UCLouvain

École polytechnique de Louvain

Deep Reinforcement Learning applied to a whole-energy system

Authors: **Denis Pinsar, Hadrien Plancq**
Supervisor: **Francesco Contino**
Readers: **Emmanuel De Jaeger, Xavier Rixhon**
Academic year 2020–2021
Master [120] in Mathematical Engineering and Master [120] in Electro-mechanical Engineering

Abstract

The European Union intends to move towards zero carbon emission by 2050. There is no doubt that the decisions taken by the politicians in the coming years will play an important role in our ability to reach this emission reduction target. Unfortunately, the human brain is not capable of considering all the policy actions and their impacts in terms of cost and emissions reduction on a country-size whole-energy system.

The aim of this master's thesis is to place the politician in different scenarios and determine which are the most appropriate levers to activate for the purpose of helping the energy transition. The policy framework of the politician is to impose taxes or incentives on resources and technologies used in the Swiss energy system. To do so, we have implemented a deep Q-learning model that will provide an appropriate path to follow in terms of political actions.

This paper is a first attempt to apply a deep Q-learning model on a whole-energy system in order to find an appropriate series of actions which allows to reach a reduction emissions target while minimizing the cost. The results obtained on a simplified problem meet the objectives set. However, the computational time of the model representing the energy system (EnergyScope TD) is the limiting factor for considering scenarios that are closer to reality. This work is therefore the foundation on which improvements could be made to enable the model to obtain results suitable for the real world.

Acknowledgements

First of all, we would like to thank our advisor, Professor Francesco Contino, who allowed us to work on this subject. Thanks to his precious advice and the trust he placed in us, we were able to complete our master's thesis.

We would like to thank Xavier Rixhon in particular for his weekly follow-up throughout the year. He was always available to answer our questions and give his opinion on the quality of our work. Once again, we would like to express our profound gratitude to Xavier Rixhon and Professor Francesco Contino for teaching us the efficient way of transmitting a scientific message.

We are grateful to Professor Emmanuel De Jaeger for accepting to be a reader of this master's thesis.

Thanks to Stefano Moret and Gauthier Limpens for developing EnergyScope Typical Days that allowed us to represent an energy system. And thanks to AMPL for providing us their solver.

We would also like to thank the entire team with whom we had our weekly meetings: Guillaume André, Anthony Devresse, Benjamin Goffaux and Xavier Rixhon.

Our last thanks will go to Michel Henry who kindly accepted to review this master's thesis.

Contents

Acronyms

CR cumulative reward.

DQL deep Q-learning.

ESTD EnergyScope Typical Days.

GWP global warming potential.

QL Q-learning.

RL reinforcement learning.

Introduction

Climate change is a growing concern since the second part of the 20th century. This goes without saying, the way human is living is not sustainable. Our energy consumption is the main reason for the greenhouse gas emissions. That is why, each country energy system has to be redesigned to meet the political objectives. Europe has the ambition of becoming carbon-neutral by 2050, i.e. to balance CO_2 emissions using CO_2 removal.

We are facing one of the biggest challenges humankind has ever encountered. Even though small scale (individual or small communities) actions are really important for the transition, there is no doubt that political decisions will have a major impact on our ability to reach the emissions target [1]. The goal of this thesis is to give tools to a politician to help establish a carbon-neutral state.

In the literature, there is a framework to facilitate the participation of scientists to policy design towards energy transition. However, this framework is mainly concerned with reconciling the desires of shareholders and scientists in the context of local energy projects. It cannot be applied to a country-size energy system [2].

One of the biggest obstacles in the energy transition is the complexity of energy systems, especially for a country-scale energy system. It is impossible for a human brain to investigate each policies combination that a politician could establish in order to select the most appropriate one. That is why, reinforcement learning (RL) tools are currently interesting to investigate in the context of complex energy systems. This field of machine learning is particularly suited for decision-making.

Perera and Kamalaruban give a good overview of what is already done concerning RL in the field of energy systems [3]. They explain that a RL approach on a whole-energy system has never been investigated in the literature. Especially using deep Q-learning (DQL), which is an application of deep reinforcement learning, a sub-field of RL. Indeed, only publications related to the adjustment of a controller for a low-energy building system [4] and related to the control of the electricity

dispatch [5, 6] have reported the use of DQL.

Contino, Moret and Limpens et al. justify the interest of looking at an energy system in its entirety for an energy transition context [7]. We will put ourselves in the shoes of a politician who tries to reach an emission target while minimizing the cost of the system. To do so, this politician will be able to impose taxes or incentives on resources and technologies of the energy system.

The goal of this master's thesis is thus to answer the following research questions:

- Which series of incentives and taxes can a politician establish on a country-scale whole-energy system in order to reach a CO_2 emission target ?
- Practically, how to determine this optimal sequence of actions by implementing a deep Q-learning (DQL) model on a whole-energy system ?

To represent a country-scale energy system, several models exist and are compared in [8]. We have decided to go with EnergyScope Typical Days (ESTD) [8] for several reasons. First of all, it is a multi-sector model which means that the sectors of electricity, heat and mobility are represented with the same level of detail. Moreover, this model takes into account the financial aspect of the system through its investment and operating costs. This is essential because money plays an important role in political decisions nowadays. Finally, it is an open-source model.

ESTD is based on linear programming, a mathematical method that is used to determine the best possible outcome or solution from a given set of parameters or list of requirements, which are represented in the form of linear relationships [9]. The goal of ESTD is simple: satisfy a known energy demand at any time while minimizing the total cost of the system and respecting a constraint on the CO_2 emissions. In the context of this master's thesis, this constraint will be set far greater than the current emissions of the energy system in order to drive emission reductions by taxes and incentives.

Chapter 1 will present the energy model used in this thesis, EnergyScope TD. In Chapter 2, a theoretical overview of deep Q-learning is presented. Chapter 3 will allow the selection of an appropriate convergence criterion that will be used in the following chapters. The practical implementation of the DQL model is done in Chapter 4. We will see, among other things, the different scenarios in which the politician will be immersed and a first estimation of the hyper-parameters of our model. The next two chapters present the two kinds of results that we obtained. The goal of Chapter 5 is to vary the hyper-parameters of the model to see which

ones have an impact on the convergence speed. It will therefore allow us to improve the performances of the model. CHAPTER 6 will be dedicated to the analysis of the impact on the energy system of the series of actions provided by the DQL model. A discussion will be held to take a step back from the information presented in this report.

Chapter 1

Energy model description

The energy system is represented using EnergyScope Typical Days (ESTD). A brief introduction is given about this method and its main features.

An energy system can be divided into three main categories: the **resources**, the **end-use demand** and the **energy conversion** to make the link from the resources to the end-use demand. The working principle of ESTD is detailed through an explanation of the three categories. This is illustrated thanks to a simple example of an energy system represented in FIGURE 1.1.

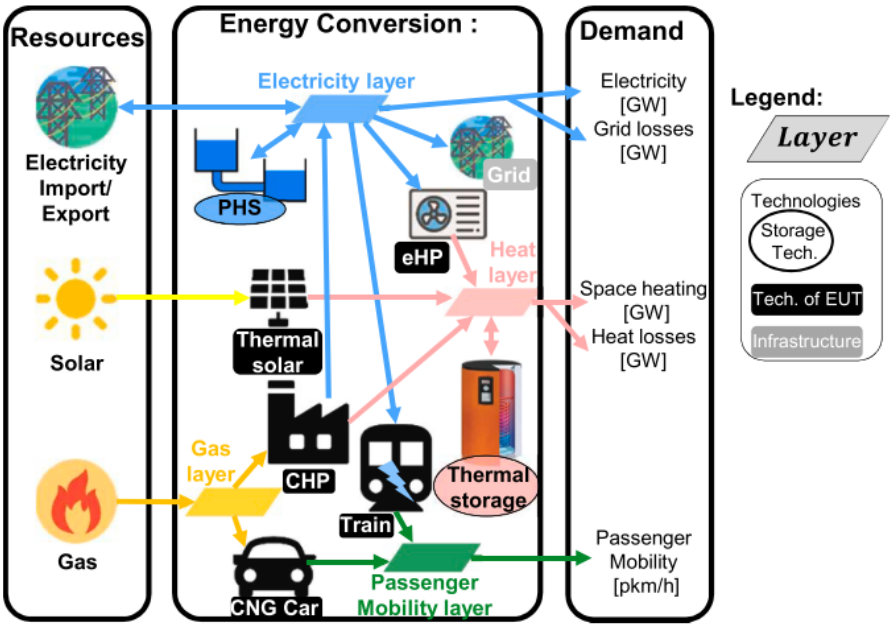


Figure 1.1: Simplified example to illustrate EnergyScope TD working principle [8].

Resources

ESTD needs to know all the resources which can be exploited by the energy system with their operation cost [M€/GWh], their global warming potential (GWP) [ktCO_2/GWh] and their availability [GWh/y]. In the example, there are 3 resources available: electricity import/export, solar energy and natural gas.

End-use demand

It is important to make the difference between two concepts: end-use demand and final energy consumption. To illustrate with the example of a boiler, the end-use demand is the amount of heat produced while the final energy consumption is the amount of natural gas consumed.

For an energy system, all the end-use demand can be classified into three categories visible in FIGURE 1.1: electricity, heating and mobility. Each of the categories can be decomposed into end-use types. For the heat, they are: high temperature heat for industry demand and low temperature for sanitary water and space heating. The mobility demand can be decomposed into 2 end-use types: passenger and freight mobility. The system requires the detailed end-use demand.

Energy Conversion

There are three different elements in the energy conversion part:

- **Technology of end-use type**: used to convert energy. Such as thermal solar installations that convert solar energy into heat.
- **Storage technology**: used to store energy. When we deal with an increasing share of renewable in the energy system, storage becomes really important. Most of renewable resources are not available at all times, therefore a focus is made on storage in order to meet the energy demand. In the simplified example, there is among others a pumped hydro storage (PHS). It can store electricity under the form of potential energy of water when there is too much electricity production and produces electricity when the demand for electricity exceeds the production.
- **Infrastructure**: used to transport energy as the power grid in the example.

Output of EnergyScope TD

ESTD returns three outputs of interest in the context of this work:

- The CO_2 emissions;
- The cost;
- The energy mix.

Indeed, the goal of this master's thesis is to reach a CO_2 emissions target while minimizing the cost. Moreover, the energy mix is used to analyze the impact of the political actions on the energy system.

Chapter 2

Deep Q-Learning overview

In order to act on the energy system presented in CHAPTER 1, the politician has to develop a decision-making mechanism. To do so, we will use a deep Q-learning model, which is particularly suited for decision-making. Once trained, the model provides an optimal series of political decisions in line with the objectives.

Before explaining in detail how this model works, one first needs to understand the basics of RL and how it can be applied to this problem. In addition, we will also look at Q-learning (QL) to be able to fully understand DQL algorithm. As we shall see, QL could theoretically have been applied to our problem. After presenting the two techniques, the reasons for using DQL rather than QL will be discussed.

2.1 Bases of Reinforcement Learning

The basic principle of reinforcement learning is illustrated in FIGURE 2.1. An agent operating in an **environment** which is initially unfamiliar with it. At any time t, the agent is in a defined **state** s_t and can take a series of defined **actions** a_t. When the agent performs an action a_t, it lands in a new state s_{t+1} and receives by the environment a measure of the quality of the action (or series of actions) it has just taken. This is called **reward** $R_{t+1} = R(s_t, a_t)$. The objective of the agent is to find, whatever its initial state, the series of actions that will maximize the reward received.

Thus, reinforcement learning requires defining an environment for the agent, a set of states it can reach, a series of actions it can undertake, and a mean of making the agent understand the value of the actions it undertakes, i.e. a reward function.

The following subsections explain the implementation choices regarding actions,

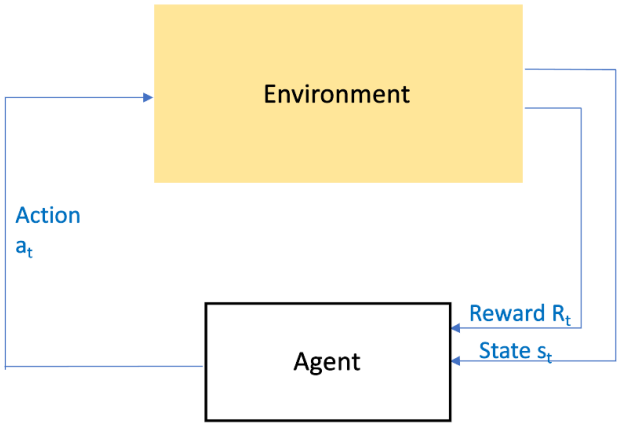


Figure 2.1: The feedback loop performed by a simple RL algorithm.

states, rewards and environment of our model. The objective is to explain the intuition behind these choices. A more practical explanation of these specifications will be made in CHAPTER 4 after a detailed explanation of the DQL model.

2.1.1 Actions

Given that the objective of the agent is to find a sequence of political actions that would lead a whole-energy system achieving its greenhouse gas emission reduction targets, the agent would be able to play on several factors:

- **The resources price**: this action is equivalent to the introduction of a tax or incentive on the resource;
- **The maximum resources capacity**: this action is equivalent to change the maximum quantity of resources that the energy system can use;
- **The technologies cost**: this action is equivalent to the introduction of a tax or incentive on the use of certain technologies;
- **The maximum technologies capacity**: this action is equivalent to changing the limit on the use of certain technologies.

However, given the multitude of resources and technologies that ESTD represents, it is not conceivable to train the agent on so many different actions. Indeed, each new action added to the possibilities of the agent will lengthen the training time, as it will be explained later in this thesis. It is therefore necessary to limit the number of different actions that the agent can perform in order to keep the

training time within an acceptable range (from a few hours to a few days). For example, it will not be possible for the agent to put incentives on the import of highly polluting resources such as coal. It should also be noted that in the context of a reinforcement learning algorithm such as ours, the action space must be discrete. Therefore, the possible actions will undergo a discretization in Chapter 4.

In addition, limits will be set on the possible actions of the agent in order to stay in a plausible scenario. For example, it will not be possible for the agent to lower the price of a certain resource or technology indefinitely in order to make it free. Nor will it be possible for the agent to change the price of certain resources, such as wind, which is free and would be materially impossible to tax or to incentivize (unlike wind and solar technologies which can be, using incentives like green certificates). Nor will it be possible for the agent to increase the maximum installed capacity of certain technologies higher than a threshold specific to the inherent constraints (e.g. it is not possible to cover the entire surface of a country with solar panels). A practical implementation of these limits will be detailed in Section 4.2.

2.1.2 States

The model starts from a reference state s_0 which corresponds to the current situation of the energy system studied in terms of resource/technology prices, maximum capacities, emissions and cost. This state is therefore called the reference state and is denoted by a vector of zeros.

Each element of this vector represents an action lever for the agent. For ease of understanding, let us take the example of the price of natural gas. Depending on the scenario, it is possible for the agent to move each lever up or down i.e. to decrease or increase the price of natural gas. Therefore, if the agent decides to decrease the price of natural gas twice, the cell corresponding to this lever in the state vector will be set to -2. If the agent decides to increase the price of natural gas 3 times, the corresponding cell will be set to 3.

It is important to emphasize the difference between actions and state in our implementation. Indeed, an action represents the fact of increasing or decreasing the cost or the emissions, there can be two different actions for a resource or a technology. While in the state vector, there is only one cell per resource or technology. The action to decrease will be associated to -1 in the state while the action to increase will be associated to +1, as illustrated in Figure 2.2.

The other states will therefore be defined by the set of actions that the agent has carried out since the initial state, i.e. the set of modifications made to the

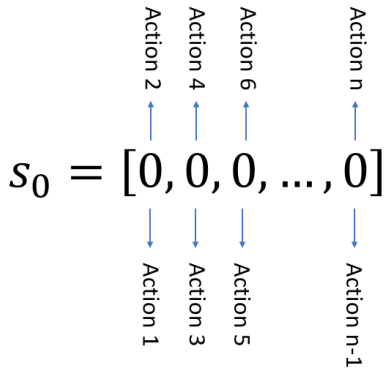


Figure 2.2: Reference state with n possible actions. A single element of the vector represents two different actions on the same resource or technology.

parameters of the model. For example, if from the reference state of FIGURE 2.2 the action number 3 was done twice and action number 6 was done 3 times then the resulting state would be $[0, -2, 3, \cdots, 0]$.

Since ESTD is deterministic, the emissions and cost of a state can be directly deduced from the state vector using ESTD, it would be redundant to include them in this state vector.

2.1.3 Reward

The reward function $R_t = R(s_t, a_t)$ is the feedback that will enable the agent to improve and understand the direction to take in its process of improvement. The agent has a primary and a secondary objective. The main objective is to reduce the greenhouse gas emissions to below the emissions threshold desired after the series of actions taken by the politician. The second objective of our agent is to reduce the annual cost of the system as much as possible, with the condition that the emissions threshold remains the priority.

The reward can be divided into two parts:

- **Transition reward**: reward that the agent will get after each of its actions. Since the climate decisions of public and private actors are governed mainly by money, this reward depends only on the cost of the system. If this cost increases as a result of a chosen action, the reward will be negative. If it decreases, the reward will be positive. This reward is also scaled according to the importance of the increase/decrease of the system cost i.e. the more

the cost decreases, the more the reward is high and vice versa.

- **Final reward**: reward that the agent will get when it has performed the maximum number of actions that it is possible to do (i.e. when it has reached the deadline date of 2050). If the emission reduction target is met, this reward will be positive, otherwise it will be negative. It is a fixed reward whose sign only changes. This reward does not depend on how far the agent is from the emission target. Indeed, from a political point of view, the aim is usually to go below a fixed emission threshold.

Mathematically, the reward R_t can be formulated as the sum of the transition reward R_{tr} plus the final reward R_f: $R_t = R_{tr} + R_f$ with

$$R_{tr} = w_{tr}\left(1 - \frac{Cost_{t+1}}{Cost_t}\right) \tag{2.1}$$

$$R_f = \begin{cases} 0 & \text{if condition (1)} \\ w_f & \text{if condition (2)} \\ -w_f & \text{if condition (3)} \end{cases} \tag{2.2}$$

where the different conditions correspond to:

1. Until the agent has done the maximum number of actions;
2. If the maximum number of actions is met and the emission target is reached;
3. If the maximum number of actions is met and the emission target is not reached.

With w_{tr} and w_f being the weights of the two rewards. The weighting between these two rewards is important because it depends on how much money the politicians are willing to invest to meet the greenhouse gas reduction targets. To illustrate, here is the two extreme cases:

- In a case where only money interests the political agent, the final reward is set to 0 ($w_f = 0$). The agent does not care about reaching the emissions target. This implies that only the transition reward will have an effect on the decisions of the agent. The agent will therefore obtain the final state in which the system is the least costly, regardless of the emissions.
- In a case where the political agent is only interested in meeting the emissions reduction target, the transition reward is set to 0 ($w_{tr} = 0$). The agent does not care about the cost as long as it reduces the emissions. The agent will therefore obtain a final state in which the system emits less than the emission target, regardless of the cost.

The reality lies between these two extremes. The choice made for the values of w_{tr} and w_f thus indirectly induces a maximum budget that the agent can spend during his actions.

2.1.4 Environment

Finally, the environment is the process that, when given a starting state and an action, will return the next state and the reward obtained by that action. For example, in the context of the scenario presented in FIGURE 2.2, if the agent is in the state $[0, 2, 0, \cdots, 0]$ and performs action number 3, then the environment will return the state $[0, 1, 0, \cdots, 0]$ and a reward according to the reward function seen in the previous section.

To be able to do reinforcement learning, the environment in which the agent evolves must take the form of a Markov Decision Process. It is equivalent to have each state satisfying the Markov property. This property can be summarized as follows: when the agent is in a state s_t, its probability to arrive in any other state s_{t+1} does not depend on the states it has passed through or the actions it has taken to reach state s_t, i.e. does not depend on previous states $s_0, s_1, \cdots, s_{t-1}$. Since ESTD is deterministic, this property is respected.

2.2 Q-Learning

Since the algorithms behind QL and DQL have a common base (as you will find out in SECTION 2.3), understanding DQL requires firstly an explanation of QL.

The idea behind Q-learning is to map each state-action pair (s_t, a_t) to a Q-value $Q(s_t, a_t)$. This value corresponds to the expected reward if the agent decides to take action a_t when it is in state s_t. Once trained, the agent will always choose the action with the best Q-value in the state it is in. The Q-learning algorithm is illustrated in FIGURE 2.3 and is divided into three steps, with the last two steps being executed in a loop until the learning process is completed. These steps are explained in detail below.

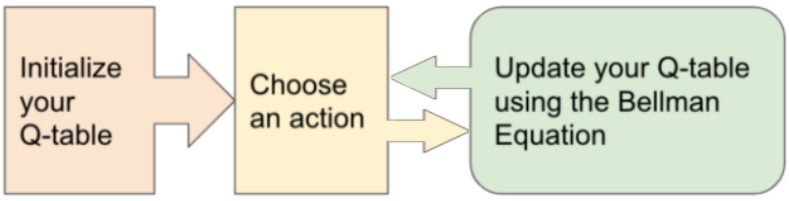


Figure 2.3: Q-learning algorithm [10].

Initialization of the Q-table

The **Q-table** is the table containing the Q-vales of each state-action pair. At initialization, it is filled with zeros. This represents the fact that the agent does not know anything about the environment at the beginning. It means that there is no action that is better than the others.

Epsilon-Greedy Exploration Strategy

During the learning process, it is essential that the agent explores the space of possible states so that each of the reachable states is taken into account in the calculation of the Q-table. To do this, the agent will start by taking only random actions and will gradually focus on the actions that have the highest expected reward i.e. the highest Q-value. This process is called epsilon-greedy strategy.

Each time the agent needs to undertake an action, a random float $x \in [0, 1]$ is drawn and compared to a threshold value $\epsilon \in [0, 1]$ that will evolve during training:

- If $x < \epsilon$, a random action is taken. The agent explores;
- If $x \geq \epsilon$, the best-know action is taken, the one with the highest Q-value. The agent exploits.

At the beginning, ϵ is set to 1 and will slowly decrease following an inverted exponential, as shown in FIGURE 2.4.

This strategy enables a trade-off between exploration and exploitation. As the training progresses, the value of ϵ decreases to replace exploration by exploitation. The formula used to update ϵ at each step s is the following:

$$\epsilon_s = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min})e^{-\beta s} \ ,$$

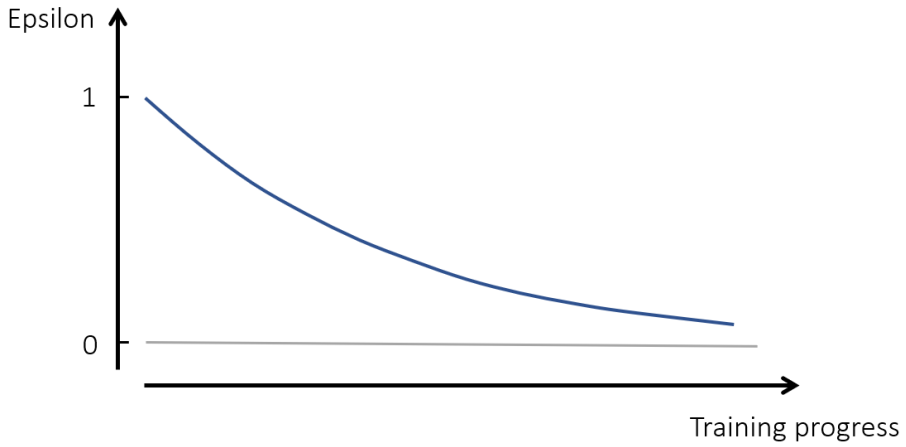


Figure 2.4: Epsilon decreases exponentially during the learning process.

where β being the decay factor. It influences the rate at which epsilon decreases. The larger the decay factor, the faster ϵ will decrease, and vice versa. This hyper-parameter is case-specific and its effect on the convergence of our algorithm will be analyzed in CHAPTER 5.

The iterative process to convergence

The process of learning the Q-table is an iterative process which, starting from the initially all-zeros Q-table, will gradually converge towards stable Q-values. This process is based on the Bellman equation [10]:

$$\begin{aligned} Q_{new}(s_t, a_t) = (1-\alpha)Q(s_t, a_t) \\ + \alpha \left[R_t(s_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \right] . \end{aligned} \tag{2.3}$$

The parameters of this equation are as follows:

- $\alpha \in [0, 1]$ is the learning rate. This parameter influences the part of the new Q-value $Q_{new}(s_t, a_t)$ that will be directly inherited from the old Q-value $Q(s_t, a_t)$. If α is set to 0 then $Q_{new}(s_t, a_t) = Q(s_t, a_t)$ and the algorithm does not learn. If α is set to 1 then $Q_{new}(s_t, a_t)$ does not depend on $Q(s_t, a_t)$ and is only dependent on the second part of equation (2.3), there is no memory of the past Q-values. In order to have a stable and steady convergence of the Q-values, α is usually set to 0.7 as a rule of good practice[11].

- $\gamma \in [0, 1]$ is the discount factor. As explained by Morales, Eduardo F. et al.:

  - If $\gamma = 0$ the policy is greedy, i.e. it chooses the best action only for the current state.

– If $\gamma > 0$ then future rewards are taken into account. The Q-values of the next states in the chain have their importance fading by a factor γ at each step. With a higher γ the policy is optimized for gains further in time, but takes more time to converge [11].

The chosen value for this parameter is $\gamma = 0.618$ because this value works in a functional DQL model[10].

Intuitively, equation 2.3 can be understood as follows: in state s_t, the new expected reward by taking action a_t is equal to a combination between the current expected reward and the transition reward from state s_t to state s_{t+1} plus the final reward of state s_{t+1} plus the best expected reward in state s_{t+1} (among all a_{t+1} possible actions in this state).

The learning process starts in the reference state s_0. At each step, the agent will choose an action a_t among the possible actions. This choice is made following the epsilon-greedy strategy explained above. Then, the environment returns to the agent the new state s_{t+1} in which it arrives and the transition reward R_t obtained as a result of its action. Finally, the Q-value $Q_{new}(s_t, a_t)$ is updated using the Bellman equation. This loop is expressed graphically in FIGURE 2.5.

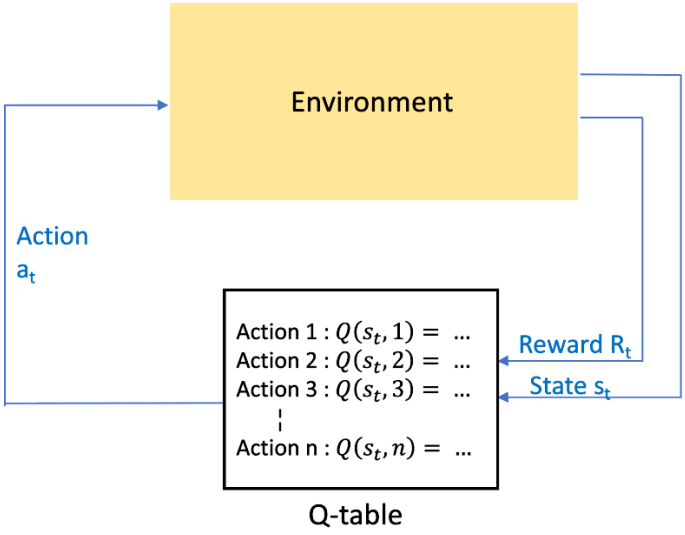


Figure 2.5: The feedback loop of a Q-learning algorithm.

When the agent reaches the maximum number of actions it is allowed to take in a row, the process starts again from the reference state s_0. After a certain amount

of time (depending on the size of the Q-table, the learning rate α, the discount factor γ and the decay factor β), the Q-values will not change anymore. That means that the Q-table has converged, the model is trained and ready to be used. In order to find the optimal pathway (i.e. the one that maximizes the reward), the best-known action must be taken in each state.

However, Q-learning uses a table of size [number of actions $\times$ number of states] to store the Q-values. Indeed, this table stores a Q-value for each state-action pair. The memory required for this table then grows $\mathcal{O}$(number of actions $\times$ number of states) and the time required for training grows Ω(number of actions $\times$ number of states). In the context of a complex environment with a lot of states and actions, the time needed to train such a model can exceed several days and take up a lot of memory space.

2.3 Deep Q-Learning

Deep Q-learning overcomes the time and memory issue of Q-learning by replacing the Q-table with a deep neural network, as illustrated in FIGURE 2.6. In this section, we will dive into the working of deep Q-learning and how a DQL model works.

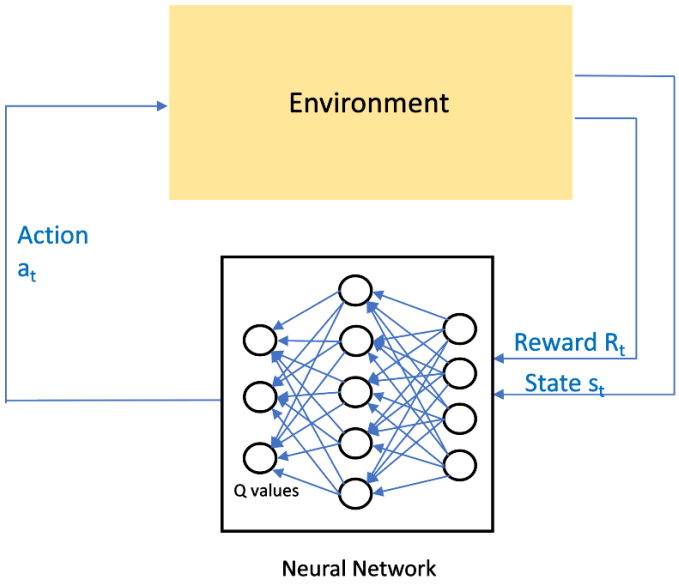


Figure 2.6: The feedback loop of a deep Q-learning algorithm.

2.3.1 Quick reminder on deep neural networks

A neural network is inspired by the neuron network present in our brain. Here is a quick description of all its components:

- **Neurons:** The neurons are organized in layers with the signal going from one layer to the next. The first and last layers are called respectively input and output layers. Between these two layers, they are one or more hidden layers. A neural network is called a deep neural network as soon as it has more than one hidden layer.

- **Connections:** A connection is the link between the output of a neuron to the input of another. A weight is associated with every connection. The weights are the key variables that evolve during the learning process of the neural network.

- **Activation function:** The output of a neuron is computed as a weighted sum of all its inputs passed throughout an activation function: $N_{out} = f(w_0 + \sum_{i=1}^{n} w_i x_i)$ where x_i are the outputs of the neurons of the preceding layer and w_i are the weights associated with the connections between these neurons and the neuron N. Note that the activation function $f(x)$ is usually non-linear and/or discontinuous, which allows the neural network to learn non-linear relations between input and output. The path of a signal going through a neural network layer is synthesized in FIGURE 2.7. Our model uses the ReLU (Rectified Linear Unit) activation function also known as the ramp function. It is the most common activation function used for the hidden layers [12]. It allows to output directly the input if it is positive and returns zero otherwise. Mathematically, it can be seen as:

$$f(x) = \begin{cases} 0 & \text{if x} \leq 0 \\ x & \text{otherwise.} \end{cases} \tag{2.4}$$

  This function is illustrated in FIGURE 2.7.

Input and output vectors

When using a Q-table, to get the Q-value, it is necessary to access the cell of the table corresponding to the pair (state, action). Unlike deep neural networks that only take as input the state of the agent and give as output an estimate of the Q-value of each of the actions (see FIGURE 2.8).

The data required by the input layer is the state of the agent, the number of neurons in this layer is thus equal to the length of the state vector. As for the

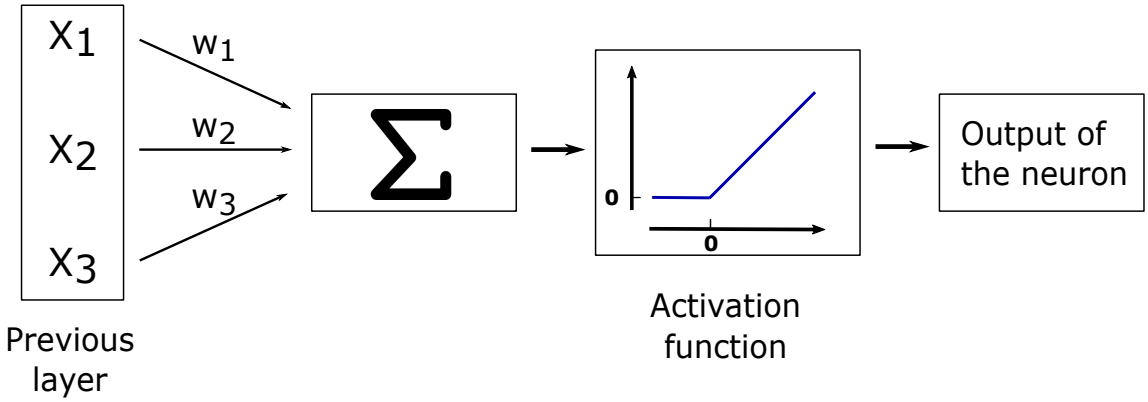


Figure 2.7: From previous layer to the output of a neuron.

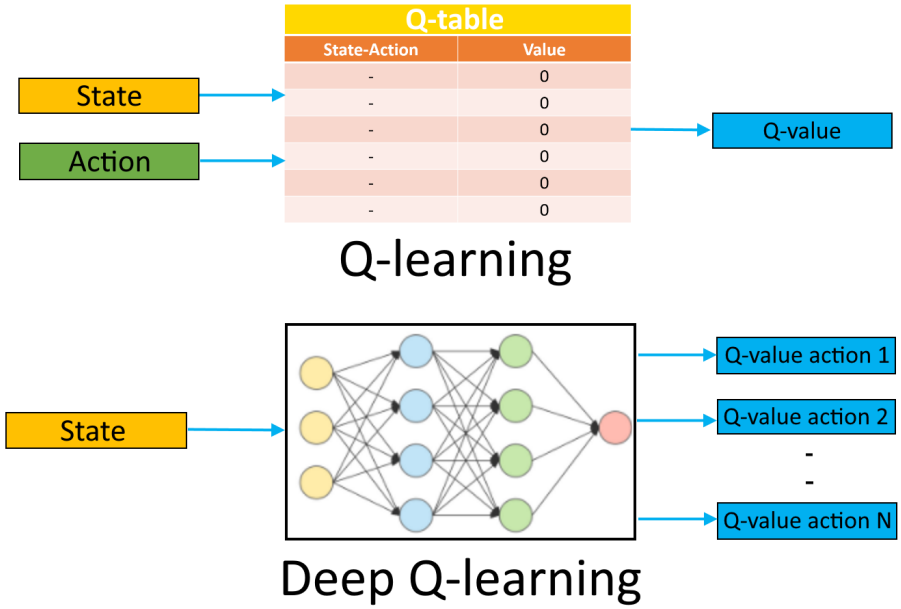


Figure 2.8: The neural network estimates the Q-values of all possible actions in the state [13].

output layer, there is a neuron for each of the possible action since the output of these neurons is an estimate of the Q-value associated with the respective action.

2.3.2 Training the model

The process of training a deep Q-learning model is synthesized in FIGURE 2.9.

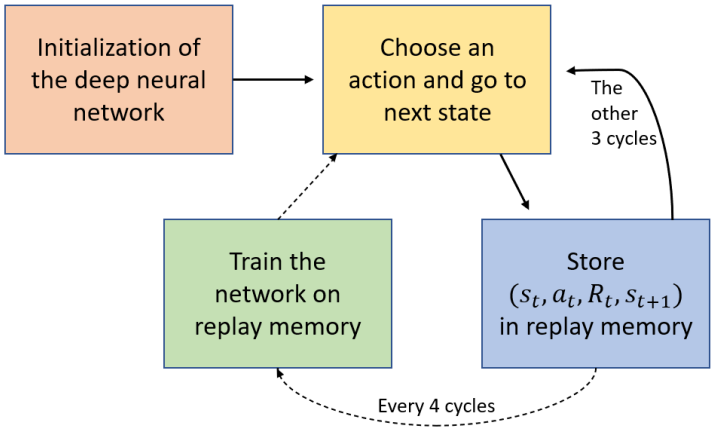


Figure 2.9: Deep Q-learning algorithm.

First, the neural network is initialized with random weights. Then, a loop is started until the end of the training. This loop consists of two main steps and a third step that only takes place one cycle out of four. As you will see later, this frequency is the result of trial and error in order to find a good trade-off between a sufficient training of the neural network and a training time that is not too high:

- The agent chooses an action by following the epsilon greedy strategy explained earlier in SECTION 2.2. Once the action is performed, the agent arrives in the next state s_{t+1} and receives a reward R_t.
- The starting state s_t, the performed action a_t, the ending state s_{t+1} and the obtained reward R_t are called an experience and are stored in a list called **replay memory**. This list has a maximum length of 10,000 experiences and works on a First In First Out (FIFO) basis. The maximum size of the replay memory should be large enough to allow the training process to take place on sufficiently diverse experiments. However, it should be completely renewed several times during the training and should not use too much memory. The maximum length of 10,000 fits to these requirements.
- As soon as the replay memory contains more than 1000 experiences, a training step takes place every four cycles. This threshold of 1000 is set arbitrarily and ensures that training is carried out on a sufficient number of experiences

to avoid overfitting at the beginning of the learning process. During this step, the neural network is trained on the experiences stored in the replay memory. This training step is called **experience replay**. As it is a complex process, it deserves a section of its own and is therefore dealt with in the following section.

At the end of this learning process, the neural network is trained to best estimate the Q-values of the actions for each state it is in.

2.3.3 Experience replay

First introduced by Google's Deep Mind AlphaGo [14], experience replay is a technique that is now used in most DQL implementations. This technique allows better use of past experiences by learning them multiple times. This is key when gaining real-world experience is costly, as in our case where ESTD has to be run for each experience.

As the functioning of experience replay is complex, it will be explained step by step following a graphical illustration available in FIGURE 2.10. As a reminder, this process takes place every four cycles from the moment the replay memory contains more than 1000 experiences. The numbers in the figure correspond to the numbers in the following list:

1. A subset of 128 experiences called **batch** is drawn at random from the experiences in the replay memory. As explained by Goodfellow I., Bengio Y. et al., it is good practice to take a batch size that is a power of 2. Indeed, this batch size allows optimal performance on some GPUs and is therefore used as standard. Apart from this rule of good practice, the larger the batch size the better, provided that the batch fits into your GPU memory. Typical batch sizes range from 32 to 128, we then chose a size of 128 [15].

2. For each of these experiences:

   (a) The neural network is used to estimate the Q-values of the starting state $Q_{values}(s_t)$;

   (b) The neural network is used to estimate the Q-values of the ending state $Q_{values}(s_{t+1})$;

   (c) The Bellman equation is used to derive the new Q-value associated with

taking action a_t in state s_t:

$$Q_{new}(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha \left[R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \right] . \tag{2.5}$$

Intuitively, this equation can be understood as follows: In state s_t, the new estimated reward by taking action a_t is equal to a mix between the current estimated reward and the reward from state s_t to state s_{t+1} plus the best estimated reward in state s_{t+1} (among all a_{t+1} possible actions in this state;

(d) The old Q-value for $Q(s_t, a_t)$ is replaced by the new in the vector of all the Q-values $Q_{values}(s_t)$.

3. All 128 states s_t are combined in an array of size [length of state vector $\times$ 128].

4. All updated Q-values vectors $Q_{values}(s_t)$ are combined in an array of size [length of action space $\times$ 128].

5. The neural network is trained in parallel on the 128 experiences. This training allows the neural network to integrate the logic behind this change in Q-values in order to have a better estimation of the Q-values in the future (of any state s_t, not only the 128 selected here). It should be noted that since the weights of the neural network are not reset before this training step, the previous trainings of the neural network are not forgotten. This training is done on top of the previous ones in order to make the neural network more accurate in its Q-value estimation.

For an extensive understanding of the DQL algorithm, a pseudo-code is available in APPENDIX D.

2.4 Comparison between Q-Learning and Deep Q-Learning

After discussing Q-learning and deep Q-learning, a comparison can be made to justify the interest of selecting DQL in this master's thesis. The main advantage of this technique over QL is that it allows to learn a more complex environment because it is not limited by the size of a Q-table.

Furthermore, QL requires the Bellman equation to be run several times on each of the cells of the Q-table in order to converge, whereas DQL does not even require the exploration of all the states. Indeed, the neural network that replaces the Q-table is a good generalization tool. It allows to give an approximation of the Q-values of an unknown state by generalizing from the Q-values of similar states.

Moreover, while QL once converged gives for sure the best possible solution, DQL will usually give a solution close to the best but not exactly that. This is because the deep neural network learns to best estimate the Q-values of the entire set of states, and may therefore be less accurate on rarer states or edge cases, of which the optimal solution is usually one.

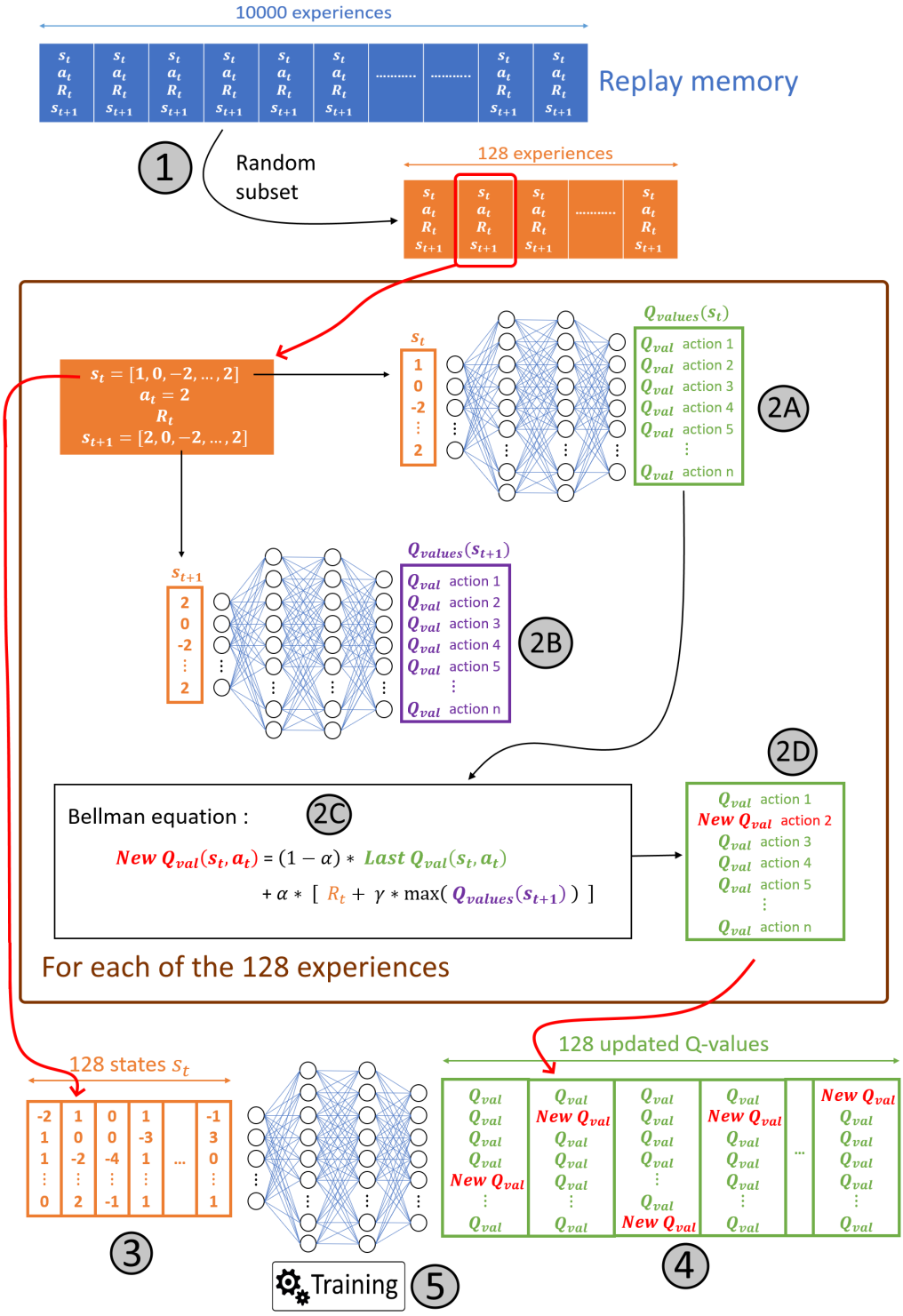


Figure 2.10: Illustration of the training of the neural network using the replay memory.

Chapter 3

Convergence criterion

An important aspect of the training of the neural network is its duration. Checking when the neural network is sufficiently trained will allow us to avoid too long or too short training time. To do so, a convergence criteria has to be set.

Two different methods were investigated to define an appropriate convergence criterion for our neural network. The first one is based on the weights and the second on the reward. In order to analyze these two criteria, both methods were tested on the same simulation, i.e. the same neural network architecture, actions list, training parameters,... The goal of this section is to focus on the convergence of the neural network, neither on the simulated scenario nor on the analysis of the corresponding results. This part will be done in the next chapters.

3.1 Convergence based on the weights

The most used criterion to verify the convergence of a neural network is the weights stabilization. Indeed, if the weights stop varying, it means that the neural network keeps providing the same solution.

Figure 3.1 shows the evolution of the weights between the two hidden layers during the training process. A step represents a complete series of actions that a politician can take to reach the emission target by 2050. At the end of the training, a part of the weights stabilizes but some of them do not stop varying. And as it is explained in Section 2.3, the output of a neuron is based on the weighted sum of all its inputs. That is why, even if the model is efficient after about 5000 steps as you will see in Section 3.2, the output of some neurons will continue to vary since a part of the weights do not stabilize.

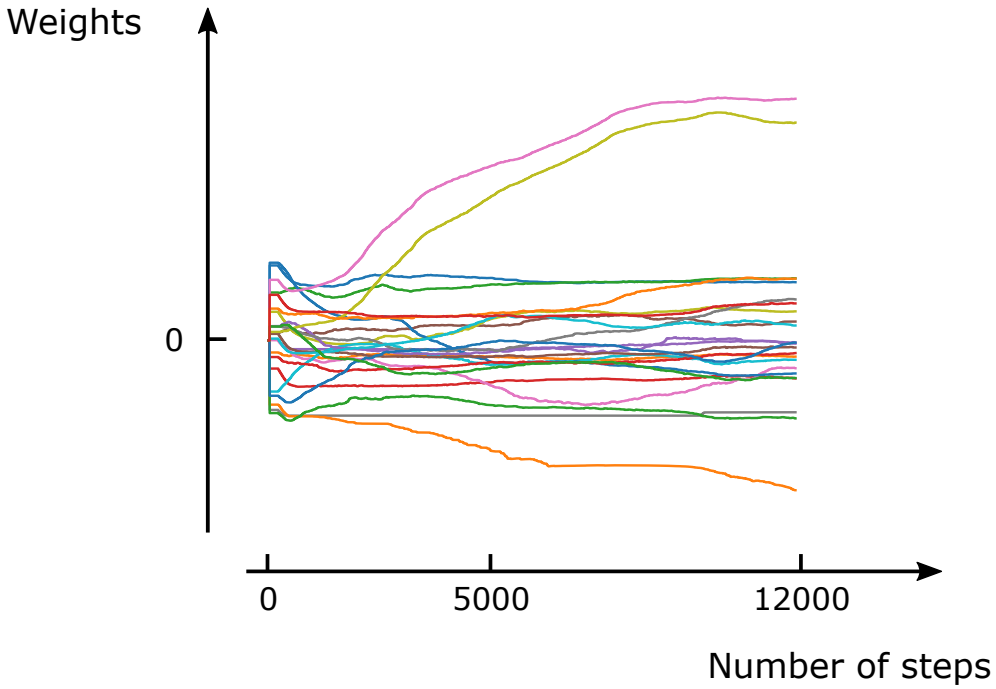


Figure 3.1: Some of the weights do not stop varying during the learning process.

The fact that the weights of our neural network keep varying can be explained through the use of experience replay. As explained in Section 2.3, the replay memory contains 10,000 experiences which corresponds to information related to 10,000 individual taken actions. Concerning the scenario investigated here, 12,000 steps are done which represents a total of 60,000 individual actions since a complete series is composed of five actions. It means that the replay memory will change entirely multiple times during the training. To be precise, since the replay memory works on a First In First Out basis, it will be entirely renewed every 10,000 steps. There will therefore be 6 complete reconstructions. Moreover, the experiences are taken at random in the replay memory. Thus, the training is based on a set of experiences that keep changing during the whole learning process. Which implies that the weights also keep changing.

Since there will always be a part of the weights of the neural network that will vary during the training, this weight-based method cannot be used to determine the convergence of our neural network.

3.2 Convergence based on the cumulative reward

Since analyzing the weights of the neural network is not consistent, another method is to evaluate the neural network continuously during the training. First, it is important to describe what an evaluation of the neural network consists of.

In real life, evaluate the neural network means that a politician takes a complete series of actions and then analyzes the impact on the energy system. Practically, the evaluation of the neural network is illustrated in FIGURE 3.2 and can be broken down into several steps:

1. The reference state s_0 is given as input of the neural network. The **cumulative reward (CR)** is initially set to zero.
2. A loop is set up between the green and the yellow block until the total number of actions that the agent can take is reached i.e. to the number of actions that the politician can take to reach the emissions target. At each step of the loop:
   - The neural network takes as input the current state of the agent and returns the Q-values associated with each possible actions;
   - The agent chooses the action with the highest Q-value and reaches a new state. The reward associated with the action chosen is added to the cumulative reward. This reward is computed thanks to the reward function in equations (2.1) and 2.2).

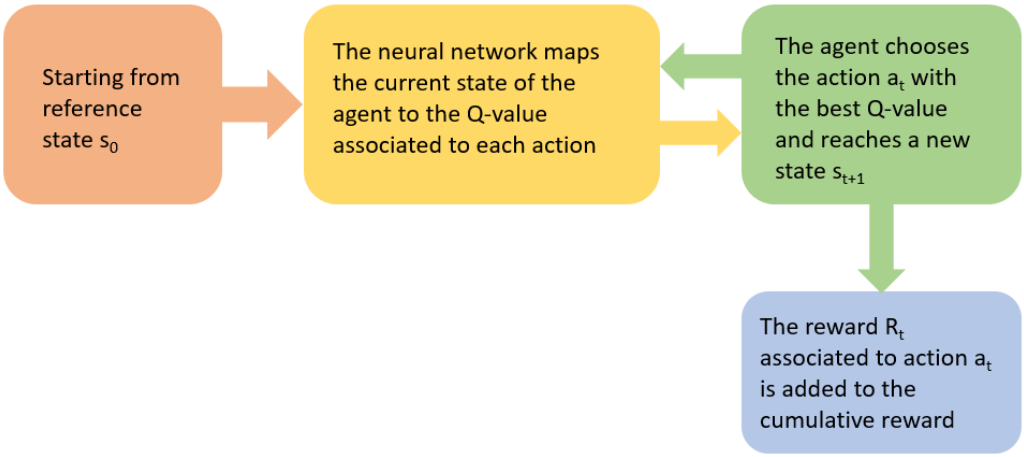


Figure 3.2: Illustration of the evaluation of the neural network.

At the end of the evaluation of the neural network, two results are available:

- The series of actions taken by the agent;
- The cumulative reward, which an indicator of the quality of the series of actions taken by the agent.

The series of actions taken by the agent is not interesting to analyze the convergence of the neural network. On the other hand, the cumulative reward is an indicator of the quality of the result provided by our model and is therefore useful to analyze the convergence. Indeed, from a political point of view, it is not the actions taken that are important but the impact on the emissions and cost of the energy system. This is represented through the cumulative reward. In this master's thesis, we assume that two different series of actions with the same cumulative reward are equally good. On the contrary, two series of actions reaching the same final state but with different cumulative rewards (due to the path taken i.e. the order in which the actions are taken) will not be equivalent.

To analyze the convergence, the neural network will be evaluated continuously during the training to get the progress of the cumulative reward. The result of the scenario investigated is shown in FIGURE 3.3. A moving average was applied to the results in order to make them more readable and therefore easier to analyze.

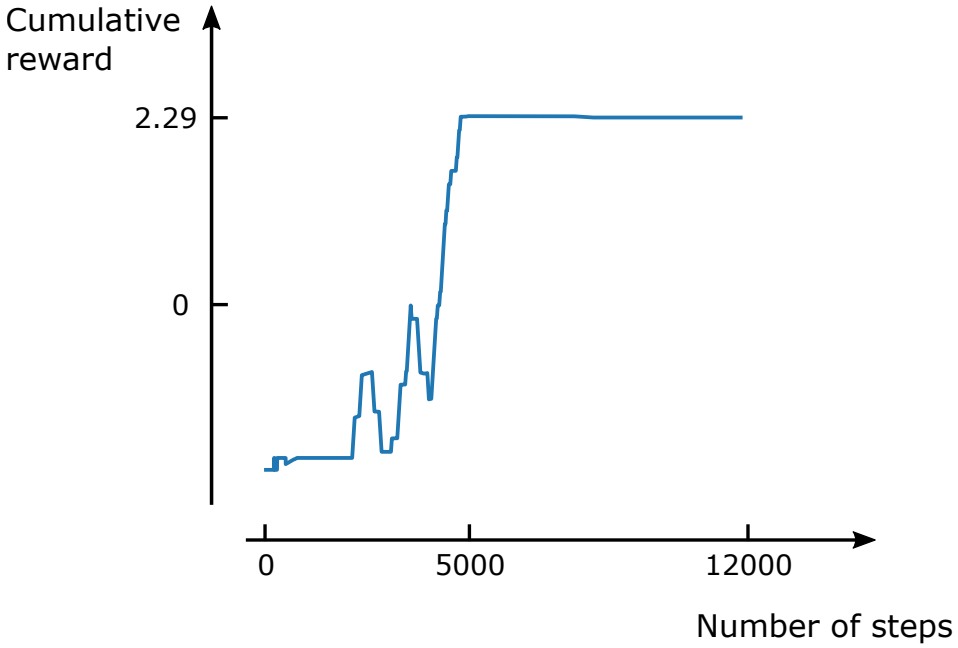


Figure 3.3: The cumulative reward obtained during the training of the neural network converges after about 5000 steps.

The oscillations are due to the fact that the model first explores before focusing on what it has found best. Then, the convergence of the neural network can clearly be observed since the cumulative reward remains constant after about 5000 step until the end of the training.

Even if the model converges towards a solution, it is important to check the quality of this solution. Indeed, if the hyper-parameters of the model are not appropriate, a neural network can converge to a cumulative reward which is much smaller than the one associated with the optimal solution i.e. the path taken by the agent (the series of actions taken by the politician) is not optimal in terms of emissions reduction and cost. The optimal solution can be considered as the series of actions encountered by the agent during the training that leads to the highest cumulative reward (among those encountered). However, the agent does not necessarily consider all possible series of action. This is why the solution we define as optimal may not be the best possible one, but will not be far from it. Indeed, the agent investigates many different paths thanks to the exploration part of the training.

Comparing, through the cumulative reward, the series of actions taken by the agent when the neural network is trained with the optimal solution gives a good idea of the quality of the convergence.

In the scenario investigated, the neural network has converged to a cumulative reward of 2.29 as it can be seen in FIGURE 3.3. The cumulative reward associated with the optimal solution is 2.25. The solution provided by our trained neural network has a cumulative reward 2% higher than the one associated with the optimal solution. This confirms that our neural network is able, from explored experiences, to generalize in order to find solutions that have not been explored.

In this scenario, the solution provided by the neural network is better than the optimal one. If the opposite happens, we need to define what is an acceptable solution in order to consider if our neural network is trained. A solution will be considered acceptable if the cumulative reward provided by our trained neural network CR_{model} is at least 80% of the cumulative reward associated with the optimal solution $\mathrm{CR}_{op,sol}$. However, it is possible that the neural network provides a high cumulative reward for only a short time before dropping due to the exploration. There are therefore two criteria for a solution to be considered acceptable:

- $\mathrm{CR}_{model} \geq 0.8 * \mathrm{CR}_{op,sol}$;
- This first criterion must be respected for a sufficient period of time. A human intervention is required to determine the length of this period.

In conclusion, the analysis of the convergence of the neural network will be systematically based on the cumulative reward and not on the convergence of its weights. This allows us to have a good indicator of the quality of the series of actions taken by our agent.

Chapter 4

Practical implementation of the model

Having reviewed the theory behind our model and how its convergence is evaluated, we are now ready to apply it on a practical case. The first thing to do was to determine the country-scale energy system on which the model is applied. Then, the actions available for the political agent had to be discretized. Based on this discretization, three scenarios of increasing complexity could be built. Each one has a different list of possible actions that can be performed by the agent. After noticing that experience replay was causing convergence difficulties, we designed a solution to counter this. Finally, the architecture of our neural network and the different training parameters will be presented.

4.1 Energy system exploited by the deep Q-learning algorithm

As it was already mentioned in the introduction, the goal of this master's thesis is to develop a deep reinforcement learning method on a whole-energy system. Obviously, simulations are required to test the different parameters of the model. This is why, it is important to use a **simplified energy system** in order to remain within a reasonable calculation time (from a few hours to a few days).

Since ESTD was a collaborative project with a Swiss engineer, the data provided with the model represent the Swiss energy system in 2011. The goal of this master's thesis is to apply DQL on any country-size whole-energy system and not especially on the Belgium one. Therefore, for the sake of simplicity, we have decided to use the data already provided.

Using the complete model of ESTD requires about 40 seconds to run a simulation. For the most complex scenario investigated (see SECTIONS 4.3 and 4.5), about 180,000 runs has to be done. It would result in about 83 days of simulation only for the ESTD part, it does not include time needed for the training of the neural network (one or two additional days). A solution to reduce the computational time is then to use a simplified model. Since ESTD requires a lot of time to compute and optimize the storage part of the energy system, the storage can be removed from the complete model of the Swiss energy system. Using this simplified model, ESTD requires only about seven or eight seconds to run. The computation time of the simplified ESTD is reduced by a factor five compared to the complete model.

In terms of calculation time, the simplified model is therefore more interesting than the complete one. However, it is important to check that removing the storage from the energy system does not significantly impact the cost and emissions. TABLE 4.1 shows the comparison between the two models and the two sankey diagrams can be found in APPENDIX A.

	Complete model	Simplified model	Variation
GWP [$MtCO_2$/y]	22.47	28.89	28.5 %
Cost [b€/y]	14.33	18.55	29.5 %
Time [s]	≈ 45	$\approx 7 - 8$	-500 %

Table 4.1: Given the need to reduce the calculation time of ESTD, we considered that a 30% loss of accuracy was worth a five-fold reduction in calculation time. That is why, the simplified model will be used by our DQL model.

Another method used to reduce the computational time of about two or three seconds per run is to store only the interesting outputs of ESTD for the model (cost and CO_2 emissions).

However, despite the use of a simplified model, the computation time needed to train the model remains of about 16 days. In order to overcome this problem, we set up a **cache system**. Its principle is to save the ESTD results on the first encounter with a state and then reuse this save rather than running again ESTD. Indeed, two ESTD runs on the same state give the same result because ESTD is deterministic. Moreover, as the learning process starts again regularly from the reference state s_0, states close to s_0 are encountered several hundred times throughout the process.

Thanks to this trick, when a state is already saved in the cache file, the time required to obtain the ESTD result is now less than one millisecond. Indeed, the

cache files created in the scenarios we have considered are less than one MB in size, so access to the data can be done almost instantaneously. The total training time of the model is thus reduced to 1-2 days, with the majority of this time being devoted to the DQL part and not the ESTD execution. A discussion of the feasibility of a cache file in a much more complex scenario will be conducted in the discussion of this thesis.

Based on the current emissions of the simplified ESTD model we will use, we had to impose a reduction target. Indeed, this limit represents the commitments that the politician is making. It should therefore not be too low to remain reachable, but not too high either so that a certain number of actions must be taken to reach this target. An appropriate value is 28.50 [$MtCO_2/y$], as you will see in CHAPTER 6.

4.2 Discretization of the actions

As a reminder of SECTION 2.1.1, the agent can change the cost and the capacity of resources and technologies. These have an initial value given by the ESTD data. This value will be called the reference value. When an action is taken, the variation (positive or negative) in cost or capacity is always a fraction of this reference value. As explained in SECTION 2.1.1, an action cannot be performed as many times as desired, there are physical limits.

We made the choice to fix for each action the variation in cost or capacity to 10% of the reference value and each action can be performed a maximum of five times i.e. an increase or decrease of 50% from the reference value at maximum. The value of 10% represents a good compromise between real life feasibility and a sufficient increase/decrease to be felt on the results of ESTD. Regarding the limit of use of each action, we first had to fix the total number of actions that the agent will perform in order to reach the emission target. This value is set to 5 actions in a row for scenario 1 and 10 actions in a row for scenarios 2 and 3. This makes it possible to envisage scenarios of variable complexity and simulation time. If we allow the agent to use the same action ten times, it may exploit only one action. On the contrary, if the number of times each action can be used is too low, the agent may be forced to take one of the least efficient actions in terms of emissions and cost reduction. The maximum number of uses of each action therefore allows the agent to play with this trade-off.

The total number of actions the agent will be able to take as well as the limit of use per action will have a great impact on the training time of the model. Indeed, if

the number of different actions and the maximum number of uses of these increases, the number of different series of actions that the agent can take will also increase. The neural network will therefore need a longer training time.

4.3 Scenarios investigated

Three different scenarios were investigated, they are numbered from one to three in increasing order of complexity. The more complex a scenario is, the more different actions the agent can take. TABLE 4.2 groups the different scenarios with the possible actions to be performed by the agent.

It is important to remember the difference between actions and state in our implementation as explained in SECTION 2.1.2. In the state vector, there is only one element per resource or technology while there can be two actions per resource or technology (increase or decrease). In TABLE 4.2, the writing with the square brackets ([-5,5] for example) does not represent a state vector. It is used to illustrate the range of use of an action (increase or decrease at most 5 times and thus decrease or increase the reference value of 50%).

	Wood capacity	NG price	Diesel price	Solar capacity	Hydro capacity
Scenario 1	$[-5, 5]$	$[-5, 5]$	/	/	/
Scenario 2	$[-5, 5]$	$[-5, 5]$	$[0, 5]$	/	/
Scenario 3	$[0, 5]$	$[-5, 5]$	$[0, 5]$	$[0, 5]$	$[0, 5]$

Table 4.2: Possible actions to be taken by the agent for each scenario investigated.

In scenario 1 for example, the agent will be able to act on the wood capacity and on the price of natural gas. This means that the state will be represented by a vector composed of two elements but there will be a total of four different actions (increase and decrease the wood capacity and increase or decrease the price of natural gas). Scenario 3 is more complex, indeed, the agent will be able to act on the wood, solar and hydro capacity but also on the price of natural gas and diesel. Except for the natural gas price, the agent will only be able to perform actions in order to increase the price or the capacity. It means that there is a total of 6 different actions and the state is represented by a vector composed of 5 elements.

Choices had to be made to decide which actions the agent could take. Indeed, increasing the number of possible actions increases significantly the computation time. To decide on which resource or technology the agent can act, the sankey diagram (see APPENDIX A) as well as the primary energy mix that can be seen in FIGURE 4.1 are used. The exact amount of resources used by the energy system

can also be found in Appendix A. They allow to know the resources and technologies used in the energy mix and in which quantity. To reduce the complexity and therefore the computation time, it is necessary to think in a goal of energy transition to decarbonize the energy system. Reducing the capacity of renewable energies is in opposition to the energy transition, as well as the reduction of the diesel price since it is a resource with high global warming potential. The natural gas is a non-renewable resource but with a low global warming potential. That is why, for an energy transition, we can assume that it will be first used to replace other non-renewable resources with high global warming potential. Then it will be replaced by renewable energies. That justifies the fact that the agent can always increase or decrease the price of natural gas. However, we decided not to rely exclusively on these assumptions. Therefore, we also used our model to check these behaviors before deciding the list of actions of the agent. Indeed, if an action is never chosen by the agent, it means that it represents less interest than the others.

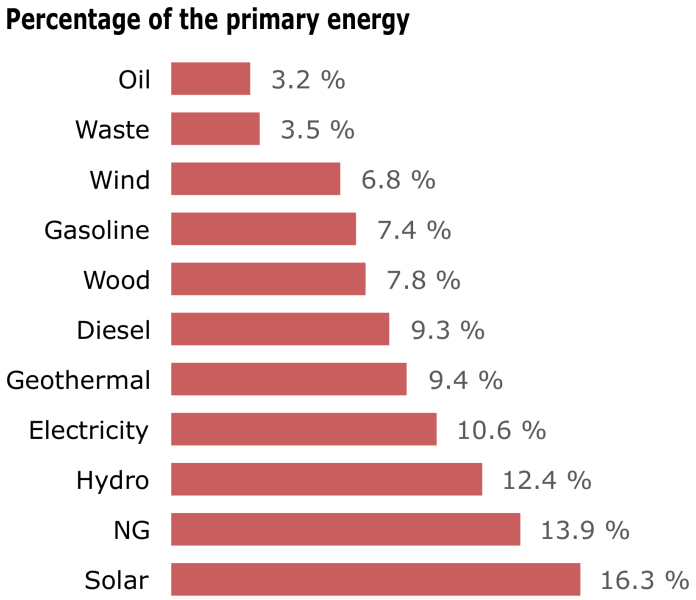


Figure 4.1: Renewable energies already have an important role in the reference energy mix.

4.4 Improvement of experience replay

During the validation of our model on the first scenario, we encountered a convergence problem linked to the functioning of experience replay. Indeed, the number

of different paths that the agent can take is very high compared to the number of near-optimal paths (the ones we want to obtain). Because of this, experiences that are part of near-optimal paths are only minimally represented in the replay memory.

As explained in SECTION 2.3 , the neural network is trained to best estimate the Q-values $Q(s_t, a_t)$ of each possible action a_t when the agent is in a state s_t. The replay memory on which the neural network is trained contains mostly experiences that are not part of optimal paths, these being a minority. Consequently, the neural network is not efficient at predicting the Q-values of these states, which are more an exception than the norm.

In order to compensate for this lack of representativeness within the replay memory, we decided to no longer draw fully at random the 128 experiences on which the training takes place. These 128 experiences are instead chosen as follows:

- 20 experiences are chosen from the top 10% of experiences with the best reward;
- 108 experiences are still taken at random.

More visually, this change is equivalent to replacing step one in FIGURE 2.10 by FIGURE 4.2.

Thanks to this modification, the experiences that contribute most to the increase in reward are more represented in the training. This allows the neural network to better learn to estimate the Q-values of these states.

In FIGURE 4.3 you can see the difference in convergence between the use of a classic experience replay (see SECTION 2.3) and the use of the improved experience replay. These two simulations were done on the first scenario using the same hyper-parameters. The model using the classic replay memory has not converged after 12,000 steps while the model using the improved replay memory has converged after about 5000 steps, which shows the benefit of making this change.

This improvement of experience replay is used for all the results of this master's thesis.

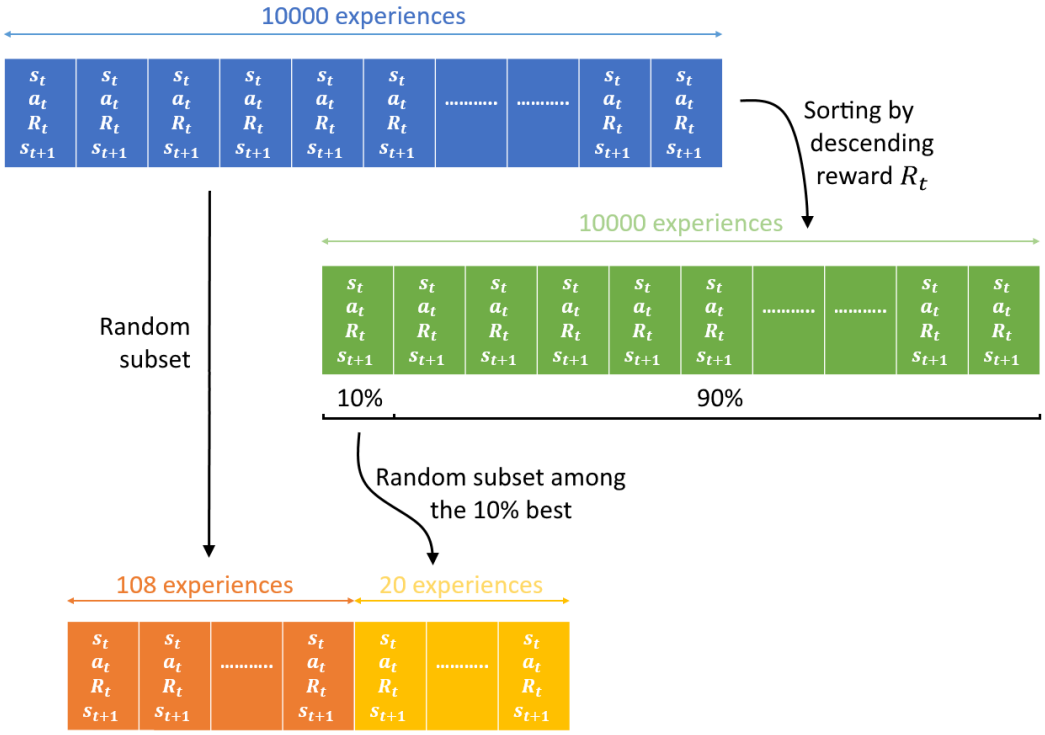


Figure 4.2: The new approach for drawing the 128 experiences on which the training takes place.

4.5 First estimate of the hyper-parameters of the deep Q-learning algorithm

We seek to estimate at best a set of parameters which provides a working model. Since DQL requires a lot of tweaking to determine optimal hyper-parameters, most of the parameter choices are based on good practice rules and trial and error. This chapter aims to give a functional model, which will then undergo an optimization of its hyper-parameters in Chapter 5.

Number of hidden layers

A neural network without hidden layers can only represent linearly separable decisions [16]. That is why, adding one or more hidden layers is essential for our model. The goal of this master's thesis is to develop a deep Q-learning model. It is thus characterized by a neural network with two or more hidden layers in order to learn complex mappings (see Section 2.3). We decided to limit ourselves to two hidden layers, increasing this number induces a longer training time. Moreover, it

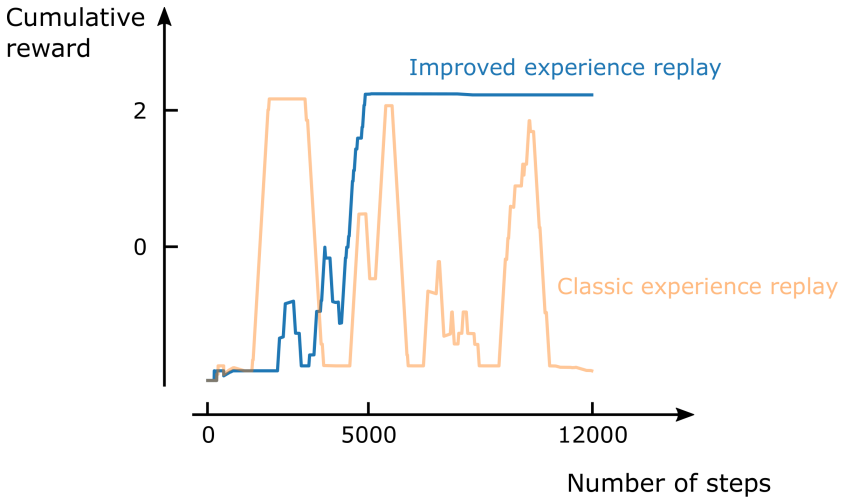


Figure 4.3: The model using the improved experience replay converges after about 5000 steps while the model using the classic experience replay does not converge after 12000 steps.

also increases the risk of overfitting, which means that the model will simply learn by heart the states encountered and their Q-values, it will not be able to generalize to new states.

Number of neurons in the hidden layers

Concerning the number of neurons in the hidden layers, three main rules of thumb can be found in the literature [16]:

- between the number of neurons in the input and output layer;
- about 2/3 of the sum of the neurons contain in the input and output layer;
- less than twice the number of neurons in the input layer.

It is obvious that increasing the number of neurons in the hidden layers will increase the complexity of the training and will therefore require more time. Indeed, the number of links (and thus the number of weights to determine) increases when the number of neurons increases. To understand this, we can take the simple example with a neural network illustrated in FIGURE 4.4. If a fourth neuron is added in the second hidden layer, seven supplementary links are required out of a total of 29. That represents an increase of about 24%.

Moreover, the phenomenon of overfitting can occur if there are too many neurons in these layers [16]. However, if there are not enough neurons in the hidden layers,

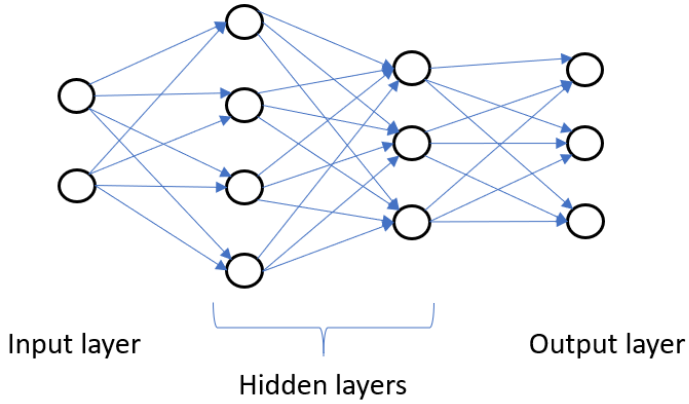


Figure 4.4: Example of neural network.

the opposite phenomenon may occur: underfitting i.e. there are too few neurons to correctly represent the mapping between input and output. The neural network will not be able to represent the complexity of the problem [16].

It is therefore important to find a trade-off that will be appropriate for our model. We noticed, through trial and error, that the performances of our model are better by increasing the number of neurons in the hidden layers compared to the rules of thumb. This will be confirmed in Chapter 5. That is why, we decided to take a reference model with a number of neurons in the hidden layers higher than the good practice rules. The number of neurons present in every layer for each scenario investigated are shown in Table 4.3.

	Input layer	Hidden layer 1	Hidden layer 2	Output layer
Scenario 1	2	8	4	4
Scenario 2	3	10	5	5
Scenario 3	5	14	7	6

Table 4.3: Number of neurons present in each layer for the three scenarios investigated.

Reward function

The shape of the reward function has been presented in Section 2.1.3, the weights of the two functions are still to be determined. To do this, we tested different combinations in order to find one that gave us satisfactory results in terms of

convergence and cumulative reward. The transition and final reward are computed as:

$$R_{tr} = 10\left(1 - \frac{Cost_{t+1}}{Cost_t}\right) \tag{4.1}$$

$$R_f = \begin{cases} 0 & \text{if condition (1)} \\ 2 & \text{if condition (2)} \\ -2 & \text{if condition (3)} , \end{cases} \tag{4.2}$$

where the three conditions are the same as in equation (2.2):

1. Until the agent has done the maximum number of actions;
2. If the maximum number of actions is met and the emissions target is reached;
3. if the maximum number of actions is met and the emissions target is not reached.

Number of neural networks

As explained in Section 2.3, the training of the neural network is based on the estimation of the Q-values $Q(s_t, a_t)$ corresponding to the choice of taking action a_t from state s_t. This estimation is done thanks to the Bellman equation (equation (2.3)). In order to make $Q(s_t, a_t)$ closer to the desired value, the weights of the neural network have to be adapted. If the weights change, it means that the value of $Q(s_{t+1}, a_{t+1})$ (and other Q-values) will also be impacted. However, $Q(s_{t+1}, a_{t+1})$ is used to compute $Q(s_t, a_t)$, this could make the training unstable [17].

A solution to counteract this, is to use during training two neural networks with the exact same architecture. The neural network on which the training is performed as usual is called the main network and the second one is called the target network. The idea behind using a second neural network is that:

- The weights of the main network are updated at each step of the learning.
- The weights of the target network are not updated every step but are copied from the weights of the main network every 10 steps. This value comes from a functional DQL model [10]. The target network is thus never trained, its weights are just synchronized.
- The main network is used to determine which action a_t has to be taken in state s_t. However, it is the target network that is used to evaluate the corresponding Q-values needed for the training of the main network.

This solution will bring more stability in the training and will be used for the training of our neural network. A discussion on the interest of using this target network is done in CHAPTER 5.

Decay factor

As a reminder of SECTION 2.2, the decay factor β influences the trade-off between exploration and exploitation during training through the update of the variable epsilon:

$$\epsilon_s = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min})e^{-\beta s} \ . \tag{4.3}$$

A rule of thumb in reinforcement learning is to always have 10% exploration at the end of the training. To follow this rule, equation (4.3) becomes equation (**??**).

$$\epsilon_{end} = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min})e^{-\beta N_{step}} \ , \tag{4.4}$$

where N_{step} is the total number of steps i.e. the number of series of actions taken by the agent. This number will be determined by trial and error. There is no way to determine this number in advance because it depends on the intrinsic parameters of the neural network, the training parameters and the scenario investigated. Indeed, the more complex a scenario is, the more experiments the neural network will need to be trained.

The method to determine the decay factor is as follows:

1. Determine ϵ_{max} and ϵ_{min}. ϵ_{max} is set to 1 in order to have 100% exploration at the beginning of the training. ϵ_{min} is set to 0.01 since at the end of the training, epsilon ϵ_{end} has to be around 0.1 and N_{step} is not infinite i.e. the second term of equation (4.4) is not equal to zero.

2. Determine for each scenarios the decay factor β and N_{step} by trial and error with the aim of setting ϵ_{end} around 0.1. For a constant ϵ_{end}, an increase in the decay factor β results in a decrease in the number of steps N_{step}, and vice versa. There is no other approach than trial and error to estimate the value of these parameters, as they are specific to our problem and the amount of exploration required for convergence.

The values of the different parameters for the three scenarios investigated are summarized in TABLE 4.4.

As explained in SECTION 4.3, the third scenario is the most complex i.e. it is the one with the most number of different pathways that the agent can take. This is why, after trial and error, we realized that having more than 10% exploration at the end of training improved convergence.

	N_{step}	Decay factor	ϵ_{end}
Scenario 1	12,000	0.0002	0.1
Scenario 2	15,000	0.00015	0.114
Scenario 3	18,000	0.0001	0.174

Table 4.4: For scenarios 1 and 2, there is 10% exploration at the end of the learning process. However, there is still a little more exploration for scenario 3.

4.6 Validation of the model

Now that the practical implementation of our model is complete, it is time to validate it by checking its performance. To do this, we will use the convergence criterion explained in CHAPTER 3. For each scenario investigated, the evolution of the cumulative reward during the training as well as the cumulative reward associated with the optimal solution will be shown. The graphs of scenario 1, 2 and 3 can be found in FIGURE 4.5, 4.6 and 4.7 respectively. As a reminder, a step represents a complete series of actions taken by the agent.

For the first scenario, the same conclusion as for FIGURE 3.3 can be derived since it was the scenario used to find the convergence criterion.

For the second scenario, the cumulative rewards associated with the first 2000 steps have been removed from the graph for the sake of readability. This part of the training is not useful for the validation of the model.

For scenarios 2 and 3, the cumulative rewards of the optimal solution are both equal to 2.52. However, the cumulative reward of the third scenario converges closer to this optimal value. One explanation could be that the parameters of scenario 3 are more adapted.

For the three scenarios, the final reward R_f associated with the last action taken by the agent is equal to 2, it means that the emissions target is reached. The main objective of the agent is achieved.

The model associated with each scenario has therefore been validated because they converge and give acceptable results. In the next chapter, we will vary the different hyper-parameters one by one. By doing so, we will see if there is a way to obtain better convergence speed by refining the first estimates made in this chapter. In CHAPTER 6, we will analyze the real-world consequences on the energy system of the actions taken by the political agent.

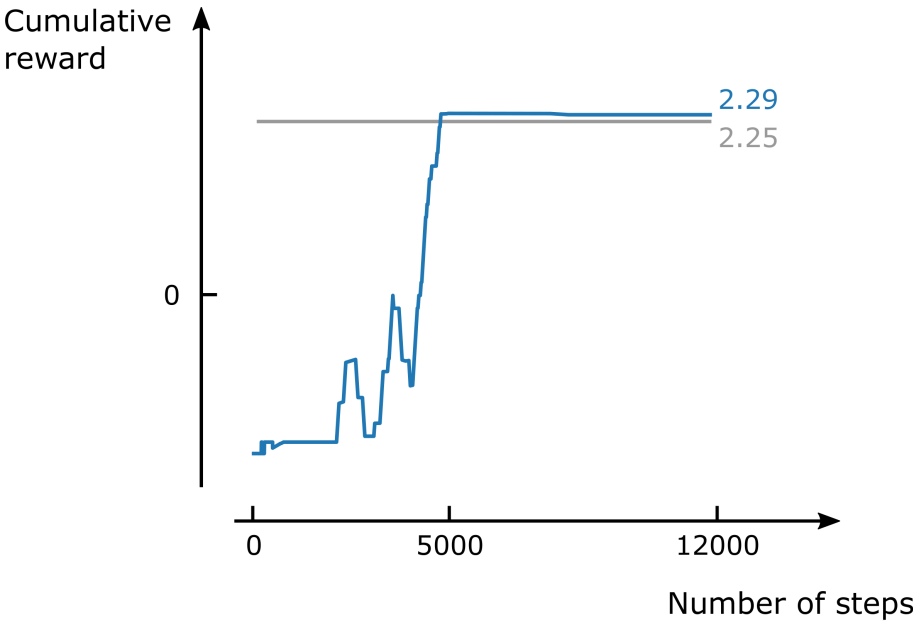


Figure 4.5: The model converges after about 5000 steps towards a cumulative reward better than the one associated with the optimal solution. It means that the neural network is able to generalize from explored experience.

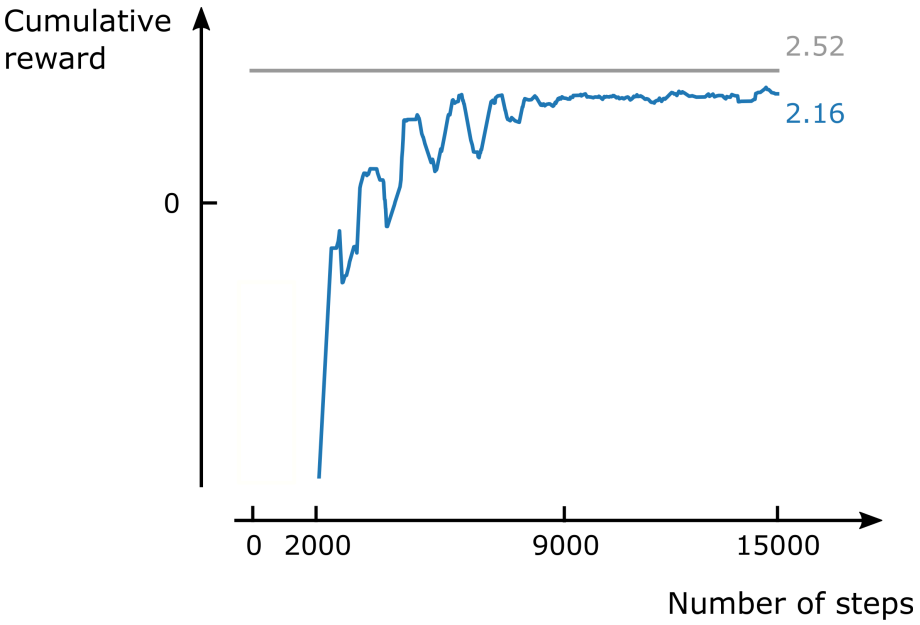


Figure 4.6: The model converges after about 9000 steps towards a cumulative reward of about 85% of the optimal one.

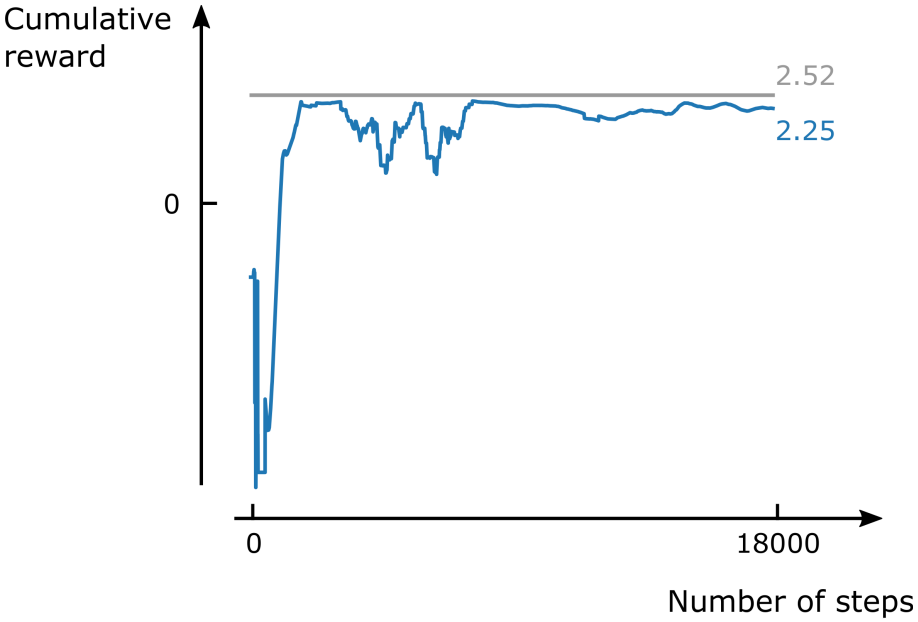


Figure 4.7: The model converges after about 7500 steps towards a cumulative reward of about 90% of the optimal one.

Chapter 5

Tuning of the hyper-parameters of the model

The model that will be used in the rest of this thesis was established in the previous chapter. We can now tune this model by playing on its hyper-parameters. Indeed, these were first estimated in the previous chapter using rules of thumb and trial and error. We can now study the effects of each hyper-parameter on convergence in order to find the optimal ones. The goal of this chapter is to reduce the convergence time of the DQL part of the algorithm while keeping the energy result close to the optimal one. To do so, the convergence criterion seen in Chapter 3 will be used.

Firstly, we will discuss the methodology used to conduct the analysis of the effect of the parameters. Then, as DQL is partly based on randomness through the epsilon greedy strategy explained in Section 2.3, an analysis of the effect of randomness on the speed of convergence of the algorithm will be carried out. This will allow us to distinguish the impact of a parameter variation from the effect of randomness.

Finally, an analysis of the effect on speed of convergence of the following parameters will be done:

- The weights w_{tr} and w_f associated with the transition reward and the final reward function;
- The use, or not, of a second neural networks during the training (the target network);
- The number of neurons in each of the two hidden layers;
- The decay factor β used in the epsilon greedy strategy.

5.1 Methodology of the tuning

Testing each parameter individually from their reference value is not sufficient as it can lead to confounding. Indeed, the effect (synergetic or dysergetic) of some combination of parameters must be taken into account. Experiments varying several parameters at the same time must be conducted.

On the other hand, given the time involved in each simulation and the number of parameters to be varied, it is not possible to test all possible combinations of parameters. This would require $3^4 = 81$ experiments just by limiting ourselves to an increase and a decrease of each of the 4 parameters. Moreover, these tests must be carried out over 3 scenarios (ranging from a few hours to a few days in computation time).

Our method of analysis must find a balance between these two extremes. For this purpose, we have opted for a so-called "greedy" method. In formal terms, a greedy method consists of choosing at each stage the choice giving the best short-term result, in the hope of obtaining a global optimum at the end.

Applied to our case, we tested each parameter one after the other, and when a change of parameter brings a benefit of convergence, this change is retained in the following experiments. It is certain that this method will not offer the best combination of parameters, but it will allow us to get close to it.

5.2 Effect of randomness on the convergence speed

Before the effect of parameters variation on convergence can be examined, an analysis of the randomness factor of the simulations must be carried out. Indeed, when analyzing a result, it is necessary to be able to distinguish between a variation due to a parameter change and a variation due to chance.

One technique for removing the randomness factor from a result is to use the Monte Carlo method. This method applied to our problem would consist of running each of the simulations a large number of times and then averaging the cumulative reward curves. This cancels out the randomness of each individual simulation. Unfortunately, as the calculation time required for each of the simulations can be more than a day, it is not possible to apply this method to each experiment required to tune the model. Indeed, using Monte Carlo with n samples would multiply the total simulation time by a factor n.

In order to estimate the role of chance in the convergence of our simulations, we decided to run the reference case 5 times for each scenario. Such a small sample size does not allow us to use the Monte Carlo method, but it does give us an indicator of the importance of randomness in our simulations. Hence, thanks to this measure, we will be able to distinguish between a parameter whose influence is not sufficient to be distinguished from randomness and a parameter whose influence is sufficiently marked to be considered more than due to randomness.

FIGURES 5.1, 5.2 and 5.3 show the five cumulative reward curves of the non-tuned model for scenarios 1, 2 and 3 respectively.

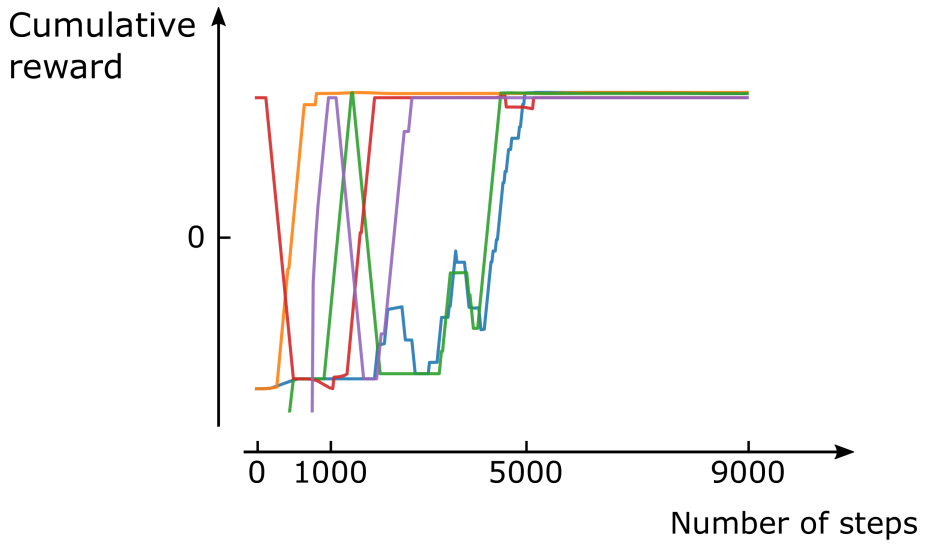


Figure 5.1: For scenario 1, the convergence of the 5 simulations takes place after between 1000 and 5000 steps. The graph has been cut after 9000 steps for clarity reasons.

These figures show that the random nature of DQL plays a significant role in the convergence speed of a simulation. Therefore, the results obtained when the hyper-parameters are varied will be analyzed with respect to this randomness. In order to be able to deduce an increase or decrease of the convergence following a parameter variation, the results will have to be sufficiently marked that they are not mistaken for randomness. For example, for scenario 3, if changing a parameter results in an improvement of 2000 steps in the convergence speed, this change will not be considered conclusive. Indeed, FIGURE 5.3 shows that the effect of randomness is of the order of 4000 steps difference between the fastest and the slowest.

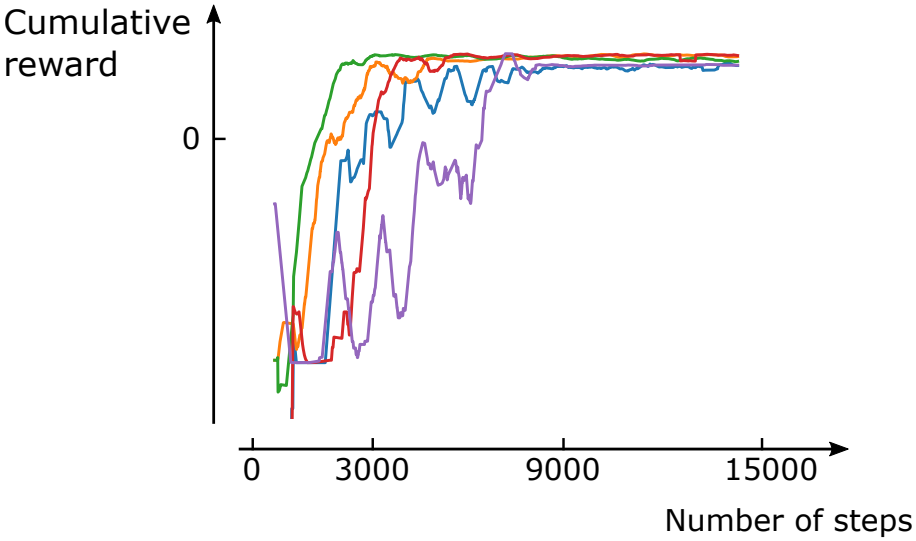


Figure 5.2: For scenario 2, the convergence of the 5 simulations takes place after between 3000 and 9000 steps.

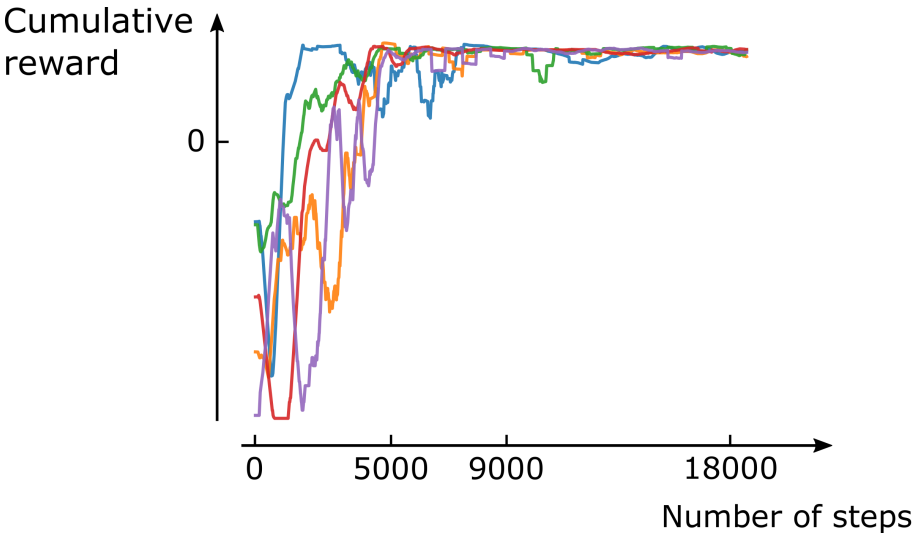


Figure 5.3: For scenario 3, the convergence of the 5 simulations takes place after between 5000 and 9000 steps.

5.3 Tuning of the four selected hyper-parameters

5.3.1 Weights of the reward function

The shape of the reward function has been presented in Section 2.1.3. The weights of the equations (2.1) and (2.2) are subject to a tuning. As a reminder,

the values of these weights for the non-tuned model are $w_{tr} = 10$ and $w_f = 2$, as seen in equations (4.1) and (4.2).

For each of the three scenarios, convergence was analyzed with two pairs of different weights:

- To give more importance to the transition reward and therefore to the cost variation: $w_{tr} = 20$ and $w_f = 1$. This curve is called **transition importance**;
- To give more importance to the final reward and therefore to the emissions reduction target: $w_{tr} = 5$ and $w_f = 4$. This curve is called **termination importance**.

Due to these various reward functions, the cumulative reward curves for these different cases do not have the same scale. Each figure in this section is accompanied by a table showing for each tested reward function the final cumulative reward obtained from the model and the best cumulative reward found during the exploration.

Intuitively, one might think that the weighting of the reward function has no direct influence on the convergence speed as it is not a parameter of the DQL model in the strict sense. Since the way in which the weights are changed will bring more importance to the decrease in cost or emissions, it can be expected that the agent will not converge to the same final state depending on the weights imposed. Indeed, if more importance is given to the transition reward (and thus to the cost reduction), we can assume that the costs will be reduced more significantly and vice versa. In this section, we will therefore try to verify whether or not these behaviours are true.

Scenario 1

The evolution of the cumulative reward for the three pairs of weights is shown in FIGURE 5.4 for scenario 1. No impact on convergence can be observed when changing the weights of the reward function. Indeed, the instability of the curve of transition importance may just be due to randomness.

Now that we have seen that the three pairs of weights tested converge approximately at the same time, we will look at the cumulative reward, the associated final state and the impact of the five actions on the energy system in terms of cost and emissions.

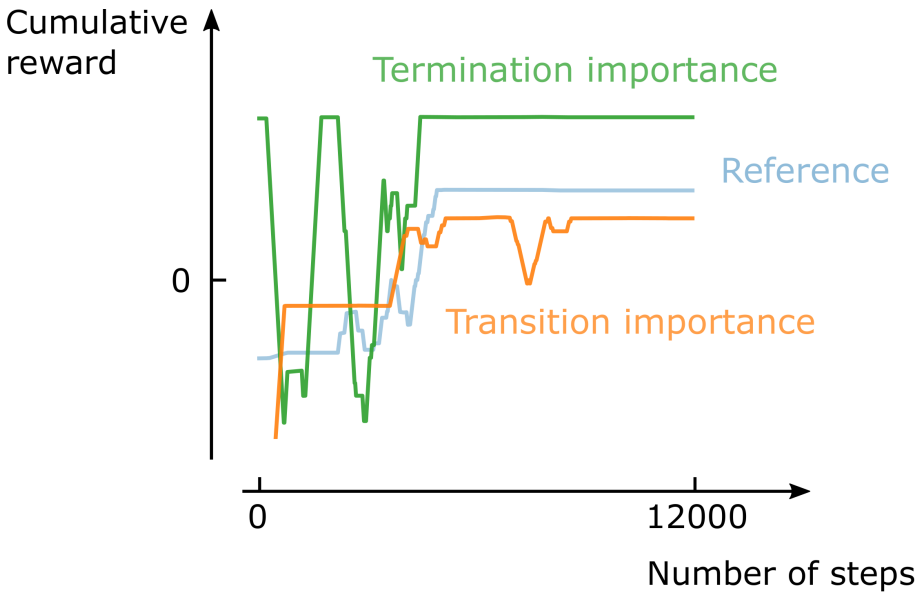


Figure 5.4: For scenario 1, all three curves converge at approximately the same time. An instability happens after convergence for the curve associated with the weights that give more importance to the transition reward.

Table 5.1 compares the cumulative reward from the trained model with the optimal cumulative reward found during training. The values of the "CR after convergence" column correspond to the cumulative rewards at which the curves in Figure 5.4 converge. For all three cases, the neural network is able to generalize from explored experience. We were already able to conclude this for the non-tuned model (see Section 4.6).

Reward function	Best CR	CR after convergence	Variation
Termination importance	4.12	4.15	0.7 %
Reference	2.25	2.29	1.8 %
Transition importance	1.50	1.59	6 %

Table 5.1: All cumulative rewards associated with the model after convergence are higher than the optimal found during training.

Table 5.2 shows the different final states associated with the optimal CR found during training as well as those after model convergence. When the final states are the same but the associated CR are different, this means that the actions taken by the agent are the same but taken in a different order.

Obviously, all simulations where the agent ends up in the same final state will

Reward function	Optimal final state	Final state after convergence
Termination importance	[-3,2]	[-3,2]
Reference	[-3,2]	[-3,2]
Transition importance	[-4,1]	[-3,2]

Table 5.2: The final state, whether after convergence or the one associated with the optimal CR, are all the same excepted the one associated with the optimal CR found during the training for the transition importance case.

have the same impact on the energy system since the actions performed are the same (even if the order is different) and ESTD is deterministic. The costs and emissions for the two different final states encountered can be found in TABLE 5.3.

	Reference state	State [-3,2]	State [-4,1]
GWP [$MtCO_2$/y]	28.89	28.46	28.49
Cost [b€/y]	18.55	18.02	17.98

Table 5.3: The final state [-4,1] reduces the cost slightly more (0.2%) than the final state [-3,2] but its emissions are slightly higher (0.1%).

In the case of scenario 1, the differences in terms of costs and emissions are too small (in the order of a tenth of a percent) to be able to draw conclusions regarding the impact of the weights of the reward function on the energy system.

Scenario 2

The impact on convergence of changing the weights of the reward function can be analyzed using FIGURE 5.5. The termination importance curve converges about 4000 steps before the two others. However, we have seen in SECTION 5.2 that, in scenario 2, the effect of randomness covers about 6000 steps. Therefore, we cannot conclude that the weights associated with the reward functions have an impact on the speed of convergence.

The dent presented on the termination curve just before the end of the simulation is caused by the moving average applied to the data. On the original data, there is an instantaneous drop at this location which is reflected on a few hundred steps due to the moving average. The original cumulative reward without applying the moving average can be seen in APPENDIX B (FIGURE B.1). This can happen in DQL implementations when weights are changed in an inappropriate way. It sends the model into a bad final state (i.e. low associated cumulative reward) for a short period of time before being rectified.

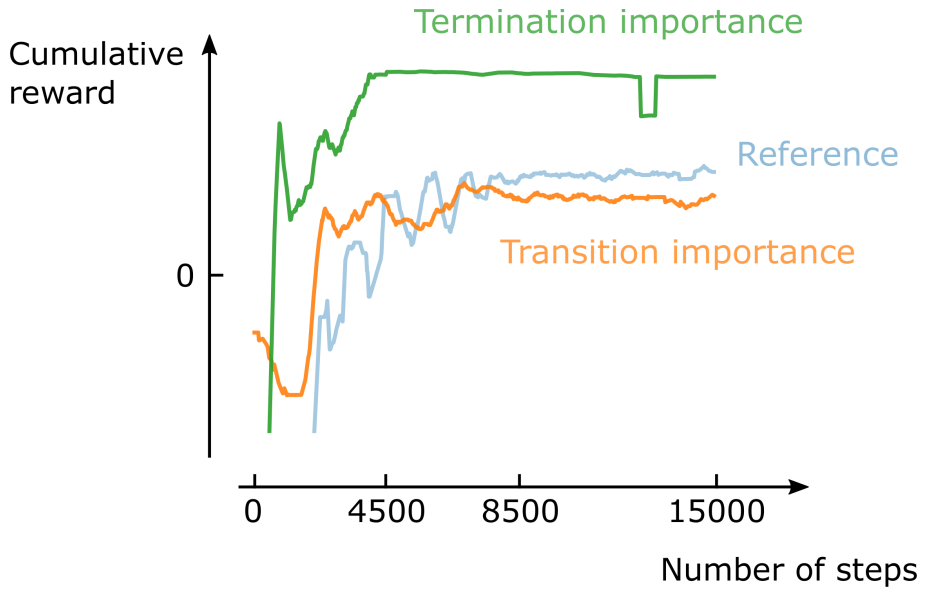


Figure 5.5: For scenario 2, the termination importance curve seems to be more stable after the convergence excepted the dent presented just before the end of the simulation.

The cumulative rewards for the second scenario can be found in Table 5.4. The more important the final reward is compared to the transition reward, the closer the CR after convergence is from the optimal CR. This can be explained by the fact that the final reward has a constant positive value if the emission target is reached. This is why it is necessary to look at the impact on the energy system in order to draw conclusions.

Reward function	Optimal CR	CR after convergence	Variation
Termination importance	4.22	4.08	-3.3 %
Reference	2.52	2.16	-14.3 %
Transition importance	2.03	1.69	-16.7 %

Table 5.4: Increasing the importance of the final reward R_f gives a cumulative reward closer to the optimal found during training. On the contrary, increasing the importance of the transition reward R_{tr} gives a cumulative reward further away from the optimal one.

The different final states can be found in Table 5.5.

The impact on the energy system in terms of cost and emissions of the different series of actions are illustrated in Figure 5.6. The exact values of costs and

Reward function	Final optimal state	Final state after convergence
Termination importance	[-5, 1, 2]	[-3, 3, 4]
Reference	[-5, 1, 4]	[-3, 3, 4]
Transition importance	[-5, 0, 5]	[-5, 2, 3]

Table 5.5: In none of the three cases does the model arrive in the optimal final state.

emissions can be found in Appendix B (Table B.1).

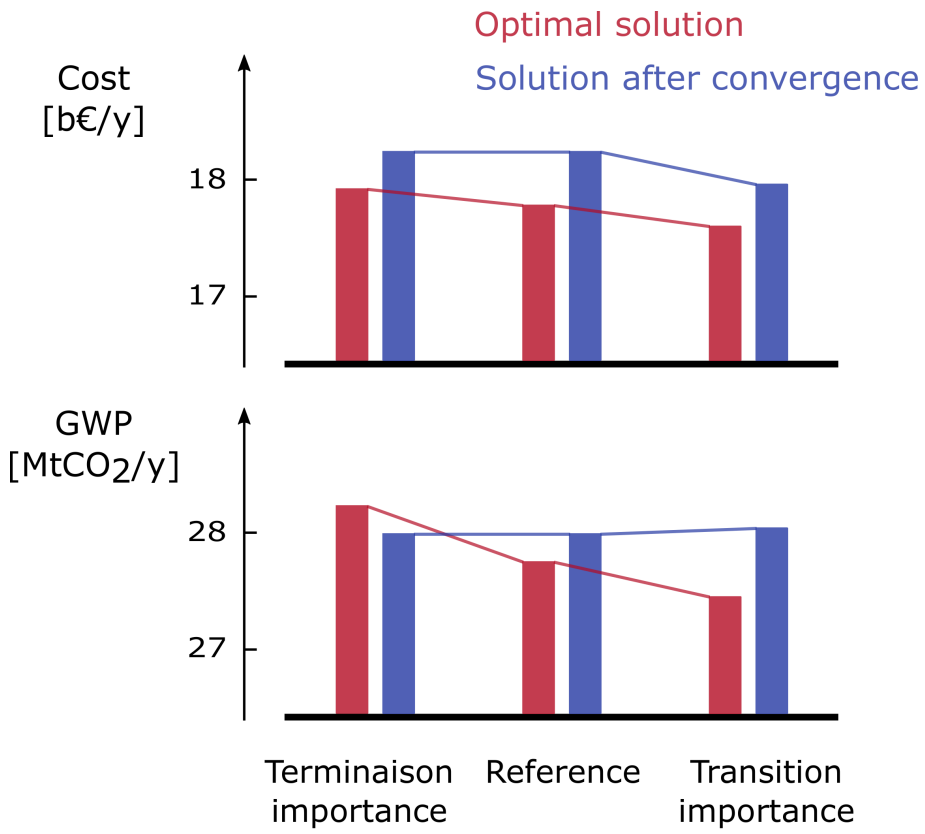


Figure 5.6: For scenario 2, the greater the transition reward is, the lower the cost of the energy system becomes. Regarding emissions, the optimal solution and the one after convergence have an opposite trend.

Concerning scenario 2, in terms of emissions, it is probably due to the randomness of the simulations that the two solutions have an opposite trend. This is why, no conclusion can be drawn.

Scenario 3

FIGURE 5.7 shows the evolution of the cumulative rewards of the three cases for the third scenario. No impact can be noticed from the weight change on convergence speed. However, there is a great instability at the end of the training for the termination importance case.

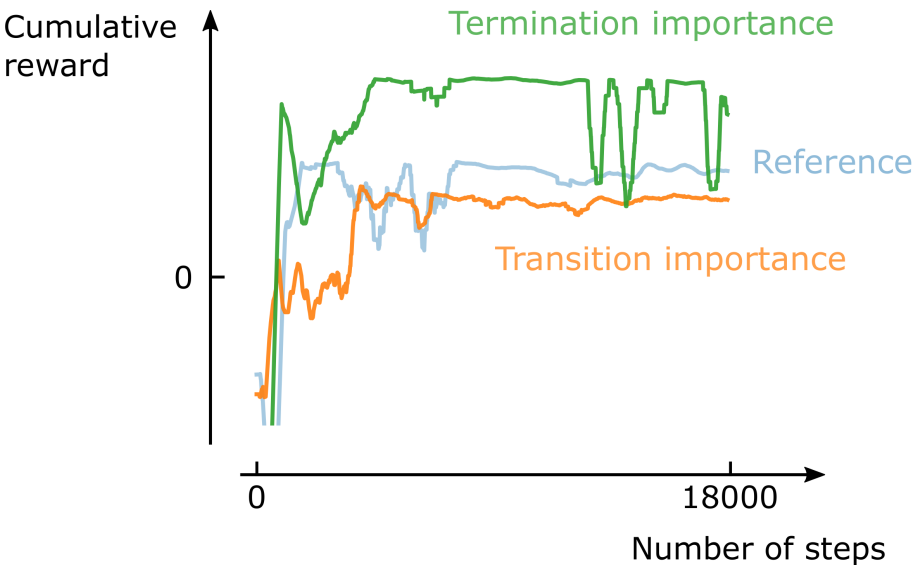


Figure 5.7: For scenario 3, the three curves converge in the same range.

The different cumulative rewards can be found in TABLE 5.6. The same conclusion as for scenario 2 can be drawn.

Reward function	Optimal reward	CR after convergence	Variation
Termination importance	4.24	4.01	-5.4 %
Reference	2.52	2.25	-10.7 %
Transition importance	2.03	1.60	-21.2 %

Table 5.6: As in TABLE 5.4, increasing the importance of the transition reward gives a cumulative reward further away from the best one.

The different final states can be found in TABLE 5.7.

The impact of the different cases on the cost and emissions of the energy system can be seen in FIGURE 5.8. Again, the exact values of the cost and emissions can be seen in APPENDIX B (TABLE B.2).

Reward function	Final optimal state	Final state after convergence
Termination importance	[-5, 0, 4, 0, 1]	[-2, 4, 3, 1, 0]
Reference	[-5, 0, 5, 0, 0]	[-1, 0, 5, 1, 3]
Transition importance	[-5, 1, 4, 0, 0]	[-4, 2, 4, 0, 0]

Table 5.7: In none of the three cases does the model arrive in the optimal final state.

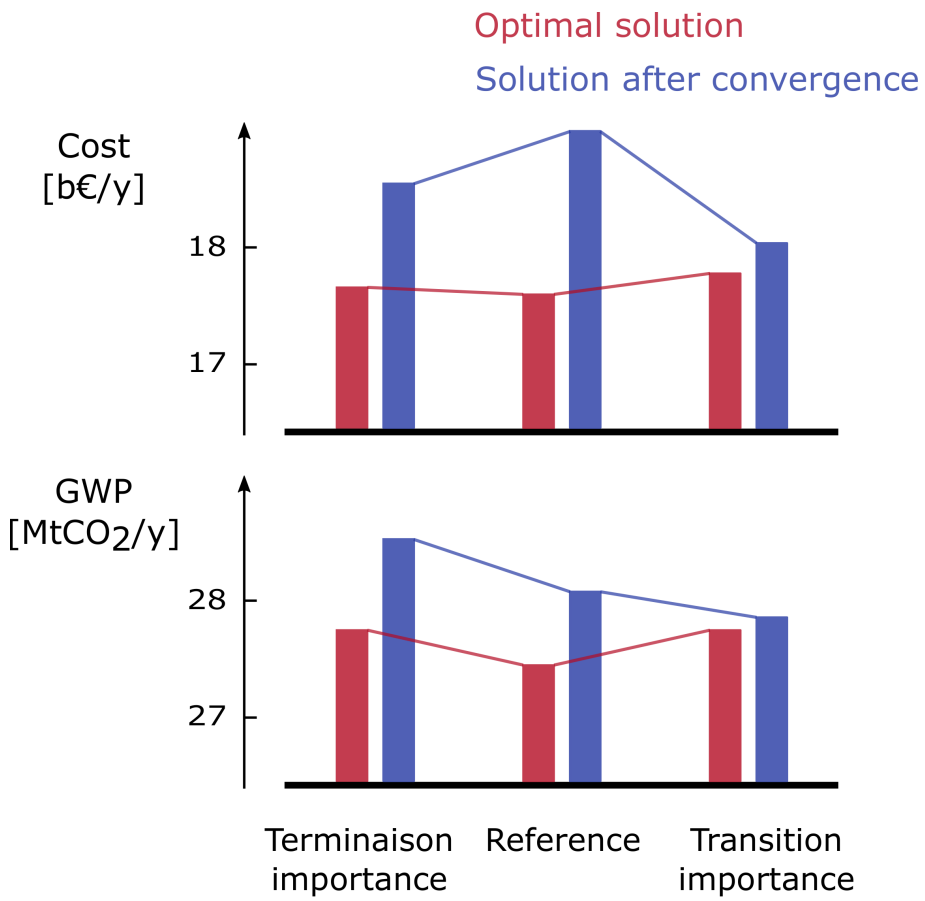


Figure 5.8: For scenario 3, no trend in the three cases can be observed in terms of reduction or increase of the cost but also concerning the emissions.

Conclusion on this parameter

As we expected at the beginning of this section, the convergence speed does not change enough to say that this parameter has a significant impact on it. However, the cumulative reward curve for the termination importance case seems more stable than the other two. However, when instability occurs, it is more important than for the other two cases. Therefore, these weights are not modified from the non-tuned

model.

In conclusion, this parameter has mainly an impact on the final state and thus on the cost and emissions of the energy system. It therefore has a physical impact. It can be explained by the fact that the transition reward is received by the agent during its exploration and therefore influences the states towards which it will move. Unfortunately, contrary to what we could have imagined, no trend could be identified to allow the control of costs and emissions from the coefficients of the rewards functions.

5.3.2 Benefits of using a target network

As explained in Section 4.5, it is usual to use 2 neural networks to increase the stability of the training process. In this section, we will compare the use of a target neural networks or not. Figures 5.9, 5.10 and 5.11 show the cumulative reward curves of the model for scenarios 1, 2 and 3 respectively. The curve labeled "reference" uses a target and a main network while the curve labeled "no target network" uses only a main network.

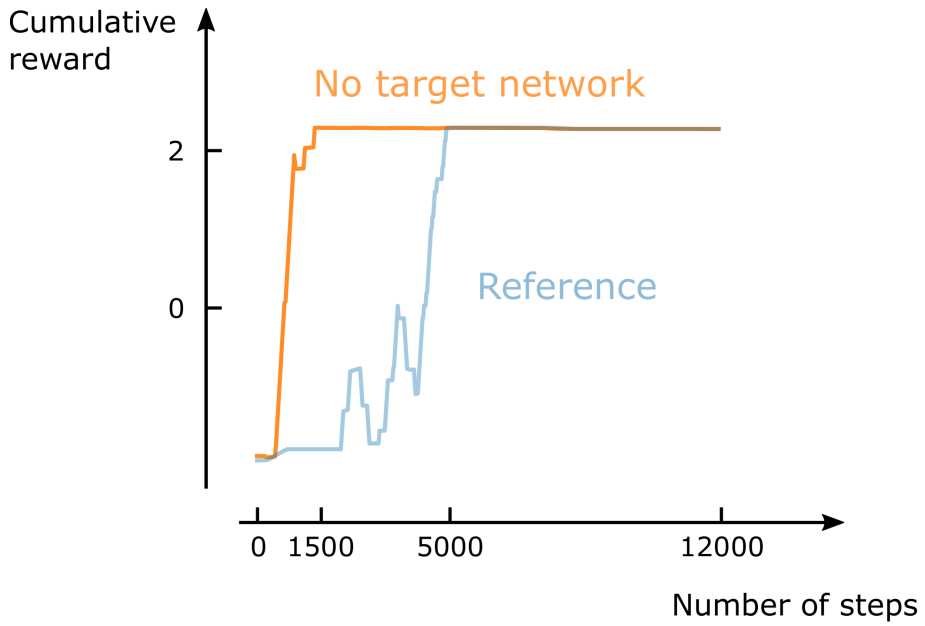


Figure 5.9: The difference in convergence speed (3500 steps) is of the same order as the difference in convergence speed that could be induced by randomness (4000 steps) for scenario 1.

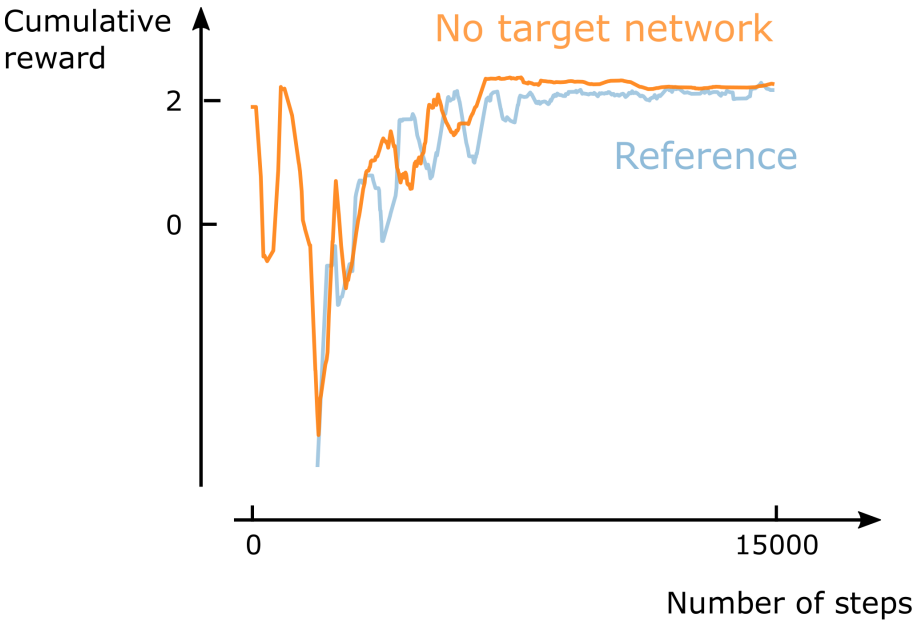


Figure 5.10: The two curves have a fairly similar speed of convergence for scenario 2.

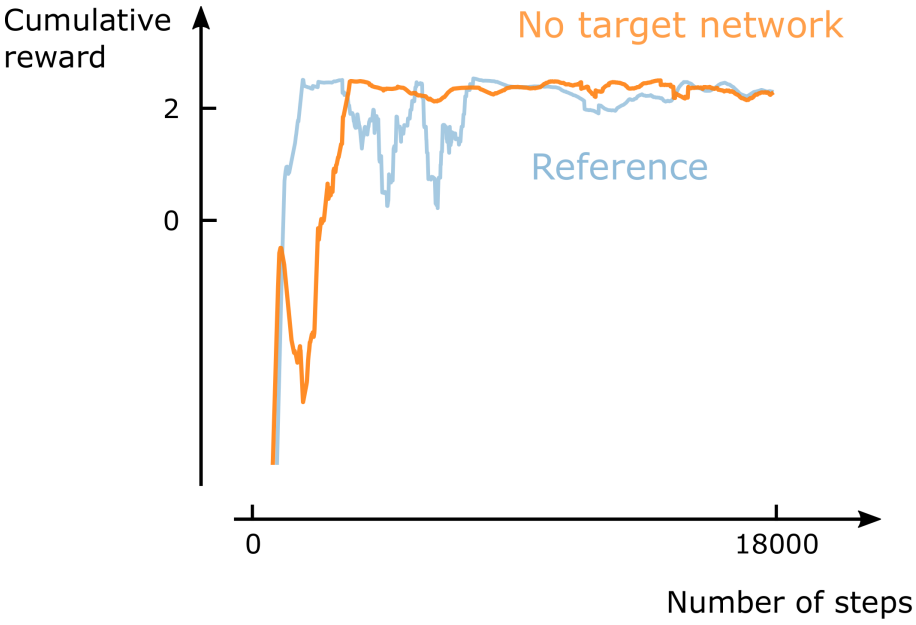


Figure 5.11: The two curves do not differ enough in terms of speed of convergence for scenario 3.

Except for scenario 1 where the difference in convergence is more marked in favor of a single neural network, using one or two neural networks does not seem to make a significant difference. The interest of changing this parameter is not marked enough to differentiate it from the randomness inherent in the DQL model. This parameter will therefore not be modified and we will continue to use a target network for training as advised by the rules of good practice.

5.3.3 Number of neurons in the hidden layers

The number of neurons in each of the two hidden layers of our neural network could be fixed according to good practice rules, as explained in Section 4.5. For each scenario, the exact number of neurons in each layer is available in Table 4.3.

In this section, an analysis of the effect of the variation of the number of these neurons on the convergence is conducted. To do so, the number of neurons in the hidden layers is doubled and halved with respect to the non-tuned model, for each of the 3 scenarios. The consequences of this change on the cumulative reward curves for scenarios 1, 2 and 3 are available in Figures 5.12, 5.13 and 5.14 respectively.

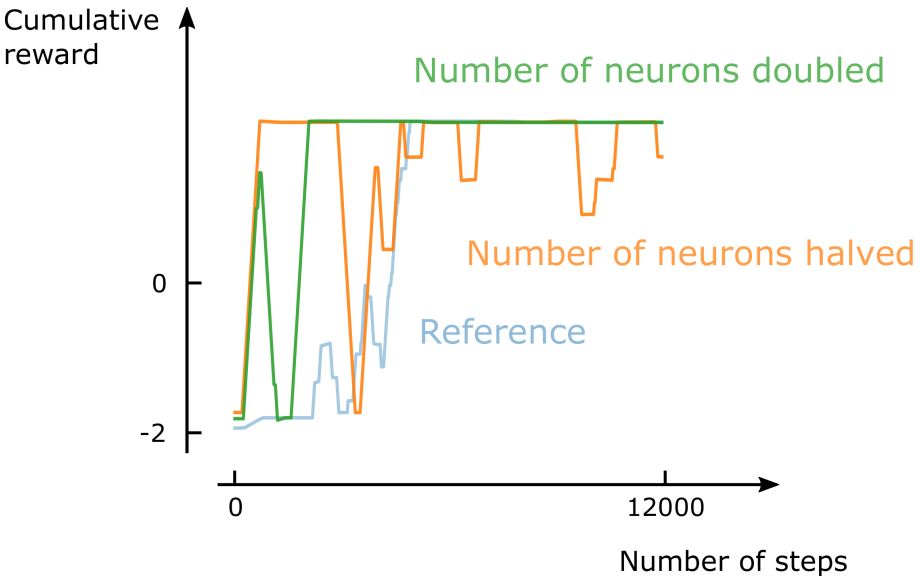


Figure 5.12: For scenario 1, convergence is fastest when the number of neurons is doubled. When the number of neurons is halved, the curve of cumulative reward seems unstable.

A clear trend can be seen in the results of the three scenarios. The higher the number of neurons in the hidden layers, the faster the cumulative reward curve

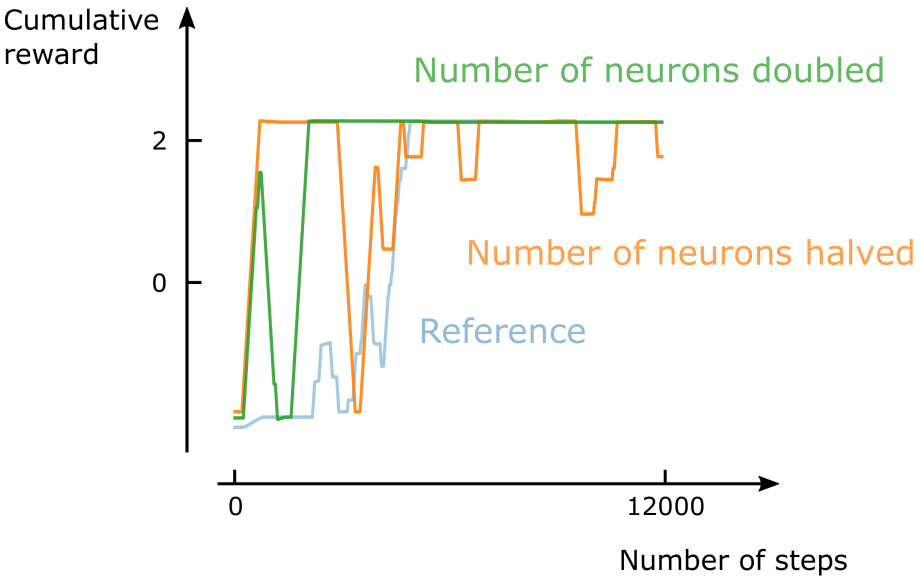


Figure 5.13: For scenario 2, convergence is fastest when the number of neurons is doubled. Moreover, the cumulative reward curves are more and more stable as the number of neurons increases.

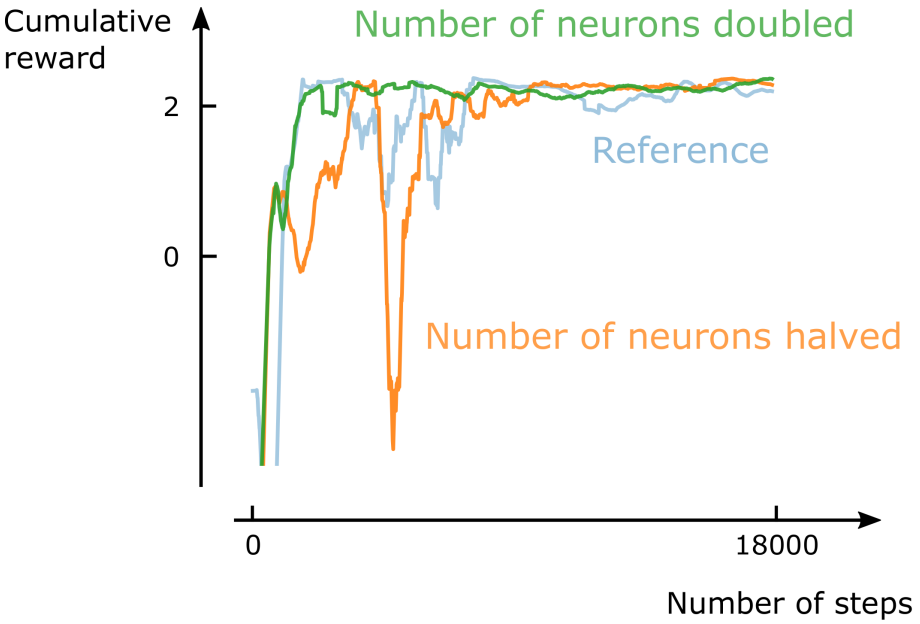


Figure 5.14: For scenario 3, the convergence speed is identical between the doubled number of neurons and the non-tuned model. On the other hand, the convergence curves are more and more stable as the number of neurons increases.

converges and the less the curve oscillates.

In order to see if this improvement in convergence persists when the number of neurons in the hidden layers is further increased, FIGURE 5.15 compares, for scenario 2, the cumulative reward curves when the number of neurons is doubled and quadrupled. A further increase in the number of neurons per hidden layer does not improve convergence speed or stability for this scenario.

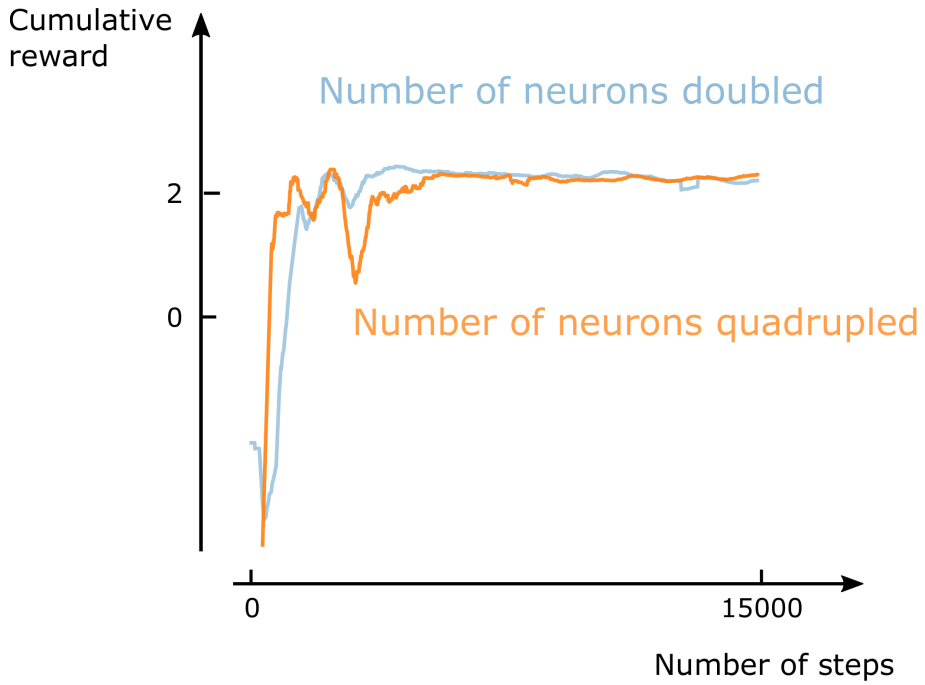


Figure 5.15: The two curves converge in the same order of speed for scenario 2.

Scenarios 1 and 3 have a neural network architecture very similar to scenario 2. Indeed, the input and output layers of the 3 scenarios have a similar number of neurons as seen in TABLE 4.3. Therefore, it can be assumed that the neural networks for these two scenarios will react in a similar manner to a quadrupling of the number of neurons. Moreover, the greater the number of neurons in the hidden layers, the more likely the neural network is to overfit. It is therefore wiser to restrict oneself to the smallest increase in the number of neurons that makes a significant improvement.

The results obtained when varying the number of neurons per hidden layer is significant enough not to be the result of chance inherent in the DQL model. Therefore, in accordance with our greedy methodology, a neural network whose

number of neurons in the hidden layers is doubled will be used in the following experiments. The number of neurons used in each layer for the three scenarios thus becomes as shown in TABLE 5.8.

	Input layer	Hidden layer 1	Hidden layer 2	Output layer
Scenario 1	2	16	8	4
Scenario 2	3	20	10	5
Scenario 3	5	28	14	6

Table 5.8: The number of neurons in the hidden layers has been doubled compared to the non-tuned model.

5.3.4 Decay factor

As explained in SECTION 2.2, the decay factor β influences the rate at which the parameter ϵ of the epsilon-greedy strategy decreases. This parameter makes it possible to play on the trade-off between exploration and exploitation during the training of the model. In this section, an analysis of the effect of this parameter on convergence is carried out.

The objective behind these experiments being to decrease the convergence time, the decay factor will only be increased. Indeed, decreasing the decay factor would lead to an increase in exploration, which is not necessary because the results are already close enough to optimality. On the other hand, an increase of the decay factor leads to a decrease in the exploration and thus, a decrease in the number of steps N_{steps} necessary for epsilon to reach its final value (around $\epsilon = 0.1$). Therefore, in parallel with an increase in the decay factor, a decrease in the simulation time will occur. The interest of this section is to see how far this decay factor can be increased while keeping an acceptable convergence for each scenario.

Scenario 1

In TABLE 5.9, you can see the values of decay factor β that were tried out for scenario 1. A multiplication of β by a factor of two leads to a halving of the number of training steps. In order to keep the figures of this section clear, experiments that do not provide any particular information will not be plotted.

As you can see in FIGURE 5.16, experiments ranging from $\beta = 0.00015$ to $\beta = 0.0024$ offer sufficient convergence. In each of these experiments the cumulative reward obtained at the end of the training is close to the optimal cumulative reward

Decay factor β	Simulation duration (steps)
Reference 0.00015	12000
0.0003	6000
0.0006	3000
0.0012	1500
0.0024	750
0.0048	375
0.0096	188

Table 5.9: For scenario 1, the decay factor is varied from 0.00015 to 0.0096 by doubling it in each experiment.

for a sufficient time (see CHAPTER 3).

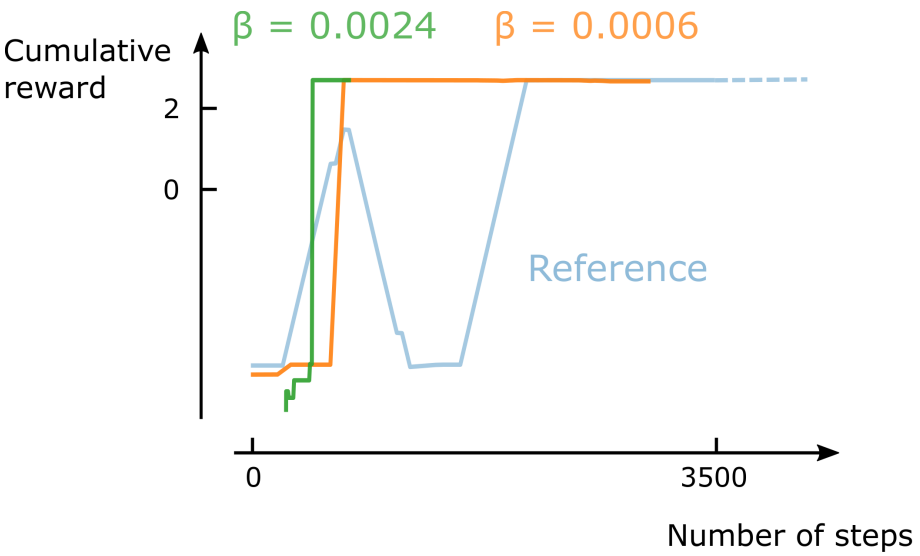


Figure 5.16: The 3 curves converge towards a similar cumulative reward.

In contrast, experiments for higher values of β do not converge optimally. Indeed, FIGURE 5.17 shows that the cumulative reward obtained for $\beta = 0.0096$ is negative while the one obtained for $\beta = 0.0048$ is close to optimal but does not converge long enough to be considered acceptable.

Based on these results, the optimal decay factor for scenario 1 is $\beta = 0.0024$. This is the largest possible decay factor for which convergence remains optimal. This change in parameter represents a reduction in training time by a factor of 16.

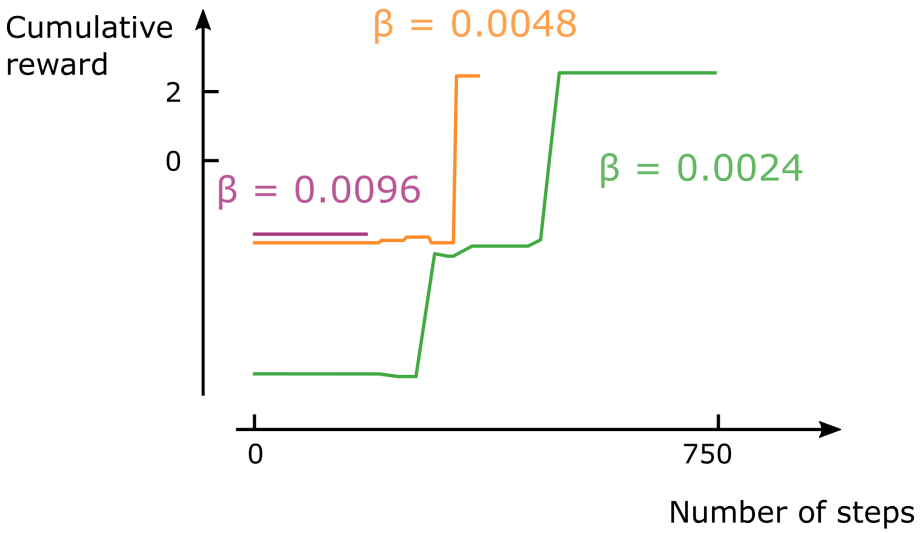


Figure 5.17: Only the curve $\beta = 0.0024$ offers a cumulative reward near the optimal for a sufficiently long time.

The DQL part of the training for this scenario, which previously took a few hours, now takes less than an hour.

Scenario 2

In TABLE 5.10, you can see the values of decay factor β that were tried out for scenario 2.

Decay factor β	Simulation duration (steps)
Reference 0.00015	15000
0.0003	7500
0.0006	3750
0.0012	1875
0.0024	937
0.0048	468

Table 5.10: For scenario 2, the decay factor is varied from 0.00015 to 0.0048 by doubling it in each experiment.

Experiments ranging from $\beta = 0.00015$ to $\beta = 0.0006$ offer sufficient convergence but do not provide additional information. In each of these experiments, the cumulative reward obtained at the end of the training is close to the optimal one.

In FIGURE 5.18 you can see that the convergence curves $\beta = 0.0012$ and $\beta = 0.0024$ are optimal. On the other hand, the convergence curve $\beta = 0.0048$, although obtaining a final cumulative reward close to the optimal one, does not allow sufficient convergence. Indeed, the convergence does not take place for a sufficiently long time.

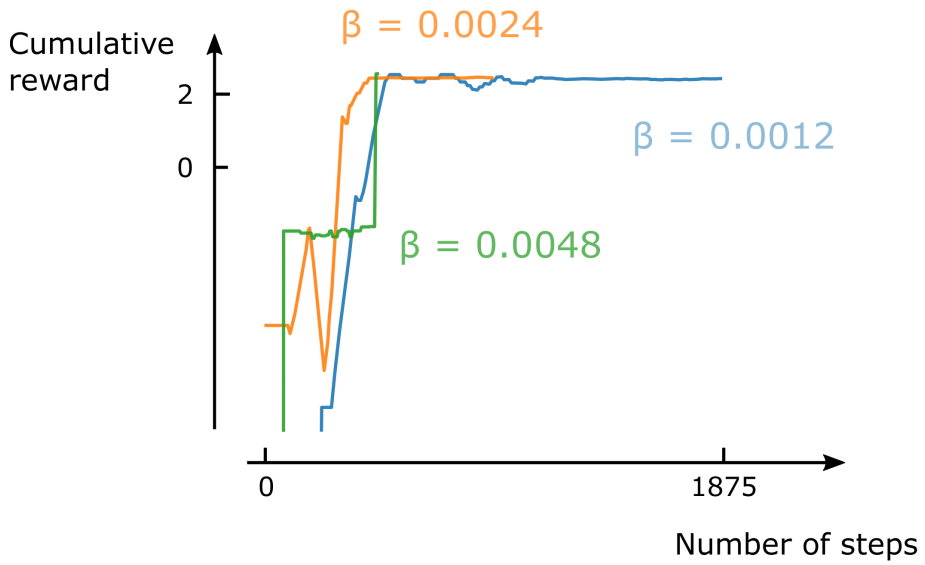


Figure 5.18: The curves $\beta = 0.0012$ and $\beta = 0.0024$ converge while the curve $\beta = 0.0048$ reaches a sufficient cumulative reward for a short period of time.

Based on these results, the optimal decay factor for scenario 2 is $\beta = 0.0024$. As for scenario 1, this change in parameter represents a reduction in training time by a factor of 16. The DQL part of the training for this scenario, which previously took about a day, now takes less than two hours.

Scenario 3

In TABLE 5.11, you can see the values of decay factor β that were tried out for scenario 3.

The experiments where the decay factor was varied from $\beta = 0.0001$ to $\beta = 0.0004$ offer sufficient convergence but do not provide additional information. In FIGURE 5.19 you can see that the cumulative reward curves $\beta = 0.0008$ and $\beta = 0.0016$ are optimal. On the other hand, the convergence curve $\beta = 0.0032$, although obtaining a final cumulative reward close to the optimal one, does not allow sufficient convergence. Indeed, the convergence curve oscillates and does not

Decay factor β	Simulation duration (steps)
Reference 0.0001	18000
0.0002	9000
0.0004	4500
0.0008	2250
0.0016	1125
0.0032	563

Table 5.11: For scenario 3, the decay factor is varied from 0.0001 to 0.0032 by doubling it in each experiment.

stabilize near the optimal cumulative reward value.

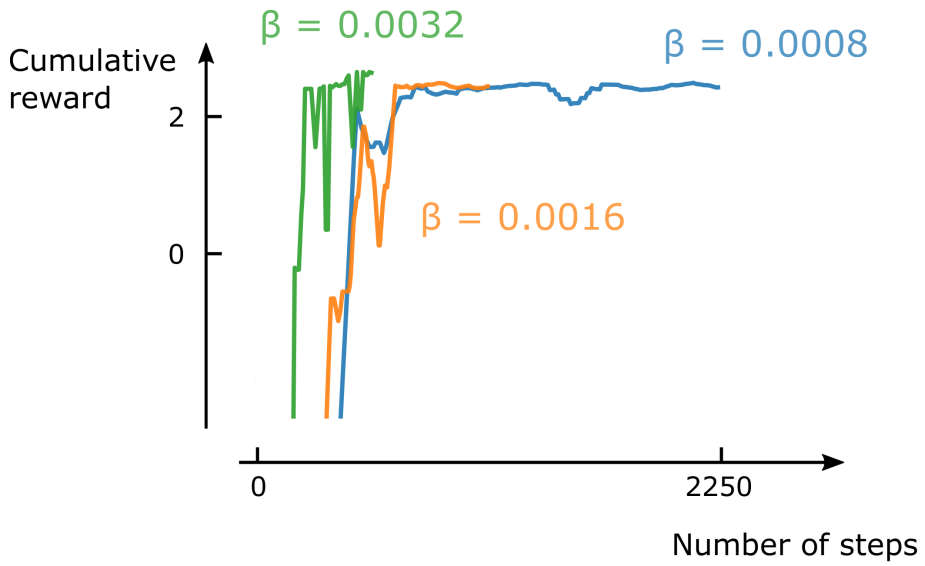


Figure 5.19: The curves $\beta = 0.0008$ and $\beta = 0.0016$ converge while the curve $\beta = 0.0032$ does not stabilize.

Based on these results, the optimal decay factor for scenario 3 is $\beta = 0.0016$. This change in parameter represents a reduction in training time by a factor of 16. The DQL part of the training for this scenario, which previously took about a day and a half, now takes less than three hours.

Conclusion on this parameter

An analysis of the optimal value of the decay factor resulted in a reduction of the training time for the DQL part of the simulation by a factor of 16, for each scenario. It is interesting to note that the choice of decay factor made by trial and error in Section 4.5 was found to be very far from the optimal value. One explanation for this is that in this first estimation, the number of neurons per hidden layer was not optimal. An increase in the decay factor without a prior increase in the number of neurons per hidden layer did not lead to a significant improvement in convergence. These two parameters therefore have a synergetic effect. The significant improvement in convergence following these experiments is not only the result of the increase in the decay factor but of the combined increase in the decay factor and the number of neurons per hidden layer.

The decay factors for each of the scenarios thus becomes as shown in Table 5.12.

Scenario	Decay factor β
1	0.0024
2	0.0024
3	0.0016

Table 5.12: The decay factors were divided by 16 compared to the non-tuned model.

5.4 Conclusion of the tuning

As a result of the analyses conducted in this chapter, the changes to the DQL model parameters are as follows:

- For each of the scenarios, the number of neurons per hidden layer of the neural network is doubled compared to the first estimate made in Section 4.5. The number of neurons in the hidden layers for each scenario is presented in Table 5.8;
- For each of the scenarios, the decay factor β is multiplied by 16 compared to the first estimate made in Section 4.5. The decay factors for each scenario is presented in Table 5.12.

The other two parameters analyzed in this section (weight of the reward function and use of a target network) remain unchanged from the reference model seen in

Section 4.5. Indeed, the effect of these parameters on convergence is not strong enough to differentiate it from the randomness inherent in the DQL model. We also noticed that we could not find any particular trend in the weights and evolution of cost and emissions i.e. it is not because we give more importance to the transition reward that the cost will decrease more significantly.

These changes provide greater convergence stability and a 16-fold decrease in simulation time of the DQL part of the simulation. The greedy tuning method allowed us to obtain such results without having to explore all possible parameter combinations.

Now that the theory behind the model has been explained in Chapter 2, that the practical implementation choices have been made in Chapter 4 and that an analysis of the optimal parameters has been carried out in this chapter, the model is ready to be used. Chapter 6 is devoted to the analysis of the actions carried out by the agent and the consequences of these actions on the energy system.

Chapter 6

Impact on the energy system

In the previous chapter, we varied various parameters from their non-tuned values found in Chapter 4. This allowed us to find new hyper-parameters that improved the convergence of our model while keeping an acceptable solution (see Chapter 3). It is now time to analyze the impact of the series of actions decided by our agent on the reference energy system (the one presented in Section 4.1). In this chapter, the way to analyze the impact of actions on the energy system will be systematically to look at their effect on costs, emissions and energy mix.

For each of the three scenarios investigated, a reminder from Chapter 4 of the different actions that the agent can take is done. The model (with the hyper-parameters from the tuning) will be used to determine the list of actions taken by the politician. The impact of these actions on the reference energy system will be presented. Finally, we will compare this solution with the optimal one found during the training. This will allow us to analyze the differences in impact on the energy system of these two series of actions.

As a reminder of Section 4.1, the cost and the CO_2 emissions of the reference energy are respectively 18.55 [b€/y] and 28.89 [MtCO_2/y]. The energy mix was presented in Figure 4.1.

6.1 Scenario 1

In the first scenario investigated, the politician takes a total of 5 actions. When making a decision, he can choose from four different actions each time: increase or decease the wood capacity and the price of natural gas.

Energy results provided by the tuned-model

The path taken by the agent is illustrated in FIGURE 6.1.

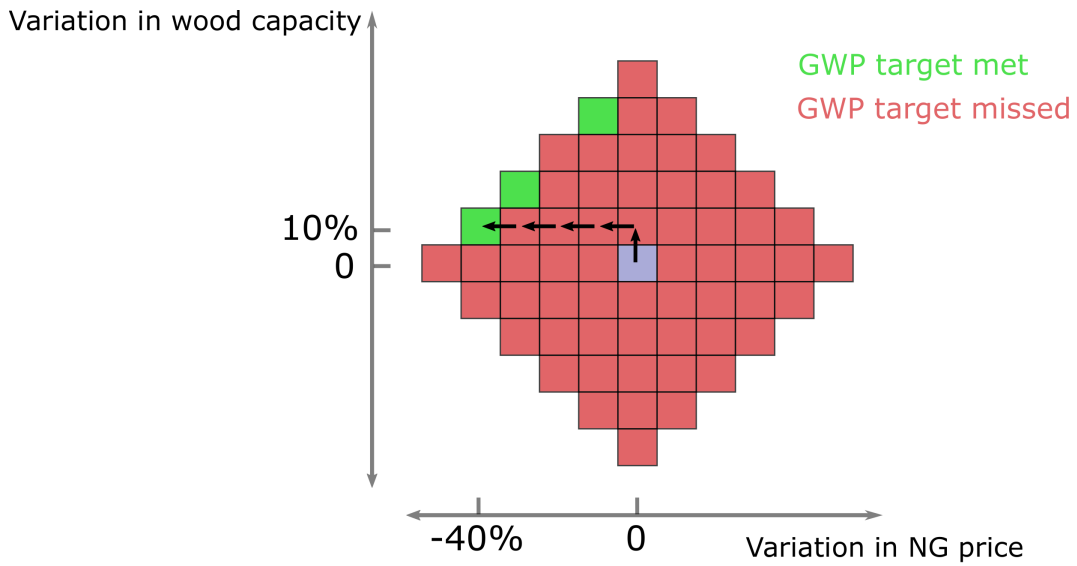


Figure 6.1: The agent increases the wood capacity, reduces four times the cost of natural gas and goes below the emissions threshold.

By carrying out this sequence of actions, the politician is fulfilling his commitments. Indeed, the emissions of the energy system are reduced to 28,492 [$MtCO_2/y$] while the target is set at 28,500 [$MtCO_2/y$] (see SECTION 4.1). TABLE 6.1 shows the comparison in terms of costs and emissions between the reference energy system and the one after carrying out the series of actions.

	Reference energy system	Modified energy system	Variation
GWP [$MtCO_2/y$]	28.89	28.49	-1.4 %
Cost [b€/y]	18.55	17.98	-3.1 %

Table 6.1: As a result of the actions taken by the agent, emissions are reduced by 1.4% and costs by 3.1%.

The impact of the series of actions taken by the agent on the energy mix is illustrated in FIGURE 6.2. The exact amount of these resources used by the energy system can be found in APPENDIX C (TABLE C.1).

The impact of the actions on the energy system is the result of the optimization carried out by ESTD, we cannot act directly on this result. The only way to

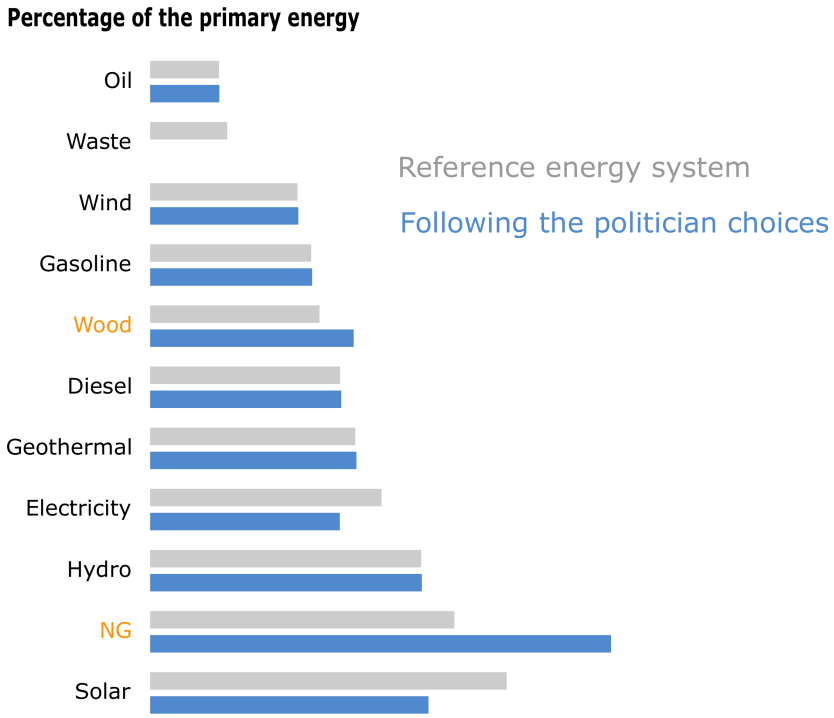


Figure 6.2: The increase in natural gas and wood use is compensated by the decrease in the use of solar technology and import electricity as well as by the removal of waste in the primary energy mix.

guide our agent to change the energy mix in a certain manner is to change the different possible actions it can undertake. For example, if we notice that the energy system is using wind power at its maximum capacity and we want to guide the energy transition towards a greater use of wind turbines, we can give the agent the opportunity to increase this wind turbine capacity. If this action is interesting in terms of costs and emissions, it will then use it and the energy mix will be modified accordingly.

Comparison with the optimal solution found during the training

The series of actions with the best cumulative reward found during the training sends the agent to the same final state as presented before i.e. the agent increases the wood capacity and decreases four times the cost of the natural gas. However, the cumulative reward is not the same (as it is explained in Section 4.6), which means that the five actions are the same but they are taken in a different order. Since the actions taken are the same and ESTD is deterministic, the two solutions

will have exactly the same impact on the energy system (cost, emissions and energy mix).

6.2 Scenario 2

In the second scenario, the agent can do the same 4 actions as in scenario 1 but in this case he can also increase the price of diesel. Also, it can now take 10 actions in a row.

Energy results provided by the tuned-model

Since the state of the agent is characterized by a vector containing more than two elements, it is impossible to represent the path taken by the agent graphically as in FIGURE 6.1. Indeed, a third dimension would be necessary. Another illustration of the path taken by the agent can be found in FIGURE 6.3.

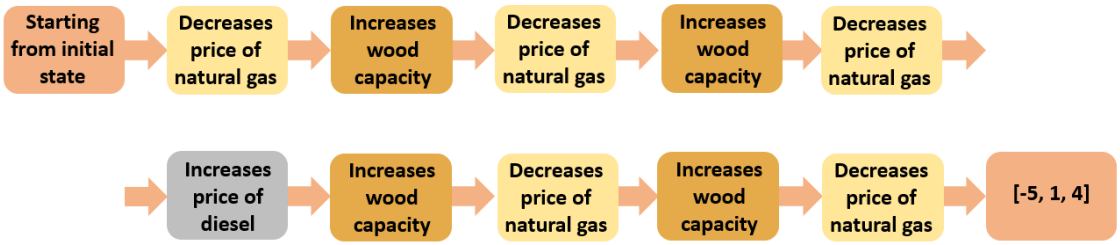


Figure 6.3: The agent reduces five times the price of natural gas, increases four times the wood capacity and increases once the price of diesel.

TABLE 6.2 shows the costs and emissions of the reference energy system and after applying the series of actions. As in the first scenario, the emission target is reached.

	Reference energy system	Modified energy system	Variation
GWP [$MtCO_2$/y]	28.89	27.71	-4.1 %
Cost [b€/y]	18.55	17.74	-4.4 %

Table 6.2: As a result of the actions taken by the agent, emissions are reduced by 4.1% and costs by 4.4%.

The impact of the series of actions on the energy mix can be seen in FIGURE 6.4. Again, the exact amount of these resources used by the energy system can be

found in Appendix C (Table C.2).

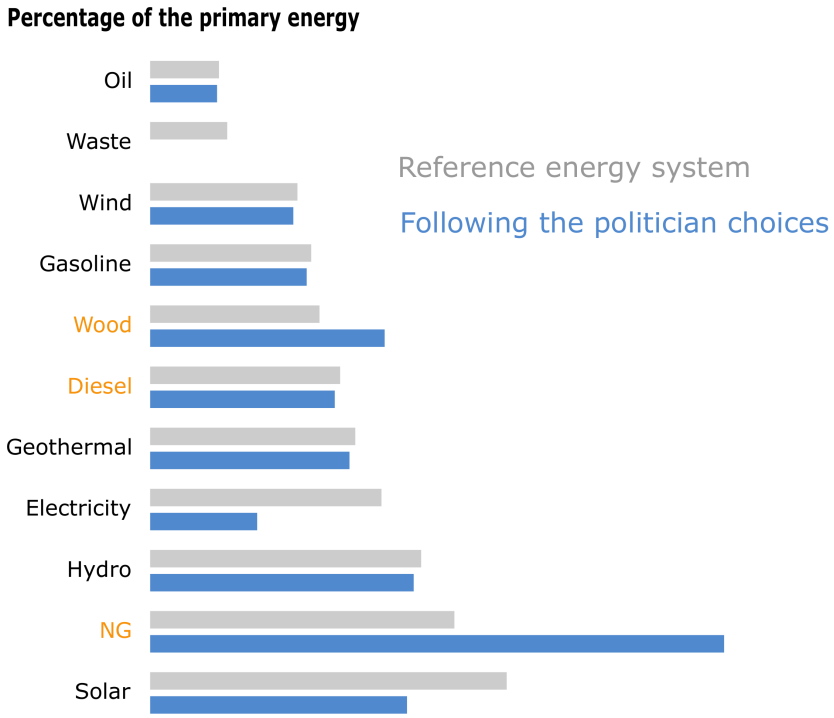


Figure 6.4: As for the first scenario, the increase in natural gas and wood use is compensated by the decrease in the use of solar technology and import electricity as well as by the removal of waste in the primary energy mix.

Comparison with the optimal solution found during the training

As for scenario 1, the solution provided by the trained model is the same as the optimal one. These two solutions thus have the same impact on the energy mix.

6.3 Scenario 3

For the third scenario, the agent has the choice among 6 different actions: increase or decrease the price of natural gas, increase the wood capacity, the diesel price, the solar capacity and the hydro capacity. The agent can do 10 actions in a row.

Energy results provided by the tuned-model

The series of actions taken by the agent can be seen in Figure 6.5.

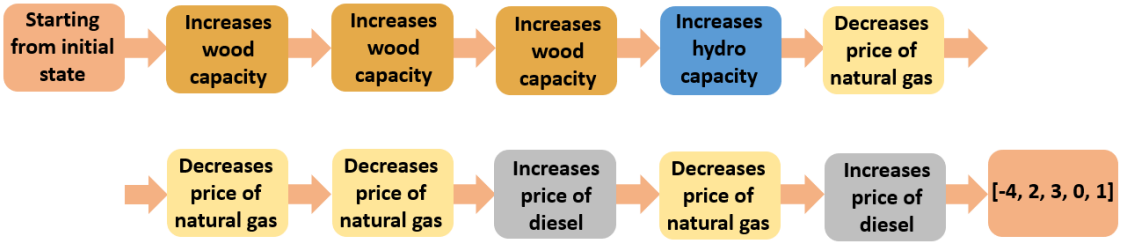


Figure 6.5: The agent decrease four times the price of natural gas, increases twice the diesel price, increases three times the wood capacity and increase once the hydro capacity.

The impact of applying the series of actions in terms of cost and emissions can be seen in TABLE 6.3. As for the first two scenarios, the emissions target is met.

	Reference energy system	Modified energy system	Variation
GWP [$MtCO_2$/y]	28.89	27.47	-4.9 %
Cost [b€/y]	18.55	17.86	-3.7 %

Table 6.3: As a result of the actions taken by the agent, emissions are reduced by 4.9% and costs by 3.7%.

The impact of the series of actions on the energy mix can be seen in FIGURE 6.6. Again, the exact amount of these resources used by the energy system can be found in APPENDIX C (TABLE C.3).

Comparison with the optimal solution found during the training

In contrast with the two other scenarios, the final states of the solution provided by the trained model and the optimal solution are not the same. Indeed, the optimal final state is [-5, 0, 3, 2, 0]. It means that the agent reduces five times the price of natural gas, increases three times the wood capacity and increases twice the solar capacity. Compared to the final state provided by the trained model, it reduces the price of natural gas once more and increases the solar capacity twice more. On the other hand, it does not change the price of diesel or the capacity of hydro.

A comparison in terms of cost and emissions can be found in TABLE 6.4. Both solutions are below the emissions threshold. Since the cumulative reward of the optimal solution is greater than the one associated with the solution after convergence, it is logical that its cost is lower.

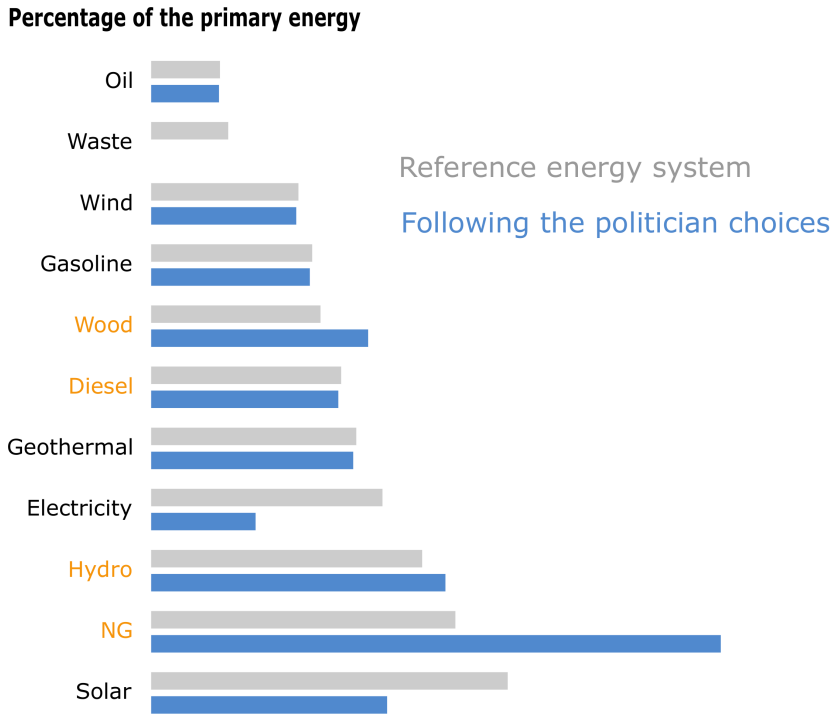


Figure 6.6: As for the first two scenarios, the increase in natural gas and wood use is compensated by the decrease in the use of solar technology and import electricity as well as by the removal of waste in the primary energy mix.

	Solution after convergence	Optimal solution	Variation
GWP [$MtCO_2/y$]	27.47	27.95	1.7 %
Cost [b€/y]	17.86	17.65	-2.4 %
Cumulative reward	2.27	2.45	7.9 %

Table 6.4: The best solution found during training has higher emissions but lower cost.

The comparison of the two solutions in terms of energy mix is shown in FIGURE 6.7. Again, the exact amount of these resources used by the energy system can be found in APPENDIX C (TABLE C.4). It seems strange that the system uses less natural gas in the case of the optimal solution while its price is reduced one more time. Hydro is more used in the case of the trained model solution because its capacity is increased unlike the other solution. The optimal solution uses more solar energy but this is not related to the fact that its capacity has been increased. Indeed, in none of the solutions its capacity is used at the maximum. The wood is used in the same quantity, this is because its capacity has been increased the same number of times in both solutions and its capacity is used at the maximum. Diesel

is used in the same amount in both solutions, although its price has been increased twice in the optimal solution. No action has been taken on imported electricity in the two solutions but it is used more in the optimal solution.

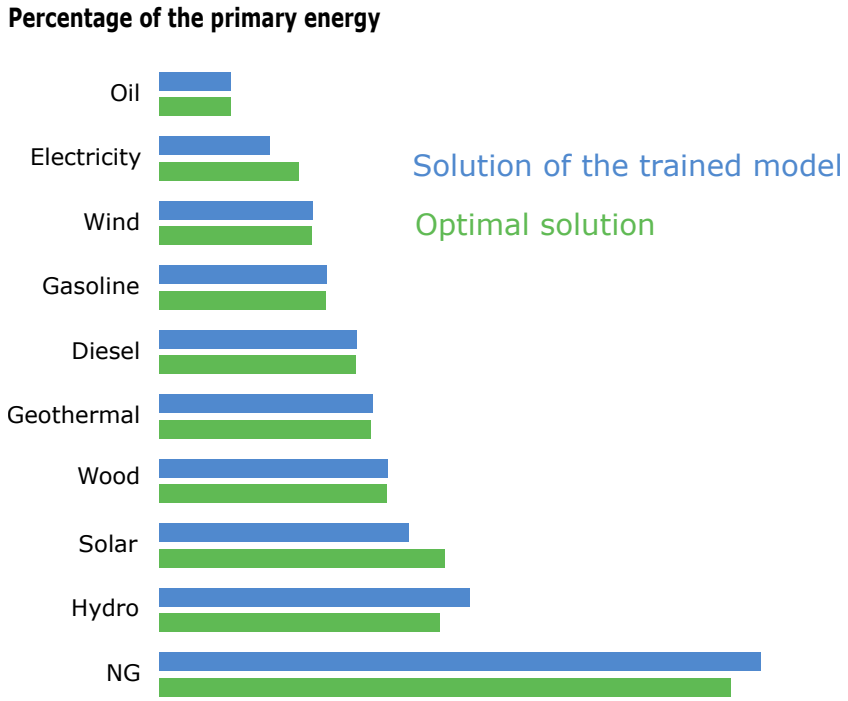


Figure 6.7: The energy mix of the two solutions are very similar even if their final states are not the same.

6.4 Analysis of the energetic results

In conclusion, we have noticed that for all three scenarios, the emission target is reached while decreasing the cost of the energy system. The tuned model therefore respects the objectives that were set out. The reductions in costs and emissions compared to the reference energy system (the one without having taken any action on it) are shown in the Table 6.5.

	Scenario 1	Scenario 2	Scenario 3
GWP [$MtCO_2/y$]	-1.4 %	-4.1 %	-4.9 %
Cost [b€/y]	-3.1 %	-4.4 %	-3.7 %

Table 6.5: Emissions are reduced by 1.4% to 4.9% and costs by 3.1% to 4.4% depending on the scenario.

Looking at the energy mix for the three scenarios after carrying out the series of actions, a similar trend can be observed:

- Reduction of the solar energy: this is strange because it is a more expensive technology than, for example, wind power and ESTD in itself is not constrained with the emissions. This could be explained by the fact that we have taken storage out of the energy system and that solar energy is more intermittent;
- Increase of the natural gas: the actions lower its price, so it becomes more attractive;
- Decrease of the import electricity: it is a relatively expensive resource with a high GWP compared to others;
- Increase of the use of wood: this is strange because even if its capacity is increased, wood is the most expensive resource used by ESTD;
- Removal of waste: this is strange because ESTD considers this resource as free.

We cannot explain all these consequences that actions have on the energy system. Indeed, how ESTD reacts to the changes in prices and capacities imposed by our model is beyond our control.

We have noticed that, even if the solution provided by the trained model does not reach the same final state as the optimal solution, the differences in terms of energy mix are limited. However, for scenarios 1 and 2, they reach the same final state, so there is no difference in the impact on the energy system.

Discussion

The purpose of this discussion is to take a step back from the information presented earlier in this master's thesis and to take a critical look at the limitations of our model as well as some of the choices we have made during the implementation. This will allow the reader to have an overview of the main limitations and points of discussion regarding our model.

Usage of a cache file

First of all, as explained in Section 2.3, the main argument that led us to implement a DQL model rather than a QL model is the absence of a Q-table. Indeed, this table takes up memory space and replacing it with a neural network allows for memory savings and easier application to complex models. Therefore, one point that can be discussed is the introduction, in Section 4, of a cache system allowing to save the emission and cost values related to the states already explored by the algorithm. Indeed, this cache principle is nothing more than a table stored in an external file.

In terms of memory space, a Q-table has a size of [number of different states $\times$ number of different actions] while the cache file has a size of [number of different states $\times$ 3]. The size ratio between these two tables is therefore $\frac{\text{number of different actions}}{3}$, which can be considered as a linear ratio considering the small number of actions in the scenarios considered in this work.

Moreover, the disadvantage of DQL is that it does not always give the optimal result but rather a result close to optimality. QL, on the other hand, will always give the optimal result.

In view of these arguments, one might think that it would have been better to implement a QL model for our problem rather than a DQL model. If ESTD was running instantaneously this would have been the case but ESTD takes about 8 seconds to run (simplified version, see Section 4.1). By using a neural network,

DQL is able to learn the logic of the actions of the agent on ESTD results. Therefore it does not need to explore the entirety of the existing states, as shown in Section 4.6 when the agent reaches a final state that it had never explored. On the other hand, QL requires the exploration of all states and therefore requires an ESTD run for each of them. For this reason, it is preferable to use a DQL model for our problem despite the memory usage of the cache and a result that may not be the most optimal. This superiority of DQL over QL will become more pronounced as the scenario becomes more complex. Indeed, the size of the cache file grows less quickly than that of a Q-table and the ability of the DQL not to have to explore all the states makes it more scalable.

Bias introduced by the operator

Another point of discussion is the bias introduced by requiring the agent to either increase or decrease (and not both) the cost or the capacity. Indeed, one of the objectives of using a model based on reinforcement learning is that it is not biased as a human might be. The model using reinforcement learning will initially consider all actions equally and it is the training that will highlight the sequence of actions that are interesting for achieving the objectives.

For scenarios 2 and 3, we indirectly introduce a bias by deciding that for certain resources the price can only be decreased and not increased (see Section 4.2). For example, following the observation that a decrease in the price of diesel would not lead to a decrease in emissions and costs, we decided to remove this possibility from the agent. While this decision has the effect of reducing the training time of the model, it introduces a bias.

For the scenarios considered in this document, this bias is not a problem because these scenarios include few actions and it is possible to verify that these removed actions are not part of an optimal sequence of action. However, for scenarios of high complexity such as in the real world, a restriction of certain actions that might seem logical to a human operator could prevent the agent from taking an optimal decision sequence. Indeed, if the number of actions on which the agent can play is very large, it is difficult for a human operator to consider all the possible sequences of actions.

Impact of the decisions of the agent on the energy system

An important aspect to highlight is that the optimal series of actions obtained for each of the three scenarios have minimal impact on the energy system. Indeed, the emission reductions for all three scenarios are less than 5% (see CHAPTER 6). In order to achieve greater emission reductions, it would be necessary to increase the number of actions that the agent can take in a row, to give the agent the possibility of playing with a greater number of different actions or to increase the 10% change in resource price or capacity of each action.

Unfortunately, the time needed to train a model of this complexity makes this impossible to consider. Indeed, despite a reduction to a few hours for the DQL part of the algorithm, the ESTD part still requires a high computation time. The use of the cache file only saves time when the states have been sufficiently explored because ESTD must be run once for each state. As the number of states increases exponentially with the number of possible actions for the agent, the time needed for training rises accordingly.

It is also interesting to discuss the nature of the actions performed by our political agent. Indeed, the latter only has the possibility to play on the capacities and prices of resources and technologies through taxes and incentives. Because of his position, it would be possible for a politician to take measures that affect the habits of the citizens, for example by making public transport free. Such a measure would lower the demand for private mobility by increasing public mobility demand. These kinds of actions are difficult to implement in a scenario that can be used by our model. Indeed, they have a direct impact on the demand for energy and not on the price of resources so they require an estimate of their effect on demand.

Stochastic aspect of the real world

In this master's thesis, we have considered the prices of the different resources and technologies as fixed at the reference value of the Swiss energy system in 2011. In the real world, it is obvious that the price of these resources and technologies is uncertain. Different prices from those considered by the model may change the final energy mix. Therefore, a policy sequence that would theoretically bring emissions below a certain threshold would in reality risk not reaching that threshold due to uncertainties. The action sequences obtained by our model are not robust to uncertainties.

Conclusion

Politicians have an important role to play in the energy transition. Indeed, imposing taxes and incentives on certain resources and technologies can help the energy system to take an appropriate path towards emissions reduction. The aim of this master's thesis is to find an appropriate policy actions to apply to a country-scale whole-energy system in order to go below a certain emission target while minimizing the corresponding cost. To this end, we have implemented a DQL model that exploits the results given by ESTD (cost and emissions) and provides, for each scenario envisaged, a sequence of actions allowing to reach the emissions target.

First, we looked in detail at the theory behind the DQL model. To do this, the understanding of reinforcement learning and Q-learning is required. A comparison between QL and DQL allowed us to conclude that DQL is more appropriate for the purpose of this thesis. Then, we studied how to analyze the convergence of our model in order to determine when the model could be considered trained. The evolution of the value of the cumulative reward was used for this purpose.

Then we saw how our DQL model was implemented in practice. We presented the interest of using a simplified version of the Swiss energy system represented by ESTD as well as the way the different policy actions are discretized. We immersed our politician in three different scenarios i.e. he has different levers at his disposal that he can operate. Moreover, we improved the experience replay algorithm in order to improve the convergence of our model. Then, a first estimation of the hyper-parameters of our model for each scenario was performed.

Having obtained this model, we were able to tune it by varying four hyper-parameters to see if they have an effect on the speed of convergence. Indeed, the aim was to reduce the number of steps needed (and thus the computation time) to obtain a solution considered acceptable (see Chapter 3). We noticed that the weights of the reward functions as well as the use of a target network did not have a significant impact on the speed of convergence. However, doubling the number of neurons in the hidden layers and multiplying the decay factor by 16 compared to

the estimates made in CHAPTER 5 drastically increase the speed of convergence. After adapting these hyper-parameters, we were able to test our model on the three scenarios presented.

This tuned model provides to the politician a series of actions capable of bringing the energy system below the emissions threshold for all three scenarios. In addition, the cost is reduced by 3 to 4% depending on the scenario. This shows that the objectives set are being fulfilled.

Using a DQL model on a whole-energy system of the complexity of a country has never been done before. The DQL has already been applied to other energy-related problems, it was missing to apply it to our case study. We realized (thanks to CHAPTER 5) that with tuning, it is possible to drastically reduce the calculation time for the DQL part of the algorithm. However, if we want to consider a real case with a high number of different possible actions, the calculation time of ESTD is the limiting factor. We quickly reach exponential computation times for over-simplified scenarios that do not reflect reality.

This is why, the DQL model implemented in this thesis could be used with another model that simulates the energy system in a much faster way. Indeed, ESTD is used as a black box and any other model that can represent a country-size whole-energy system could be used. Therefore, a future path that could be considered would be to replace ESTD with a neural network whose objective is to estimate the costs and emissions of a whole-energy system. In other words, this neural network would learn to imitate the behavior of ESTD and its results. Indeed, given the ability of our neural network to generalize to states it has not explored, there is no doubt that replacing ESTD with a neural network is possible. The computation time of a neural network is almost instantaneous, this would make it possible to envisage complex scenarios that are closer to reality. This work is therefore the foundation on which improvements could be made to enable the model to obtain results suitable for the real world.

Bibliography

[1] M. B. Lindberg, J. Markard, and A. D. Andersen, "Policies, actors and sustainability transition pathways: A study of the EU's energy policy mix," *Research Policy*, vol. 48, no. 10, pp. 1–15, 2019. [Online]. Available: https://doi.org/10.1016/j.respol.2018.09.003

[2] Y. Tanaka, A. Chapman, T. Tezuka, and S. Sakurai, "Putting the process into the policy mix: Simulating policy design for energy and electricity transitions in Japan," *Energy Research and Social Science*, vol. 70, no. July, p. 101702, 2020. [Online]. Available: https://doi.org/10.1016/j.erss.2020.101702

[3] A. T. Perera and P. Kamalaruban, "Applications of reinforcement learning in energy systems," *Renewable and Sustainable Energy Reviews*, vol. 137, no. October 2020, p. 110618, 2021. [Online]. Available: https://doi.org/10.1016/j.rser.2020.110618

[4] Z. Yu and A. Dexter, "Online tuning of a supervisory fuzzy controller for low-energy building system using reinforcement learning," *Control Engineering Practice*, vol. 18, no. 5, pp. 532–539, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.conengprac.2010.01.018

[5] T. Yu, H. Z. Wang, B. Zhou, K. W. Chan, and J. Tang, "Multi-Agent Correlated Equilibrium Q(λ) Learning for Coordinated Smart Generation Control of Interconnected Power Grids," *IEEE Transactions on Power Systems*, vol. 30, no. 4, pp. 1669–1679, 2015.

[6] T. Yu, L. Xi, B. Yang, Z. Xu, and L. Jiang, "Multiagent Stochastic Dynamic Game for Smart Generation Control," *Journal of Energy Engineering*, vol. 142, no. 1, p. 04015012, 2016.

[7] F. Contino, S. Moret, G. Limpens, and H. Jeanmart, "Whole-energy system models: The advisors for the energy transition," *Progress in Energy and Combustion Science*, vol. 81, 2020.

[8] G. Limpens, S. Moret, H. Jeanmart, and F. Maréchal, "EnergyScope TD: A novel open-source model for regional energy systems," *Applied Energy*, vol. 255, 2019.

[9] Techopedia, "Linear programming (lp)," https://www.techopedia.com/definition/20403/linear-programming-lp, (accessed February 3, 2021).

[10] M. Wang, "Deep q-learning tutorial: mindqn," https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc, (accessed April 7, 2021).

[11] E. F. Morales and J. H. Zaragoza, "An introduction to reinforcement learning," *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, pp. 63–80, 2011.

[12] V. Jain, "Everything you need to know about "activation functions" in deep learning models," https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253, (accessed May 19, 2021).

[13] P. Singh, "Reinforcement learning : Deep q networks," https://spraphul.github.io/blog/RL3, (accessed April 10, 2021).

[14] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[16] J. Heaton, "The number of hidden layers," https://www.heatonresearch.com/2017/06/01/hidden-layers.html, (accessed May 4, 2021).

[17] J. TORRES.AI, "Deep q-network (dqn)-ii," https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c, (accessed May 25, 2021).

Appendices

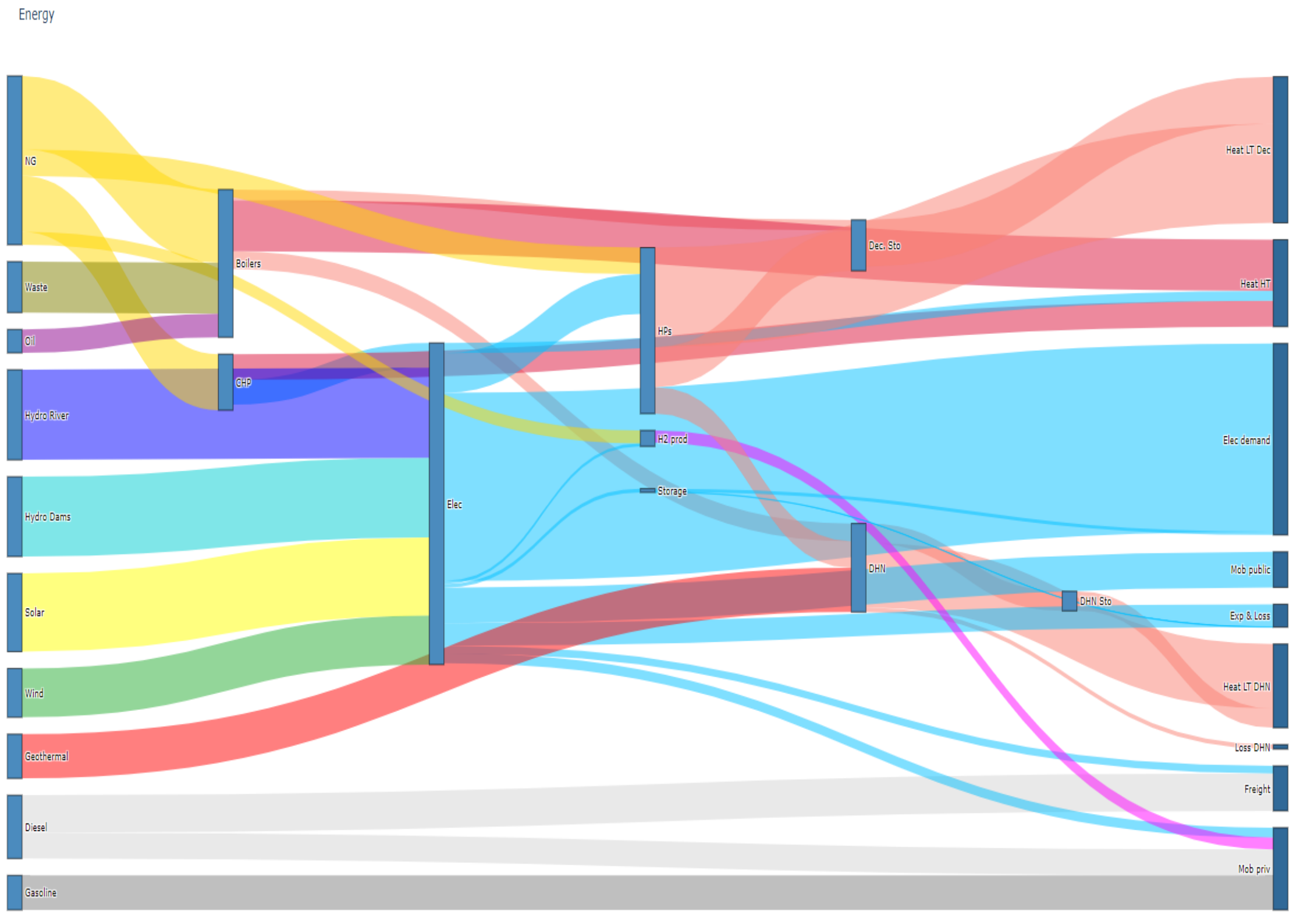
Appendix A

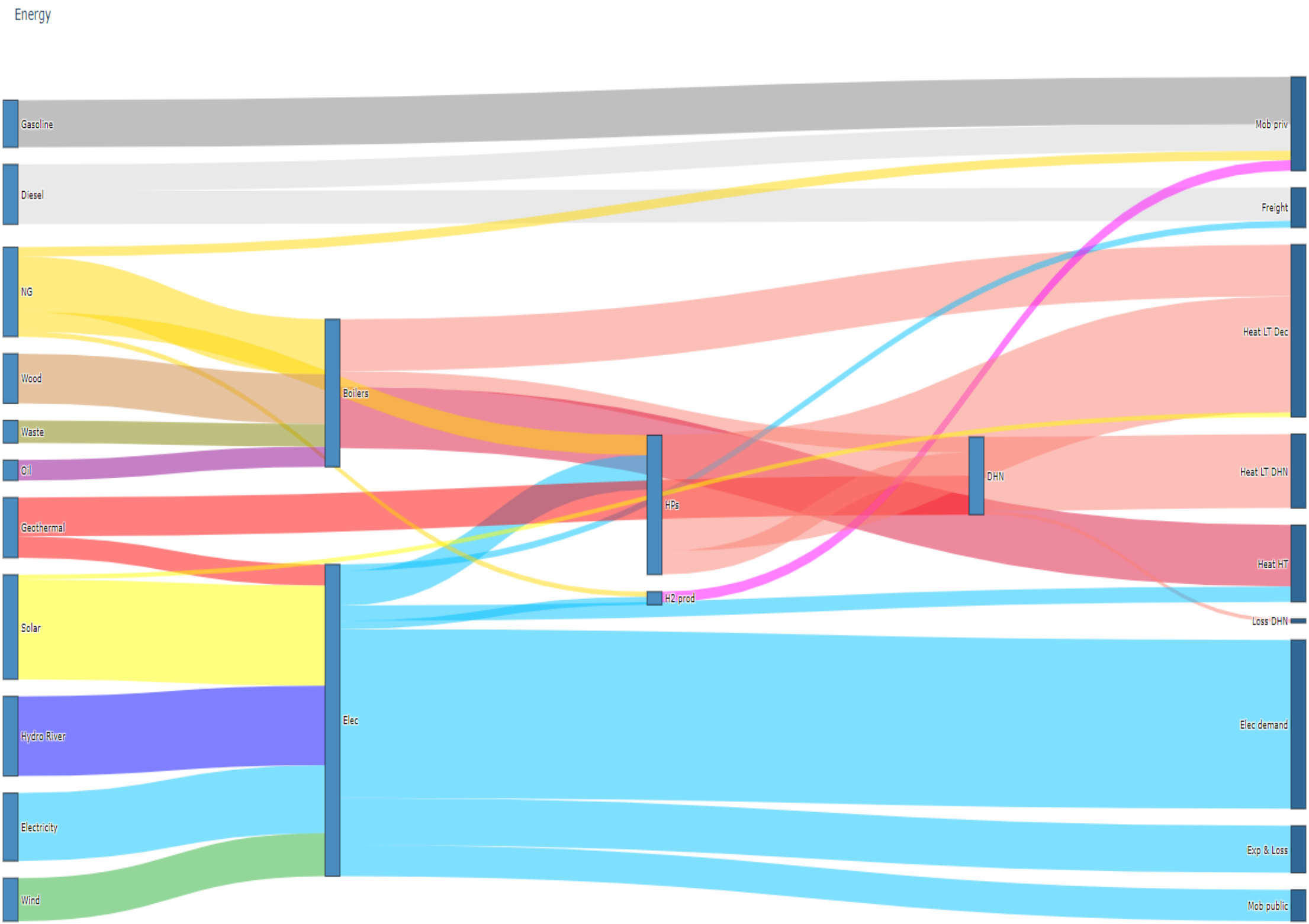
Practical implementation of the model - energy system part

	Solar	Hydro	Electricity	Wind	Wood	Waste	Oil	NG	Geothermal	Gasoline	Diesel	Total
Primary energy [TWh]	25.9	19.7	16.8	10.7	12.3	5.6	5	22.1	14.9	11.7	13.8	158.5

Table A.1: Primary energy mix of the simplified energy system

On the next two pages, the sankey diagrams of the complete energy system and the simplified one are presented.


Energy
NG
Waste
Oil
Hydro River
Hydro Dams
Solar
Wind
Geothermal
Diesel
Gasoline
Boilers
CHP
Elec
HPs
H2 prod
Storage
Dec. Sto
DHN
DHN Sto
Heat LT Dec
Heat HT
Elec demand
Mob public
Exp & Loss
Heat LT DHN
Loss DHN
Freight
Mob priv


Energy
Gasoline
Diesel
NG
Wood
Waste
Oil
Geothermal
Solar
Hydro River
Electricity
Wind
Boilers
HPs
H2 prod
DHN
Elec
Mob priv
Freight
Heat LT Dec
Heat LT DHN
Heat HT
Loss DHN
Elec demand
Exp & Loss
Mob public

Appendix B

Parameters analysis

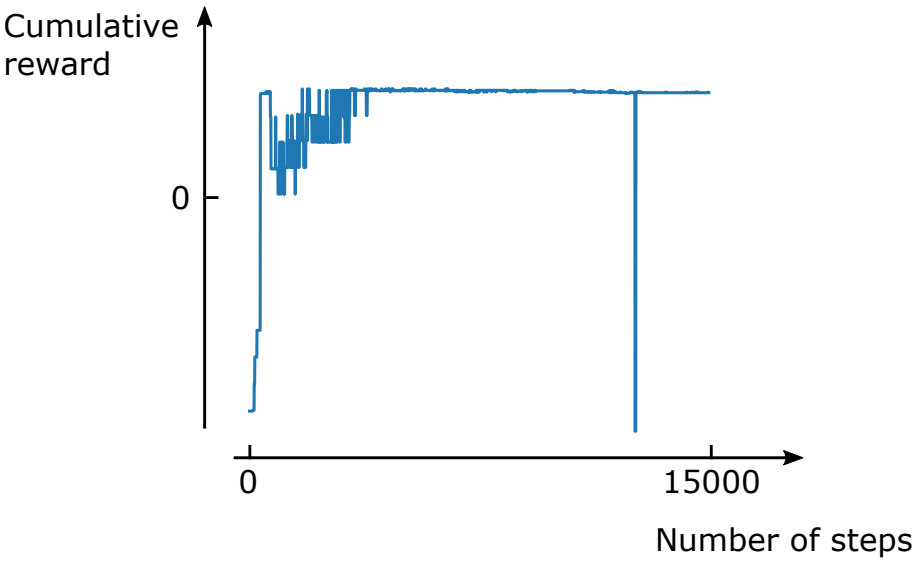


Figure B.1: Cumulative reward for the second scenario without applying the moving average

Reward function	Termination importance	Reference	Transition importance
Cost after convergence	18.26	18.26	17.92
Cost optimal solution	17.88	17.74	17.56
Emissions after convergence	27.95	27.95	28.00
Emissions optimal solution	28.19	27.71	27.41

Table B.1: Exact values of the cost and emissions for scenario 2

Reward function	Termination importance	Reference	Transition importance
Cost after convergence	18.51	18.96	18.00
Cost optimal solution	17.62	17.56	17.74
Emissions after convergence	28.49	28.04	27.82
Emissions optimal solution	27.71	27.41	27.71

Table B.2: Exact values of the cost and emissions for scenario 3

Appendix C

Impact on the energy system

	Solar	Hydro	Electricity	Wind	Wood	Waste	Oil	NG	Geothermal	Gasoline	Diesel	Total
Primary energy [TWh] reference	25.9	19.7	16.8	10.7	12.3	5.6	5	22.1	14.9	11.7	13.8	158.5
Primary energy [TWh] following actions	20.1	19.7	13.7	10.7	14.7	0	5	33.3	14.9	11.7	13.8	157.6

Table C.1: Comparison in term of resources between the reference energy system and the energy system following the politician choice for scenario 1

	Solar	Hydro	Electricity	Wind	Wood	Waste	Oil	NG	Geothermal	Gasoline	Diesel	Total
Primary energy [TWh] reference	25.9	19.7	16.8	10.7	12.3	5.6	5	22.1	14.9	11.7	13.8	159.5
Primary energy [TWh] following actions	19.2	19.7	8	10.7	17.2	0	5	42.9	14.9	11.7	13.8	163.1

Table C.2: Comparison in term of primary energy between the reference energy system and the energy system following the politician choice for scenario 2

	Solar	Hydro	Electricity	Wind	Wood	Waste	Oil	NG	Geothermal	Gasoline	Diesel	Total
Primary energy [TWh] reference	25.9	19.7	16.8	10.7	12.3	5.6	5	22.1	14.9	11.7	13.8	159.5
Primary energy [TWh] following actions	17.4	21.7	7.7	10.7	16	0	5	42	14.9	11.7	13.8	160.9

Table C.3: Comparison in term of primary energy between the reference energy system and the energy system following the politician choice for scenario 3

	Solar	Hydro	Electricity	Wind	Wood	Waste	Oil	NG	Geothermal	Gasoline	Diesel	Total
Primary energy [TWh] Scena 3	17.4	21.7	7.7	10.7	16	0	5	42	14.9	11.7	13.8	160.9
Primary energy [TWh] best solution	20.1	19.7	9.8	10.7	16	0	5	40.2	14.9	11.7	13.8	161.9

Table C.4: Comparison in term of primary energy between the energy system following the politician choice and the optimal solution for scenario 3

Appendix D

Pseudo-code and structure of the program

Algorithm 1: Deep Q-learning model applied to a whole-energy system

Initialize replay memory $\mathcal{R}$;
Initialize main neural network $\mathcal{M}$ to random weights;
Initialize target neural network $\mathcal{T}$ to weights of the main network;
Initialize TensorFlow environment $\mathcal{D}$;
Set $\epsilon = 1.0$;
for *episode = 1,* N_{steps} **do**
    Reset environment;
    Set $CR = 0$;
    Set $N_{actions} = 0$;
    while $N_{actions} <$ *Max number of actions in a row* **do**
        if *rand in [0,1]* $< \epsilon$ **then**
            Select a random action a_t;
        else
            Estimate Q-values using main network $Q_{val}(s_t) = \mathcal{M}(s_t)$;
            Select $a_t = max(Q_{val}(s_t)$;
        end
        Go to next state $(s_{t+1}, R_t) = \mathcal{D}(s_t, a_t)$;
        Append experience (s_t, a_t, R_t, s_{t+1}) to $\mathcal{R}$;
        if $N_{actions}$ *% 4 == 0* **then**
            Train $\mathcal{M}$ using ALGORITHM 2;
        end
        $N_{actions} += 1$;
        $s_t = s_{t+1}$;
        $CR = CR + R_t$;
    end
    if *episode % 10 == 0* **then**
        Copy weights from $\mathcal{M}$ to $\mathcal{T}$;
    end
    Update ϵ using Epsilon-Greedy strategy
end

Algorithm 2: Training of the main neural network $\mathcal{M}$

Replay memory $\mathcal{R}$, main neural network $\mathcal{M}$ and target neural network $\mathcal{T}$ are given in argument to this function;
if *size($\mathcal{R}$)<1000* **then**
    return;
else
    batch1 = random sample of $\mathcal{R}$ of size 108;
    $\mathcal{R}_{sorted}$ = copy of $\mathcal{R}$ sorted in descending order;
    $\mathcal{R}_{sorted,10\%}$ = first 10% of $\mathcal{R}_{sorted}$;
    batch2 = random sample of $\mathcal{R}_{sorted,10\%}$ of size 20;
    batch = append(batch1,batch2);
    foreach *experience* (s_t, a_t, R_t, s_{t+1}) *in batch* **do**
        Predict $Q_{val}(s_t) = \mathcal{M}(s_t)$;
        Predict $Q_{val}(s_{t+1}) = \mathcal{T}(s_{t+1})$;
        Using Bellman equation, update in $Q_{val}(s_t)$ the Q-value of action a_t :
            $Q(s_t, a_t) = (1 - \alpha) * Q(s_t, a_t) + \alpha * [R_t + \gamma * max(Q_{val}(s_{t+1})]$;
    end
    Using TensorFlow built-in function, fit $\mathcal{M}$ to map states s_t to updated Q-values $Q_{val}(s_t)$;
end

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**