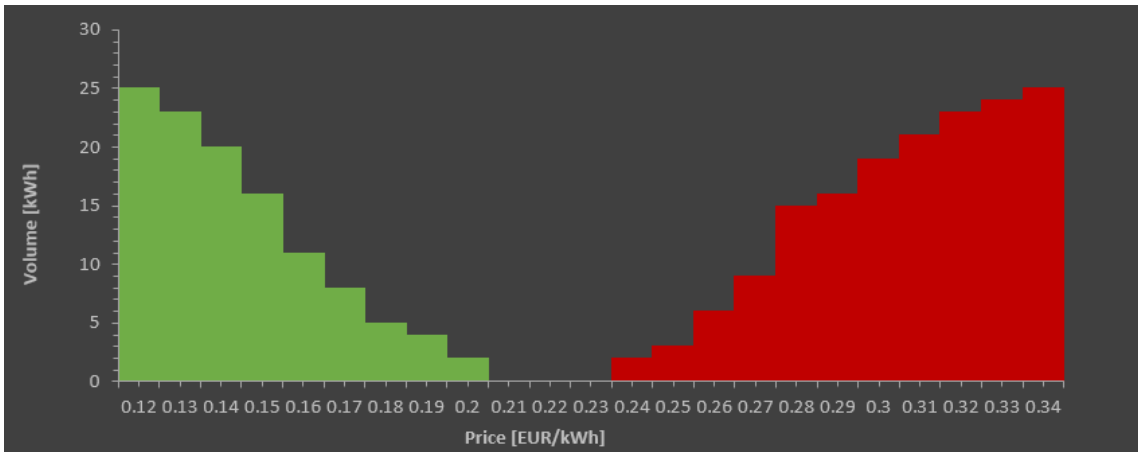

Figure 2.3: Illustration of the order book with standing bids and standing asks in the trading sessions of a specific product.

2.7 Matching Mechanism

The market uses a *Pro Rata matching algorithm* to determine trades between orders. A match occurs whenever the limit price of an order crosses the spread. For instance, when a bid price is higher than the lowest selling price in the order book, or when an ask price is lower than the highest buying price in the order book.

Once there is a match, the trade is executed at the best available price with the maximum tradable quantity possible. In case the incoming order takes all available quantity at the best price, a second match is executed at the next best price if the spread is still crossed. This is repeated until the entire quantity of incoming order is executed, or the spread is not crossed anymore. Moreover, in case of multiple orders standing at the best price when a match is executed, the quantities are distributed proportionally to the quantity of the original order, thus the name *Pro Rata matching algorithm*.

Besides, the transaction price is always determined by the standing order and accepted by the incoming order. Therefore, the standing order is considered the price maker and the incoming order, the price taker.

2.8 Balancing Settlement

The *market holdings* of each participant is defined as the total quantity of a specific product that the peer traded in the market. This quantity represents the amount of electricity that the participants commit to produce/consume during the physical exchange interval of that product.

When the physical exchange interval ends, any difference between the actual production/-consumption and the market holdings is then cleared in a balancing settlement. The settlement is done through a fixed feed-in tariff paid out for any unit of excess and a fixed price charged for any unit of deficit.

2.9 Autonomous Agents

The need for autonomous agents in markets that work with the mechanisms explained in this chapter was the motivation for the investigation done in [10]. The decision-making problem confronted by the agents involved in the market is a process with the potential to be automated.

Reinforcement Learning (RL) is a mathematical framework with algorithms that are used to learn policies for autonomous agents facing sequential decision-making problems. In this project, we investigated the use of one of these algorithms, called *Deep Q-Network* (DQN), to solve the formulated problem. Other studies [13, 14] also investigated the application of RL algorithms to automate the activity needed in electricity markets. The next chapter will elaborate on the RL framework and the *DQN algorithm*.

3 Reinforcement Learning

As explained previously, the decision-making problem of agents involved in electricity markets that follows the market mechanisms explained in Chapter 2 has the potential to be automated. Manually deciding when to place orders in the market and deciding from which product to buy electricity becomes a hassle depending on the number of transactions to be done. An autonomous agent that follows a determined policy represents the trading strategy of a peer placing buy and sell orders in the market to achieve a goal.

Reinforcement Learning is a mathematical framework used to solve sequential decision-making problems/stochastic optimization problems. The general functionality is based on the agent's experience and exploration of the environment. I.e., the problem is iteratively solved by an agent interacting with the environment (trying to solve the problem) and improving its policy based on rewards received as a response to its actions. The fact that the agent learns how to improve its policy for a specific environment, without any previous knowledge about the interaction, is the main characteristic of RL, and that is what makes this class of algorithms unique.

The objectives of this chapter are to describe the RL problem in a broad sense, to explain the Markov Decision Process framework that models the agent's decision-making problem, and to provide the explanation of the RL algorithm used to learn policies for the autonomous agents – *Deep Q-Network*.

3.1 Rewards

The most fundamental building block of RL is the *reward*. A *reward*, denoted as r_{t+1}, is a scalar feedback signal indicates how well the agent did in step t. RL is based on the *reward hypothesis*, which states that all agent's goals can be described by the maximization of the *expected future cumulative rewards*. The reason for not maximizing immediate rewards is that actions may have long term consequences, and sometimes it is better to sacrifice immediate rewards in order to achieve a higher long-term reward. According to [15], all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

The definition of goals in terms of rewards usually results in a *reward function*, and the process of constructing this function is referred to as *reward design*. Examples of rewards designed to achieve goals of specific applications that can be solved using the RL framework are given in Table 3.1.

Definition 1 (reward hypothesis): all goals can be described by the maximisation of expected cumulative reward.

Definition 2 (reward function): the function that computes r_{t+1} based on the action taken by the agent in time step t.

Definition 3 (reward design): the process of constructing the reward function.

Table 3.1: Examples of rewards designed to achieve goals of specific applications that can be solved using the RL framework.

Application	Positive reward	Negative reward
Control of a helicopter	Following desired trajectory	Crashing
Playing Backgammon	Winning the game	Losing the game
Control of robot walk	Forward motion	Falling over
Control of power station	Producing power	Exceeding safety thresholds

When dealing with stochastic environments, we cannot be sure of rewards happening far in the future, so sometimes we would like to value short term rewards more than long term rewards. It is possible to weight the importance of future rewards by including a discount factor to the calculation of the return, denoted by $\gamma \in [0, 1]$. The return of time step t, denoted as R_t, is the sum of discounted future rewards and can be computed according to Equation 3.1.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-1} r_{t+T} = \sum_{k=0}^{T-1} \gamma^k r_{t+k+1} \tag{3.1}$$

When $\gamma = 0$, future rewards are completely ignored, and the agent can be considered short-sighted. On the other hand, if $\gamma = 1$, future rewards are not discounted at all, and it is the usually right choice for agents interacting with deterministic environments.

3.2 Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework that uses the reward building block to model the agent-environment interaction. This framework is used to formulate the RL problem.

In a MDP, an agent observes an environment represented by a current state, denoted by $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states. Based on the observation, the agent takes an action, denoted by $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of actions available. The environment changes its state to s_{t+1}, and sends back to the agent a numerical reward, denoted by $r_{t+1} \subset \mathbb{R}$. The diagram of the agent-environment interaction modeled as an MDP is shown in Figure 3.1.

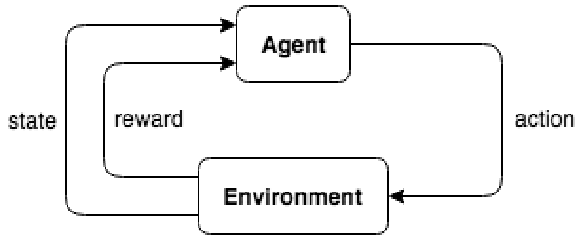

Figure 3.1: Diagram that represents the agent-environment interaction modelled as a Markov Decision Process.

A single transition of the MDP can be fully described by $\mathcal{T}_t = < s_t,\ a_t,\ s_{t+1},\ r_{t+1} >$. The time horizon of the agent-environment interaction, denoted by $t \in [0, T]$, can be characterized by the terminal time T. In case of a finite horizon ($T \neq \infty$), the MDP is referred to as an episodic problem. The episode is defined as the set of transitions $\mathcal{T}_t$, from $t = 0$ to $t = T$.

An essential property that allows us to use RL algorithms to find solutions for decision-making problems is called *Markov Property*, and it states that "the future is independent of the past" [15]. This property can be thought in terms of states, and, in this case, a state is called Markov if it contains all useful information from history. I.e., a state s_t is Markov if and only if $\mathbb{P}[s_{t+1} \mid s_t] = \mathbb{P}[s_{t+1} \mid s_1, s_2, ..., s_t]$.

Although many RL algorithms build on this assumption, it requires the agent to observe a full representation of the environment state, which is not always true. Real-world problems usually involve decision-making under uncertainty, and agents cannot fully observe the environment state. For these cases, the MDP is called Partially Observable Markov Decision Process (POMDP). The decision-making process analyzed in this project is a POMDPs, since we always expect randomness and exogenous information not available to the agent. From now on, the state s_t always refers to a possibly imperfect observation of the environment's state.

3.3 Policy

The next building block needed to formulate the RL problem is called policy. A policy is a mapping function from states to probabilities of selecting an action. It is denoted by π, where $\pi(a \mid s)$ is the probability that $a_t = a$ if $s_t = s$. At each time step, the agent uses the policy to select its action based on his observation of the environment [16].

The RL problem is solved by finding the optimal policy, denoted as π^*, that maximizes the expected future reward from all states. In practice, it is often difficult to find the optimal policy or even know if it has been found. However, in many cases, a reasonable estimate is an acceptable solution.

3.4 Value Functions

Almost all RL algorithms involve approximating *value functions*, which represent estimates of the expected return. Below, we describe two value functions that are building blocks of RL algorithms: *State value function* and *Action value function* (or *Q-value function*).

3.4.1 State value function

The state value function, denoted as $V_\pi(s)$, is the expected return when observing state s and following policy π after that. Formally, it can be defined as:

$$V_\pi(s) = \mathbb{E}_\pi[R_t \mid s_t = s] \tag{3.2}$$

State value functions have a particular recursive property. For any policy π and any state s, the following condition holds between $V_\pi(s)$ and the value of its possible successor states:

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[R_t \mid s_t = s] \\
&= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s]
\end{aligned}
\tag{3.3}
$$

Equation 3.3 is referred to as *Bellman equation* for state value function.

The *optimal state value function*, detoned as $V_*(s)$, is the maximum state value function over all policies:

$$V_*(s) = \max_\pi V_\pi(s) \tag{3.4}$$

3.4.2 Action value function (Q-value function)

The action value function (or *Q-Value function*), denoted as $Q_\pi(s,a)$, is the expected return of taking action a, when observing state s, under policy π. Formally, it can be defined as:

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_t \mid s_t = s, a_t = a] \tag{3.5}$$

The same recursive property can be applied to the action value function. For any policy π, state s, and action a, the following condition holds between $Q_\pi(s,a)$ and the value of its possible successor states:

$$Q_\pi(s,a) = \mathbb{E}_\pi[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a] \tag{3.6}$$

Equation 3.6 is referred to as *Bellman equation* for action value function.

The *optimal action value function*, denoted as $Q_*(s,a)$, is the maximum action value function over all policies:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a) \tag{3.7}$$

In the following chapters we always refer to the *action value function* as the *Q-value function*, and we omit the π subscript.

3.5 Deep Q-Network (DQN)

Now that the building blocks of RL were introduced, we can explain the algorithm investigated to solve the decision-making problem of this project. *Deep Q-Network*, or simply

DQN, is an algorithm that updates the parameters of an Artificial Neural Network (ANN) to improve an approximation of the *Q-Value function* of a MDP.

We can define the ANN that approximates the optimal action value function as $Q(s, a, \theta)$, where θ is the set of weights and biases to be learned. To learn θ, we need first to define a loss function L to be minimized. Based on the recursive property of the Bellman equation, it is possible to improve $Q(s, a)$ by bootstrapping, i.e., using an estimate obtained by taking a step in the environment.

Considering a transition $\langle s_t, a_t, s_{t+1}, r_{t+1} \rangle$, the loss L is defined according to Equation 3.8.

$$L = \Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big]^2 \tag{3.8}$$

Now, it is possible to update θ using *Stochastic Gradient Descent (SGD)* on L with respect to θ, with a step-size controlled by the learning rate α, according to Equation 3.9.

$$\theta \leftarrow \theta - \alpha \nabla_\theta L. \tag{3.9}$$

However, some tricks have to be used to overcome the challenges faced during the training loop of the algorithm. The issues are discussed in the following sections, and the tricks used to solve the problems are explained.

3.5.1 Exploration-Exploitation Dilemma

Exploitation is taking the right action to maximize the expected reward on the one-step, but exploration may produce a greater total reward in the long run [15]. I.e., when the agent is exploiting, he is taking an action to maximize the expected return. On the other hand, during exploration, the agent is taking a random action to verify if actions that were not taken before could produce higher returns. Since it is impossible to explore and exploit any single action selection concurrently, the conflict between exploration and exploitation is a dilemma confronted by the RL agents.

To overcome this challenge, a technique called *decaying epsilon-greedy* was introduced to the *DQN* algorithm. In this method, the agent selects random actions with a probability equal to ϵ during the training loop. The value of ϵ is linearly reduced from ϵ_{start} to ϵ_{final} during the first ϵ_{steps} steps of the training loop.

3.5.2 Training Instability

RL is known to be unstable or even to diverge when a nonlinear function approximator such as an ANN is used to represent the *Q-Value funtion* [17]. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between $Q(s_t, a_t)$ (the action-values) and $r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$ (the target values).

Following the work of [17], two key methods address these issues: *experience replay buffer* and *target network*.

Experience replay buffer - In this method, there is a buffer that saves past experiences (state, action, reward, next state). When calculating the loss to update θ, we evaluate a batch sampled from this buffer, with size denoted as B. The size of the buffer is denoted as E_{size}.

Target network - We keep a copy of our network and use it for obtaining the $Q(s_{t+1}, a)$ in Equation 3.8. The parameters of the target network, denoted as $\hat{Q}$, are synchronized every T_{sync} steps.

3.5.3 Final DQN Algorithm

Considering these tricks, the loss of Equation 3.8 was modified, and the final version is expressed in Equation 3.10.

$$L = \frac{1}{B} \sum_{k=1}^{B} \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a) - Q(s_t, a_t) \right]^2 \tag{3.10}$$

Algorithm 1 describes the pseudo-code of the complete *DQN algorithm* used to learn policies for the decision-making problem formulated in this project.

Algorithm 1 DQN Algorithm.

```
1: Initialize networks $Q(s,a)$ and $\hat{Q}(s,a)$ with random weights
2: Initialize empty replay buffer
3: Initialize $\epsilon$ equal to $\epsilon_{start}$
4: With probability $\epsilon$ choose a random action $a_t$, otherwise $a_t = argmax_a Q(s,a)$
5: Execute action $a_t$ in the environment, observe reward $r_{t+1}$ and next state $s_{t+1}$
6: Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in the replay buffer
7: Sample a random batch of experiences from the replay buffer
8: Calculate target: $r_{t+1}$ if the episode has ended at this step, otherwise: $r_{t+1} + \gamma max_a \hat{Q}(s_{t+1}, a)$
9: Calculate loss according to Equation 3.10
10: Update $Q(s,a)$ applying SGD, minimizing $L$ in respect to model parameters
11: Copy weights from $Q$ to $\hat{Q}$ every $T_{sync}$ steps
12: Repeat from step 4 until problem is solved
```

In this project, we implemented the *DQN algorithm* using PyTorch [18]. The full implementation of the algorithm is available in the GitHub repository listed in Appendix A.1.

3.5.4 Hyperparameters

The *DQN algorithm* has a set of hyperparameters that have to be configured correctly to make the training feasible. In this project, we have only investigated the impact of γ. The size of the experience replay buffer was defined according to the training speed (the simulations were getting too slow as we increased the buffer size). We have not focused on investigating the optimal selection of hyperparameters, so we used values applied by other studies [17].

The list of hyperparameters with their explanation is available in Table 3.2. The values used in the analysis are defined in each chapter.

Table 3.2: List of hyperparameters and their respective description.

Hyper-parameter	**Description**
γ	discount factor for future rewards
B	batch size that is obtained from the experience replay buffer
E_{size}	experience replay buffer size
T_{sync}	target synchronization rate
α	learning rate
ϵ_{start}	value of ϵ in the beginning of training
ϵ_{final}	value of ϵ in the end of training
ϵ_{steps}	steps to get the final value of ϵ

3.6 Summary

In this chapter, we described the RL framework used to solve sequential decision-making problems. We explained the building blocks of a MDP, the *DQN algorithm*, and listed the hyperparameters that need to be configured.

The next chapter will formulate the decision-making problem that was solved in this project using the *DQN algorithm* explained before. We provide all assumptions made to the model and explain the rules of the agent-environment interaction. In summary, the problem consists of an agent interacting with an environment that represents an electricity market with the same mechanisms explained in Chapter 2, buying electricity to meet its flexible demand.

4 Problem Formulation

As stated in Chapter 1, this project is an extension of the work done in [10]. Esch implemented an autonomous bidding agent for a peer-to-peer electricity market with the same framework explained in Chapter 2, however, a general agent and environment were considered, and several simplifications were made. For the sake of this project, we use the same market framework used by Esch, but we tailor the interactions between agent and environment to a specific case study.

We investigate automating the decision-making problem of one of the biggest consumers of the Svalin community trading electricity in the respective peer-to-peer electricity market. Thus, the agent-environment interaction and the decision-making problem considered in the project are not general anymore so that we can provide more realistic insights into the results.

In the following sections, we explain the assumptions made, formulate the decision-making problem and its goal, and give details about how the agent-environment interaction was modeled.

The implementation of the agent-environment interaction was done according to the *OpenAI Gym* framework [19], and it is available in the GitHub repository listed in Appendix A.1. The author of [20] also used the same framework to implement environments for similar problems.

4.1 Environment

The environment that the agent is interacting with is a representation of the electricity market of the Svalin community that follows the framework explained in Chapter 2. In this environment, there are multiple products available that can be traded by the agent in completely independent trading sessions. Throughout each trading session, the respective order book is dynamically changing according to the orders placed by market participants and trades that are occurring.

In our trading problem definition, we are only allowing the agent to buy electricity from the market – actions to place sell orders are not available. Therefore, we represented the market only by the lowest price of the selling orders available in the order book in each time step of the trading session duration. Since we are not considering the amount of electricity available at each price level of the order book, we assumed that there is always an *infinite* amount of electrical energy available at the lowest selling price – also known as *infinite market depth.*

We also assumed that the orders placed by the agent do not influence the state of the order book and price dynamics. Therefore, the trading session's price dynamics can be simply modeled as a price evolution. Following the definitions of [10], this assumption and the *infinite market-depth* show that we are working under a *Liquidity taker assumption.*

Assumption 1: *The environment that the agent is interacting with is a representation of the electricity market of the Svalin community, assuming an infinite market-depth. I.e., there is always an infinite amount of electricity available at the lowest selling price. We modeled the environment under the Liquidity taker assumption.*

4.2 Market Price Dynamics

The dynamics of the price during the trading session of each product is a result of the trading strategies adopted by the market participants, and it has to be modeled in our environment. Two approaches were investigated in this project: usage of real market data and mathematically modeling prices using a stochastic process.

4.2.1 Real Data

In the current P2P electricity market implemented in the Svalin community, the policy to define the price of orders placed by the agents is deterministic. The price is linearly decreasing throughout the trading sessions, while the quantities are selected according to the most recent forecast. However, there are three reasons for not selecting the data of the real electricity market of the Svalin community to model the price dynamics:

- Svalin's electricity market has low liquidity.
- There is not enough historical data available – considering the number of episodes that need to be simulated to learn a policy.
- The current agents have a simplified trading strategy (linear policy).

Therefore, the data of the real electricity market of the Svalin community was not used to model the price dynamics.

Another real market data investigated is the historical data of the EPEX SPOT Continuous Intraday market of Germany [21], which has mechanisms and framework with similarities to the Svalin's market [10]. Considering a dataset with historic anonymous orders placed to this market, we simulated a continuous double auction market following the framework explained in Chapter 2, where the orders of the dataset were placed throughout several trading sessions.

The market dynamics resultant from this simulation was a market with low liquidity, where the products are not available for the most part of the trading sessions. Therefore, we decided not to use the historical data of the EPEX SPOT Continuous Intraday market of Germany to model the price dynamics.

4.2.2 Stochastic Process

According to the literature [22, 23], one possible way to model the price dynamics of an intraday electricity market is to use a mean-reversion stochastic process called Ornstein-Uhlenbeck (OU) process. Considering that the concepts of the market in this project are similar to the intraday market, we decided to follow this modeling strategy.

The OU Process is mathematically described by Equation 4.1.

$$dx = \psi\ (\Upsilon - x)\ dt + \frac{\sigma}{\sqrt{c}}\ dW \tag{4.1}$$

In the context of the project, x is the price to be modeled, Υ is the long-run average followed by the price dynamics, ψ is the mean-reversion speed, σ is the parameter that represents the volatility of the price, and $c = \frac{1}{dt}$ is the Brownian scaling factor used to scale the Gaussian noise (dW) according to the time step size (dt) used in the simulation.

When applying the model to multiple products and, consequently, multiple prices, some requirements must be met. As a first requirement, we would like to keep some correlation between those prices, since they refer to electrical energy that is being traded for time intervals that are close to each other. However, if we were to model all prices with the same OU Process, i.e., with the same parameters, they would be the same product from a statistical point of view.

Therefore, we modeled the prices with separate OU processes, where each process has its own reversion mean. For the reference price ($p_{t,0}$), the OU process reverts to a mean that follows a *price trend*, denoted as p_{trend}. On the other hand, the OU processes of other prices ($p_{t,i}$) revert to the price of the previous product $p_{t,i-1}$. By doing this, we keep prices correlated but statistically different.

Assumption 2: *The dynamics of the prices in the environment were modeled using separate OU processes correlated with each other.*

We can mathematically formulate the price dynamics according to Equation 4.2.

$$p_{t,i} = \begin{cases} p_{t-1,i} + \psi\ (p_{trend} - p_{t-1,i})\ dt + \frac{\sigma}{\sqrt{c}}\ dW, & \text{if } i = 0 \\ p_{t-1,i} + \psi\ (p_{t,i-1} - p_{t-1,i})\ dt + \frac{\sigma}{\sqrt{c}}\ dW, & \text{if } i \neq 0 \end{cases} \tag{4.2}$$

4.3 Agent

Aiming to tailor the problem to the specific case study of the Svalin community, we defined a specific agent to the problem: we assumed that the agent interacting with the environment is one of the main consumers of the community. Considering how the problem was formulated, there is no difference if the agent represents the main electrical vehicle (EV) charging station or the heat pump.

Assumption 3: *The agent operating in the electricity market is one of the main consumers of the Svalin community: main EV charging station or heat pump.*

The choice of representing an agent that buys electricity for one of these two consumers was made due to the physical and operational characteristics of the devices that allow analyzing the decision-making problem from a specific point of view: electrical energy flexibility.

The electrical energy flexibility characteristic of the agent can be described by a certain amount of electrical energy that can be flexibly consumed throughout a specific time interval without impairing the device operation. We define the amount of electrical energy that can be flexibly consumed as the *flexible demand* (denoted as D_{flex}), and the time interval that it refers to as the *flexibility window* (denoted as W_{flex}).

For instance, if we consider an agent with the following energy flexibility characteristic:

- $D_{flex} = 10$ kWh
- $W_{flex} = [13{:}00, 14{:}00]$

There is no difference (in a device operation point of view) between consuming the flexible demand of 10 kWh from 13:00 to 13:30 or consuming it from 13:45 to 14:00. The decision to consume the electrical energy in the first or second half-hour is reflected in the agent's behavior in the electricity market, and it is the decision-making problem we want to approach in the project. For an EV charging station, this can be seen as the time window that a car is parked and available for charging a certain amount of electricity.

4.4 Agent-Environment Interaction

In our model of the environment, we assumed that the agent only places orders to buy electricity. The agent is never operating with a speculative behavior (buying when prices are low and selling when prices are high, or vice-versa) or selling electricity to regret from a purchase.

Assumption 4: *The agent operating in the electricity market only places buying orders.*

Also, we assumed that the agent only places orders that are matched instantaneously. It means that the agent is always a liquidity taker, accepting the price of the selling order with the lowest price that is available in the order book – market modeled under the *Liquidity taker assumption*.

Assumption 5: *The orders placed by the agent are always matched instantaneously with the selling order with the lowest price in the order book.*

Finally, we also fixed the amount of electricity in the orders placed by the agent. In this case, the agent does not have to decide the quantity to be bought; therefore, the only decision to be made is to buy the fixed amount or not. We denote the fixed quantity in the orders as β. The motivation behind this assumption was to decrease/limit the action/state space's dimensions, and, consequently, reduce the problem's complexity.

Assumption 6: *The orders placed by the agent have a fixed quantity to be bought, denoted as β.*

4.5 Decision-making Problem

As explained in Chapter 2, each product being traded in the proposed electricity market framework refers to electrical energy to be produced or consumed in a specific time interval in the future (also called the physical exchange). Considering the agent's flexibility capability explained in the last section, the decision-making problem to be solved can be formulated as to decide how much electricity to buy of each product available in the market, in order to meet the demand of the agent during its *flexibility window*.

Moreover, considering that the price of each product being traded is varying during the trading session according to the dynamics of the market, the agent also needs to decide when it is the best moment of the trading session to place the buying orders.

The goal of the agent in this context can be defined in many ways, such as the minimization of demand peaks or minimization of CO2 emissions. First, we investigated the decision-making process only with the goal of reducing costs. Later, we introduced product preferences to the problem and included it to the agents' goals.

4.6 Markov Decision Process

The agent-environment interaction explained before was mathematically modeled as a Markov Decision Process (MDP), which is suitable for solving the decision-making problem using the Reinforcement Learning algorithm explained in Chapter 3: *Deep Q-Network*.

In each time-step, the agent interacts with the environment with one action from the action space. In each transition of the environment, the agent has an opportunity to place an order to buy electricity in the market until the end of the trading session. Considering that the trading sessions of available products open at time-step $t = 0$ and end at time-step $t = T$, the MDP considered in the project is an episodic problem with *finite horizon*.

4.6.1 State Space

The state of the environment at time step t, denoted as s_t, is defined as:

$$s_t = [\ \hat{p}_{t,0} \quad \hat{p}_{t,1} \quad \dots \quad \hat{p}_{t,N-1} \quad \mu_t \quad \omega_t\ ]^\top \tag{4.3}$$

Where $\hat{p}_{t,i}$ are the normalized prices of products i at time step t, μ_t is the progress of the current episode, and ω_t is the ratio between current holdings and agent's flexible demand. Therefore, the dimension of the state space is equal to $N + 2$, and it was defined in the simulation model with continuous variable according to:

$$S_t = \left\{ \begin{array}{c} \hat{p}_{t,0} \in \mathbb{R} \quad , \quad 0 < \hat{p}_{t,0} \leq 1 \\ \hat{p}_{t,1} \in \mathbb{R} \quad , \quad 0 < \hat{p}_{t,1} \leq 1 \\ \dots \\ \hat{p}_{t,N-1} \in \mathbb{R} \quad , \quad 0 < \hat{p}_{t,N} \leq 1 \\ \omega_t \in \mathbb{R} \quad , \quad 0 \leq \omega_t \leq \omega_{max} \\ \mu_t \in \mathbb{R} \quad , \quad 0 \leq \mu_t \leq 1 \end{array} \right\}$$

The prices were normalized by the maximum price that peers are allowed to set in their orders. In the environment model, we assumed a maximum price of 1.0 [EUR/kWh] for the orders placed in the market.

4.6.2 Action Space

Considering the assumption that the agent can only place orders that are meant to buy electricity with fixed quantity, and that the orders are always matched instantaneously, the problem was simplified to an environment with discrete action space. It means that, for a market modeled with N available products, the agent can either buy the fixed quantity β from one of the available products or not place any order.

In this case, the action space is a discrete variable with dimension equal to $N+1$, where N is the number of available products. The mapping from the discrete variable to action taken by the agent is the following:

$$a_t = \begin{cases} 0, & \text{buy quantity } \beta \text{ of } \textit{Product 0} \text{ at price } p_{t,0} \\ 1, & \text{buy quantity } \beta \text{ of } \textit{Product 1} \text{ at price } p_{t,1} \\ 2, & \text{buy quantity } \beta \text{ of } \textit{Product 2} \text{ at price } p_{t,2} \\ ... & \\ N-1, & \text{buy quantity } \beta \text{ of } \textit{Product N-1} \text{ at price } p_{t,N-1} \\ N, & \text{do nothing} \end{cases}$$

It is worth remembering that the model of the environment considers an infinite depth, so the available quantity of product i at price p_i is always greater than quantity β. In addition, price p_i refers to the lowest selling price of the available orders in the order book for product i.

According to the actions taken by the agent throughout the trading session (a_t for $t \in [0, T]$), the holdings of each product (H_i) is determined according to Equation 4.4.

$$H_{i,T} = \sum_{t=0}^{T} b_{i,t} \cdot Q \tag{4.4}$$

Where,

$$b_{i,t} = \begin{cases} 1, & \text{if } a_t = i \\ 0, & \text{otherwise} \end{cases}$$

The agent's total holdings in each step of the trading session is calculated according to Equation 4.5.

$$H_T = \sum_{i=0}^{N} H_{i,T} \tag{4.5}$$

4.7 Rewards

Based on the *rewards hypothesis*, all agent's goals can be described by the maximization of the *cumulative rewards*. The *reward function* is used to give rewards to the actions taken by the agent and it was designed to reflect its goal.

Considering the goal minimizing costs, we designed a reward function that gives intermediate rewards r_{t+1} for every action a_t taken by the agent according to Equation 4.6, where $\hat{p}_{t,i}$ is the normalized price paid by the agent in the transaction to buy the fixed quantity of product i.

$$r_{t+1} = \frac{1}{e^{\hat{p}_{t,i}}} - \frac{\hat{p}_{t,i}}{e} \tag{4.6}$$

From 4.6, we can see that the rewards which are given to the agent increase when the prices decrease. The RL are maximizing the expected future rewards. Consequently, the electrical energy costs are minimized (original decision-making problem goal).

The choice of the reward function aims to control the range of rewards given and provides an exponential characteristic. It is possible to visualize the range of rewards given in the agent-environment interaction, if we plot the function for $\hat{p}_{t,i} \in [0, 1]$.

In Figure 4.1, we see that the intermediate rewards range from $r_{t+1} = 1$ (when $\hat{p}_{t,i} = 0$) to $r_{t+1} = 0$ (when $\hat{p}_{t,i} = 1$).

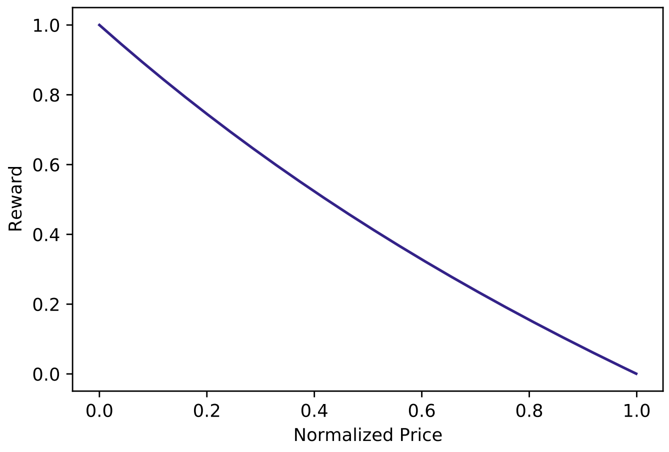

Figure 4.1: Reward function for $\hat{p}_{t,i} \in [0, 1]$.

4.7.1 Rewards - Exceptional cases

Exceptional cases were added to the reward function, according to specific conditions. Therefore, the scalar feedback is adjusted depending on the outcome of action taken. We added these exceptional cases, so the decision-making problem's goal is more clearly reflected in the rewards given. The propositions that control the exceptional cases of the rewards given are described as follows.

Proposition 1 *The agent did not place a buy order. In this case, $a_t = N$.*

Proposition 2 *The agent placed a buy order and the total holdings would exceed its demand ($\omega_t > 1$).*

Proposition 3 *The agent placed a buy order, however, the price of the product selected by the agent is not the lowest price available in the market. In this case, $a_t \neq \min_i p_{t,i}$.*

Then, we defined the reward function, including the exceptional cases based on described propositions, according to Equation 4.7.

$$r_{t+1} = \begin{cases} 0, & \textbf{if:} \\ & \quad \text{Proposition 1 is true OR} \\ & \quad \text{Proposition 2 is false AND Proposition 3 is true} \\ -0.1, & \textbf{if:} \\ & \quad \text{Proposition 1 is false AND Proposition 2 is true} \\ \frac{1}{e^{\hat{p}_{t,i}}} - \frac{\hat{p}_{t,i}}{e}, & \textbf{if:} \\ & \quad \text{Proposition 1 is false AND Proposition 2 is false} \end{cases} \tag{4.7}$$

4.8 Basis for Policy Performance Comparison

Throughout the report, the performance of policies learned by RL agents are compared to the performance of two naive policies (*Random Naive Policy* and *Minimum Naive Policy*), and to the *Optimal Behaviour with Full Knowledge.* They are described as follows.

4.8.1 Random Naive Policy

The *Random Naive Policy* consists of an agent that, in every time step, takes a random action (according to a discrete uniform distribution) until the demand boundary is exceeded. I.e., it keeps placing buy orders in the trading session of random products, until the sum of quantity holdings exceeds the main boundary.

The mathematical definition of the *Random Naive Policy* is expressed in Equation 4.8.

$$\pi(s_t) : a_t = \begin{cases} \mathit{unif}\{0, N\}, & \text{if } \omega_t < 1 \\ N, & \text{otherwise} \end{cases} \tag{4.8}$$

4.8.2 Minimum Naive Policy

The *Minimum Naive Policy* consists of an agent that, in every time step, takes a **specific action** until the main boundary is exceeded. It differs from the previous policy because it only places orders in the trading session of the product with the lowest price. I.e., the agent keeps placing orders in the trading session of the product with the lowest price until the sum of quantity holdings exceeds the main boundary.

The mathematical definition of the *Minimum Naive Policy* is expressed in Equation 4.9.

$$\pi\left(s_t\right) : a_t = \begin{cases} \min\limits_i \hat{p}_{t,i}, & \text{if } \omega_t < 1 \\ N, & \text{otherwise} \end{cases} \tag{4.9}$$

Considering the way the rewards were designed, it is possible to conclude that the performance of the *Minimum Naive Policy* is always going to be higher than the performance of the *Random Naive Policy*, because the positive rewards are only given to the agent when placing buy orders to the trading session of the product with the lowest price (always the case for the *Minimum Naive Policy*).

4.8.3 Optimal Behaviour with Full Knowledge

The *Optimal Behaviour with Full Knowledge* refers to the maximum return that an agent could have received for a specific episode. This value is only calculated after all transitions of the episode occurred (i.e., when trading sessions ended) and the evolution of prices over time for all products is available.

Considering an agent that needs to place K buy orders of fixed quantity β in order to meet its flexible demand D_{flex}, we can define a vector $\boldsymbol{\hat{p}^{min}}$ which contains the K-lowest normalized prices that occurred on the episode, and calculate the maximum return R_{max} of the *Optimal Behaviour with Full Knowledge* according to Equation 4.10.

$$\boldsymbol{\hat{p}^{min}} = \begin{bmatrix} \hat{p}^{min}_0 & \hat{p}^{min}_1 & \dots & \hat{p}^{min}_K \end{bmatrix}$$

$$R_{max} = \sum_{k=0}^{K} \frac{1}{e^{\hat{p}^{min}_k}} - \frac{\hat{p}^{min}_k}{e} \tag{4.10}$$

In all analyses done in the next chapters, the term *Performance* refers to the ratio between the return obtained in an episode and the return R_{max}.

4.9 Summary of Analyses

In this Chapter, we formulated the decision-making problem of the agent, provided all the rules of the agent-environment interaction, and defined the assumptions that were made to the electricity market model.

The decision making problem was solved using the *DQN algorithm* explained in Chapter 3 and three different analyses were made:

- Chapter 5 investigates the impact of changing the discount factor applied to the rewards received by the RL agent.

- Chapter 6 analyzes the impact of increasing the dimensions of the environment's state/action spaces on the problem's complexity.

- Chapter 7 exploits the fact that electricity is a differentiable product in the proposed market framework, and modifies the reward function used in the agent-environment interaction to account for the agent's product preferences

5 Discount Factor Analysis

The sequential decision-making problem formulated in the last chapter was solved with the *DQN algorithm* explained in Chapter 3. The agents learned policies that are used to select an action based on the state observed during the agent-environment interaction.

As a first investigation of the results obtained with the algorithm, we propose analyzing the impact of the discount factor, denoted as γ, on the policies learned by the RL agents.

In the following sections, we describe the assumptions made to the decision-making problem and model of the environment, show the training and performance results for the agents, analyze how each agent explored the environment, visualize the policies learned, and discuss the results obtained.

5.1 Assumptions

To investigate the impact of the discount factor on the policies learned by the reinforcement learning agents, we decided to train individual agents (with the *DQN algorithm*) considering three different values of γ:

- $\gamma = [0, 0.9, 1]$

It was also assumed an environment that models the electricity market with 36 time steps ($T = 36$) and 3 products available ($N = 3$). Considering it, for each episode, the agents have to make 36 sequential decisions that refer to buying electricity from one of the 3 products (or not) to achieve their goal.

According to the problem formulation of Chapter 4 and the assumptions described above, the state space has 5 dimensions ($N + 2$), and the state at time step t is defined as follows:

$$s_t = \begin{bmatrix} \hat{p}_{t,0} & \hat{p}_{t,1} & \hat{p}_{t,2} & \mu_t & \omega_t \end{bmatrix}^\top$$

The action space of the MDP has 4 dimensions ($N + 1$), and the mapping from discrete variables to actions is represented as follows:

$$a_t = \begin{cases} 0, & \text{buy quantity } \beta \text{ of } \textit{Product 0} \text{ at price } p_{0,t} \\ 1, & \text{buy quantity } \beta \text{ of } \textit{Product 1} \text{ at price } p_{1,t} \\ 2, & \text{buy quantity } \beta \text{ of } \textit{Product 2} \text{ at price } p_{2,t} \\ 3, & \text{do nothing} \end{cases}$$

Concerning the price dynamics, it was assumed that the *price trend* (p_{trend}), which drives other products' prices dynamics, is linearly dropping over the trading session. Therefore, the product prices are expected to follow a dropping trend throughout the trading session. Equation 5.1 defines the *price trend* over the trading session.

$$p_{trend,t} = \begin{cases} 0.5, & \text{for } t = 0 \\ p_{trend,t-1} - 0.012, & \text{for } t = 1, 2, ..., 36 \end{cases} \tag{5.1}$$

Also, we defined the fixed quantity β so that the agent has to place 18 actions representing buying electricity from one of the 3 products (actions 0, 1, or 2) to meet the device's flexible demand D_{flex}. Therefore:

$$\beta = \frac{D_{flex}}{18}$$

The motivation for these two assumptions is that, intuitively, we expect that the optimal policy reflects the behavior of buying from the cheapest product in the last 18 steps of the episode. Therefore, we can verify if the policy learned by the agent is sacrificing immediate rewards to get a higher return in the episode or not.

The parameters of the *OU Processes* used to model the price dynamics are shown in Table 5.1. Figure 5.1 (left) shows an example of the prices' dynamics in the environment for a single episode with 36 time-steps, where it is possible to verify the dropping price trend. Figure 5.1 (right) shows the distribution of products' prices over 1000 episodes.

Table 5.1: Parameters used for the OU processes that model the price dynamics.

OU parameter	**Value**
ψ	0.01
dt	8
σ	1
c	0.125

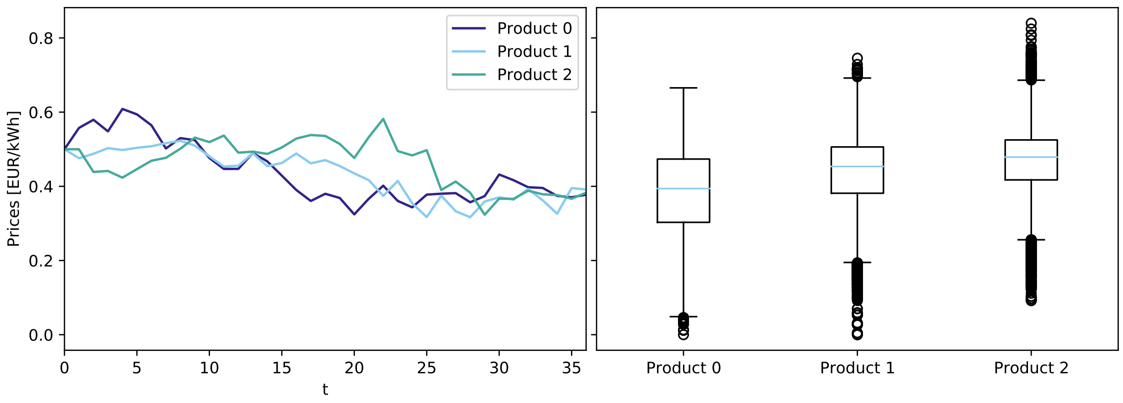

Figure 5.1: An example of the prices' dynamics in the environment for a single episode (left). The distribution of products' prices for 1000 episodes (right).

For the *Q-value function* approximation, we implemented a fully-connected ANN that contains an input layer with 5 nodes, 2 hidden layers with 64 nodes each, and an output layer with 3 nodes. The ANN receives the state observed by the agent as input, and it outputs the estimated expected future return of taking each action.

5.2 Training Results

The agents were individually trained for 10^6 steps, considering the same hyperparameters (besides γ), and the *performance* was monitored throughout the training. The performance here refers to the ratio between the episode's return and maximum return that could be obtained. The hyperparameters are specified in Table 5.2, and Figure 5.2 shows the moving average (100 episodes) of the agents' performance throughout the training loop.

Table 5.2: Hyperparameters used in the training.

Hyper-parameter	**Value**
γ	$[0, 0.9, 1.0]$
B	32
E_{size}	500×10^3
T_{sync}	10×10^3
α	10^{-4}
ϵ_{start}	1.0
ϵ_{final}	0.02
ϵ_{steps}	10^5

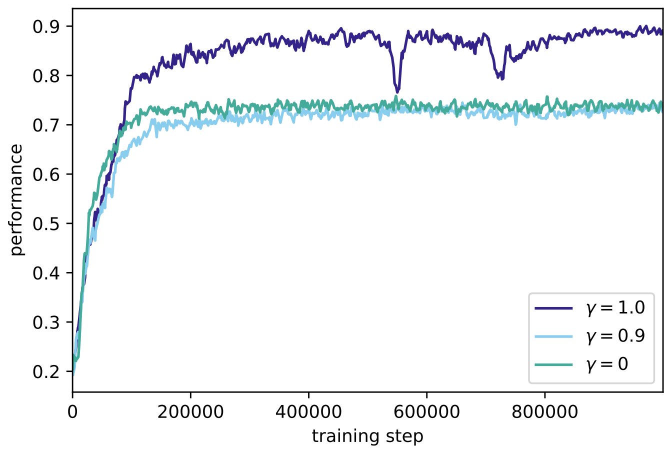

Figure 5.2: Agents' performance (moving average over 100 episodes) throughout the training loop.

From Figure 5.2, it is possible to verify that agents smoothly converged to a policy in a similar number of time steps (around 2×10^5). However, for agents considering $\gamma = [0, 0.9]$, the algorithm converged to a policy with lower performance compared to the agent with $\gamma = 1$.

Considering that during the training loop the agent is still taking random actions (with a probability of 2% after 10^5 steps), the best way to compare performances is testing the policies with agents interacting with the environment outside of the training loop – this was done in the next section.

5.3 Performance Results

The policies learned for the agents were tested in the environment for 1000 episodes, and its performances were compared to the baseline policies explained in Chapter 4 (*random* and *minprice*). The agent-environment interactions were implemented with 1000 fixed random seeds, so all agents/policies experienced the same price dynamics in the environment (Figure 5.1 - right). Figure 5.3 shows the comparison of the agents' performance over 1000 episodes.

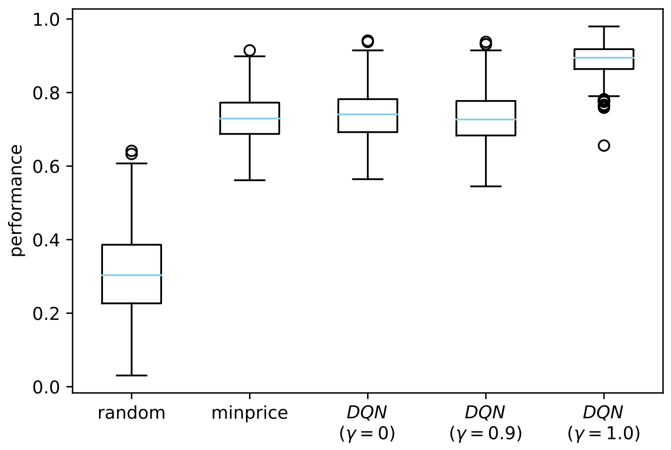

Figure 5.3: Distribution of agents' performance over 1000 episodes. The Figure compares the performances of the 2 baselines (*random* and *minprice*) with the DQN algorithm considering $\gamma = [0, 0.9, 1]$.

From Figure 5.3, it is possible to verify that the performance of agents considering discount factors $\gamma = [0, 0.9]$ were similar to the performance of the *minprice baseline policy.* An intuition that answers why the performances are similar comes from the behavior of the *minprice baseline policy* and the impact of γ in the returns:

As explained in Chapter 3, the problem was formulated such as the *Return* in time step t, denoted as R_t, is calculated according to Equation 5.2.

$$R_t = \sum_{k=0}^{36-t} \gamma^k r_{t+k} \tag{5.2}$$

From Equation 5.2, we can verify that $\gamma = 0 \rightarrow R_t = r_t$, i.e., the return of time step t completely ignores that future rewards and it is equal to the reward received in time step t. When we increase the value of γ, the agent starts considering future rewards to compute the return, however, for $\gamma = 0.9$, the future reward is discounted in around 85% in only 18 steps.

The policy that results from agents with $\gamma = [0, 0.9]$ reflects agents that optimize their policies to maximize immediate rewards over future rewards. I.e., they do not consider the fact that if they do not buy electricity in the first steps of the trading sessions, they can get lower prices in the future (due to our dropping price assumption). Consequently, the expected policy that the agents learn for these values of γ is a market participant who buys electricity from the cheapest product until it meets the demand. This is precisely the behavior of the *minprice baseline policy*, so, intuitively, the performances of these policies are expected to be similar.

5.4 Environment Exploration

Another way to verify the agents' behavior of prioritizing immediate rewards over future rewards is by analyzing how they explored the environment's state space throughout the training.

Because of the assumptions made to the environment, we know that the lowest prices occur in the second half of the trading sessions; therefore, if the agent reaches this point with $\mu_t < 0$ (demand not met yet), he could get the highest rewards of the episode. However, when the agent is optimizing for immediate rewards, he will not wait for the rewards in the second half of the trading session, reaching that point with $\mu_t \approx 1$. We can verify this by looking at the states visited by the agents during the training loop.

Figure 5.4, Figure 5.5, and Figure 5.6 represent how the agent explored the environment's state space on two dimensions (ω_t and μ_t) for $\gamma = [0, 0.9, 1]$, respectively. The exploration plots are represented by histograms that count how many times the agent visited a state with specific values of ω_t and μ_t.

From Figure 5.4, Figure 5.5, we can see that the agents considering $\gamma = [0, 0.9]$ were not experiencing states with $\mu_t > 0.5$ **and** $\omega_t < 1.0$. Therefore, they were never visiting the states that allow them to receive higher rewards – remember that even if prices are low, the agent gets negative rewards when buying electricity with $\omega_t >= 1$ (i.e., when the demand was already met). This confirms the fact that, if we consider $\gamma = [0, 0.9]$, the policies learned optimize for immediate rewards and never wait to buy electricity in the future.

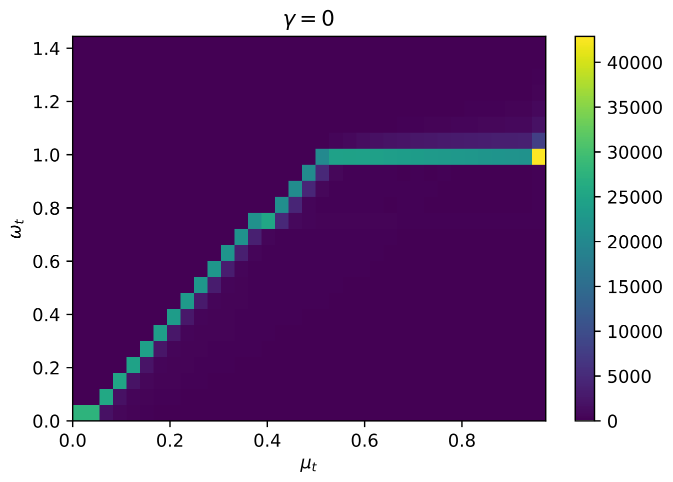

Figure 5.4: Visualization of how the agent explored the state space, considering the DQN algorithm with $\gamma = 0$.

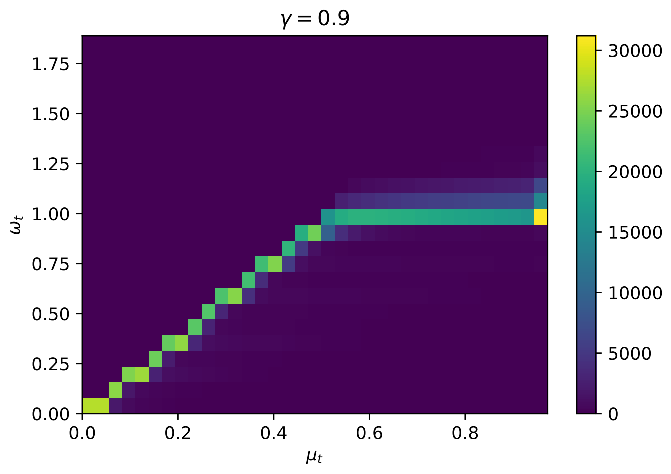

Figure 5.5: Visualization of how the agent explored the state space, considering the DQN algorithm with $\gamma = 0.9$.

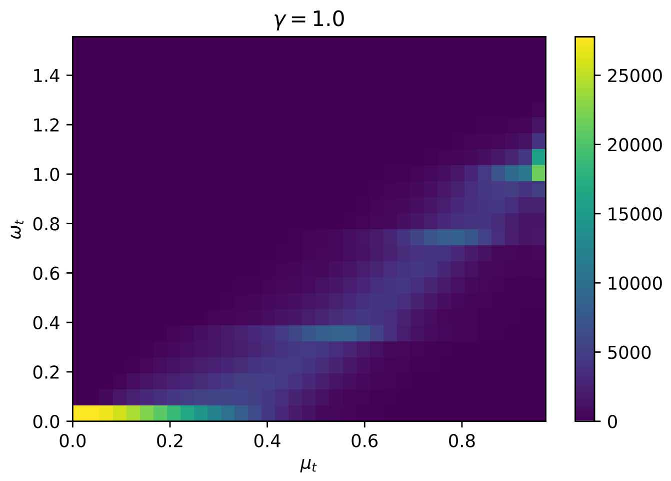

Figure 5.6: Visualization of how the agent explored the state space, considering the DQN algorithm with $\gamma = 1$.

On the other hand, from the exploration representation of Figure 5.6, we can see that the agent that was trained with $\gamma = 1$ is indeed experiencing the states that allow to get higher rewards ($\mu_t > 0.5$ *AND* $\omega_t < 1.0$). Intuitively, this represents a policy with the behavior of sacrificing immediate rewards to get higher future rewards.

5.5 Policies Visualization

The policies learned by the agents take as input the current state observed by the agent (5 dimensions) and output the action that maximizes the expected future return (1 dimension). In order to visualize the policies learned by the agents, we ran 1000 episodes, logged the state and action taken by the agent, and reduced the number of dimensions of the input state by analyzing the number of times each action was taken for different values of μ_t. We analyzed three different cases:

- **Case** 1: When $\hat{p}_{t,0} < \hat{p}_{t,1}$ and $\hat{p}_{t,0} < \hat{p}_{t,2}$ (cheapest product: $p_{t,0}$).

- **Case** 2: When $\hat{p}_{t,1} < \hat{p}_{t,0}$ and $\hat{p}_1 < \hat{p}_{t,2}$ (cheapest product: $p_{t,1}$).

- **Case** 3: When $\hat{p}_{t,2} < \hat{p}_{t,0}$ and $\hat{p}_{t,2} < \hat{p}_{t,1}$ (cheapest product: $p_{t,2}$).

Figure 5.7, Figure 5.8, and Figure 5.9 shows the number of times each action was taken for different values of μ_t, in the three cases analyzed, considering $\gamma = 0$, $\gamma = 0.9$, and $\gamma = 1$, respectively.

In Figure 5.7, it is possible to verify that the policy is clearly split in 2 parts: action taken for $\mu_t < 0.5$ and action taken for $\mu_t > 0.5$. As discussed before, the reinforcement learning agent with $\gamma = 0$ optimizes the policy to maximize immediate rewards, and never sacrifices immediate rewards to get higher rewards in the future. The result is a policy that always places orders in the cheapest product until the demand is met, and after that, places no order.

This behavior is exactly what the policy of Figure 5.7 represents. We can see that for $\mu_t > 0.5$, the demand is already met, and the agent takes action number 3 (do not place any order) no matter which product is the cheapest. For $\mu_t < 0.5$, the agent takes the action that represents buying electricity from the cheapest product.

When we increased γ to 0.9, we see from Figure 5.8 that the policy is not clearly split into 2 parts as in the previous case, but there are some cases that the agent places orders when $\mu_t > 0.5$. What happens is that the future rewards that might occur at the end of the trading session are not propagated to the very beginning.

Figure 5.9, on the other hand, shows that when we train the reinforcement learning agent with $\gamma = 1$, the policy learned has the characteristic of sacrificing immediate rewards in the beginning of the trading session to maximize the expected future return. As we can see, action number 0 is the one taken more often. This is due to the fact that the price dynamics used in the model resulted in a price distribution that product 0 often has the lowest prices (see Figure 5.1).

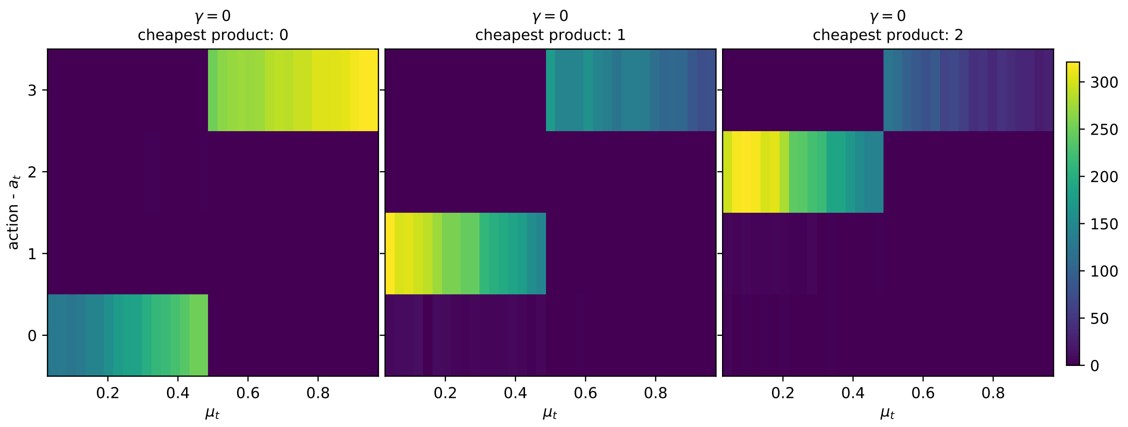

Figure 5.7: Number of times each action was taken for different values of μ_t, in the three cases analyzed: cheapest product 0, cheapest product 1, cheapest product 2, and considering $\gamma = 0$.

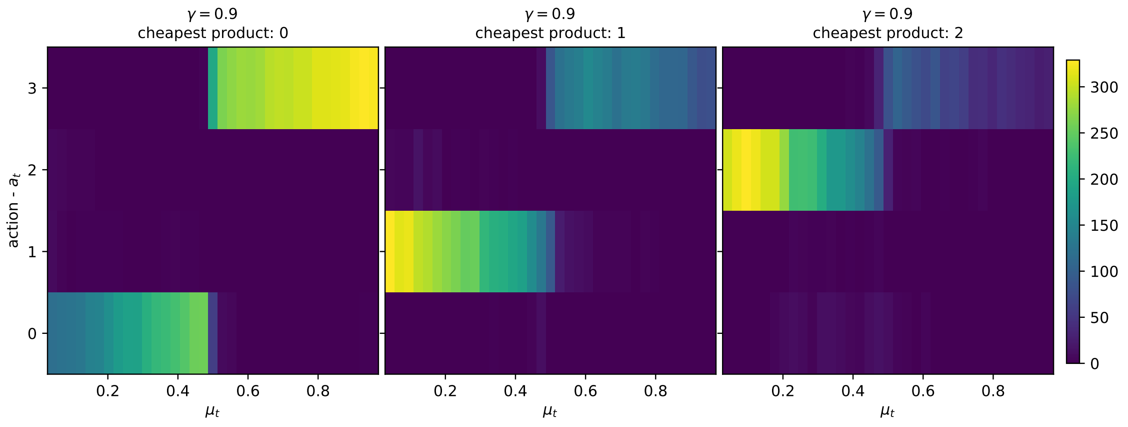

Figure 5.8: Number of times each action was taken for different values of μ_t, in the three cases analyzed: cheapest product 0, cheapest product 1, cheapest product 2, and considering $\gamma = 0.9$.

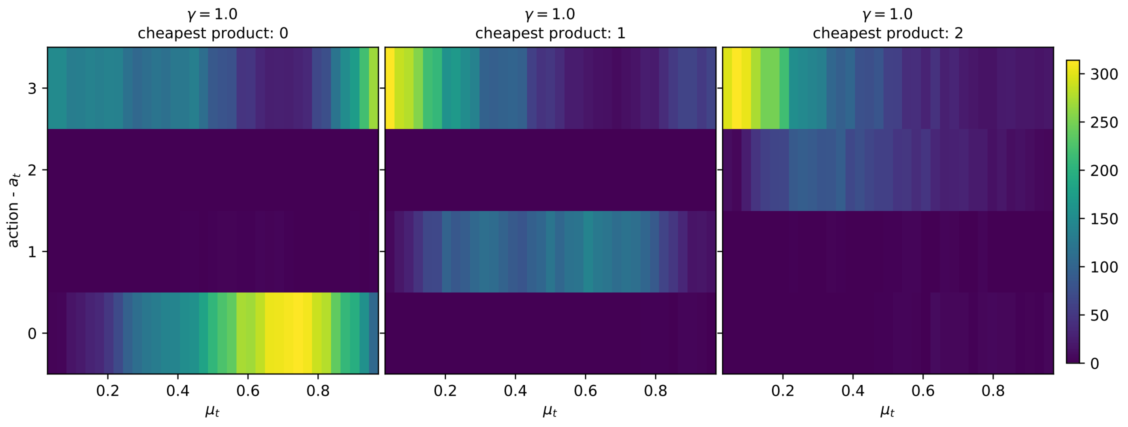

Figure 5.9: Number of times each action was taken for different values of μ_t, in the three cases analyzed: cheapest product 0, cheapest product 1, cheapest product 2, and considering $\gamma = 1$.

5.6 Discussion

From the analysis of performances and environment exploration, we could verify that selecting the correct discount factor γ has a significant impact on the policy learned by the *DQN algorithm*. For $\gamma = 0$ and $\gamma = 0.9$, the agents could not learn that it was better to buy electricity at the end of the trading sessions (optimal policy behavior). When increasing the discount factor to $\gamma = 1$, the agent learned that it was worth it to sacrifice immediate rewards for higher future rewards. The impact of γ on the final performances of the agents was significant. Considering it, for the decision-making problem we formulated, it does not make sense to use a discount factor different than 1 to the calculation of the returns.

In the next chapter, we analyze how the problem's complexity grows when increasing the number of time steps in the trading session or increasing the number of products available. From now on, we decided to use $\gamma = 1$ in all analyses.

6 Curse of Dimensionality

After investigating the impact of the discount factor γ on the policies learned, the suitability of the DQN algorithm was tested for an environment modeled with more dimensions, resulting in higher problem complexity. We investigated increasing the following environment's dimensions:

- T - the number of time steps that the agent is allowed to place orders during the trading sessions.

- N - the number of products available in the market.

As a consequence of increasing the number of time steps in the trading session (T), the state space of the environment becomes larger. On the other hand, increasing the number of products available in the market (N) results is an action-space and a state-space with more dimensions – one extra dimension for each product added.

In the following sections, we describe the assumptions made for the environment, investigate the training and performance results for each scenario, visualize the policies learned by the agents, evaluate the *Q-value function* approximation, and, finally, discuss the outcomes obtained.

6.1 Assumptions

We trained agents for four different cases, combining two values of N and T with each other, as follows:

- **Case** 1: 36 time steps/3 products ($T = 36$ and $N = 3$).

- **Case** 2: 36 time steps/8 products ($T = 36$ and $N = 8$).

- **Case** 3: 288 time steps/3 products ($T = 288$ and $N = 3$).

- **Case** 4: 288 time steps/8 products ($T = 288$ and $N = 8$).

The assumptions made in Chapter 5 related to the *price trend* dropping over trading sessions, and the definition of the quantity β in order to be able to meet the demand in half of the trading session are still valid for the environment used in this chapter. However, the price trend (p_{trend}) and quantity β are now defined accordingly to the number of time steps (T) we considered in each scenario.

Equation 6.1 and Equation 6.2 show the mathematical formulation of p_{trend} and β. Note that for an environment considering 288 time steps, the variation of p_{trend} in each simulation step is smaller, and the quantity β fixed in the orders placed is also smaller (when the values are compared to an environment with 36 time steps).

$$p_{trend,t} = \begin{cases} 0.5, & \text{for } t = 0 \\ p_{trend,t-1} - 0.012 \times \frac{36}{T}, & \text{for } t = 1,\ 2,\ ...,\ \text{T} \end{cases} \tag{6.1}$$

$$\beta = \frac{D_{flex}}{0.5 \times T} \tag{6.2}$$

6.2 Training Results

The agents were trained for 5×10^6 steps, considering the same hyperparameters, and the performance was monitored throughout the training. Similarly to Chapter 5 analysis, the performance is calculated by the ratio between the sum of rewards received in the episode and the maximum reward that could be received.

The hyperparameters are specified in Table 6.1, and Figure 6.1 shows the moving average (100 episodes) of the agents' performance throughout the training loop. Note that, based on the analysis made in Chapter 5, it was decided to continue the simulations only with $\gamma = 1$ (i.e., not discounting future rewards at all).

Table 6.1: Hyper-parameters used in the training.

Hyper-parameter	**Value**
γ	1.0
B	32
E_{size}	500000
T_{sync}	10000
α	0.0001
ϵ_{start}	1.0
ϵ_{final}	0.02
ϵ_{steps}	100000

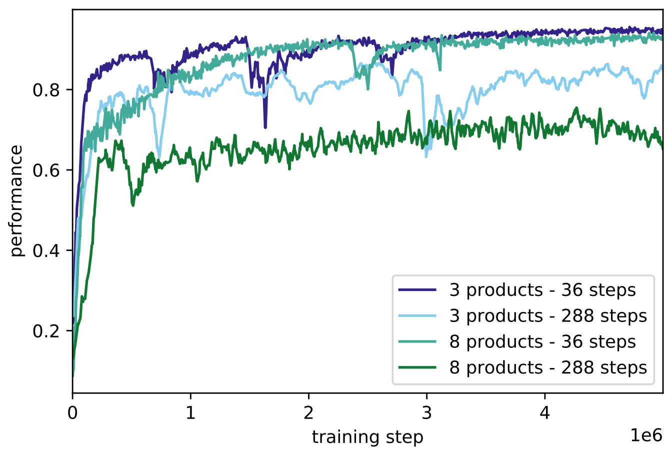

Figure 6.1: Agent's performance along the training loop (rolling average over 100 episodes) for the 4 cases analyzed.

The evolution of performances throughout the training, which is observed in Figure 6.1, shows that the agents trained for problems with 36 steps (both 3 and 8 products) converged to the same value (around 0.9). On the other hand, the scenarios considering 288 steps converged to different values: 0.8 for 3 products and 0.7 for 8 products. Therefore, increasing the number of steps from 36 to 288 had a higher impact on the problem complexity than increasing the number of products from 3 to 8.

6.3 Performance Results

The performance of the 4 policies learned were tested outside of the training loop (without any random action being taken) over 1000 episodes. Similarly to Chapter 5 analysis, the agent-environment interactions were implemented with 1000 fixed random seeds, so all agents/policies experienced the same price dynamics in the environment.

Figure 6.2 shows the comparison of the agents' performance over 1000 episodes for the 4 scenarios. The plots on the left side refer to environments considering 36 steps, while the plots in the right, 288 steps. Also, the plots in the upper part of the figure refer to environments considering 3 products, while the plots in the lower part, 8 products.

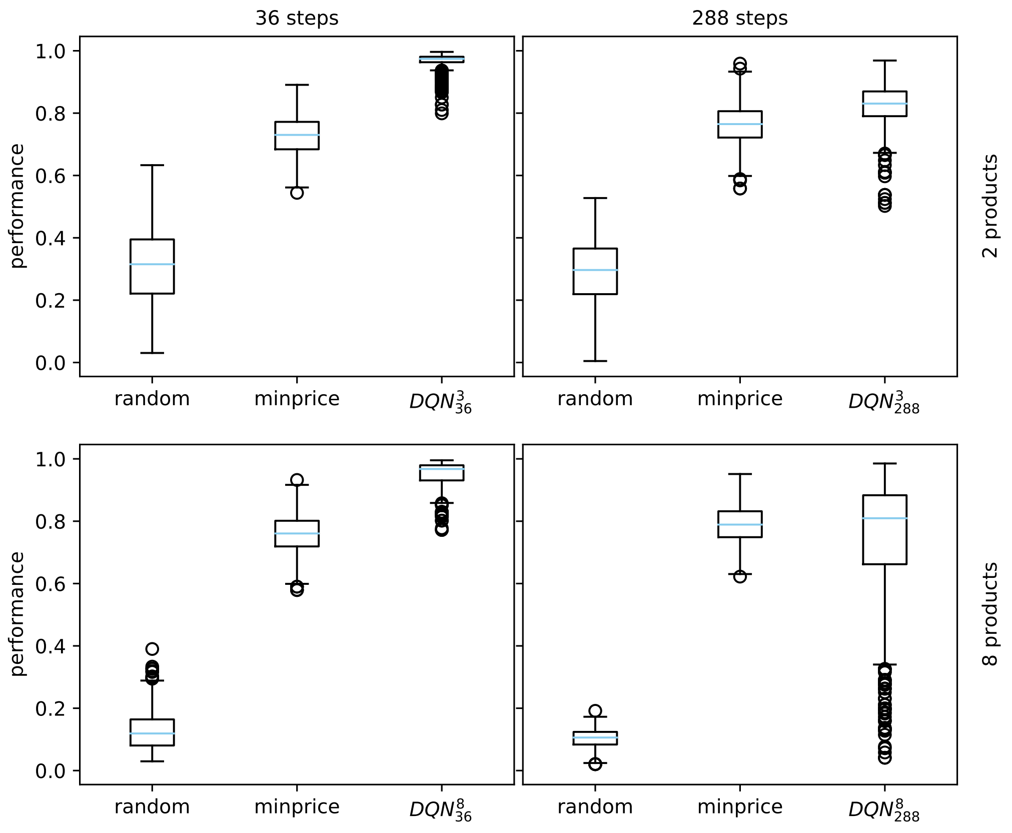

Figure 6.2: Distribution of agents' performance over 1000 episodes, for the 4 scenarios.

As expected, we can see in Figure 6.2 that the performance of the *Random Baseline* decreases significantly by increasing the number of products from 3 to 8. Intuitively, when we increased the number of products available in the market, the probability of the random action to be the cheapest product is reduced from $\frac{1}{4}$ to $\frac{1}{9}$.

Besides, we can also verify in Figure 6.2 that the performance of the *minprice baseline* does not have a significant difference when increasing the complexity of the problem. This happens because the policy does not have any randomness associated with its decisions: it always places orders to buy the cheapest product until demand is met.

Finally, we can verify the that the performance of the DQN agents outperformed the *random baseline* in all scenarios of the problem's dimensions complexity, and outperformed the *minprice baseline* in the following scenarios: *3 products/36 steps, 3 products/288 steps, and 8 products/36 steps.* For the scenario with *8 products/288 steps*, on average, the agent outperformed the *minprice* baseline; however, in several episodes, the DQN policy has low performance.

For the scenarios with *3 products/36 steps and 8 products/36 steps*, the policies learned by the agent have an average performance that is close to *optimal performance with full knowledge*, and they reach the maximum returns that can be obtained in the environment in some episodes (performance = 1).

6.4 Policies Visualization

As explained in Chapter 3, in the DQN algorithm, we use an ANN as a function approximator for the *Q value function* of the *MDP*. The *Q value function* takes as input the state of the environment and outputs the expected future return of each available action. The policy (action to be taken in each state) can then be determined according to: $\pi(s) = \text{argmax}_a Q(s)$.

During the 1000 episodes that were simulated to test the policies, we logged: the states used as input to the ANN and the action taken by the agent. It is possible to visualize the policy (with reduced dimensions) by plotting the number of times each action was taken by the agent, for different values of μ_t.

In Figure 6.3 and Figure 6.4, we show the policy visualization for problems considering 3 products. Note that we separated the policy visualization according to the cheapest product of the given state. Also, Figure 6.5 shows the distribution of prices for the environment's model over 1000 episodes.

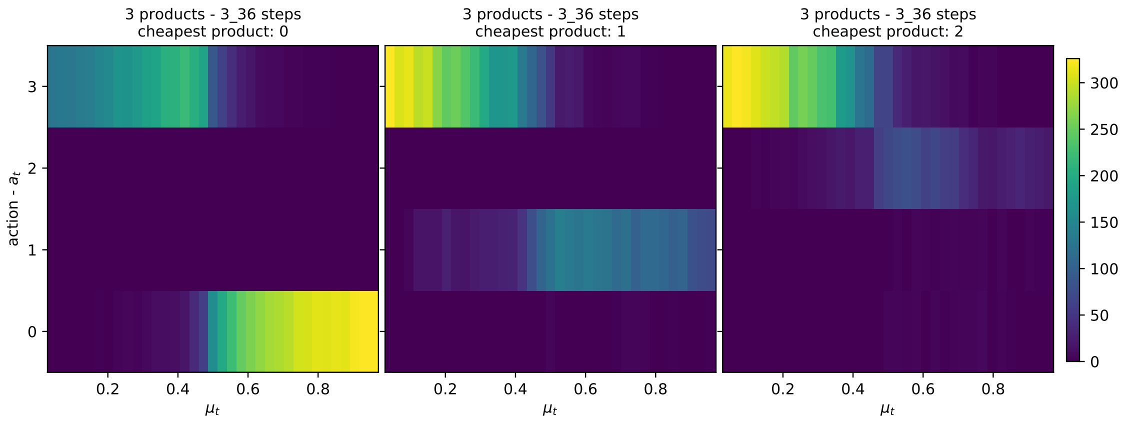

Figure 6.3: Number of times each action was taken for different values of μ_t, considering and environment with *3 products/36 steps.*

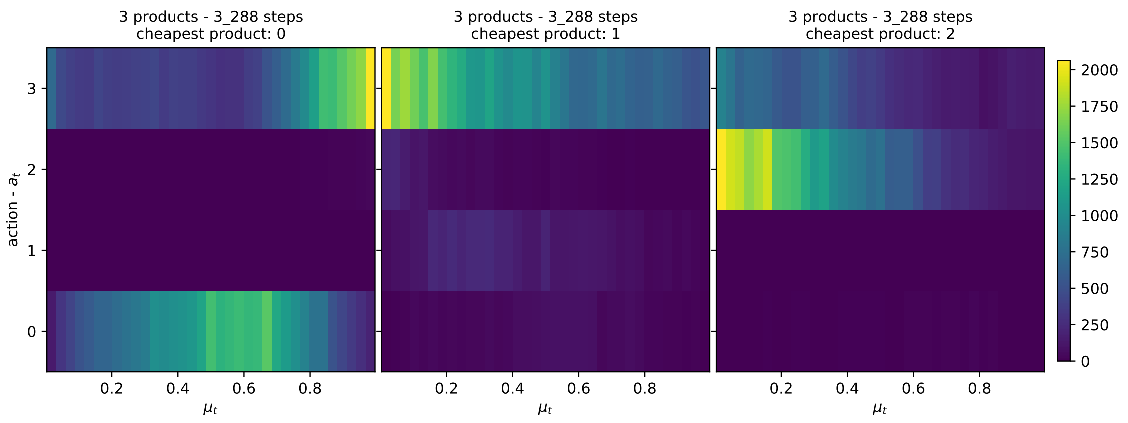

Figure 6.4: Number of times each action was taken for different values of μ_t, considering and environment with *3 products/288 steps*.

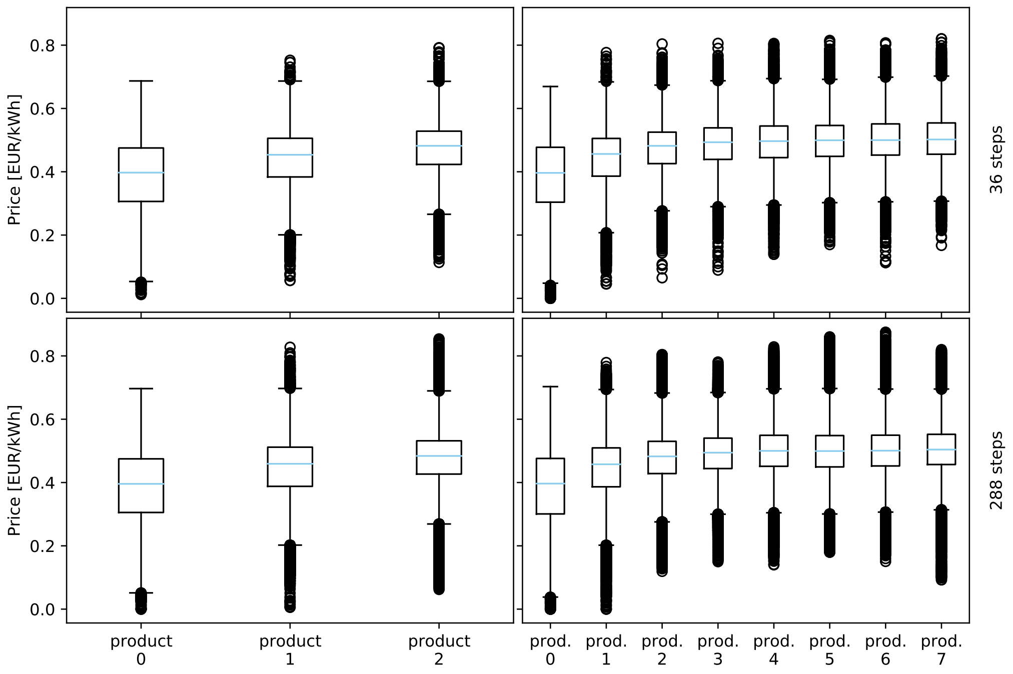

Figure 6.5: Distribution of prices over 1000 episodes, for the 4 scenarios analyzed.

The policy visualization for the problem with *3 products/36 steps* (see Figure 6.3) shows that the behavior of the agent reflects not buying electricity in the first half of trading session, and then buy electricity from the cheapest product in the second half. Considering that the number of times action 0 is taken, one might think that the agent tends to buy electricity from Product 0. However, this happens due to the price dynamics used in the model: the price of Product 0 is usually lower than other products for all environment scenarios (see the prices' distribution in Figure 6.5).

On the other hand, the agent trained considering an environment with 3 products and 288 steps has a different behavior (see Figure 6.4). The agent did not learn that he can wait for the second half of the trading session to buy electricity and get the highest rewards. Increasing the number of steps from 36 to 288 increased the complexity of the problem, and the number of experiences that the agent had with the environment were not enough to learn in the optimal behavior.

If we compare the performance of the agent for *3 products/288 steps* with the performance of the *minprice baseline* (see Figure 6.2), we see that the first slightly outperforms the second. This is confirmed by the behavior visualized in Figure 6.4 - the agent places orders in the middle of the trading session while the *minprice baseline* places orders in the first half of the trading sessions. Considering that prices are usually higher in the beginning than in the middle, the baseline performance is indeed lower.

Figure 6.6 and Figure 6.7 show the policy visualization for the problem considering 8 products (the analysis was not divided by the cheapest product). It was decided not to divide the analysis according to the cheapest product for the 8 products case because it would increase the number of plots to eight, making the analysis too complex.

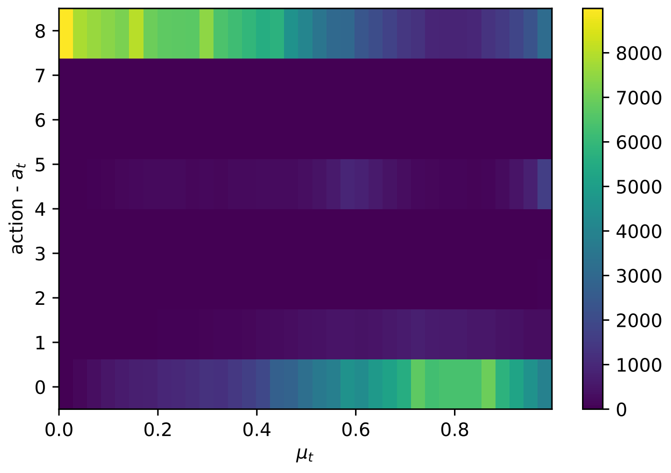

Figure 6.6: Number of times each action was taken for different values of μ_t, considering and environment with *8 products/36 steps*.

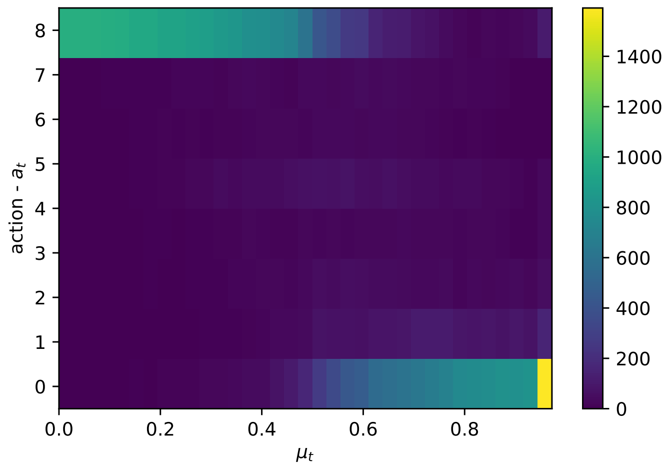

Figure 6.7: Number of times each action was taken for different values of μ_t, considering and environment with *8 products/288 steps*.

Comparing the distribution of actions of Figure 6.6 and Figure 6.7, we can see that, for an environment modeled with 36 time steps, the agent follows a policy that is more similar to the optimal behavior (not buying electricity in the first half of the trading session and buying from the cheapest product in the second half). We can state that because in Figure 6.7 the actions are more distributed than the actions in Figure 6.6 (compare the magnitudes of the color-legends).

With a policy that has more distributed actions like the one of Figure 6.7, we can expect (considering the assumptions made and price dynamics used) that the performance of the agent for different episodes is going to vary depending on the price dynamics. This is exactly the performance distribution of Figure 6.2 (bottom-right), for the scenario with *8 products/288 steps*.

We can also verify from the prices' distribution of Figure 6.5 that product 0 is usually the cheapest product in the trading session of all scenarios. Consequently, in Figure 6.6 and Figure 6.7, we see that the number of times that Action 0 was taken is higher when compared to others – in this case, the agent is buying from the cheapest product.

As explained before, the agents' selection of actions that resulted in the distributions of Figure 6.3, Figure 6.4, Figure 6.6, and Figure 6.7 is based in the *Q-value* function approximator, which predicts the expected future return given a state s. In the next section, we will analyze how well the neural network used to estimate the *Q-Value* function performed, by comparing the predictions with the actual future return obtained in the episodes.

6.5 Evaluating the Q-Value Function Approximation

The architecture of the neural networks used to approximate the *Q-value function* depends on the number of products of the scenario analyzed. For the environment considering 3 products, the input layer contains 5 nodes, 2 hidden layers with 64 nodes each, and an output node with 3 nodes (1 node for each available action). For the environment considering 8 products, the input layer contains 10 nodes, 2 hidden layers with 64 nodes each, and an output node with 8 nodes (1 node for each available action).

The architecture used for the function approximator consists in a fully-connected neural network in both cases. Figure 6.8 illustrates the fully-connected neural networks used in the scenarios with 3 product (left) and 8 products (right) – the architecture does not change for scenarios with different time steps.

To investigate how well the neural networks were approximating the expected future rewards, we logged the output of the *Q-value function* in the node of the action selected by the agent and the actual future rewards obtained in the specific episode. We simulated 1000 episodes of agent-environment interaction and calculated the average of these 2 expected future rewards (DQN prediction and Target) for each simulation step. The 4 scenarios and agents explained before were still being considered. Figure 6.9 shows future rewards curves (*DQN Prediction* and *Target*) for the 4 scenarios analyzed.

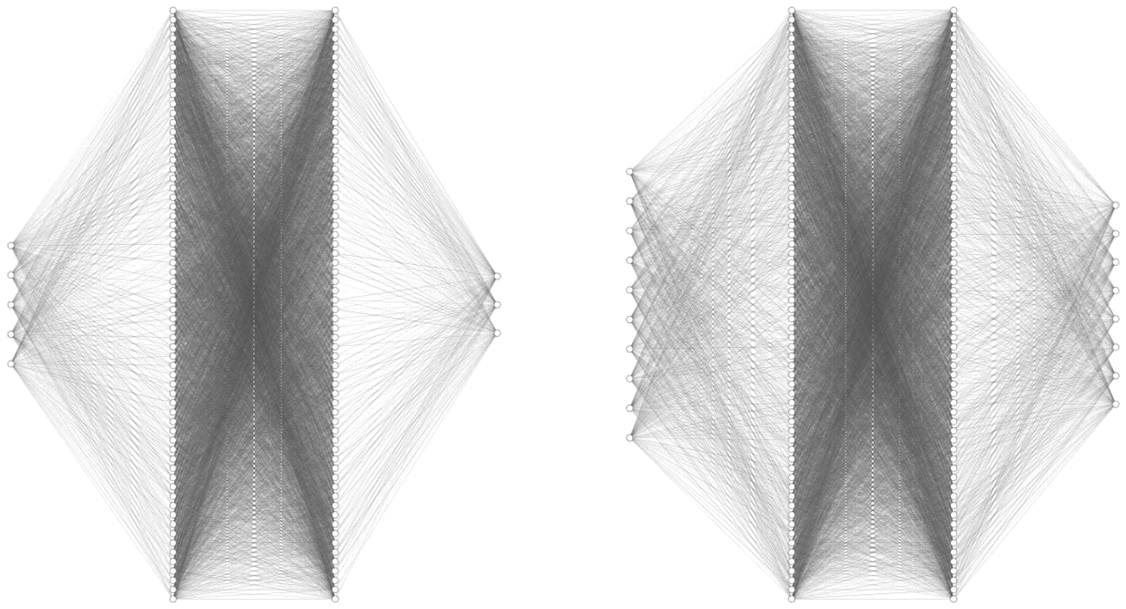
Figure 6.8: Architecture of the fully-connected neural networks used to approximate the *Q-value function* in the scenarios with 3 product (left) and 8 products (right).

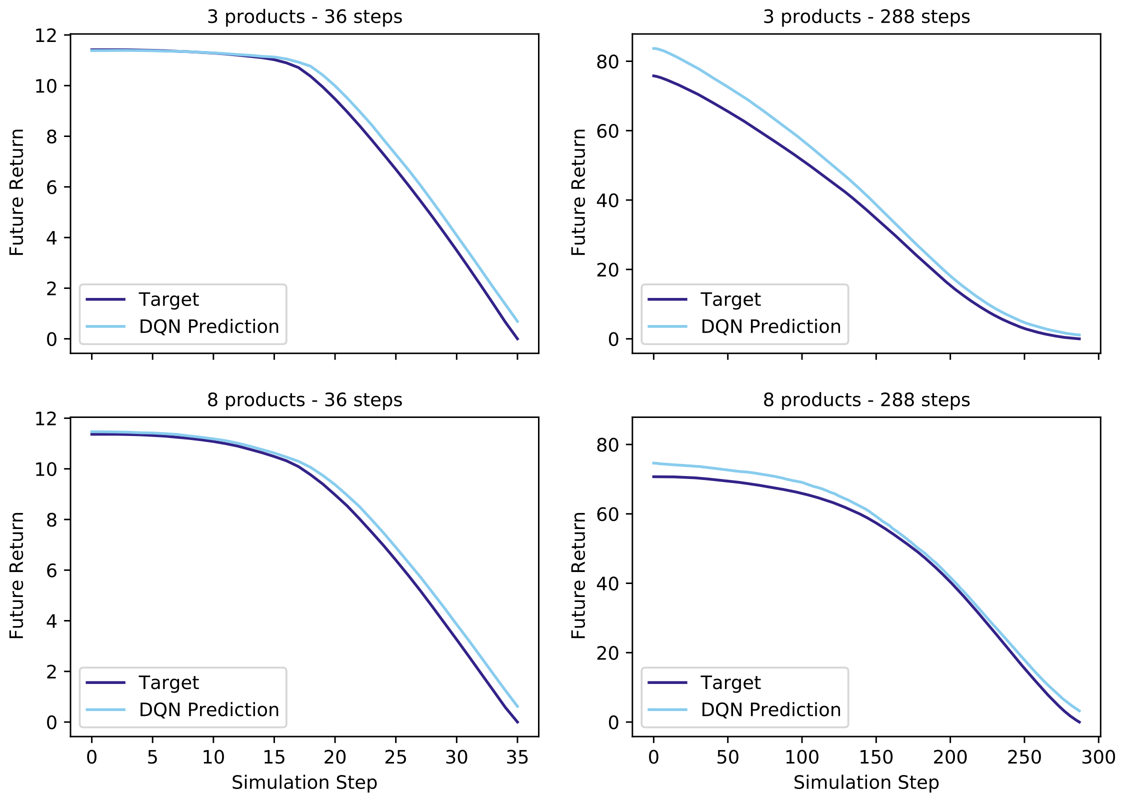

Figure 6.9: Average over 1000 episodes of the expected future return (*DQN Prediction* curve) and the actual future return (*Target* curve).

From Figure 6.9, we can see that the neural networks were successful in approximating the *Q-value functions* for all 4 scenarios. Even for scenarios that the policy learned by the agent resulted in lower performances (for instance, *8 products/288 steps*) the DQN prediction curves are similar to the Target (actual value) curves. This happens because the performance is not only associated with how well the neural network approximates the *Q-value function*. The performance is associated with how well the *Q-value function* is approximated and also to the policy that this *Q-value function* refers. If the policy is not close to the optimal policy, the performances will be low even if the neural network is perfectly representing the Target.

We can intuitively verify which policies are closer to the optimal policy by analyzing the derivative of the curves in Figure 6.9. When the derivative is almost equal to zero ($\frac{df}{dt} \approx 0$), it means that the agent is not getting any reward (probably not buying electricity). When the derivative of the curve is negative, it means that the agent is buying electricity from the cheapest product and getting rewards.

In the scenarios of *36 steps* (Figure 6.9, left) is clear that the behavior is to not buy any electricity in the first half of the trading session (because the derivative is close to zero) and buy from the cheapest product in the second half (negative derivative). Considering the assumptions made, this is precisely the optimal behavior.

The agents with *288 steps* (Figure 6.9, right) have curves that are a bit more rounded when compared to the other ones. The derivatives in these 2 cases reflect policies that are completely different from the optimal behavior, resulting in lower performances.

6.6 Discussion

In this chapter, we analyzed the impact of increasing the environment's dimensions on the problem's complexity. The results showed that increasing the number of time steps from $T = 36$ to $T = 288$ had a significant impact on the DQN agent's performance. Some improvements to the training strategy that could improve the results obtained for $T = 288$ include:

- Increasing the amount of steps for the exploration phase (ϵ_{steps}).
- Increasing the number of training steps.

Increasing the number of products, on the other hand, did not have a significant impact on the agent's performance. However, this might be true only under the price dynamics that we assumed in the environment. Under this assumption, prices of *product 0* are usually the lowest (both for $N = 3$ and $N = 8$), so the problem does not become that different. If the distribution of prices were more uniform (E.g., without the droping price trend assumption), they would have probably learned policies with lower performance when $N = 8$.

In the next chapter, we investigate the modification of the reward function of the environment in order to account for the eventual agent's product preferences based on non-price attributes. The purpose of introducing this is, for instance, to be able to train agents that have goals that are not only aiming to reduce costs, but also to buy more electricity for renewables or local producers.

7 Accounting for Product Preferences

All the agents trained in Chapter 5 and Chapter 6 were not taking into account any product preferences based on non-price attributes that the market participant might have. The policies learned by those agents were only optimizing with the goal of reducing electricity costs, without considering any other differences between products.

In this chapter, we investigate the modification of the reward function of the environment to account for the agent's product preferences based on non-price attributes. We included a vector of *preference factors*, denoted as $\dot{\chi}$, that reflects the willingness of the agent to buy each product available in the market. Equation 7.1 defines $\dot{\chi}$.

$$\dot{\chi} = [\ \chi_0 \quad \chi_1 \quad \cdots \quad \chi_i\ ] \tag{7.1}$$

In the following sections, we describe the assumptions made to the decision making problem and model of the environment, explain the details of how $\dot{\chi}$ was introduced to reward function, show the training and performance results for the agents, analyze how each agent explored the environment, visualize the policies learned, and discuss the results obtained.

7.1 Assumptions

For the analysis of this chapter, we considered an environment that models an electricity market with three products available ($N = 3$) and 36 time steps ($T = 36$). Moreover, the assumption for p_{trend} used in other chapters, which forces the prices to drop over the trading session, is still valid. The value used for β, which defines the fixed quantity in the orders placed by the agents, was defined similarly to Chapter 6, according to Equation 6.2. Consequently, the agent has to place 18 buy orders to meet the flexible demand (D_{flex}).

In addition to the usual assumptions, we considered that the products are differentiated according to the time of delivery, and from the energy source they are being produced. We assumed that:

- *Product 0* and *Product 1* refer to electricity generated from non-renewable energy sources.
- *Product 2* refers to electricity generated from renewables.

Note that we could have differentiated products according to other non-price attributes (such as being produced locally or not), and the analysis/results would still be valid.

To introduce the preferences in the analysis, we considered 3 agents that prefer buying electricity from renewables, however, with different *preference intensity.* In this project, *preference intensity* represents how much agent i is willing to pay extra for buying the product he prefers and is denoted as I_i. We assumed that *Agent 2* is the one willing to pay more for electricity produced from renewables, followed by *Agent 1* and *Agent 0*, therefore:

$$I_0 < I_1 < I_2$$

One way to train agents that can learn a policy that considers product preferences is to design a reward function in the environment that accounts for that. From Equation 4.7, it is clear that the reward that is given back to the agent by the environment is not dependent on anything else but the product's price – so, this has to be modified.

We included a constant to the reward function, denoted as χ_i, which depends on the product preferences of the agent – larger values of χ_i reflect higher values of *preference intensity* (I_i). Moreover, we must remove **Preposition 3** from the conditions of the reward function; otherwise, the agent would never get rewards when placing orders in a product that is not the cheapest. The reward function considering these modifications is described in Equation 7.2, and its value is calculated according to **Proposition 1** and **Proposition 2**.

Proposition 1 - The agent did not place a buy order. In this case, $a_t = N$.

Proposition 2 - The agent placed a buy order and the total holdings would exceed its demand ($\omega > 1$).

$$r_{t+1} = \begin{cases} 0, & \textbf{if:} \\ & \quad \text{Proposition 1 is true} \\ -0.1, & \textbf{if:} \\ & \quad \text{Proposition 1 is false AND Proposition 2 is true} \\ \frac{\chi_i}{e^{\hat{p}_{t,i}}} - \frac{\hat{p}_{t,i}}{e}, & \textbf{if:} \\ & \quad \text{Proposition 1 is false AND Proposition 2 is false} \end{cases} \tag{7.2}$$

The value of χ_i used to calculate the reward is dependent on the product that was bought with the action taken and should reflect the willingness of the agent to buy that product. To reflect the assumptions we made about agents' preferences, we defined the values of χ_i for each agent, as described in Table 7.1.

Table 7.1: Values of χ_i considered for each Agent.

	χ_0	χ_1	χ_2
Agent A	1.0	1.0	1.10
Agent B	1.0	1.0	1.25
Agent C	1.0	1.0	1.50

The impact of changing χ_i on the scalar feedback given by the reward function of Equation 7.2 is illustrated on the curves of Figure 7.1. It is possible to see that, if increasing the value of χ_i, the scalar feedback is moved to higher rewards for the same product price. Consequently, the rewards take into consideration the agent's product preferences.

Considering the assumptions made, we expect that the policies learned by these agents should reflect their willingness to buy from *Product 2*.

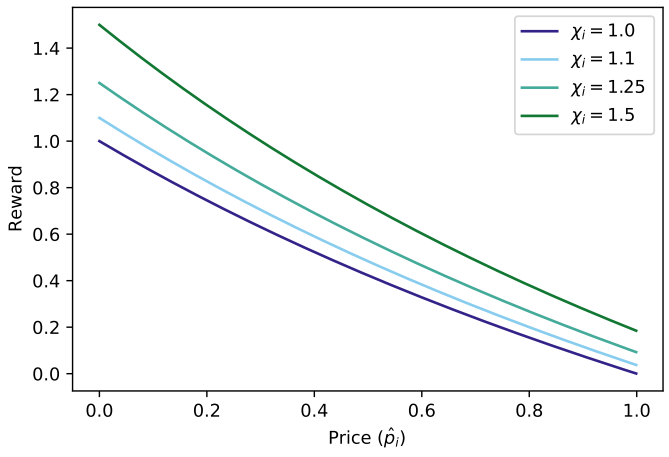

Figure 7.1: Impact of changing χ_i in the rewards given by the environment according to the price paid by the product.

7.2 Training Results

The agents were trained for 5×10^6 steps, considering the same hyperparameters, the performance was monitored throughout the training. Like other chapters' analyses, the performance is calculated by the ratio between the sum of rewards received in the episode and the maximum reward that could be received.

The hyperparameters are specified in Table 7.2, and Figure 7.2 shows the moving average (100 episodes) of the agents' performance throughout the training loop.

Table 7.2: Hyper-parameters used in the training.

Hyper-parameter	**Value**
γ	$[0, 0.9, 1.0]$
B	32
E_{size}	500000
T_{sync}	10000
α	0.0001
ϵ_{start}	1.0
ϵ_{final}	0.02
ϵ_{steps}	100000

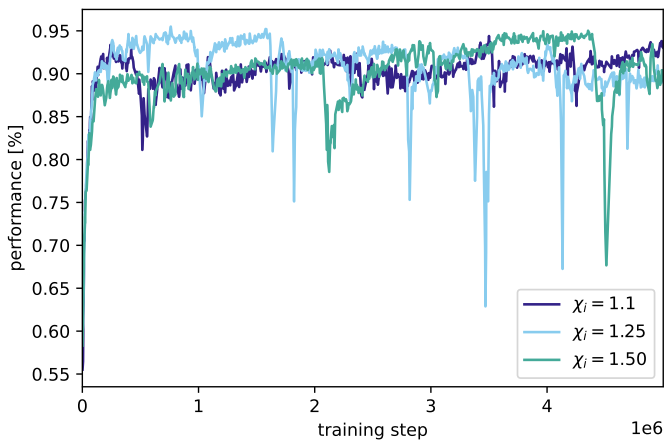

Figure 7.2: Agent's performance along the training loop (rolling average over 100 episodes) for agents considering $\chi_2 = 1.1$ (dark blue), $\chi_2 = 1.25$ (light blue), and $\chi_2 = 1.5$ (green).

In Figure 7.2, it is possible to see that the agents converged to similar performance values. The three agents reached performances of 0.95 during the training loop. However, the training was less stable compared to agents trained without χ.

Comparing the three curves of Figure 7.2 with the curve of Figure 6.1 that refers to *3 products/36 steps*, we can conclude that introducing the changes to the reward function in order to account for preferences reduced the training stability and increase the time for convergence.

One possible explanation for this impact is the fact that we removed **Proposition 3** from the reward function. This proposition was previously forcing the environment to send *null* rewards to the agent if placing orders in a product that was not the cheapest. Without this proposition, the difference in the reward given to the agent is smaller when the products have similar prices. By removing **Proposition 3**, the overall goal of the agent is less clearly defined in the reward function.

7.3 Performance Results

The three agents' policies were tested over 1000 episodes, experiencing the same randomness and price dynamics (1000 fixed random seeds were used for the simulations). The distribution of performance of the agents in this 1000 episodes are shown in Figure 7.3, Figure 7.4, and Figure 7.5. The three performances are compared to the performances of agents following policies of the *random baseline* and the *minprice baseline* – as explained in Section 4.8.

The first thing that is possible to verify in Figure 7.3, Figure 7.4, and Figure 7.5 is the fact that the *random baseline* performed better in the current environment if compared to the results of Chapter 5 and Chapter 6. When we removed **Proposition 3** from the reward function, rewards are given to the agent even if he is buying electricity from a product that is not the cheapest. Consequently, when taking random actions, we get higher rewards compared to the situation that we had in Chapter 5 and Chapter 6 – taking **Proposition 3** into account.

Also, we could see that the agents' policy performed significantly better than both baselines in all scenarios. Therefore, introducing the *preference factor* χ did not compromise the suitability of the DQN algorithm.

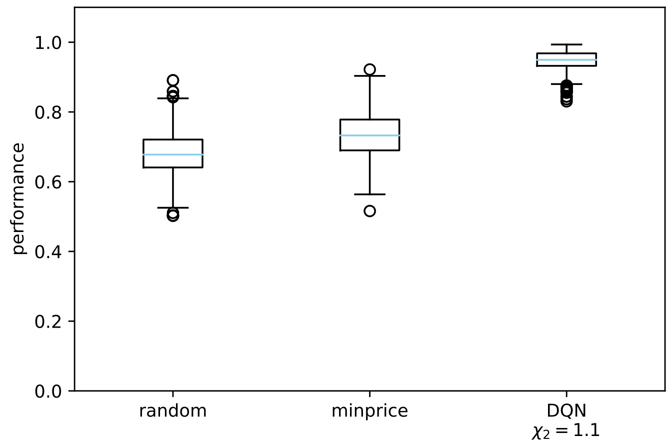

Figure 7.3: Distribution of agents' performance over 1000 episodes. The Figure compares the performances of the 2 baselines (*random* and *minprice*) with the DQN algorithm considering $\chi_2 = 1.1$.

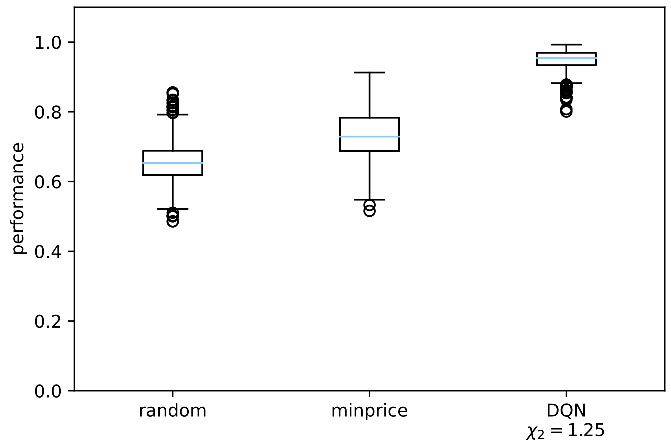

Figure 7.4: Distribution of agents' performance over 1000 episodes. The Figure compares the performances of the 2 baselines (*random* and *minprice*) with the DQN algorithm considering $\chi_2 = 1.25$.

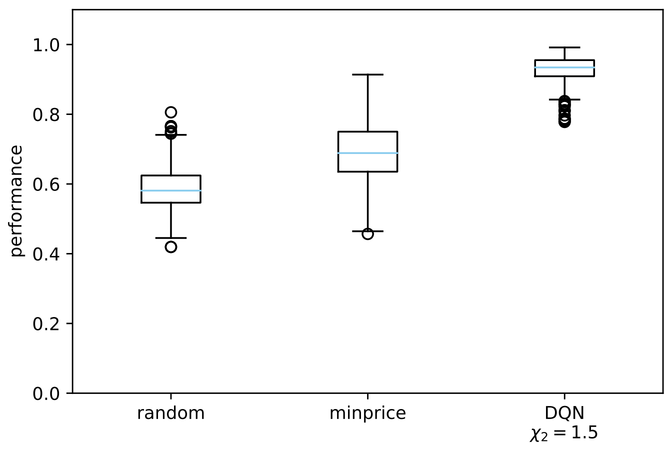

Figure 7.5: Distribution of agents' performance over 1000 episodes. The Figure compares the performances of the 2 baselines (*random* and *minprice*) with the DQN algorithm considering $\chi_2 = 1.5$.

7.4 Policies Visualization

Similar to what was done in previous chapters, we reduced the dimensions of the state space to be able to visualize the policies learned for each agent. The policies are visualized along the μ_t dimension (progress of the episode).

To achieve this, during the simulation of the 1000 episodes, we logged the states observed by the agent and the actions taken subsequently. Considering that the policies are used to select the actions to be taken, if we count the number of times each action was selected for different values of μ_t, we are able to visualize the policy in this dimension. Figure 7.6, Figure 7.7, and Figure 7.8 show the visualization of the policies for *Agent A*, *Agent B*, and *Agent C*, respectively.

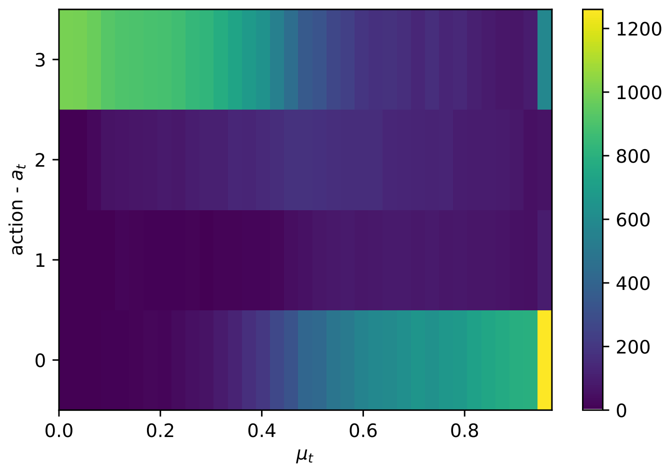

Figure 7.6: Number of times each action was taken for different values of μ_t, considering $\chi_2 = 1.10$.

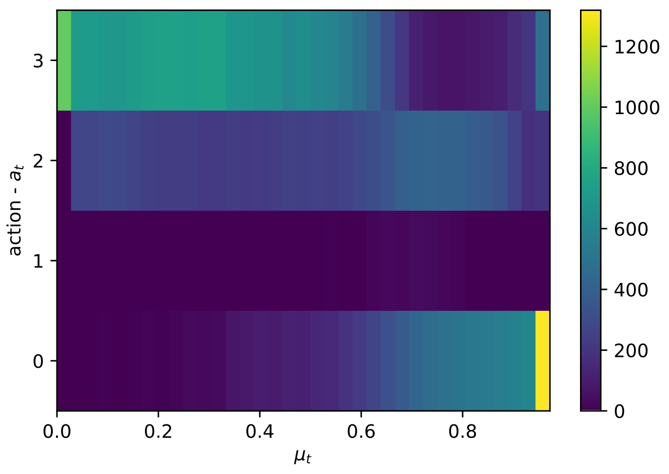

Figure 7.7: Number of times each action was taken for different values of μ_t, considering $\chi_2 = 1.25$.

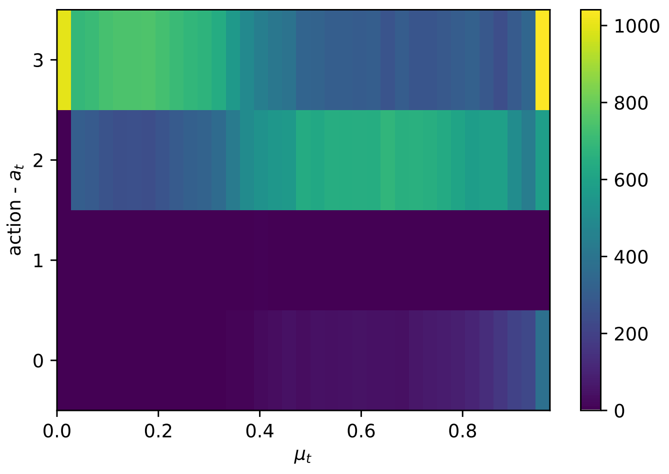

Figure 7.8: Number of times each action was taken for different values of μ_t, considering $\chi_2 = 1.50$.

When we compare the number of times the *action 2* (that refers to buying electricity from *Product 2*) in Figure 7.6, Figure 7.7, and Figure 7.8, it is clear that this number increases when increasing the *preference factor* χ_2. The learned policies indicate that the expected result of χ_i was achieved – reflect the agent's willingness to pay more for an specific product.

In Table 7.3, it is possible to verify the number of times the agents took each action over the 1000 episodes. We can see that increasing χ_2 from 1.10 to 1.25 changed the number of times the preferred product (*Product 2*) was selected from 3758 to 9685 – an increase of 158%. Moreover, increasing χ_2 from 1.25 to 1.50 resulted in a growth of 73%. Therefore, the relationship between χ and the agent's behavior of buying the preferred product is not linear.

Table 7.3: Number of times the agents took each action over the 1000 episodes.

	Product 0	**Product 1**	**Product 2**	**Not Buying**
Agent A ($\chi_2 = 1.10$)	13313	1575	3758	17354
Agent B ($\chi_2 = 1.25$)	8445	206	9685	17664
Agent C ($\chi_2 = 1.50$)	2001	21	16756	17222

7.5 Discussion

The results of the previous sections support the fact that we were able to account for agent's product preferences by introducing a vector of *preference factors*, denoted as $\dot{\chi}$, to the environment's reward function. In the analysis, we assumed that $\dot{\chi}$ was related to a specific non-price attribute (electricity generated from renewables or not). However, the method would provide the same results for different non-price attributes (e.g., electricity generated locally or not) or for a combination of many non-attributes.

The results also indicate that the relationship between the *preference factor* χ_i and the number of times the preferred product is going to be selected among available products is not linear. Therefore, setting the correct values of χ_i for an agent to reflect the exact willingness to buy a specific product require a trial and error method.

8 Conclusion

This work investigated the use of the DQN algorithm for learning a policy used to automate the decision-making problem of an agent interacting with an environment representing the P2P electricity market of the Svalin's community.

We started by reviewing the electricity market framework proposed by [4], which provides the mechanisms of the Svalin's market. Next, we reviewed the RL framework and explained the algorithm investigated in the project (DQN). The RL framework is suitable to solve sequential decision-making problems/stochastic optimization problems, and its general functionality is based on the agent's experience and exploration of the environment. We selected this method to solve the decision-making problem based on the work of [10]. Then, we formulated the decision-making problem and its goals, explained the assumptions made to the environment model, and give details about how the rules of the agent-environment interaction. Finally, three analyses were made to the problem: we investigated the impact of the discount factor on the agent's performance, verified the suitability of the method to models with higher complexity, and introduced the possibility of expressing agent's product preferences based on non-price attributes in the decision-making problem.

The discount factor investigation showed that the three γ values tested had significantly different performances. The agent trained considering $\gamma = 1$ performed better and was able to learn a policy that reflects the optimal behavior with full-knowledge. Also, we verified that the value of γ impacted the way agents explored the environment. In the scenarios of $\gamma = 0$ and $\gamma = 0.9$, the agents did not experience the states that provide the highest rewards. Finally, the visualization of policies showed that agents with $\gamma = 0$ and $\gamma = 0.9$ were short-sighted, while the agent with $\gamma = 1$ was sacrificing immediate rewards to get higher rewards in future.

The analyses of the method's suitability for scenarios considering environments modeled with more dimensions and higher complexity were based on cases considered a different number of time steps ($T = 36$ and $T = 288$), and products available ($N = 2$ and $N = 6$). Based on the results, changing the number of products available in the market is less critical than increasing the number of time steps in the simulations. However, this might be true only under the price dynamics that we assumed in the environment (dropping price trend assumption).

Finally, to introduce the possibility of expressing an agent's product preferences based on non-price attributes, we modified the reward function of the environment by including a vector of *preference factors*, denoted as $\dot{\chi}$, that reflects the agent's willingness of buying each product. The policies learned by agents considering different values for the preference factor of a specific product indicate that the method was successful in modifying the behavior according to preferences. The *preference intensity* of the agents can be controlled by selecting the right values for $\dot{\chi}$. However, the relationship between $\dot{\chi}$ and the number of times the preferred product is selected is not linear.

8.1 Challenges

The main challenges faced during this work were mostly related to the amount of time needed to train agents. For instance, the definition of hyperparameters in the DQN algorithm was essential to make the training feasible. However, sometimes a prolonged time was needed to realize that agents were not learning anything due to wrong hyperparameters. Moreover, increasing some hyperparameters, e.g., the experience replay buffer size (E_{size}) and batch size (B), was reducing the training speed significantly. Also, the challenge of selecting the right complexity for the environment model and suitable price dynamics was a complicated decision to make.

8.2 Future Work

Some topics that could be explored to extend the work of this project include:

- Analyze the suitability of the method for an environment with non-stationary dynamics. For instance, it is possible to investigate how many extra steps the DQN algorithm needs to adapt its policy to an abrupt change in the environment (for instance, a sudden change from a dropping price trend to a rising trend). Methods that automate the adjustment of the learning rate are already studied [24].

- Include more dimensions to the state-space representation. The dimensions could include forecasts, order book data, set of trades executed in the market.

- Investigate the impact of introducing transaction fees in the model of the environment.

- Reformulate the problem and agent-environment interaction to include actions that refer to placing sell orders.

- Investigate the decision-making problem in a multi-agent perspective. Similarly to the work of [25], but considering a P2P electricity market as the environment.

References

[1] Pierre Pinson et al. “The Emergence of Consumer-centric Electricity Markets”. In: *Distribution & Utilization* 34.12 (2017), pp. 27–31.

[2] Merlinda Andoni, Valentin Robu, and David Flynn. “Blockchains: Crypto-control your own energy supply”. In: *Nature* (2017). ISSN: 14764687. DOI: 10.1038/548158b.

[3] Murray Goulden et al. “Smart grids, smart users? the role of the user in demand side management”. In: *Energy Research and Social Science* (2014). ISSN: 22146296. DOI: 10.1016/j.erss.2014.04.008.

[4] H S Esch et al. “Online matching and preferences in future electricity markets”. In: *Proceedings of the Nineteenth Yale Workshop on Adaptive and Learning Systems.* APA, 2019. ISBN: 2016112530.

[5] Thomas Morstyn et al. “Using peer-to-peer energy-trading platforms to incentivize prosumers to form federated power plants”. In: *Nature Energy* (2018). ISSN: 20587546. DOI: 10.1038/s41560-017-0075-y.

[6] Chris Giotitsas, Alex Pazaitis, and Vasilis Kostakis. “A peer-to-peer approach to energy production”. In: *Technology in Society* (2015). ISSN: 0160791X. DOI: 10.1016/j.techsoc.2015.02.002.

[7] Tiago Sousa et al. “Peer-to-peer and community-based markets: A comprehensive review”. In: *Renewable and Sustainable Energy Reviews* (2019). ISSN: 18790690. DOI: 10.1016/j.rser.2019.01.036. arXiv: 1810.09859.

[8] G. Le Ray and P. Pinson. “The ethical smart grid: Enabling a fruitful and long-lasting relationship between utilities and customers”. In: *Energy Policy* 140 (2020), p. 111258. ISSN: 0301-4215. DOI: https://doi.org/10.1016/j.enpol.2020.111258. URL: http://www.sciencedirect.com/science/article/pii/S0301421520300185.

[9] Etienne Sorin, Lucien Bobo, and Pierre Pinson. *Consensus-based approach to peer-to-peer electricity markets with product differentiation.* 2018. arXiv: 1804.03521 [cs.SY].

[10] Helge Stefan Esch. *Autonomous Bidding Agent in P2P Electricity Market.* 2020.

[11] Warren B. Powell. “A unified framework for stochastic optimization”. In: *European Journal of Operational Research* (2019). ISSN: 03772217. DOI: 10.1016/j.ejor.2018.07.014.

[12] *The Energy Collective project.* URL: https://the-energy-collective-project.com/context/ (visited on 02/19/2020).

[13] Y. Ye et al. “Deep Reinforcement Learning for Strategic Bidding in Electricity Markets”. In: *IEEE Transactions on Smart Grid* 11.2 (2020), pp. 1343–1355.

[14] Markus Peters et al. “A reinforcement learning approach to autonomous decision-making in smart electricity markets”. In: *Machine Learning* 92 (July 2013). DOI: 10.1007/s10994-013-5340-0.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, Second Edition: An Introduction - Complete Draft.* 2018. ISBN: 9780262039246.

[16] Warren B. Powell. *Clearing the Jungle of Stochastic Optimization.* 2014. DOI: https://doi.org/10.1287/educ.2014.0128.

[17] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* (2015). ISSN: 14764687. DOI: 10.1038/nature14236.

[18] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[19] Greg Brockman et al. *OpenAI Gym.* cite arxiv:1606.01540. 2016. URL: http://arxiv.org/abs/1606.01540.

[20] José Vázquez-Canteli et al. “CityLearn v1.0: An OpenAI Gym Environment for Demand Response with Deep Reinforcement Learning”. In: Nov. 2019, pp. 356–357. ISBN: 978-1-4503-7005-9. DOI: 10.1145/3360322.3360998.

[21] EEX Group. *EPEX SPOT.* URL: https://www.epexspot.com/. (accessed: 02.08.2020).

[22] Enrico Edoli, Stefano Fiorenzani, and Tiziano Vargiolu. “Optimal Trading Strategies in Intraday Power Markets”. In: *Optimization Methods for Gas and Power Markets: Theory and Cases.* London: Palgrave Macmillan UK, 2016, pp. 161–184. ISBN: 978-1-137-41297-3. DOI: 10.1057/9781137412973_8. URL: https://doi.org/10.1057/9781137412973_8.

[23] Marco Piccirilli and Tiziano Vargiolu. *Optimal Portfolio in Intraday Electricity Markets Modelled by Lévy-Ornstein-Uhlenbeck Processes.* 2018. arXiv: 1807.01979 [q-fin.PM].

[24] Chang Xu et al. *Reinforcement Learning for Learning Rate Control.* 2017. arXiv: 1705.11159 [cs.LG].

[25] Jose Vazquez-Canteli et al. “Multi-agent reinforcement learning for adaptive demand response in smart cities”. In: *Journal of Physics: Conference Series* 1343 (Nov. 2019), p. 012058. DOI: 10.1088/1742-6596/1343/1/012058. URL: https://doi.org/10.1088%5C%2F1742-6596%5C%2F1343%5C%2F1%5C%2F012058.

A Appendices

A.1 GitHub Repository

The environment implemented according to the OpenAI Gym framework and the DQN algorithm implemented using PyTorch are available in the following GitHub repository:

https://github.com/lucasblt/DQN4P2Pmarket