



UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

DEPARTAMENTO DE INGENIERÍA MECÁNICA

ENERGY-OPTIMIZING FAILURE-RESILIENT
AUTOMATIC CONTROLLER FOR A WATER HEATING
SYSTEM THROUGH DEEP REINFORCEMENT LEARNING

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN
CIENCIAS DE LA INGENIERÍA, MENCIÓN MECÁNICA

ADRIÁN FELIPE RIEBEL BRUMMER

PROFESOR GUÍA:
ENRIQUE LÓPEZ DROGUETT

PROFESOR CO-GUÍA:
JOSÉ CARDEMIL IGLESIAS

COMISIÓN:
RODRIGO PASCUAL JIMÉNEZ

SANTIAGO DE CHILE

2022

**RESUMEN DE LA TESIS PARA OPTAR AL
GRADO DE:** Magíster en Ciencias de la Ingeniería,
Mención Mecánica
POR: Adrián Felipe Riebel Brummer
FECHA: 2022
PROFESOR GUÍA: Enrique López Droguett

Agente optimizador de energía resiliente a fallas para un sistema de calentamiento de agua mediante aprendizaje reforzado profundo

En el contexto climático actual, reducir el impacto ambiental de la humanidad es más importante y urgente que nunca. Esto debe lograrse sin dejar de lado las mejoras que se han conseguido durante las últimas décadas en la calidad de vida de la gran mayoría de la población mundial; mejoras que han elevado la esperanza de vida, la salud y la educación a niveles sin precedentes, y que en parte se deben a la industrialización. Por lo tanto, el objetivo de la humanidad no debiese ser la abolición de la industrialización, sino un modelo productivo que pueda mantenerse en el largo plazo, y en lo posible debe lograrse rápido. En este contexto, un modelo de producción energética libre de carbono es esencial. Una forma obvia de conseguirlo es mediante el uso de energías limpias y renovables. En adición a esto, se puede reducir la necesidad energética optimizando su consumo.

El trabajo presentado en este documento intenta abordar la optimización en el control de sistemas que usan energía de fuentes renovables. En particular, el método conocido como “Deep Reinforcement Learning” (aprendizaje reforzado profundo) se usa para entrenar a agentes autónomos que controlan el sistema de calentamiento de agua sanitaria usado para entregar agua a los camarines en el área de deportes del edificio ubicado en Beauchef 851, Santiago de Chile, el cual pertenece a la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile.

“Reinforcement Learning” es un área de aprendizaje de máquinas que estudia la optimización de tareas de control. Un agente es entrenado para ejecutar las mejores acciones posibles sobre un ambiente, con el objetivo de obtener mayores “recompensas”, las cuales son una función que depende de los efectos que las acciones del agente producen sobre el ambiente. “Deep Reinforcement Learning” es la sub-área de Reinforcement Learning que estudia el uso de redes neuronales profundas como agentes que toman las decisiones.

En el estudio aquí presentado, una plataforma simple para el entrenamiento de redes neuronales densas es desarrollada con el objetivo entrenar a agentes que controlen el sistema ya mencionado; luego, se define una función recompensa considerando las características del sistema y el objetivo del proceso de entrenamiento, que es optimizar el uso de energía mientras se provee agua caliente a los camarines. Además, los agentes se entrenan para controlar una versión del sistema sujeta a fallas de los componentes, de forma de que no se interrumpa el suministro de agua caliente en caso de falla.

Los resultados muestran un claro éxito del método presentado, tanto para optimizar el uso de energía como para manejar el sistema cuando ocurren fallas. Sin embargo, es posible que la simulación tenga demasiadas simplificaciones con respecto al sistema real, lo cual podría producir un desempeño deficiente de los agentes si éstos se pusieran en práctica con el actual avance del estudio. Por lo tanto, el siguiente paso obviamente consiste en añadir más complejidades a la simulación, lo cual probablemente llevará a la necesidad de usar redes neuronales y métodos de Deep Reinforcement Learning más sofisticados.

**RESUMEN DE LA TESIS PARA OPTAR AL
GRADO DE:** Magíster en Ciencias de la Ingeniería,
Mención Mecánica
POR: Adrián Felipe Riebel Brummer
FECHA: 2022
PROFESOR GUÍA: Enrique López Droguett

Energy-Optimizing Failure-Resilient Automatic Controller for a Water Heating System through Deep Reinforcement Learning

In the current climatic context, reducing the environmental impact of mankind is more important and urgent than ever. This must be achieved without leaving aside the important improvements that have been achieved in the quality of life of most people in the world during the last decades; improvements which have increased life expectancy, health and education to unprecedented levels, and which have been partly due to industrialization. Therefore, the goal of mankind should not be to abolish industrialization, but to find a production model that can be maintained in the long term, and this must be hopefully achieved quickly. In this context, a carbon-free energy production model is essential. An obvious way of achieving it in the short term is by the use of clean and renewable energy sources. In addition to this, the need for energy can be reduced by optimizing its consumption.

The work presented in this document tries to address the control optimization of systems that use renewable energy sources. In particular, the method known as “Deep Reinforcement Learning” is used to train an autonomous controlling agent that optimizes the performance of a sanitary water heating system that is used to supply warm water to the dressing rooms in the sports area of the building located in Beauchef 851, Santiago de Chile, which belongs to the Faculty of Physical and Mathematical Sciences (FCFM) of the University of Chile.

Reinforcement Learning is an area of Machine Learning that studies the optimization of control tasks. An agent is trained to execute the best possible actions on an environment, with the goal of obtaining the best possible “rewards”, which are a function that depends on the effects that the actions of the agent produce on the environment. Deep Reinforcement Learning is the sub-area of Reinforcement Learning that studies the use of Deep Neural Networks as the decision-making agent.

In the study presented here, a simple Dense Deep Neural Network-training platform is developed to train agents in order to control the water heating system mentioned above; then, a reward function is formulated considering the characteristics of the system and the objective of the training process, which is to optimize the use of energy while effectively supplying warm water to the dressing rooms. Moreover, the agents are trained to control a failure-subject version of the same water heating system, so that the supply of warm water is not interrupted when a failure occurs.

The results show a clear success of the method presented, both at optimizing the energy use, as well as at handling the system when failures occur. However, it is believed that the simulation that is used to model the water heating system has too many simplifications with respect to the actual system, which can lead to poor performance of the agents if they were put into practice in the current stage of the study. Therefore, the obvious next step would be to introduce more of the complexities that the actual system has into the simulation; this would probably lead to the need to use more sophisticated Deep Neural Networks and Deep Reinforcement Learning techniques.

A mi familia

Agradecimientos

Estando en la etapa final de este proceso, no puedo dejar de mencionar los importantes aportes que diferentes personas han hecho para que hoy pueda estar entregando esta tesis. Agradezco sinceramente:

A mi profesor guía, Doctor Enrique López Droguett, por haber confiado en mí y por todo el tiempo dedicado a darme consejos y a conversar los contenidos que finalmente han quedado plasmados en este trabajo. Gracias además por haberme motivado a ir más allá de lo que en un principio pensé que era posible.

A mi profesor co-guía, Doctor José Miguel Cardemil Iglesias, quien siempre tuvo la disponibilidad para evaluar mi trabajo y para hacer detallados análisis que finalmente se tradujeron en importantes aportes a esta tesis.

Al profesor Dr. Rodrigo Pascual Jiménez, miembro de la comisión evaluadora, por el tiempo dedicado a escuchar mi propuesta y por sus valiosos consejos; y al profesor Ramón Frederick González, coordinador del programa de Magíster, por su constante disponibilidad para resolver cada una de mis dudas.

A toda mi familia, especialmente a mi círculo más cercano: mis hermanos Andrea, Alex y Tito; mis padres Ricardo y Ellen; y Mariela. Ustedes son por lejos las personas más importantes en mi vida, quienes más han aportado a que hoy yo sea quien soy. Esta tesis va dedicada a ustedes.

A todas las personas que, a lo largo de mi vida, han formado parte de ella. Entre otros, me gustaría mencionar a mis compañeros y amigos del colegio, a mis amigos de la Universidad, y obviamente a la gente de la Andinia. Comenzar a mencionar nombres en este punto sería imposible, ya que indudablemente caería en el error de omitir nombres de forma involuntaria; además, una lista completa ocuparía varias páginas. Sin embargo, confío en que cada persona, sabiendo lo que ha significado para mí, sepa que es parte de esta mención.

Table of content

Chapter 1: Introduction.....	1
1.1. On our effect on the planet	1
1.2. Sustainability	2
1.3. Solar water heating	3
1.4. Policy optimization.....	4
1.5. Reinforcement Learning and Deep Reinforcement Learning.....	4
1.6. Objectives	5
1.6.1. General objective.....	6
1.6.2. Specific objectives.....	6
1.7. Structure of the Thesis.....	6
Chapter 2: Literature Review	8
2.1. Literature on solar water heating.....	8
2.2. Literature on policy optimization	8
2.3. Summary.....	10
Chapter 3: Theoretical framework.....	11
3.1. Dense Deep Neural Networks	11
3.1.1. Gradient Descent	12
3.1.2. Momentum	13
3.1.3. Backpropagation.....	14
3.1.4. Softmax activation.....	17
3.1.5. DNN initialization	19
3.2. Deep Reinforcement Learning Algorithms	19
3.2.1. Basic Concepts	20
3.2.2. Policy gradient methods and value methods	21
3.2.3. REINFORCE algorithm	22
3.2.4. Actor-Critic Algorithm.....	22
3.2.5. Q-Learning and Deep Q-Learning.....	24
3.2.5.1. Q-Values.....	24
3.2.5.2. Q-Learning	24
3.2.5.3. Deep Q-Learning algorithm	25
3.2.5.4. Double DQN.....	27
3.2.5.5. Prioritized Experience Replay.....	27

3.3. Reliability theory	28
3.3.1. Hazard rate.....	30
3.3.2. Exponential distribution	31
3.3.3. Series system	31
3.3.4. Parallel system.....	32
3.4. Discrete-time Markov chains.....	32
3.4.1. Geometric distribution.....	33
3.4.2. Steady-state probabilities.....	34
3.5. Heat pipe evacuated tube solar collectors.....	37
3.6. Heat pumps and refrigeration systems.....	38
3.7. TRNSYS.....	40
Chapter 4: Development of the training platform	42
4.1. System under study.....	42
4.2. Actions.....	44
4.3. Rewards	46
4.4. Environment state	48
4.5. TRNSYS-Python connection.....	49
4.6. Introducing stochastic failures.....	53
4.6.1. Failure rates of individual devices.....	54
4.6.2. Construction of the Markov chains	56
4.7. Pseudo-code versions of the Python Scripts.....	60
4.7.1. Initializer.....	60
4.7.2. Code to train the network	61
4.7.2.1. Main code	61
4.7.2.2. Interaction at 8.00 AM.....	62
4.7.2.3. Interaction from 10.00 AM to 8.00 PM.....	62
4.7.2.4. Interaction at 10.00 PM	63
Chapter 5: Methodology.....	64
5.1. Stages of the study.....	64
5.2. Result Analysis	64
Chapter 6: Results and Discussion	68
6.1. Comparison of DRL algorithms	68
6.2. Comparison to a non-smart-controlled baseline.....	74
6.2.1. Testing method	75

6.2.2. Results	76
6.3. Comparison of different network architectures and training hyperparameters	77
6.3.1. Comparing Different Architectures	79
6.3.2. Comparing Different Discount Factors	81
6.3.3. Comparing traditional DQN to Double DQN.....	84
6.3.4. Effect of momentum.....	85
6.4. Behavior comparison under different reward parameters	86
6.4.1. Changing the value of α_1	87
6.4.2. Changing the value of α_2	94
6.4.3. Changing the value of α_3	97
6.4.4. Effect of the α_4 parameter.....	98
6.5. System subject to failures	99
6.5.1. Training is carried out with the Markov chains of the real system	101
6.5.2. Training is carried out with planned failure cycles	108
6.5.3. Alternative Markov chains	114
6.5.4. Effect of momentum for failure-subject agents.....	117
6.5.5. Agent selection and final testing	119
Chapter 7: Conclusions.....	126
7.1. Accomplishment of objectives	126
7.2. Future work	127
8. Glossary	129
9. Bibliography	130
Annexes	134
Annexed A. Further details about the water heating system simulation	134
Annexed A.1. Detailed flow diagrams	134
Annexed A.2. Parameters of the elements in the system simulation (Types)	135
Annexed A.3. Elements (Types) with external files.....	138
Annexed A.4. Imposed Temperatures	142
Annexed B. All results of Section 6.5.	143

Figure Index

Figure 1..... 1	Figure 31..... 72	Figure 61 95	Figure 91119
Figure 2.....2	Figure 32..... 73	Figure 62 96	Figure 92121
Figure 3..... 5	Figure 33..... 73	Figure 63 96	Figure 93121
Figure 4..... 11	Figure 34..... 74	Figure 64 97	Figure 94123
Figure 5..... 12	Figure 35..... 76	Figure 65 98	Figure 95124
Figure 6..... 33	Figure 36..... 77	Figure 66 99	Figure 96124
Figure 7..... 38	Figure 37..... 77	Figure 67 102	Figure 97125
Figure 8..... 38	Figure 38..... 78	Figure 68 103	Figure 98125
Figure 9..... 39	Figure 39..... 79	Figure 69 104	Figure 99134
Figure 10..... 40	Figure 40..... 80	Figure 70 105	Figure 100134
Figure 11..... 41	Figure 41..... 80	Figure 71 106	Figure 101135
Figure 12..... 41	Figure 42..... 81	Figure 72 106	Figure 102141
Figure 13..... 42	Figure 43..... 82	Figure 73 107	Figure 103143
Figure 14..... 43	Figure 44..... 82	Figure 74 107	Figure 104143
Figure 15..... 44	Figure 45..... 83	Figure 75 108	Figure 105144
Figure 16..... 45	Figure 46..... 83	Figure 76 108	Figure 106144
Figure 17..... 49	Figure 47..... 84	Figure 77 109	Figure 107145
Figure 18..... 50	Figure 48..... 84	Figure 78 111	Figure 108146
Figure 19..... 51	Figure 49..... 85	Figure 79 111	Figure 109147
Figure 20..... 52	Figure 50..... 87	Figure 80 111	Figure 110147
Figure 21..... 53	Figure 51..... 88	Figure 81 112	Figure 111148
Figure 22..... 58	Figure 52..... 89	Figure 82 112	Figure 112148
Figure 23..... 59	Figure 53..... 90	Figure 83 113	Figure 113149
Figure 24..... 59	Figure 54..... 90	Figure 84 113	Figure 114149
Figure 25..... 65	Figure 55..... 91	Figure 85 114	Figure 115150
Figure 26..... 66	Figure 56..... 91	Figure 86 115	Figure 116151
Figure 27..... 66	Figure 57..... 92	Figure 87 116	Figure 117152
Figure 28..... 67	Figure 58..... 92	Figure 88 116	Figure 118153
Figure 29..... 71	Figure 59..... 93	Figure 89 117	Figure 119153
Figure 30..... 71	Figure 60..... 94	Figure 90 118	

Table Index

Table 1	46	Table 24.....	101
Table 2	56	Table 25.....	109
Table 3	56	Table 26.....	110
Table 4	57	Table 27.....	118
Table 5	59	Table 28.....	120
Table 6	59	Table 29.....	120
Table 7	59	Table 30.....	122
Table 8	68	Table 31.....	123
Table 9	69	Table 32.....	135
Table 10	69	Table 33.....	135
Table 11	70	Table 34.....	136
Table 12	75	Table 35.....	137
Table 13	78	Table 36.....	138
Table 14	79	Table 37.....	139
Table 15	81	Table 38.....	139
Table 16	83	Table 39.....	139
Table 17	84	Table 40.....	140
Table 18	86	Table 41.....	140
Table 19	87	Table 42.....	140
Table 20	94	Table 43.....	142
Table 21	98	Table 44.....	142
Table 22	99	Table 45.....	142
Table 23	100		

Equation Index

Equation 1..... 12	Equation 32..... 18	Equation 63 29	Equation 94 36
Equation 2..... 13	Equation 33..... 18	Equation 64 29	Equation 95 36
Equation 3..... 13	Equation 34..... 18	Equation 65 29	Equation 96 36
Equation 4..... 13	Equation 35..... 19	Equation 66 29	Equation 97 36
Equation 5..... 13	Equation 36..... 19	Equation 67 29	Equation 98 37
Equation 6..... 14	Equation 37..... 19	Equation 68 30	Equation 99 39
Equation 7..... 14	Equation 38..... 20	Equation 69 30	Equation 100 ... 39
Equation 8..... 14	Equation 39..... 20	Equation 70 30	Equation 101 ... 39
Equation 9..... 14	Equation 40..... 20	Equation 71 30	Equation 102 ... 39
Equation 10..... 14	Equation 41..... 21	Equation 72 30	Equation 103 ... 40
Equation 11..... 15	Equation 42..... 22	Equation 73 30	Equation 104 ... 40
Equation 12..... 15	Equation 43..... 23	Equation 74 31	Equation 105 ... 46
Equation 13..... 15	Equation 44..... 24	Equation 75 31	Equation 106 ... 47
Equation 14..... 15	Equation 45..... 24	Equation 76 31	Equation 107 ... 47
Equation 15..... 15	Equation 46..... 24	Equation 77 31	Equation 108 ... 47
Equation 16..... 15	Equation 47..... 25	Equation 78 31	Equation 109 ... 47
Equation 17..... 15	Equation 48..... 25	Equation 79 32	Equation 110 ... 47
Equation 18..... 15	Equation 49..... 25	Equation 80 32	Equation 111 ... 48
Equation 19..... 15	Equation 50..... 25	Equation 81 32	Equation 112 ... 48
Equation 20..... 16	Equation 51..... 25	Equation 82 33	Equation 113 ... 68
Equation 21..... 16	Equation 52..... 26	Equation 83 33	
Equation 22..... 16	Equation 53..... 26	Equation 84 33	
Equation 23..... 16	Equation 54..... 26	Equation 85 34	
Equation 24..... 16	Equation 55..... 27	Equation 86 34	
Equation 25..... 17	Equation 56..... 27	Equation 87 34	
Equation 26..... 17	Equation 57..... 27	Equation 88 34	
Equation 27..... 17	Equation 58..... 28	Equation 89 34	
Equation 28..... 17	Equation 59..... 28	Equation 90 35	
Equation 29..... 17	Equation 60..... 28	Equation 91 35	
Equation 30..... 18	Equation 61..... 28	Equation 92 36	
Equation 31..... 18	Equation 62..... 29	Equation 93 36	

Chapter 1: Introduction

1.1. On our effect on the planet

The biochemical processes that take place on Earth contribute to determine its climatic conditions. The carbon cycle, for example, is the process by which carbon is exchanged between living beings, the atmosphere, the soil, the hydrosphere and fossil reservoirs [1]. It is widely accepted that higher carbon dioxide (CO₂) concentrations in the atmosphere can cause an increase in the Earth's mean temperature due to the capacity of CO₂ molecules to absorb infrared radiation that would otherwise be radiated by Earth into space. This is called “greenhouse effect” and is produced by CO₂ and many other “greenhouse gases” (GHG), e.g. methane (CH₄, another carbon-based gas) and water vapor (H₂O) [2].

There is also great evidence that at this moment, human activity is breaking the balance of the carbon cycle by releasing an additional amount of fossil carbon reserves into the atmosphere; mankind has been doing this to power its technological development for more than 200 years since steam power became popular at the beginning of the 19th century. This idea is supported by studies conducted on ice core samples which contain atmospheric air from thousands and even millions of years ago; these studies allow to determine the atmospheric concentrations of CO₂ in the past. Figure 1 [3] shows that, at least for the last 800,000 years, the atmospheric CO₂ concentration has never been as high as today. It is also remarkable that the magnitude and rate of change produced in the last years is much greater than that of any previous change in the time span considered.

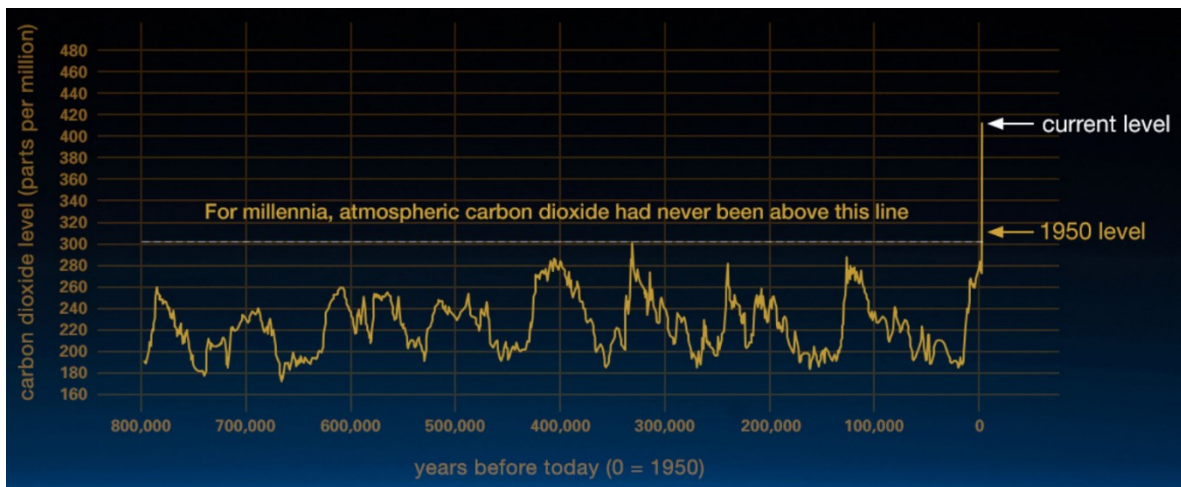


Figure 1: Atmospheric CO₂ levels during the last 800,000 years.
Source: NASA [3]; Data from: Luthi, D. et al. 2008; Etheridge, D.M. et al. 2010;
Vostok ice core data/J.R. Petit et al.; NOAA Mauna Loa CO₂ record

The results shown in Figure 1 are consistent with direct measurements of the Earth's temperature. According to data of NASA, the years 2016 and 2020 are tied as the two warmest years on record, with a global mean temperature 1.02°C higher than the 1951-1980 mean baseline [4]. Figure 2 shows a map of mean temperature differences between the 2016-2020 period and the baseline. The differences reach a maximum value of around 2°C in the red areas of the map.

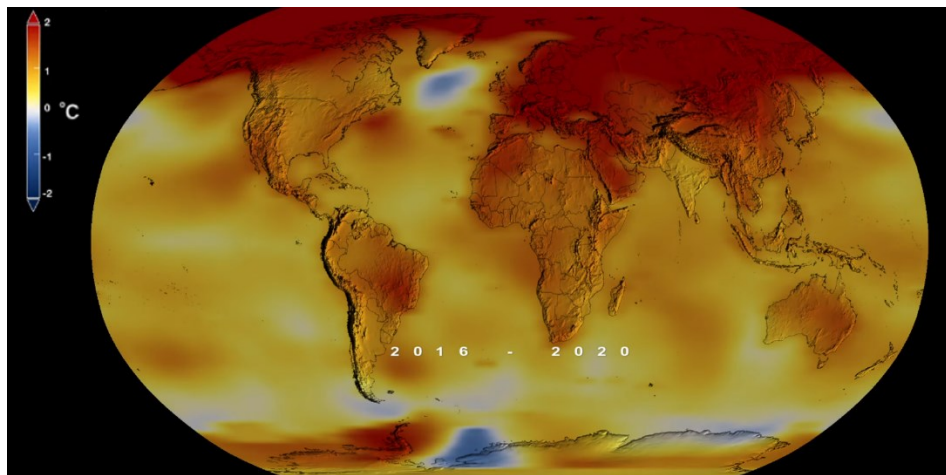


Figure 2: Temperature differences between the 2016-2020 period and the 1951-1980 baseline
 Source: NASA's Scientific Visualization Studio/Lori Perkins/Robert B. Schmunk [4]

The global temperature rise is also believed to be causing positive feedback loops that are making the rate of change even faster [5]. Warmer oceans have less capacity to dissolve CO₂, thus they release it to the atmosphere, increasing the effect. Water evaporation also increases, and a warmer atmosphere can hold more water vapor, which is a GHG as well. Melting ice releases trapped carbon dioxide and methane. Ice sheets also reflect solar radiation far better than soil, rock or liquid water; if the area covered by ice decreases, more energy from the sun is absorbed by Earth.

There is uncertainty on the future of climate, mostly because it largely depends on the decisions that mankind makes now. A paper by O'Neill et al. [6] tries to explore different scenarios from the point of view of the decisions that are made on a global scale in the coming years. The most pessimistic scenario presents a global mean temperature 5°C higher than before the industrial revolution by the end of this century; this was obtained by assuming a fivefold increase in the use of coal. This result has been criticized [7] as unrealistic, but the authors claim that they only want to present several scenarios and not predict the future. The same study is also fairly optimistic in the good-case scenarios.

It is certainly difficult (actually impossible) to predict and specify the exact path of decisions that have to be made in order to get to a global-level arrangement that leads to a systematic and orderly reduction of the use of fossil fuels. (If the future were so easy to predict based on current decisions, all the problems of the world would have already been solved. Mankind is clearly a chaotic system). But the fact that predicting the future is hard should not dissuade us from trying to make it better, and an obvious way to start is by reducing our environmental impact.

1.2. Sustainability

As already discussed, human activity is causing great changes on the planet, especially since the industrial revolution of the late 18th century. But the problems discussed above are only a tiny part of all the environmental problems that have been attributed to humans. Nowadays, concepts like global warming, ocean acidification, rising sea levels, deforestation, desertification, droughts, water pollution, plastic pollution, air pollution and mass extinction are widely accepted in the scientific community, and there is great consensus that human activity is causing, or at least contributing to, all of these problems. In recent years, this has caught the attention of the whole planet, encouraging the formation of organizations, political parties and movements that have the fight against climate change as their main motivation.

In order to understand the issue, it is fundamental to keep in mind that *any* human activity has an impact on the environment (even for the essential activity of being alive, one needs to eat, which produces an impact). Therefore, a straightforward solution that some people may advocate for is to reduce the rate of human activities in order to reduce their impact; in other words, people should reduce their consumption levels in order to reduce the impact of their consumption. This point of view could be defended by considering, as an example, the level of fashion consumption and its recent growth in developed countries. It could be argued that many people are consuming more clothes than they actually need, and due to the great impact that this industry has [8], it would be possible for people in rich countries to make a positive change by changing their habit of buying clothes.

However, there are many other parts of the world where most people have not yet achieved the minimum level of consumption that is necessary to lead a good and healthy life. Given that up to now, there are still entire countries which have not achieved a life expectancy at birth of 60 years [9], it could also be argued that many people in the world actually need to *increase* their consumption up to a point where they have access to basic services like drinking water, medical attention, electric energy, education, reliable food, hygiene, communications, transport and security.

From this, it can also be concluded that mankind cannot indefinitely reduce its consumption level. There is a minimum life standard that should be available for all people on Earth; therefore, technology needs to reach *at least* a way to sustainably manage *that* level of consumption.

The term “sustainability” refers to the idea of a production model that could be maintained in the long term without shattering the fragile balance of the Earth’s biosphere; this involves, among other things, stopping the pollution of rivers, oceans and the atmosphere, and the loss of forests. People who may be affected by the presence of industrial or agricultural activities should be considered as well. There are different ideas that are slowly being put into practice with the aim of achieving this goal; some of them are renewable energy sources, better energy management, industrial wastewater treatment and recycling. A long-term solution for the energy production issue may be nuclear fusion energy, although its commercial use might not be available at time in order to avoid the effects of global warming.

The task of avoiding future environmental damages and fixing already done damages is not easy and will require much of the human capacity during the following years to be accomplished. New technologies need to be developed and already existing technologies need to become more efficient and cheaper in order to be a viable option. The main goal of this document is to make a small contribution in that direction.

1.3. Solar water heating

The most direct way of addressing the problem of global warming is by reducing GHG emissions. In Section 2.1 it will be quantitatively argued that the use of solar energy to heat water (solar water heating, SWH) can considerably contribute to this goal. This can be done in contexts ranging from domestic to industrial purposes. Even if the desired water temperature cannot be reached with solar energy alone, this method can be used to partially heat the water flow; in this case the use of fossil fuels may not be totally avoided but it can be greatly reduced. In most SWH systems, this is the most likely situation because solar energy is intermittent; therefore, there will be moments when thermal energy is needed but not enough solar energy is available.

1.4. Policy optimization

The limitation of solar energy just mentioned can be at least partially compensated by improving the control policy of the SWH system; this means, implementing an automatic controller which operates the system in such a way as to improve the use of sustainable energy sources, thus reducing GHG emissions.

In order to achieve such a performance improvement, thermal energy storage (TES) is a minimum requirement. In the case of water heating, a TES system consists of storage tanks which, because of good thermal isolation properties, are able to hold warm water for long periods of time. A basic policy could be to store warm water at moments of high solar energy availability, in order to use it at moments of low energy availability and high demand.

However, many heating systems require a more complex policy because their performance is subject to many variables; in such cases, the optimal control policy may be not so easy to determine, and Machine Learning algorithms become a good option. The area of Machine Learning that focuses on control tasks and optimal decision making is called “Reinforcement Learning”; the basic concepts of this topic are discussed in the next section.

1.5. Reinforcement Learning and Deep Reinforcement Learning

The concept of Reinforcement Learning (RL) is quite simple, although the objective can be very hard to achieve, and the algorithms to do it are still under development ([10], [11], [12] and [13]). RL is an independent area of Machine Learning that is intimately related to optimal control and decision making. In contrast to supervised learning, which is based on learning from labels, RL is based on learning from trial and error.

The objective of RL is to train an *agent* to interact with an *environment*. The environment is usually a time-dependent simulation that can be influenced by the agent; i.e., the agent has some degree of control over the environment. In order to achieve this agent-environment interaction, the environment simulation is divided into *time steps*; on each time step, the agent is allowed to execute an *action* on the environment. The actions are chosen by the agent from a pre-established set of possibilities. The objective of the RL process is that the agent, without prior knowledge about the environment or about the effect of the actions that it executes, learns the best action to execute on each time step, only by interacting with the environment. To define how good the actions are, a *reward* function is defined (it must be a real number). For every executed action, the agent gets a reward on the next time step. The final goal of the training process is that the agent maximizes the rewards it receives. This is the core concept of RL.

To make a decision about which action is the best, on every time step the agent receives a certain amount of information from the environment which is called *observation*. Observations may contain all kinds of information about the current, past, or future conditions of the environment; however, they might not contain all the information that is necessary to define the internal state of the environment. As already discussed, the goal is that the agent learns what action to execute on each interaction (or time step) in order to maximize the rewards it receives, basing the decision on the information that it has about the environment. The agent does not have to base the decision only on the current observation, but it may also take advantage of the history of previous actions, rewards and observations. The basic interaction process is shown in Figure 3.

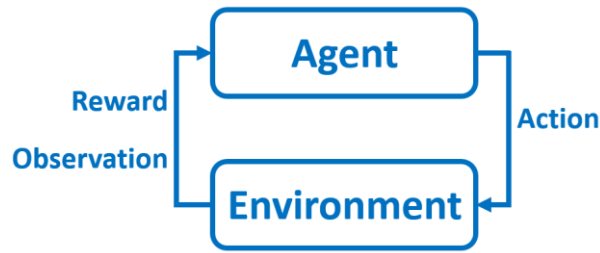


Figure 3: Interaction between the agent and the environment in RL

Training a smart artificial agent to play a videogame is a good example of a concrete situation where RL may be needed (actually, many important breakthroughs in RL have been done by experimenting on videogames; the paper by Mnih et al. [14] is a good example that will be detailed in Section 3.2). In that particular case, the observation would be the current screen image. The possible actions obviously depend on the game, but they could be for example: going forward, turning left, turning right, jumping, shooting, etc. (in the case of this study, only discrete and finite action sets will be considered). Depending on what the goal of the game is, the agent may take advantage of “remembering” previous screen images or actions that it has previously executed, i.e. the history of previous interactions. The reward function could be the points received while playing. What is important about the reward function is that the agent will always try to increase it; therefore, the reward function should completely represent what the agent is supposed to do. In other words, one has to formulate a reward function that satisfies the condition: “the higher the reward, the better”.

Deep Reinforcement Learning (DRL) is a family of methods to train a deep neural network (DNN) as the controlling agent. Here is why this may be necessary: in the case of a very simple environment, namely, an environment with few possible observations and for which the last observation is enough to determine the best action, it may be feasible to learn the best action for every possible observation, and then to store this information in a table. However, in practice, most environments have continuous observation spaces defined by several variables, thus this is not possible [14]. In this case, a feasible method to determine a good policy is to use a DNN, which evaluates the useful information as input and determines which action is the best as output. The algorithms used to train the agents (DNNs) shall be detailed in Section 3.2.

1.6. Objectives

The study presented in this document aims to create an autonomous controlling agent for a water heating system with availability of solar energy and other low-consuming heating devices.

The heating system used for the study was inspired by the system which heats the water for the dressing rooms of the sports area of the building located in Beauchef 851, Santiago, Chile. The building belongs to the Faculty of Physical and Mathematical Sciences (FCFM) of the University of Chile. In addition to solar power and thermal energy storage, the system uses heat recovery from a water chiller and air-water heat pumps.

Roughly speaking, the controlling agent controls the system by deciding which heating devices to turn on and which to turn off, in order to reduce the electric consumption of the system while fulfilling the task of delivering warm water to the dressing rooms. The agent has some instances to make this decision during each day, and the heating devices remain at the states decided by the agent until a new action is required.

The system is simulated with the TRNSYS software (Transient System Simulation Tool) [15], which is a widely used software for the simulation of all kinds of transient systems, and it is

especially demanded to simulate thermal systems and other energy-related systems. For this reason, it is also popular for the implementation of projects related to renewable energy sources.

To train the controlling agent, DRL is used; therefore, the control policy is determined by a DNN. In order to create the training platform, a connection between the TRNSYS software and the Python programming language must be established. As will be explained in Section 4.5, the connection between both programs made it unfeasible to use regular deep learning libraries for the development of the controlling agents; for this reason, the DNNs were programmed by using basic features of Python.

In addition to the basic task of controlling the system to deliver warm water while using less energy, it will be studied whether the controlling agent is able to cope with failures of the system. In this case, the objective is to train the agent to fulfill its task of delivering warm water when some of the heating devices are forcibly taken out of operation at random moments.

1.6.1. General objective

The main objective is to develop a platform that, by the use of DRL techniques, is able to train smart agents to control a water heating system, even at moments when parts of the system are out of operation due to a failure.

1.6.2. Specific objectives

The specific objectives of this study are:

1. Establishing a connection between the TRNSYS software and the Python programming language. The Python code has to be able to transmit decisions to the TRNSYS simulation, regarding which devices are used. In addition to this, the code must receive results from the simulation, use them to make decisions and impose these decisions on the simulation.
2. Showing that an effective training process of the DNNs can be achieved in a basic programming language, without the use of specialized Deep Learning libraries.
3. Defining a reward function that fulfills the condition of producing a desirable behavior of the smart agents as they try to maximize it.
4. Analyzing the training results when the hyperparameters of the training algorithm are changed. Different neural network architectures are tested as well.
5. Analyzing how the behavior of the controlling agents changes as the reward function is modified. Modifying the reward function is equivalent to changing what the agents are supposed to achieve.
6. Training agents to fulfill their task when the devices of the system are subject to random failures.

1.7. Structure of the Thesis

This thesis is structured as follows:

On Chapter 2, a review of related studies is made with the aim of analyzing the potentials and opportunities of the study that is being proposed.

On Chapter 3, the theoretical framework is discussed. Here, the mathematical foundations of the methods used to train the autonomous agents are presented, along with other concepts that are necessary to understand the study.

On Chapter 4, the characteristics of the water heating system that is used for the study are detailed, and then a step-by-step development of the training platform is presented.

On Chapter 5, the experimental methodology is presented. First, the stages of the study are detailed, and then the methods to analyze the results are explained.

On Chapter 6, the results of the conducted simulations are presented and discussed.

On Chapter 7, the conclusions that can be inferred from the results and from the study in general are mentioned, and the opportunities for future work on the subject are mentioned.

Chapter 2: Literature Review

2.1. Literature on solar water heating

According to data of the agency “Our World in Data”, 84.33% of the global amount of energy consumed in the year 2019 was produced by fossil fuels (oil, coal and gas) [16]. On the other hand, energy production accounts for around 75% of the GHG emissions produced by human activities [16].

Avoiding current and future GHG emissions makes it necessary to reduce the energy consumption as well as to increase the use of sustainable energy sources. In this context, the use of SWH is an important topic to take into account. Artur et al. [17] (2020) conducted a study in Maputo, Mozambique, concluding that the use of SWH can lead to more than 65% reduction on the electricity demand for domestic water heating (DWH). The study considered 700 households in 28 neighborhoods within the city of Maputo. The two main energy sources used for DWH were electricity and biomass, with 46% and 41% of the total energy demand for DWH respectively. The study also notes that the total electricity demand has increased an average of 9% annually during the last 15 years, due to the movement of people from rural areas into cities, leading to electricity shortages and big efforts to build new power plants. Therefore, the development of renewable energy sources could be even more important in developing countries because it can lead to reductions on environmental impact by slowing down the necessity of new power plants, and it can improve the availability of electricity.

In the case of Canada, Aguilar et al. [18] concluded through simulations that DWH represents the second-largest energy end-use by Canadian households, after space heating, with 21.7% of the domestic energy demand. Despite the fact that more than 78% of the electricity in Canada is produced with low carbon technologies (Dolter, Rivers [19]), 59% of the energy used for DWH is produced by natural gas [18], so there is potential of reducing GHG emissions by implementing domestic SWH systems. However, it is important to consider that the levels of solar radiation in Canada are considerably lower than in Maputo, so the benefits in equal conditions are lower. Jahangiri et al. [20] found that it was feasible to reduce 35% of the annual energy demand for space heating and DWH with solar collectors in the city of Regina, Canada. For the study, a flat plate collector with an area of 40m² was considered to heat a space of 80m² and to supply 110L of water at 60°C per day. The study was conducted in other Canadian cities as well, with lower results in terms of the percentage of energy supplied by solar energy.

2.2. Literature on policy optimization

One of the disadvantages of solar energy (and wind energy as well) is its intermittence, which leads to serious difficulties if one wants to not only reduce their energy consumption but to fully depend on renewable energy sources. If mankind wants to achieve a carbon-free future based on renewable energy sources, this has to be the final goal.

In this context, a step towards that goal could be trying to optimize the use of renewable energy and to maximize its exploitation. In the case of SWH, this can be achieved by the use of thermal energy systems (TES) and a good system policy. The “policy” is the set of “rules” that the automatic controller follows in order to achieve certain goals like reducing the energy cost, reducing GHG emissions or increasing the use of renewable energy sources.

Saloux and Candanedo [21] presented a rule-based control strategy in a Canadian district with SWH and TES systems, in which 34% reduction in energy costs and 29% reduction in GHG emissions were achieved in comparison with the traditional control strategy. Another study published by Tian et al. [22] also argues for the potential of solar district heating systems with smart thermal grids in Denmark. Although the levels of solar radiation in Denmark are relatively low, the study concluded that the integration of solar energy with smart control systems is competitive because of different factors: the low price of the land used for the collectors, high efficiency and high reliability of the collectors and high taxes on natural gas. Besides, 64% of Danish households are already part of district heating systems, which makes it easier to integrate solar energy into the systems.

With complex thermal systems, it may be difficult to determine an optimal strategy; in this case, RL and DRL become good options. RL and DRL algorithms have proved to be effective in different kinds of complex control tasks. Mullapudi et al. [23] presented a study where smart agents were trained to control stormwater systems by using DRL. In the study, the controlling agent gets information about flows in the system and is able to control valves in order to deviate the flows. The study argues that the DRL-trained agent significantly outperforms the uncontrolled systems. This could bring benefits like avoiding floods and reducing the need for building new drain systems. In many places, the cost of building new infrastructures is prohibitive and occasional floods are unavoidable, so these smart-controlled systems become an option to take into account.

The paper of Yang et al. [24] presents a smart controlling agent for a wind energy farm, trained by using DRL. The main goal of the study was to enhance the economic viability of this type of energy, with an energy storage system that is part of the farm, and the option of purchasing energy from external reserves. By using energy price predictions and wind availability predictions, which are generated by recurrent neural networks, the agent has to decide on the charge/discharge schedule of the energy storage system, in order to optimize the revenues of the farm. The advantage of the DRL method is that no assumptions on the probability distributions of energy price and wind availability had to be made.

In the study of Nakabi and Toivanen [25], DRL methods were used to train agents in order to optimize the performance of a microgrid that includes a wind turbine and an energy storage system. Different DRL algorithms were tested, and remarkable differences between the performances of the training methods were found. On the other hand, Lu et al. [26], presented a DRL approach to optimize the performance of microgrids by including the option of trading energy between different microgrids. Based on projections of future energy generation and future demand, the agent determined the optimal energy trading policy, reducing the overall dependence on external power plants.

In the field of optimal operation of thermal systems, Du et al. [27], Gupta et al. [28] and Brandi et al. [29] present DRL approaches to optimize the performance of Heating, Ventilation and Air-Conditioning (HVAC) systems. In the study of Lissa et al. [30], an agent that controls an integrated system of HVAC and DWH was presented. The heat source for both purposes is a heat pump, which has the option of using solar energy from photovoltaic panels; for this reason, the availability of solar energy had to be taken into account as well.

In the paper of Gao et al. [31] a Deep Reinforcement Learning technique is applied to optimize the control of an HVAC system, prioritizing the comfort of the occupants of the building and the energy consume of the system. Like this thesis, Gao et al. [31] used the TRNSYS software for the simulation of the system and the Python programming language for the development of the neural networks, but they also used the MySQL software as an interface between both programs; this

enabled them to use the Pytorch library for the development of the neural networks. The results show that the comfort level of the occupants can be enhanced along with the energy efficiency of the system by the use of DRL-trained agents.

The study on RL made by Correa et al. [32] was conducted on the same system that is used for the study presented in this document. In the study of Correa et al., the agent is able to control the operation of two heating systems: heat recovery from a water chiller and solar thermal collectors. The agent is allowed to take actions on the system three times a day. With other assumptions, a total number of 20 possible paths that the agent can follow each day are obtained. In addition to this, 5 KPIs are created based on indicators like the use of clean energy and energy consumption of the system. With this, the reward function is defined by giving different “weights” to each KPI. Then, the optimal paths, i.e. the paths which yielded the highest rewards, were determined as the weights of the KPIs were modified.

In the field of failure resilience, Dooraki and Lee [33] developed an autonomous controller for a quadcopter which was also capable of controlling the quadcopter when one or two of the rotors fail. No more than one neural network was necessary to achieve the objective; this means that the same controlling agent was capable of fulfilling the task on normal conditions as well as adapting to abnormal conditions.

2.3. Summary

The aforementioned studies were cited in order to show the potentials as well as the opportunities of the topics that are addressed in this thesis.

By taking the sources [16], [17], [18], [19] and [20] into account, it is possible to argue for both the economic viability of SWH systems and their potential of reducing greenhouse gas emissions. Studies [21] to [32] show the capacity of policy optimization methods to enhance the performance of different types of systems, including SWH. [23] to [32], in particular, show the potential of Reinforcement Learning and Deep Reinforcement Learning methods to achieve this objective. The source [33] also shows the ability of these methods to operate systems which are subject to failures.

The paper of Gao et al. [31] is similar to the study presented in this thesis because the same software was used for the simulation of the controlled system, and Deep Reinforcement Learning was used to enhance the energy efficiency of a system as well. However, Gao et al. used the MySQL software as an interface between TRNSYS and Python. One of the goals of this thesis is to show that an effective training platform for deep neural networks can be achieved without the use of specialized Deep Learning libraries, and therefore, the use of an interface between TRNSYS and Python is not necessary. This result could be useful for the later development of DRL-trained agents in environments where the access to Deep Learning platforms is not possible.

Another field that is very under-developed is the application of DRL techniques for the training of failure-resilient agents. Dooraki and Lee [33] make a great contribution in that direction, but none of the studies that were found does something similar with thermal systems.

In the last years, Machine Learning methods have become more and more popular in industrial applications, but Reinforcement Learning remains largely unknown, and as has been shown, it has the potential of largely increasing the efficiency of many different kinds of processes. For this goal to become a reality, more academic research should concentrate in this topic. This thesis aims to be a small contribution to the development of Reinforcement Learning techniques, especially in Chile and in the Faculty of Physical and Mathematical Sciences of the University of Chile.

Chapter 3: Theoretical framework

3.1. Dense Deep Neural Networks

Deep Neural Networks (DNNs) are mathematical functions that evaluate their inputs and generate outputs through successive operations, which gives them the ability to use several “levels of abstraction”, one over the other. Their name is due to the fact that in some ways they resemble the functioning of the human brain; they have even been used to better understand how the brain processes the information that it receives (Yamins, DiCarlo 2016 [34]; Walker et al. 2019 [35]). Their popularity comes from the fact that they can be “trained” by large amounts of data to find trends and correlations which are hard or impossible to find by humans.

Even though there are various types of DNNs (e.g. Dense, Convolutional, Recurrent), only Dense DNNs are used in this study, so only they will be discussed (from now on, “DNN” or simply “network” will be used to denote a Dense DNN). A common way of illustrating these networks is by diagrams like the one shown in Figure 4, which shows how the weights (lines) connect the neurons (circles) of the network. Note that the network is constructed with “layers” (input, hidden and output; the network can be made “deeper” by adding more hidden layers) which give the network its “levels of abstraction” mentioned above. Each layer can “discover” features in the results yielded by the previous one. In the specific case of Dense DNNs, all neurons of one layer are connected to all neurons of the next layer by the weights, as shown in Figure 4.

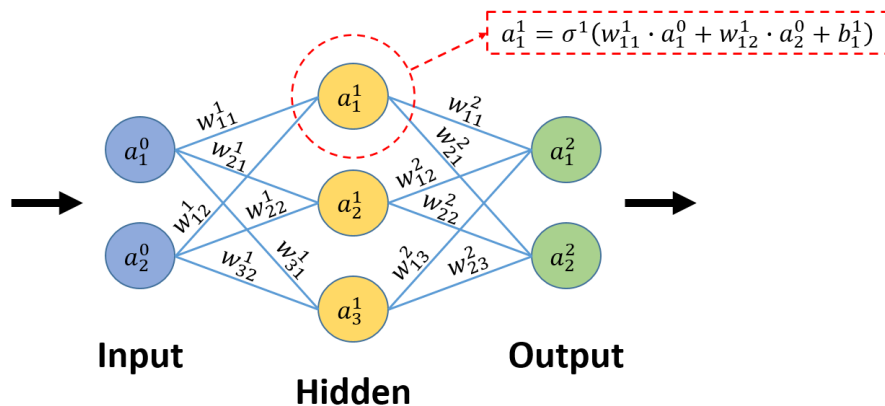


Figure 4: Visual illustration of a Dense Deep Neural Network

In the coming sections, a detailed description of the mathematical foundations of Dense DNNs will be made. This is because, as will be discussed in Section 4.5, no regular Deep Learning libraries are used for the study; therefore, the DNN training platform have to be developed with basic Python features. For these explanations, the representation and nomenclature shown in Figure 5 will be considered, where a DNN is modeled as a sequence of matrix-vector operations. The figure shows a network with three layers, but this can be generalized for more layers by adding layers at the right side.

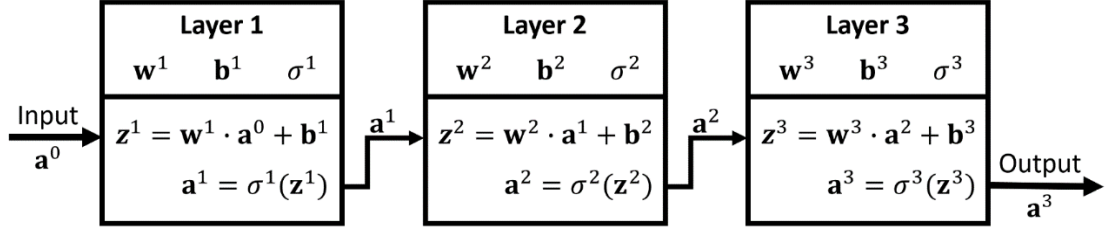


Figure 5: Dense DNN as a sequence of matrix-vector operations

In the representation of Figure 5, the i -th layer is defined by a weight matrix \mathbf{w}^i , a bias vector \mathbf{b}^i and an activation function σ^i . Note that the numbers over the letters indicate the number of the layer and not an exponent. In this representation, unlike the one of Figure 4, the weights are not connecting layers but they are part of the layers. Because of this, there is no need for an input layer; instead, the first layer receives the input as a vector \mathbf{a}^0 .

Under this representation, a DNN works as follows: the i -th layer receives an activation vector \mathbf{a}^{i-1} , either from the input or from a previous layer; then the activation vector is multiplied by the weight matrix of the respective layer \mathbf{w}^i and then the bias vector \mathbf{b}^i is added, obtaining a vector that here is denoted as \mathbf{z}^i . Finally, the activation function of the i -th layer σ^i is applied to \mathbf{z}^i , obtaining the activation vector of that layer \mathbf{a}^i . That vector is either passed to the next layer or delivered as output of the DNN. It will be assumed that the vectors \mathbf{z}^i and \mathbf{a}^i have the same number of components, i.e. the function σ^i does not change the number of components of the vector. From this, it is easy to conclude that the dimensions of the matrix \mathbf{w}^i are defined as follows: the number of columns of \mathbf{w}^i is equal to the size of the vector \mathbf{a}^{i-1} (this vector can be the output of a previous layer or the input of the DNN if $i = 1$) and the number of rows of \mathbf{w}^i is equal to the number of neurons of the layer i , i.e. the size of the vector \mathbf{a}^i .

The term “parameter” will be used to refer to any component of a weight matrix w_{jk}^i or a component of a bias vector b_j^i . The term “hyperparameter” will be used for values which are external to the network and have to be set by the programmer before the training process, like the learning rate, the momentum factor, etc. The concepts just mentioned will be clarified in the coming sections.

3.1.1. Gradient Descent

The DNN is initialized with random values in its weight matrices, and in the case of this study, the biases are initialized as zeros, so in order for the network to make precise predictions, these values must be corrected; namely, the DNN must be trained. “Gradient Descent” is the name given to the algorithm that achieves this. The weights and biases can be modified, whereas the activation functions are defined before the training process and remain fixed.

A vector which contains all the parameters of the network will be denoted as $\boldsymbol{\theta}$. All parameters are arranged in this vector, regardless of the role they play as part of a bias vector or a weight matrix.

To generalize the representation shown in Figure 5, a DNN with L layers will be assumed (considering the number of layers as equal to the number of weight matrices). The network receives an input \mathbf{a}^0 and produces an output \mathbf{a}^L . The input has an associated label or target \mathbf{a}^T , which is the output that the network should have delivered in a perfect scenario. The output of the DNN, \mathbf{a}^L , is a function of the input and the parameters of the network:

$$\mathbf{a}^L = \mathbf{a}^L(\mathbf{a}^0, \boldsymbol{\theta}) \quad (1)$$

The error between \mathbf{a}^L and \mathbf{a}^T is measured by a cost function C .

$$\text{Error} = C(\mathbf{a}^L, \mathbf{a}^T) = C(\mathbf{a}^L(\mathbf{a}^0, \boldsymbol{\theta}), \mathbf{a}^T) \quad (2)$$

Now, it would be desirable to reduce the cost function by adjusting the network parameters. The input \mathbf{a}^0 and the target \mathbf{a}^T are external data, thus they cannot be modified. However, by modifying the parameters of the DNN, it is possible to modify the output \mathbf{a}^L .

The gradient of the cost function with respect to the network parameters will be defined as:

$$[\nabla_{\boldsymbol{\theta}} C(\mathbf{a}^L, \mathbf{a}^T)]_i = \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial \theta_i} \quad (3)$$

In Equation 3, $\mathbf{a}^L(\mathbf{a}^0, \boldsymbol{\theta})$ as been simplified to \mathbf{a}^L ; however, it is important to understand that changing the parameters of the network produces a change in the cost function by changing the output \mathbf{a}^L . For this reason, it is important that the cost function is differentiable with respect to the output of the network.

The gradient of a function is the direction of steepest increase in that function; therefore, the best way to modify the parameters of the network in order to reduce the cost function is by taking a small step in the opposite direction to the gradient of the cost; hence the name of the algorithm:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} C(\mathbf{a}^L, \mathbf{a}^T) \quad (4)$$

The symbol \leftarrow indicates that $\boldsymbol{\theta}$ is updated to what is at the right side of the arrow. η is a small number known as learning rate.

Equation 4 shows a single training step when only one sample (i.e. a pair of an input vector \mathbf{a}^0 and its corresponding target vector \mathbf{a}^T) is taken into account. However, the goal is to reduce the cost function C with respect to the whole data-set (i.e. the set of all samples that are being used for training). Given that using all samples of the data-set for every single training step may be computationally too expensive, a faster technique is to take a random subset of the data-set for each training step. This technique is known as “mini-batch gradient descent” and the subsets of experiences that are used for each training step are known as “batches”. With a batch size N (i.e. N samples per batch), Equation 4 can be re-written as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{N} \cdot \sum_{i=1}^N \nabla_{\boldsymbol{\theta}} C(\mathbf{a}_i^L, \mathbf{a}_i^T) \quad (5)$$

3.1.2. Momentum

The training process of the DNN is an optimization problem, because what is being looked for is the combination of DNN parameters that minimizes the cost function with respect to the data-set as a whole; this can be thought of as the minimization process of a function (cost) in the multi-dimensional space of the parameters of the network. In order to avoid local optima and to achieve a faster training process, Momentum (Géron [36], page 361) is a useful tool. In intuitive words, Momentum is a vector the same size as the parameter vector $\boldsymbol{\theta}$ that stores the “update speed” accumulated in previous iterations. When combined with the mini-bath gradient descent, the training iterations are as follows:

$$\mathbf{m} \leftarrow \beta \cdot \mathbf{m} - \frac{\eta}{N} \sum_{i=1}^N \nabla_{\theta} C(\mathbf{a}_i^L, \mathbf{a}_i^T) \quad (6)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m} \quad (7)$$

Here, \mathbf{m} is the momentum vector and β is a number in $[0,1)$ that can be thought of as a “friction” that does not allow speed to accumulate when $\beta = 0$. From now on it will be called “momentum factor”. N is the size of the batches used to train the network and η is the learning rate. There are other proposed ways of defining momentum, all with the same goal, but this is the one that is going to be considered for the study.

3.1.3. Backpropagation

Now, the problem that still remains is: how to compute the derivatives of the cost function with respect to the parameters of the network as shown in Equation 3? Backpropagation [37] is the algorithm that solves this issue. It was first presented by Rumelhart et al. [38] in the year 1986.

For this section it is important to understand the nomenclature shown in Figure 5 (Section 3.1). Like in previous sections, a DNN with L layers is assumed. It receives an input \mathbf{a}^0 and produces an output \mathbf{a}^L . The input is associated to a target vector \mathbf{a}^T which is the desired output.

Let p be any parameter of the DNN, namely a weight or a bias, from any layer (i.e., p is any component of the vector $\boldsymbol{\theta}$). Reducing Equation 4 to a single parameter would look like:

$$p \leftarrow p - \eta \cdot \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial p} \quad (8)$$

This means that the derivative of the cost function with respect to each parameter of the network must be computed in order to update the corresponding parameter. As already discussed, the cost function is reduced by modifying the output of the DNN, which can be achieved by modifying its parameters. Therefore, it is useful to use the chain rule in the form:

$$\frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial p} = \sum_i \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_i^L} \cdot \frac{\partial a_i^L}{\partial p} \quad (9)$$

Here, a_i^L are the components of the output of the DNN. (Remember that \mathbf{a}^T is not modifiable). The derivative $\partial C(\mathbf{a}^L, \mathbf{a}^T) / \partial a_i^L$ depends on what cost function is being used. The quadratic cost (QC) function is a popular example, in which case:

$$QC(\mathbf{a}^L, \mathbf{a}^T) = \frac{1}{2} \sum_i (a_i^L - a_i^T)^2 \quad \Rightarrow \quad \frac{\partial QC(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_i^L} = a_i^L - a_i^T \quad (10)$$

In the coming sections, the derivative $\partial C(\mathbf{a}^L, \mathbf{a}^T) / \partial a_i^L$ in its general form will be used.

Now, all that is missing is to find the derivative $\partial a_i^L / \partial p$, that is, how modifying a parameter p changes the output vector \mathbf{a}^L .

Case 1: p is a parameter of the output layer

The simplest problem is to find the derivative $\partial a_i^L / \partial p$ when the parameter p is a weight or a bias of the output layer. Given that the vector \mathbf{a}^L is defined as the result of the activation function σ^L when applied to the \mathbf{z}^L vector, and by using the chain rule, this derivative is equal to:

$$\frac{\partial a_i^L}{\partial p} = \sum_j \frac{\partial a_i^L}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial p} \quad (11)$$

From now on, it will be assumed that the activation function of the output layer, σ^L , is an element-wise function; this means that every component of the \mathbf{z}^L vector determines one component of the \mathbf{a}^L vector:

$$a_i^L = \sigma^L(z_i^L) \quad (12)$$

This assumption is valid for nearly all activation functions; only the Softmax function is an important exception that will be discussed in the next section. For deeper layers (other than the output layer), it will be assumed that the activation function is always element-wise. With this, the derivative shown in Equation 11 becomes:

$$\frac{\partial a_i^L}{\partial p} = \frac{d[\sigma^L(z_i^L)]}{d z_i^L} \cdot \frac{\partial z_i^L}{\partial p} \quad (13)$$

The \mathbf{z}^L vector is defined as:

$$\mathbf{z}^L = \mathbf{w}^L \cdot \mathbf{a}^{L-1} + \mathbf{b}^L \quad (14)$$

Therefore:

$$z_i^L = \sum_j w_{ij}^L \cdot a_j^{L-1} + b_i^L \quad (15)$$

From Equation 15, it can be directly concluded that the derivatives $\partial z_i^L / \partial w_{ij}^L$ and $\partial z_i^L / \partial b_i^L$ are:

$$\frac{\partial z_i^L}{\partial w_{ij}^L} = a_j^{L-1} \quad (16)$$

$$\frac{\partial z_i^L}{\partial b_i^L} = 1 \quad (17)$$

By combining Equation 13 with Equations 16 and 17, it is possible to get expressions for the derivatives of a_i^L , i.e. the outputs of the DNN, with respect to the parameters (weights and biases) of the output layer:

$$\frac{\partial a_i^L}{\partial w_{ij}^L} = \frac{d[\sigma^L(z_i^L)]}{d z_i^L} \cdot a_j^{L-1} \quad (18)$$

$$\frac{\partial a_i^L}{\partial b_i^L} = \frac{d[\sigma^L(z_i^L)]}{d z_i^L} \quad (19)$$

Case 2: p is a parameter of a hidden layer

Now the problem becomes: how to get deeper into the other layers? This is where the *real* backpropagation property is used. It can be understood by noting that, unlike the input of the first layer, \mathbf{a}^0 , the input of the last layer, \mathbf{a}^{L-1} , can be modified since it is produced by previous layers; and this would modify the output of the DNN. Now the question becomes: how to modify the parameters of previous layers (i.e. $L - 1, L - 2, \dots$) so that \mathbf{a}^{L-1} gets modified in the right way in order to reduce the cost function? This will be done with the layer $(L - 1)$ first and then a generalization for deeper layers will be inferred.

It is useful to note that the relations shown in Equations 18 and 19 are valid for the layer $(L - 1)$ as well, only by changing the number of the layer:

$$\frac{\partial a_i^{L-1}}{\partial w_{ij}^{L-1}} = \frac{d [\sigma^{L-1}(z_i^{L-1})]}{d z_i^{L-1}} \cdot a_j^{L-2} \quad (20)$$

$$\frac{\partial a_i^{L-1}}{\partial b_i^{L-1}} = \frac{d [\sigma^{L-1}(z_i^{L-1})]}{d z_i^{L-1}} \quad (21)$$

As already discussed, for Equations 20 and 21 it has been assumed that the activation function σ^{L-1} is element-wise.

To determine how to modify the activation vector \mathbf{a}^{L-1} , the derivative of the cost function with respect to its components is computed:

$$\frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_j^{L-1}} = \sum_i \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_i^L} \cdot \frac{\partial a_i^L}{\partial a_j^{L-1}} \quad (22)$$

The derivative $\partial a_i^L / \partial a_j^{L-1}$ can be solved by applying the chain rule and Equations 12 and 15:

$$\frac{\partial a_i^L}{\partial a_j^{L-1}} = \sum_m \frac{\partial a_i^L}{\partial z_m^L} \cdot \frac{\partial z_m^L}{\partial a_j^{L-1}} = \frac{\partial a_i^L}{\partial z_i^L} \cdot \frac{\partial z_i^L}{\partial a_j^{L-1}} = \frac{d [\sigma^L(z_i^L)]}{d z_i^L} \cdot w_{ij}^L \quad (23)$$

Finally, by applying the chain rule and Equations 20, 21, 22 and 23, it is possible to obtain expressions for the derivatives of the cost function with respect to the parameters of the layer $(L - 1)$. In the case of the weights:

$$\begin{aligned} \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial w_{jk}^{L-1}} &= \sum_m \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_m^{L-1}} \cdot \frac{\partial a_m^{L-1}}{\partial w_{jk}^{L-1}} = \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_j^{L-1}} \cdot \frac{\partial a_j^{L-1}}{\partial w_{jk}^{L-1}} \\ &= \sum_i \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_i^L} \cdot \frac{d [\sigma^L(z_i^L)]}{d z_i^L} \cdot w_{ij}^L \cdot \frac{d [\sigma^{L-1}(z_j^{L-1})]}{d z_j^{L-1}} \cdot a_k^{L-2} \end{aligned} \quad (24)$$

For the biases the process is almost the same:

$$\frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial b_j^{L-1}} = \sum_m \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_m^{L-1}} \cdot \frac{\partial a_m^{L-1}}{\partial b_j^{L-1}} = \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_j^{L-1}} \cdot \frac{\partial a_j^{L-1}}{\partial b_j^{L-1}}$$

$$= \sum_i \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_i^L} \cdot \frac{d[\sigma^L(z_i^L)]}{dz_i^L} \cdot w_{ij}^L \cdot \frac{d[\sigma^{L-1}(z_j^{L-1})]}{dz_j^{L-1}} \quad (25)$$

Conclusion

It might seem that the calculations are getting more complicated for the hidden layers, but there is a simple way to generalize the method for any number of layers:

1. The derivative of the cost function with respect to an activation of any hidden layer l can be known by knowing derivatives of the cost function with respect to all activations of the next layer (the one which is closer to the output layer):

$$\frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_j^l} = \sum_i \frac{\partial C(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_i^{l+1}} \cdot \frac{d[\sigma^{l+1}(z_i^{l+1})]}{dz_i^{l+1}} \cdot w_{ij}^{l+1} \quad (26)$$

When the layer $(l+1)$ happens to be the output layer L , then the derivative $\partial C(\mathbf{a}^L, \mathbf{a}^T)/\partial a_i^{l+1}$ can be computed directly from the definition of the cost function.

This rule can be applied backwards from the output layer until reaching the first layer. This is the core of backpropagation.

2. The derivative of the activations of any hidden layer l with respect to the parameters of the same layer can be determined with the following formulas:

$$\frac{\partial a_i^l}{\partial w_{ij}^l} = \frac{d[\sigma^l(z_i^l)]}{dz_i^l} \cdot a_j^{l-1} \quad (27)$$

$$\frac{\partial a_i^l}{\partial b_i^l} = \frac{d[\sigma^l(z_i^l)]}{dz_i^l} \quad (28)$$

These two rules allow to determine the derivative of the cost function with respect to any parameter of the DNN by applying the chain rule.

3.1.4. Softmax activation

As already discussed, for the previous section it has been assumed that the activation functions of all layers are element-wise. The only exception that will be discussed is the case of the Softmax activation, which is used as the activation function of the output layer when the DNN is used to generate a set of probabilities that together should add up to one.

An example of when to use this function is when the DNN has to categorize the data samples into a finite number of classes, each sample only belonging to one class. The DNN delivers a vector of values which are greater than zero and add up to one; therefore, these values can be interpreted as the probabilities that a sample belongs to each class.

Like other activation functions, the Softmax is applied to the \mathbf{z}^L vector as discussed in Section 3.1; its formula is (Géron [36], page 164):

$$a_i^L = [\sigma^L(\mathbf{z}^L)]_i = \frac{\exp(z_i^L)}{\sum_j \exp(z_j^L)} \quad (29)$$

The number j takes the indices of all the components of the \mathbf{z}^L vector. The Softmax function, as defined in Equation 29, is not element-wise because every component of the \mathbf{a}^L vector depends on all components of the \mathbf{z}^L vector. In this case, the target vector will be defined as:

$$a_i^T = \begin{cases} 1 & \text{if } i \text{ is the class that the sample belongs to} \\ 0 & \text{if } i \text{ is NOT the class that the sample belongs to} \end{cases} \quad (30)$$

This way to define the target is intuitive because it shows that the sample has a probability equal to one of belonging to the class that it actually belongs to.

In this case, the cost function will be the categorical cross-entropy (CCE) function:

$$CCE(\mathbf{a}^L, \mathbf{a}^T) = - \sum_i a_i^T \cdot \ln(a_i^L) \quad (31)$$

Given that a_i^T is equal to zero except when i is the real class of the sample, this can be reduced to:

$$CCE(\mathbf{a}^L, \mathbf{a}^T) = - \ln(a_k^L) \quad k = \text{real class of the sample} \quad (32)$$

In this case, it is simpler to (directly) compute the derivatives $\partial C(\mathbf{a}^L, \mathbf{a}^T) / \partial z_i^L$.

When i is not the target class, i.e. $i \neq k$:

$$\begin{aligned} \frac{\partial CCE(\mathbf{a}^L, \mathbf{a}^T)}{\partial z_i^L} &= - \frac{\partial \ln(a_k^L)}{\partial z_i^L} = - \frac{\partial}{\partial z_i^L} \ln \left(\frac{\exp(z_k^L)}{\sum_j \exp(z_j^L)} \right) \\ &= - \left(\frac{\exp(z_k^L)}{\sum_j \exp(z_j^L)} \right)^{-1} \cdot \frac{\partial}{\partial z_i^L} \left(\frac{\exp(z_k^L)}{\sum_j \exp(z_j^L)} \right) \\ &= - \left(\frac{\sum_j \exp(z_j^L)}{\exp(z_k^L)} \right) \cdot \frac{- \exp(z_k^L) \cdot \exp(z_i^L)}{(\sum_j \exp(z_j^L))^2} = \frac{\exp(z_i^L)}{\sum_j \exp(z_j^L)} = a_i^L \end{aligned} \quad (33)$$

When i is the target class, i.e. $i = k$:

$$\begin{aligned} \frac{\partial CCE(\mathbf{a}^L, \mathbf{a}^T)}{\partial z_k^L} &= - \frac{\partial \ln(a_k^L)}{\partial z_k^L} = - \frac{\partial}{\partial z_k^L} \ln \left(\frac{\exp(z_k^L)}{\sum_j \exp(z_j^L)} \right) \\ &= - \left(\frac{\exp(z_k^L)}{\sum_j \exp(z_j^L)} \right)^{-1} \cdot \frac{\partial}{\partial z_k^L} \left(\frac{\exp(z_k^L)}{\sum_j \exp(z_j^L)} \right) \\ &= - \frac{\sum_j \exp(z_j^L)}{\exp(z_k^L)} \cdot \left(\frac{\exp(z_k^L) \cdot (\sum_j \exp(z_j^L)) - (\exp(z_k^L))^2}{(\sum_j \exp(z_j^L))^2} \right) \\ &= - \left(1 - \frac{\exp(z_k^L)}{\sum_j \exp(z_j^L)} \right) = a_k^L - 1 \end{aligned} \quad (34)$$

In summary, let k be the index of the real class of a sample:

$$\frac{\partial CCE(\mathbf{a}^L, \mathbf{a}^T)}{\partial z_i^L} = a_i^L - \delta_{ik} \quad (35)$$

δ_{ik} is the Kronecker Delta.

Only for the parameters (weights and biases) of the output layer, there is a slight difference in the computation of the gradient vector, in contrast with the element-wise case shown in the previous section: since in this case the derivatives $\partial C(\mathbf{a}^L, \mathbf{a}^T)/\partial z_i^L$ have been computed, the derivatives shown in Equations 16 and 17 have to be used in order to apply the chain rule.

Also, Equation 22 must be replaced with the formula shown in Equation 36. After this, the backpropagation algorithm is the same for the parameters of hidden layers.

$$\frac{\partial CCE(\mathbf{a}^L, \mathbf{a}^T)}{\partial a_j^{L-1}} = \sum_i \frac{\partial CCE(\mathbf{a}^L, \mathbf{a}^T)}{\partial z_i^L} \cdot \frac{\partial z_i^L}{\partial a_j^{L-1}} \quad (36)$$

3.1.5. DNN initialization

Before the training process, the DNN is created without previous knowledge about the information that it is going to learn. The question of how to set the initial parameters has been studied and it was found that the initialization method can greatly influence the performance of the network and its convergence time [39].

For this study, the default initialization method for the dense layers of Keras is used [40]. Keras is a widely used Deep Learning library for Python.

In Keras, by default, the bias vectors are initialized as zeros. This means that all components of the bias vectors of all layers are equal to zero at the beginning. For the weight matrices, the Glorot Uniform method is used. This method, which was presented by Glorot and Bengio in 2010 [41], implies that the initial weights of the layers (i.e. the components of the weight matrices) follow a uniform distribution:

$$w_{ij}^l \sim U\left(-\sqrt{\frac{6}{I^l + J^l}}, \sqrt{\frac{6}{I^l + J^l}}\right) \quad (37)$$

The arguments of the uniform distribution U are the limits for the random variable w_{ij}^l . I^l and J^l are the output size and the input size of the layer l , respectively. In other words, the matrix \mathbf{w}^l has I^l rows and J^l columns.

3.2. Deep Reinforcement Learning Algorithms

In the coming sections, the Deep Reinforcement Learning (DRL) algorithms that are tested during the study will be explained. However, it is necessary to clarify a few more basic concepts first. For this section it is also necessary to understand the introduction given in Section 1.5.

3.2.1. Basic Concepts

Environment state

It has been said that the agent may base its decisions on the history of previous interactions with the environment. Therefore, the history at time step t is defined as (David Silver [42]):

$$H_t = \{ O_1, A_1, R_1, O_2, A_2, R_2, \dots, A_{t-1}, R_{t-1}, O_t \} \quad (38)$$

Here, O are observations, A are actions and R are rewards. The numbers indicate the time step to which they belong. The information that the agent uses to make a decision on time step t will be called “state”, and it is any convenient function of the history H_t [42]:

$$S_t = f(H_t) \quad (39)$$

It is important to note that this “state” is the information that the agent has about the environment; however, it may not define the full internal state of the environment; this will depend on how the observations are defined. The term “state” or “environment state” are used interchangeably in this document to refer to the definition of Equation 39.

Moreover, from Chapter 4 onwards, the word “state” is used as equivalent to “observation” because, as will be discussed in that section, the agents of this study only use the latest observation as information to make a decision. Mathematically written:

$$S_t = f(H_t) = O_t \quad (40)$$

In this theoretical section, the term “state” is used in its general sense as any function of the interaction history, as shown in Equation 39.

Terminal states

An environment is defined as *episodic* if it can reach states in which the agent does not have to continue executing actions; these are called terminal states. When a terminal state is reached, the interaction between the agent and the environment finishes and another episode must begin in order to continue the training process [43].

On the convention

As already shown in Equation 38, in the next sections, S_t is used to denote the state of the environment at time step t ; therefore, O_t is the observation received from the environment at t . A_t is used to denote the action that is executed immediately after the observation O_t is received and R_t is the reward received immediately after the action A_t is executed. It is important to note this because, under this convention, the reward R_t is received along with the observation O_{t+1} .

Long Term Rewards (Returns)

This concept is one of the most important ones in RL, and it is key to understanding the mathematics behind the training algorithms.

With most environments, one has to consider that an action not only has an effect on the reward that is received immediately after that action is executed; instead, it may affect the rewards that are received many time steps later as well. This makes it necessary for the agent to take future rewards into account at the moment of deciding what action to choose. To achieve this consideration of future rewards, long term rewards, also called returns [43], are defined as:

$$G_t = \sum_{i=t}^N \gamma^{i-t} \cdot R_i = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \dots \quad (41)$$

G_t is the return of the action A_t , executed at the instant t . N is the number of time steps in the corresponding episode; therefore i takes the numbers of the time steps that happened since action A_t until the end of the episode. γ is a training hyperparameter in the interval $[0,1]$ that *is supposed to* approximate the level of influence of an action on future rewards. Although this “level of influence” depends mainly on the features of the environment, the value of γ must be set by the user and its optimal value must be found through exploration. If its optimal value is close to one, actions greatly influence future rewards; if its optimal value is close to zero, only the immediate reward is being influenced by the action.

Policy and expected returns

The term “policy” will be used to summarize the whole set of rules that are followed in order to make decisions as the environment simulation progresses. The policy could be, for example, that sometimes a completely random action is selected and sometimes the action that the agent predicts to be the best is selected. If the parameters of the DNN are modified, the policy is being modified as well.

The rewards (and therefore the returns) that the agent gets obviously depend on its policy (if they do not, then the rewards are independent from what the agent does, so the training process is pointless). Nonetheless, the agent may not have complete knowledge about the *internal* state of the environment; therefore, the rewards may not be deterministic from the point of view of the agent when a certain action is executed on a certain environment state, because the state *that the agent is seeing* may have partial information.

A property that is assumed to justify the algorithms discussed below is called “markovian property”; it implies that the pair (R_t, S_{t+1}) , as a random variable, follows a probability distribution (discrete or continuous) that only depends on the previous state S_t and action A_t (although the state S_t can consider previous observations, actions and rewards). More details on the markovian property are discussed in Section 3.4, although in that case no actions or rewards are involved in the process.

What can be concluded from the markovian property is that, given the policy that the agent is following, each environment state is associated to a certain expected return. The fact that this “expected return” can actually be very hard to determine is what makes the use of machine learning methods a very useful tool. Therefore, in the mathematical argumentations to come, the expected return is a very important concept, and the policy is what determines it. In the coming equations, the letter π will be used to denote policies in general (with all the rules that they may have), and $A_t \sim \pi$ will be used to denote that the action A_t was chosen by following the policy π .

3.2.2. Policy gradient methods and value methods

There are two big families of methods used in Deep Reinforcement Learning: policy gradient methods and value methods [43]. With all methods, the input which is given to the DNN is the state of the environment.

With policy gradient methods, the output of the DNN consists of *probabilities* of executing each possible action. The agent chooses an action by following these probabilities, and the training

process consists of increasing the probabilities of executing actions which turn out to yield high returns, thus decreasing the probability of executing actions that yield low returns.

With value methods, the DNN seeks to predict the return that the agent will get with future rewards given the current state of the environment. In methods purely based on values, the DNN has to predict the expected return for each possible action, and then the best action to take will be the one which has the largest expected return.

3.2.3. REINFORCE algorithm

The REINFORCE algorithm (Géron [36], page 617) is a policy gradient method. This means that the actions are chosen by following probabilities delivered by the agent; therefore, the Softmax activation function is used in the output layer of the DNN (see Section 3.1.4). The algorithm works roughly the following way:

1. Execute multiple episodes without making any change to the DNN.
2. Compute the return for every executed action of every episode, with the formula given by Equation 41.
3. Compute the average and the standard deviation of the returns. Normalize the returns by subtracting the average to every return and then dividing the result by the standard deviation.
4. For every state-action pair (S_t, A_t) of every episode, compute how the parameters of the network should be modified in order to increase the probability of executing A_t in the state S_t . This is done as follows: given that the activation function of the output layer is a Softmax function, a gradient is computed by using the categorical cross-entropy cost function and considering the action A_t as “target class” (as defined in Section 3.1.4).
5. Multiply each gradient by the normalized return of the corresponding action. Average all resulting vectors and use the result to update the network. Note that, if the return of an action is below the average, the parameter update will actually result in decreasing the probability of executing that action because the normalized return is negative.
6. Repeat this process several times until a good performance is reached. The parameters of the DNN must be updated in “small” steps defined by a learning rate, just like in supervised learning.

The intuition behind this method is that the agent starts its training process by executing totally random actions, and then it can look at them “in retrospect” to analyze which of them were good and which were bad; and then the network is updated with these results.

3.2.4. Actor-Critic Algorithm

The Actor-Critic algorithm (Sutton, Barto [43], page 331) combines the concepts of policy gradients and values. In order to understand it, some concepts must be discussed before. First, the concept of “state value” is defined as the expected return given the current state of the environment and the policy that the agent is using:

$$v_{\pi}(s) = \mathbb{E} [G_t \mid S_t = s, A_i \sim \pi \text{ for } i \geq t] \quad (42)$$

As already discussed, π represents the policy that is being applied, and $A_i \sim \pi$ expresses that the action A_i is chosen by following that policy. Because of the definition of returns given by Equation 41, and also by taking the markovian property into account (see “Policy and expected returns” in Section 3.2.1 for markovian property), the state value of state s can also be expressed as:

$$v_{\pi}(s) = \mathbb{E} [R_t + \gamma \cdot v_{\pi}(S_{t+1}) \mid S_t = s, A_t \sim \pi] \quad (43)$$

This “splitting” of the state value into the immediate reward and the state value of the next state is the key of the Actor-Critic algorithm. In this method two neural networks are used:

- The “actor” is a DNN that evaluates the current state of the environment and delivers probabilities of executing each possible action.
- The “critic” is a DNN that evaluates the state of the environment and predicts its state value.

The training procedure is as follows:

1. Begin an episode.
2. The actor evaluates the state S_t and delivers the probabilities of executing each action. An action A_t is executed by following the probabilities.
3. A reward R_t and a new state S_{t+1} are received. For the next steps it is assumed that S_{t+1} is not a terminal state; the exception will be discussed in the last point.
4. Compute predictions for the state values $v_{\pi}(S_t)$ and $v_{\pi}(S_{t+1})$ using the critic DNN; these predictions will be denoted as $\hat{v}(S_t)$ and $\hat{v}(S_{t+1})$. If the result $R_t + \gamma \cdot \hat{v}(S_{t+1})$ is greater than $\hat{v}(S_t)$, the action A_t is considered to be better than expected; if the opposite happens, the action is considered to be worse than expected.
5. Update the actor DNN so that if the action A_t was better than expected, the probability of executing that action on state S_t is increased, and if the action was worse than expected, the probability of executing it is decreased. This is done as follows: given that the activation function of the output layer is a Softmax function, a gradient is computed with the categorical cross-entropy cost function and considering the action A_t as “target class” (as defined in Section 3.1.4). In order to take the “quality” of the action A_t into account, the gradient vector is multiplied by the factor $\delta = R_t + \gamma \cdot \hat{v}(S_{t+1}) - \hat{v}(S_t)$ and the obtained vector is used to update the actor DNN. Note that if the value of δ is negative, the network update will result in decreasing the probability of executing the action A_t , as desired.
6. Update the critic DNN so that the prediction $\hat{v}(S_t)$ gets closer to $R_t + \gamma \cdot \hat{v}(S_{t+1})$. This is done as follows: compute the gradient using the quadratic error as cost function, and the value $R_t + \gamma \cdot \hat{v}(S_{t+1})$ as target. Update the network with the obtained gradient.
7. Repeat the process until the episode is finished. In the terminal state, the state value is zero by definition, i.e. $v(S_{t+1}) = 0$, so the critic only has to compute a prediction for $v_{\pi}(S_t)$. The whole process can be repeated for multiple episodes until a good performance is reached.

The intuition behind this process is that the actor learns which actions are good and which are bad based on the predictions of the critic, and the probabilities of executing those actions are increased or decreased according to the results obtained. This is achieved by multiplying the gradient vector by the scalar δ expressed above. In this way, the difference between how good the action was expected to be and how good it actually was is taken into account as well. Meanwhile, the critic learns to make better predictions of the state values based on the rewards that the agent gets from the environment.

Clearly, the parameter updates for both networks must be “small” steps defined by learning rates. Each DNN can have its own associated learning rate. In another variant of this method, the training steps are not executed at every time step; instead the gradients are stored and averaged every certain number of time steps. However, this was not tested in this study.

3.2.5. Q-Learning and Deep Q-Learning

3.2.5.1. Q-Values

Q-Values (Géron [36], page 623) are defined as the expected return that the agent will get after taking the action a on state s :

$$Q_{\pi}(s, a) = \mathbb{E} [G_t \mid S_t = s, A_t = a, A_i \sim \pi \text{ for } i > t] \quad (44)$$

As in previous sections, $A_i \sim \pi$ expresses that the action A_i is chosen by following the policy π . With this definition, the best action to take on state s would be the one that has the largest Q-Value. Unlike the state value function discussed above, which only takes the state as argument and determines the expected return, the Q-Value function determines the expected return for a state-action pair (s, a) . This is necessary in order for the agent to know which action is the best.

The Q-Value of a state-action pair depends on the policy that is being applied because the return depends on future rewards, which are dependent on the policy. The maximum possible Q-Value of a state action pair (s, a) is obtained when the best possible policy is applied in the next time steps. Clearly, since the goal of the agent is to find that policy, it is desirable to find the largest possible Q-Value for every state-action pair.

3.2.5.2. Q-Learning

Let $Q^*(s, a)$ be the optimal (or largest possible) Q-Value for the state-action pair (s, a) . Note that this Q-Value does not depend on the policy anymore, because it is being assumed that the best possible policy is followed after executing the action a . This value may also be expressed as:

$$Q^*(s, a) = \mathbb{E} \left[R_t + \gamma \cdot \max_{a'} \{ Q^*(S_{t+1}, a') \} \mid S_t = s, A_t = a \right] \quad (45)$$

This transformation, which is analogous to the one discussed for the actor-critic algorithm (Equation 43), consists of “splitting” the Q-Value into the immediate reward and the maximum Q-Value of the next state. This transformation also assumes the markovian property as true. With this in mind, an iterative way of converging to the optimal Q-Values of the environment is to update the Q-Value approximation \hat{Q} after every experienced transition with a learning rate η and the formula (Géron [36], page 626):

$$\hat{Q}(S_t, A_t) \leftarrow \hat{Q}(S_t, A_t) + \eta \cdot \left(R_t + \gamma \cdot \max_{a'} \{ \hat{Q}(S_{t+1}, a') \} - \hat{Q}(S_t, A_t) \right) \quad (46)$$

Up to now, no neural networks have been introduced into the algorithm. The approximations $\hat{Q}(s, a)$ can simply be recorded in a table if the state space and the action space are discrete and finite (and not very large). Then, the entries of the table could be updated according to Equation 46 until achieving convergence for every pair (s, a) . One detail is that, when S_{t+1} is a terminal state, the Q-Values of all actions are equal to zero; therefore, no predictions from the agent are needed. This method is known as “Tabular Q-Learning”.

The benefit of this method is that, after executing an action, one only has to wait until the reward R_t and the next state S_{t+1} are received, in order to update the Q-Value approximation of the previous state-action pair (S_t, A_t) . However, the update is based on the Q-Value approximations for the state S_{t+1} , so several iterations are normally needed to achieve convergence.

3.2.5.3. Deep Q-Learning algorithm

In the study presented here, the action-space is discrete and finite but the state-space is continuous and multi-dimensional; therefore, the tabular method just mentioned becomes unfeasible. Deep Q-Learning is a method for training a DNN, also known as Deep Q-Network (DQN), to evaluate the environment states and to predict the optimal Q-Values. It is inspired by the “standard” Q-Learning method discussed above. This method was shown to be very powerful by researchers of DeepMind, one of the leading companies in the area of Machine Learning and specifically in Reinforcement Learning as well. In 2015, Mnih et al. published a paper in the Nature magazine [14] where this algorithm obtained state-of-the-art results at various Atari games, surpassing all other previous training methods and achieving the level of professional human players.

The output vector of the DNN, denoted here as \mathbf{a}^L to follow the same nomenclature from previous sections, must contain as many values as possible actions the agent can choose. Each component of \mathbf{a}^L is the Q-Value prediction of one action, as a function of the state that was given to the DNN as input:

$$[\mathbf{a}^L(S_t)]_i = \hat{Q}(S_t, a_i) \quad (47)$$

In Equation 10, i is not representing a time step like the letter t ; instead, it represents the index of an action within the set of possible actions. That is why i also represents the components of the vector \mathbf{a}^L . \hat{Q} is a Q-Value prediction of the DNN.

The first step of the training process consists of executing random actions on the environment and storing these “experiences” of the agent in a “Replay Memory”, which will be used as data-set to train the DNN. When the maximum length of the Replay Memory has been reached, the oldest experiences are erased. An experience E_t is composed of:

$$E_t = \{ S_t, A_t, R_t, S_{t+1} \} \quad (48)$$

From experience E_t , a target vector will be defined so that the training method based on Gradient Descent, discussed in Section 3.1.1, can be used. The target vector \mathbf{a}^T is defined as:

$$[\mathbf{a}^T(E_t)]_i = \begin{cases} \hat{Q}(S_t, a_i) & \text{if } i \text{ is NOT the index of the action } A_t \\ Q_{\text{Target}}(S_t, A_t) & \text{if } i \text{ is the index of the action } A_t \end{cases} \quad (49)$$

In Equation 49, $Q_{\text{Target}}(S_t, A_t)$ is defined as:

$$Q_{\text{Target}}(S_t, A_t) = \begin{cases} R_t + \gamma \cdot \max_{a'} \{ \hat{Q}(S_{t+1}, a') \} & \text{if } S_{t+1} \text{ is not a terminal state} \\ R_t & \text{if } S_{t+1} \text{ is a terminal state} \end{cases} \quad (50)$$

Note that the target vector \mathbf{a}^T , which is a function of the experience E_t , is defined so that it only differs from the prediction of the network in the component that corresponds to the action A_t executed at that time step. Therefore, the quadratic cost function (defined in Equation 10) of the prediction vector \mathbf{a}^L and the target vector \mathbf{a}^T can be reduced to:

$$QC(\mathbf{a}^L(S_t), \mathbf{a}^T(E_t)) = \frac{1}{2} \cdot \sum_{i=1}^H (a_i^L - a_i^T)^2 = \frac{1}{2} \cdot (\hat{Q}(S_t, A_t) - Q_{\text{Target}}(S_t, A_t))^2 \quad (51)$$

In Equation 51, H is the number of possible actions, and a_i^L, a_i^T are the components of the prediction vector and the target vector, respectively. The cost function is reduced to a single squared difference because all other components of the aforementioned vectors are equal. With this simplification, the gradient vector, defined in Equation 3, can be expressed as:

$$[\nabla_{\theta} QC(\mathbf{a}^L(S_t), \mathbf{a}^T(E_t))]_i = (\hat{Q}(S_t, A_t) - Q_{\text{Target}}(S_t, A_t)) \cdot \frac{\partial \hat{Q}(S_t, A_t)}{\partial \theta_i} \quad (52)$$

In Equation 52, θ_i represents any parameter of the DNN, just as in Equation 3. With this, Equation 4, which is the heart of the Gradient Descent algorithm, can be re-written as:

$$\theta \leftarrow \theta + \eta \cdot (Q_{\text{Target}}(S_t, A_t) - \hat{Q}(S_t, A_t)) \cdot \nabla_{\theta} \hat{Q}(S_t, A_t) \quad (53)$$

In Equation 53, the gradient with respect to the Q-Value prediction is defined as: $[\nabla_{\theta} \hat{Q}(S_t, A_t)]_i = \partial \hat{Q}(S_t, A_t) / \partial \theta_i$. Equation 53 shows that, by using the quadratic cost function, the training step is proportional to the difference between the target and the prediction, which is analogous to the tabular Q-Learning method (see Equation 46). Given that a data-set (the Replay Memory) is being used, the mini-batch method and momentum method can be used as well.

Target Network

One problem of Deep Q-Learning, as discussed up to now, is that the network has to set its own targets, because it has to estimate the Q-Values for the state S_{t+1} (see Equation 50). The fact that the targets change as the network is trained can destabilize training. To solve this, Mnih et al. [14] used a target Network to calculate these Q-Values. The target network is a copy of the trained network that is not updated in every iteration; instead, it remains fixed for a certain number of iterations and then it is updated by copying the parameters of the trained network. Its task is to evaluate the state S_{t+1} and to predict its Q-Values in order to set the target Q-Value at S_t , according to Equation 50. With the concept of “target network”, Equation 50 could be re-written as:

$$Q_{\text{Target}}(S_t, A_t) = \begin{cases} R_t + \gamma \cdot \max_{a'} \{ \hat{Q}(S_{t+1}, a', \theta^*) \} & \text{if } S_{t+1} \text{ is not a terminal state} \\ R_t & \text{if } S_{t+1} \text{ is a terminal state} \end{cases} \quad (54)$$

Here, the third argument of the Q-Value prediction \hat{Q} specifies that that prediction is made by the target network, here denoted as θ^* .

ϵ -greedy method

As already discussed, training begins by executing random actions in order to fill the Replay Memory with experiences. At this moment, it does not make sense to execute the actions that the agent predicts to be the best because, before the DNN is trained, being right about what action is the best is merely luck. But as the agent begins to make better predictions about the Q-Values of the environment states, it would be desirable to “exploit” this knowledge by taking the actions that are predicted to be the best. This “exploitation” of knowledge is known as “greedy behavior”. However, one must always expect that the agent has something new to learn (if the agent knows the best action for all possible environment states, then it is pointless to continue the training process). Therefore, it is always good to take some random actions, in the hope that some of them happen to be unexpectedly good; if that happens, the agent has discovered a better strategy that it would not have discovered if it had behaved “greedily”. Taking random actions in the hope of

discovering better options is known as “exploration”. The issue of how to behave (greedy or exploratory) is known as “exploration-exploitation trade-off”.

The ϵ -greedy method (Géron [36], page 628) is meant to solve this issue. It consists of defining a probability of executing a random action; this probability is normally denoted as ϵ (hence the name of the method). When the agent does not select a random action, it selects the action that it predicts to be the best. The probability ϵ is usually equal to one at the beginning (because the agent does not know anything about the environment) and then it starts to decrease linearly, until it reaches a constant and small value. Although the final value of ϵ is small, it should not be equal to zero, because it is always good to have at least a small proportion of “exploratory behavior”.

The concepts discussed up to this point regarding Deep Q-Learning were included in the paper of Mnih et al. [14] in 2015. In the next two sections (3.2.5.4 and 3.2.5.5), two techniques that were introduced to the Deep Q-Learning method in later papers are explained.

3.2.5.4. *Double DQN*

It has been shown that the traditional DQN algorithm, with the target network as defined in the previous section, tends to overestimate the Q-Values of the environment states. Given that the network updates are based on Q-Value estimations, these overestimations tend to grow as the training process progresses. To solve this, van Hasselt et al. [44] created a method that they called “Double DQN”; this method is a variant of traditional DQN, with a subtle difference regarding the task that the target network fulfills and the way the Q-Value targets are set. In this algorithm, the “online network”, i.e. the DNN that is updated in every iteration, estimates the Q-Values of all actions for the state S_{t+1} , and decides which the best action to take is; then, the target network estimates the Q-Value of that action, and that estimation is used for the target.

In the traditional DQN method, by contrast, the target network decides both things: which action has the maximal Q-Value *and* its Q-Value. Both methods are tested and compared in this study: the traditional target network method and Double DQN.

3.2.5.5. *Prioritized Experience Replay*

Prioritized experience replay is a method presented by Schaul et al. [45] that is based on the following reasoning: if an experience has a surprisingly high or low target Q-Value in comparison with the prediction of the DNN, it is possible that the agent has much to learn from that particular experience, thus it will be assigned a higher probability of being sampled from the Replay Memory again. To do this, a prediction error is defined as:

$$\delta_t = \hat{Q}(S_t, A_t) - Q_{\text{Target}}(S_t, A_t) = \hat{Q}(S_t, A_t) - R_t - \gamma \cdot \max_{a'} \{ \hat{Q}(S_{t+1}, a') \} \quad (55)$$

With this definition, Schaul et al. [45] presented two different ways of calculating a priority number p_t for the experience associated to time-step t :

$$1. \quad p_t = |\delta_t| + \phi \quad (56)$$

$$2. \quad p_t = \frac{1}{\text{rank}(t)} \quad (57)$$

On the first definition, ϕ is a small value (larger than zero) which assures that no experience has zero probability of being sampled (this parameter is called ϵ in the original paper). This prioritizing method is known as “proportional prioritization” (although it is not exactly proportional to the absolute value of δ_t because of the use of ϕ). On the second definition, $\text{rank}(t)$ is the position of

the t -th experience when the absolute values of the prediction errors (as defined in Equation 55) of all experiences in the Replay Memory are ordered from largest to smallest. This is known as “rank-based prioritization”.

After obtaining the priority number of each experience, the probability of adding the t -th experience to the training batch is computed as:

$$P(t) = \frac{p_t^\alpha}{\sum_{j=1}^J p_j^\alpha} \quad (58)$$

Here, J is the total number of experiences stored in the Replay Memory. α is a hyperparameter to measure the importance given to priorities. When α is zero, the effect of prioritizing disappears.

Due to the fact that the sampling probabilities are not equal for all experiences, a bias is being introduced into the training process. To correct this, the authors of the paper (Schaul et al, 2016 [45]) propose to give a different weight w_t to each experience:

$$w_t = \left(\frac{1}{J} \cdot \frac{1}{P(t)} \right)^\psi \quad (59)$$

The parameter ψ is another training hyperparameter that must be tuned (it is called β in the original paper). The weight w_t is used to scale the gradient associated to the t -th experience at the moment of updating the network. However, before being applied, the weights are normalized by using the maximum weight of the Replay Memory, as shown in Equation 60. This is done so that no weight has an increasing effect in the training steps. With this, the “mini-batch gradient descent” method becomes as shown in Equation 61.

$$w_{t,\text{norm}} = \frac{w_t}{\max_j \{w_j\}_{j=1}^J} \quad (60)$$

$$\mathbf{m} \leftarrow \beta \cdot \mathbf{m} - \frac{\eta}{N} \sum_{i=1}^N w_{i,\text{norm}} \cdot \nabla_{\theta} QC(E_i) \quad (61)$$

In the case when $\psi = 1$, the effect of the non-uniform probabilities is completely compensated by the non-uniform weights [45]. In the case of the study presented here, the ψ parameter was always equal to zero, which means that the weights of all experiences are equal to one (in other words, the weights defined in Equation 59 are not used).

3.3. Reliability theory

Let T be a random variable that measures the moment at which a certain device stops operating properly, provided that the device started its operation at the instant $t = 0$. In other words, T represents the moment of failure of the device, and it is a random variable because in most cases the failure of any device or machine cannot be predicted with certainty. It is also being assumed that the device is not maintained during the operation (see Modarres, Kaminskiy, Krivtsov 2016 [46] for further information on reliability theory).

The most elemental function in reliability theory is the reliability function, which is defined as the probability that the failure of an item occurs after a certain instant t :

$$R(t) = P(T > t) \quad (62)$$

In other words, $R(t)$ is the probability that the device under study operates until t without failing (in this Section, t represents an instant in a continuous time spectrum, not the index of an experience like in the previous section. Given that the device starts to operate at $t = 0$, the domain of $R(t)$ are all values equal to or greater than zero).

The function $R(t)$ must meet certain conditions: its value at $t = 0$ must be one, since the device is put into operation at that moment. Its value must tend to zero (or simply *be* zero) as t tends to infinity. This is because no machine operating without maintenance can operate for an infinite amount of time. $R(t)$ must also be a decreasing function, i.e.:

$$\frac{dR(t)}{dt} \leq 0 \quad \forall t > 0 \quad (63)$$

The reliability function can never increase for the following reason: let t_1 and t_2 be two instants with $t_2 > t_1$. If the device operates until t_2 , this means that it has also operated until t_1 without failing. Therefore, operating until t_2 implies having operated until t_1 as well. On the other hand, the device could operate until t_1 but fail before t_2 . Consequently, the probability of operating until t_1 must be greater than the probability of operating until t_2 . In an extreme case, the reliability at both instants could be the same (with $dR(t)/dt$ being equal to zero between the two instants), but this would mean that the probability of failure between both instants is zero, which in practice does not happen if $R(t_1)$ and $R(t_2)$ are greater than zero.

The probability that the failure occurs between two instants t and $t + \Delta t$, with $\Delta t > 0$, can be known by subtracting the reliabilities at those instants:

$$P(t < T < t + \Delta t) = R(t) - R(t + \Delta t) \quad (64)$$

If the probability given by Equation 64 is divided by the time lapse between both instants, Δt , then the result can be interpreted as a mean “probability per time unit” of failing between t and $t + \Delta t$. This probability density can be defined “locally” for a specific instant in time by taking the limit as Δt tends to zero:

$$\lim_{\Delta t \rightarrow 0} \frac{R(t) - R(t + \Delta t)}{\Delta t} = - \frac{dR(t)}{dt} \quad (65)$$

Therefore, the negative derivative of the reliability function is commonly known as “probability density function” (PDF). Here it will be designated as $f(t)$:

$$f(t) = - \frac{dR(t)}{dt} \quad (66)$$

This function can be intuitively thought of as the local density of probability (per time unit) of the device failing at the instant t . Nevertheless, the probability of failing at an exact instant is mathematically equal to zero, and in order to obtain a failure probability, the PDF must be integrated between two different instants:

$$P(t_1 < T < t_2) = \int_{t_1}^{t_2} f(t) dt \quad (67)$$

Some properties of the function $f(t)$ are: it is always larger than or equal to zero. This is a consequence of Equations 63 and 66. The integral of $f(t)$ from zero to infinity is always equal to one. This is because of the property expressed in Equation 67, and because of the fact that the device under study *must* fail sometime between $t = 0$ and $t = \infty$.

The Mean Time to Failure (MTTF) is an important indicator that is defined as the expected failure time:

$$MTTF = \mathbb{E} [T] = \int_0^{\infty} t \cdot f(t) dt \quad (68)$$

3.3.1. Hazard rate

Suppose that a machine has been operating for a while without failing. A question worth asking is: “what is the probability that it will fail in the near future?” One might think that the answer can be calculated by integrating the PDF as expressed in Equation 67, with t_1 being the current time and t_2 some instant in the future. This would be wrong because now there is an extra information available: the device has been operating until t_1 without failing; thus, the chance of failure before t_1 can be “discarded”. Integrating the PDF, as expressed above, would yield the probability of failure between t_1 and t_2 without knowing whether the device has survived until t_1 , which in the present case is known to be true.

Formally said, the question that is being asked now is a conditional probability, the condition being that the device has operated during a certain amount of time without failing. Consequently, the probability that is being looked for is:

$$P(t < T < t + \Delta t \quad | \quad T > t) \quad (69)$$

In a general case, a conditional probability is computed with the formula:

$$P(A \quad | \quad B) = \frac{P(A \cap B)}{P(B)} \quad (70)$$

Here, $A \cap B$ is the intersection of the events A and B . Therefore, for Equation 69:

$$P(t < T < t + \Delta t \quad | \quad T > t) = \frac{P(t < T < t + \Delta t \cap T > t)}{P(T > t)} \quad (71)$$

The event $t < T < t + \Delta t$ is completely contained in the event $T > t$, thus the intersection of both events is simply $t < T < t + \Delta t$. Therefore:

$$P(t < T < t + \Delta t \quad | \quad T > t) = \frac{P(t < T < t + \Delta t)}{P(T > t)} = \frac{R(t) - R(t + \Delta t)}{R(t)} \quad (72)$$

Now, just as in the case of $f(t)$, an instantaneous density of conditional probability can be defined by dividing the result by Δt and taking the limit as Δt tends to zero:

$$h(t) = \lim_{\Delta t \rightarrow 0} \frac{1}{R(t)} \cdot \frac{R(t) - R(t + \Delta t)}{\Delta t} = -\frac{1}{R(t)} \cdot \frac{dR(t)}{dt} = \frac{f(t)}{R(t)} \quad (73)$$

The function $h(t)$ that has just been defined is known as “hazard rate” and it is an instantaneous density of *conditional* failure probability, the condition being that the device is still operating at t .

The probability of failure during a time lapse Δt in the near future can be estimated by using the hazard rate:

$$P(\text{failure in the near future} \mid \text{device is still operating at } t) \approx h(t) \cdot \Delta t \quad (74)$$

“Near future” means that the value of Δt in Equation 74 must be small, so that the resulting probability is much less than one. For large values of Δt , it can even happen that the resulting “probability” is larger than one, which clearly does not make sense.

3.3.2. Exponential distribution

The simplest case that can be imagined with the concepts just mentioned is that where the hazard rate is constant. This means that, while the device is operating, the chance of failure is always the same. In other words: provided that the device is still operating, the chance that it will fail within the next day is the same regardless of whether it was put into operation yesterday or a year ago.

Let λ be the constant hazard rate of the device under study. This value is also called “failure rate”. Because of Equation 73, it is possible to develop the following differential equation:

$$-\frac{1}{R(t)} \cdot \frac{dR(t)}{dt} = \lambda \quad (75)$$

By considering the ODE in Equation 75, and by imposing the condition $R(t = 0) = 1$, it is possible to get the solution:

$$R(t) = \exp(-\lambda \cdot t) \quad (76)$$

When a device has this kind of reliability function, the failure time is said to follow an “exponential distribution”. The expected (mean) time to failure, defined in Equation 68, is in this case:

$$MTTF = \int_0^{\infty} t \cdot \lambda \cdot \exp(-\lambda \cdot t) dt = \frac{1}{\lambda} \quad (77)$$

For the integration shown in Equation 77 it has been assumed that λ is greater than zero, which is true because λ is a failure rate.

3.3.3. Series system

Now, suppose that there is a system that is composed of many items, all of which could probably fail within some operation time. The system will be considered a “series system” if it depends on all the items of which it is composed in order to operate; i.e., if only one of the items of the system fails, the whole system fails. Therefore, in this case the term “series” has nothing to do with the spatial configuration of the items in the machine; it only means that all the items have to be operating in order for the machine to work.

Let $R_i(t)$ be the reliability function of the i -th item, with a total number of N items in the system or machine. The reliability of the system, $R_s(t)$, can be determined as:

$$R_s(t) = \prod_{i=1}^N R_i(t) \quad (78)$$

(For this case it is not necessary to have independence between the functional states of the items in the system, “independence” meaning that the functional state of any item does not affect the reliability of the others. If there were dependence, then the function $R_i(t)$ of each component must be considered only in the case where all other components of the system work. When any of them fails, then the value of the random variable T has been measured and the experiment is concluded; therefore, the case in which other items of the system have failed must not be taken into account.)

If all items have a constant failure rate (i.e. their failure times follow exponential distributions), the reliability of the system is equal to:

$$R_s(t) = \prod_{i=1}^N \exp(-\lambda_i \cdot t) = \exp\left(-\left(\sum_{i=1}^N \lambda_i\right) \cdot t\right) \quad (79)$$

Here, λ_i is the individual constant failure rate of each component of the system. This result shows that, in the case of a series system that is composed only of components with constant failure rates, the whole system has a constant failure rate that is equal to the sum of the failure rates of its components.

3.3.4. Parallel system

From a certain point of view, a parallel system can be considered as the “opposite” of a series system. Its definition is: from all the components of the system, it is only necessary that one of them operates in order for the system to be operative. In other words, the only way that a parallel system fails is that all its components fail. Its reliability, assuming that all components are operative at $t = 0$, and that the components are independent, can be calculated as one minus the probability that all components fail. For a system with N components:

$$R_s(t) = 1 - \prod_{i=1}^N (1 - R_i(t)) \quad (80)$$

(Here independence is needed, in contrast to the previous section, because in this case many combinations of operative and non-operative items are possible while the system operates. Thus, if the reliability of an item is affected by the states of the others, the reliability function becomes much more complex than Equation 80.) In the case of a parallel system where all the components have a constant and identical failure rate λ , the reliability function has the formula:

$$R_s(t) = 1 - (1 - \exp(-\lambda \cdot t))^N \quad (81)$$

This formula does not have the form of an exponential reliability; therefore, a parallel system that is composed of identical components with a constant failure rate does not have a constant failure rate itself.

3.4. Discrete-time Markov chains

A Markov process [47] (named after Andrey Andreyevich Markov, a Russian mathematician who lived from 1856 to 1922 [48]) is a mathematical model for a system that changes its state stochastically. The most important assumption is called “markovian property”, which implies that the probability of reaching a certain state at a certain moment only depends on the state in which the system was immediately before, and not on the history of previous states. In this study, only Markov processes with discrete and finite sets of possible states are considered. When a Markov

process has a discrete state space, it is commonly known as “Markov *chain*”. (The “states” discussed in this section, Section 3.4, are different from the “states” discussed in Section 3.2 regarding the Deep Reinforcement Learning process. As already discussed, states here are discrete and finite, and also the process does not involve executing actions and receiving rewards).

In discrete-time Markov chains, time is divided into discrete instants. The instants are identified as t_i , with i showing the number of the corresponding time step (i can take entire numbers equal to or larger than zero). At a certain instant t_i , given the current state of the system, there are certain probabilities of reaching a new state or staying in the same state at the next instant t_{i+1} . A way of visualizing a discrete-time Markov chain is shown in Figure 6, where a system with three possible states is considered.

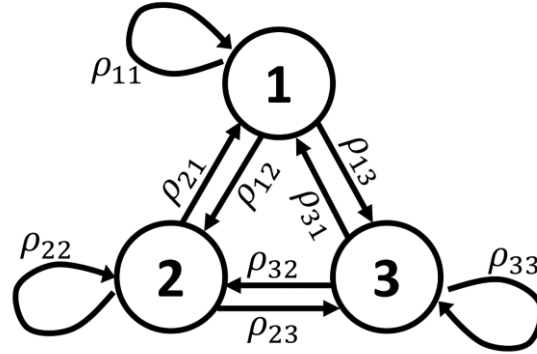


Figure 6: Illustration of a discrete-time Markov chain with three states

In Figure 6, the transition probabilities just mentioned are represented by arrows connecting the possible states of the system. ρ_{ij} is the probability of transiting from state i to state j between two consecutive instants. Although it is theoretically possible that the transition probabilities depend on time, for this study they will be assumed to be constant. A Markov process with this property is known as “time-homogeneous Markov process”.

Let $X(t_i)$ be the state of the system at t_i , with $i \in \mathbb{N}$ representing the number (in time steps, since the beginning of the process) of the instant t_i . Also, let S be the set of possible states of the system. At every moment, the system must be in one of its possible states, thus:

$$\sum_{j \in S} P[X(t_i) = j] = 1 \quad \forall i \geq 0 \quad (82)$$

Between two instants, the system must either stay in the same state or transit to another one, thus:

$$\sum_{j \in S} \rho_{ij} = 1 \quad \forall i \in S \quad (83)$$

A way of mathematically illustrating the markovian property mentioned above is:

$$P[X(t_{i+1}) = s \mid X(t_i) = s_i] = P[X(t_{i+1}) = s \mid X(t_0) = s_0, X(t_1) = s_1, \dots, X(t_i) = s_i] \quad (84)$$

3.4.1. Geometric distribution

The geometric distribution is a probability distribution that is very useful for time-discrete Markov chains. A random variable Y follows the geometric distribution if it measures the number of independent repetitions of an experiment that are needed until a certain result of the experiment,

defined as “success”, is obtained; all repetitions of the experiment have the same success probability p [49]. Therefore, the possible values for Y are all entire numbers equal to or greater than one. (There is a very similar version of the distribution that counts the number of *failed* repetitions until a success is obtained, “failure” meaning that any result other than success is obtained. In this case, the only difference is that the random variable takes values equal to or greater than zero. However, the version explained above will be used.)

The probability that the random variable Y takes a certain value k is given by [49]:

$$P(Y = k) = (1 - p)^{k-1} \cdot p \quad k \in \mathbb{Z}, k \geq 1, p \in [0, 1] \quad (85)$$

The expected value for Y is [49]:

$$\mathbb{E}(Y) = \frac{1}{p} \quad (86)$$

An example of how to use this distribution is to determine the expected permanence time of the system in a certain state. This follows a geometric distribution because one is counting the number of time steps (repetitions of the experiment) until the system leaves the initial state (success).

Let j be the state of the system at t_i . The probability of changing state in the next instant is:

$$P[X(t_{i+1}) \neq j \mid X(t_i) = j] = 1 - \rho_{jj} = \sum_{k \neq j} \rho_{jk} \quad j, k \in S; i \in \mathbb{N} \quad (87)$$

If the number of time steps of permanence in a certain state j is the random variable that is being measured, then the event “leaving state j ” can be considered as a “success” and thus the parameter p defined in Equation 85 would be equal to $1 - \rho_{jj}$ (the geometric distribution can only be used if the Markov chain is time-homogeneous; otherwise, the value of p could not be defined). The convention on the number of time steps will be that, if the system leaves state j in the first possible transition, it has spent one time step in state j . The expected value for that random variable is:

$$\mathbb{E}(\text{time steps of permanence in } j) = \frac{1}{1 - \rho_{jj}} = \frac{1}{\sum_{k \neq j} \rho_{jk}} \quad j, k \in S \quad (88)$$

3.4.2. Steady-state probabilities

First, it is necessary to clarify the concept of “absorbing states”: a state is called “absorbing” if the probability of leaving it is zero; therefore, once the system has reached an absorbing state, it will stay in that state forever.

For the Markov chains considered during this study, and also for this theoretical part, it is assumed that all states are reachable by the system regardless of the state where the Markov chain started. However, not all states need to be directly connected to each other by a nonzero transition probability. For this condition to be true, the system cannot have absorbing states.

Let $P_{ij}(m, n)$, with $n > m$, be the probability that the system is in the state j at the instant t_n , assuming that at t_m it was in the state i :

$$P_{ij}(m, n) = P[X(t_n) = j \mid X(t_m) = i] \quad i, j \in S; m, n \in \mathbb{N}; n > m \quad (89)$$

The probability $P_{ij}(m, n)$ does not depend on both values m, n but only on their difference $n - m$, because the process is time-homogeneous (the experiment does not depend on when it is initiated). Therefore, the result only depends on the number of time steps between t_m and t_n .

For simplicity, let $P_{ij}^*(n) = P_{ij}(m, m + n)$ for any natural number m . The function P^* yields the same result as P but only takes the number of time steps between the instants into account. If the Markov chain meets the condition that all states are reachable by starting in any other state, then the value of $P_{ij}^*(n)$ tends to become independent of the initial state i when the number of time steps n becomes large enough:

$$\lim_{n \rightarrow \infty} P_{ij}^*(n) = \lim_{n \rightarrow \infty} P_{kj}^*(n) \quad \forall i, k, j \in S \quad (90)$$

That same value will be designated simply as P_j , because it does not depend on the initial state:

$$\lim_{n \rightarrow \infty} P_{ij}^*(n) = \lim_{n \rightarrow \infty} P_{kj}^*(n) = P_j \quad \forall i, k, j \in S \quad (91)$$

The number P_j is the steady-state probability that the system is in the state j . This number can be interpreted as the percentage of time that the system will spend in the state j when enough time has passed so that the initial state has no effect on the future state of the system. This is called “steady-state regime”.

As discussed above, it is being assumed that all states of the system are reachable from any other state. A few examples of systems where that does not happen are:

1. If the system has an absorbing state, P_j is equal to 1 only if j is the absorbing state. Else, P_j is equal to 0.
2. If the system has several absorbing states, the system will reach one of them and stay there forever. However, that final state of the system is not determined beforehand and, most importantly, is dependent on the initial state. Thus, the probability of reaching a certain absorbing state will change depending on which the initial state was. One could propose an “analog” to P_j for this case, where P_j represents the probability of reaching the state j as final absorbing state. However, there will be several possible sets of “ P_j ’s”, one for each possible initial state. This is not what is being looked for here.
3. If the system has a state (or more than one) from which the system can “get out” but never “get in” again (because there are no transitions conducting to that state), then the value of P_j for that state will be zero. The other states, where the system can get in and out repeatedly, would have P_j ’s larger than zero.
4. If the system has a completely isolated state or a group of states which is disconnected from the rest of the states (i.e. with no transition probabilities entering or leaving the isolated group), then the final regime of the system will depend on the initial state. Therefore, it would not be possible to define a single set of “ P_j ’s”.

The idea of imposing that all states are reachable from any other state is that the system eventually reaches all of them repeatedly, so that none of the cases named above happens. If this is true, then the value of P_j is greater than zero for all states; i.e. $P_j > 0 \quad \forall j$.

At any moment, not necessarily at steady-state regime, the probability that the system is in state j at t_i can be calculated with the probabilities of the previous instant t_{i-1} :

$$P[X(t_i) = j] = \sum_{k \in S} \rho_{kj} \cdot P[X(t_{i-1}) = k] \quad j \in S ; i \in \mathbb{N} \quad (92)$$

In the steady-state regime, the probability of being in any state is constant in time, thus it does not depend on the instant. Therefore, Equation 92 becomes:

$$P_j = \sum_{k \in S} \rho_{kj} \cdot P_k \quad j \in S \quad (93)$$

If the transition probabilities ρ_{kj} are known and one wants to know the probabilities of being in each state at steady regime, a system of equations can be imposed:

$$\sum_{k \in S} (\rho_{kj} - \delta_{kj}) \cdot P_k = 0 \quad j \in S \quad (94)$$

δ_{kj} is the Kronecker Delta. Now, assuming that each state of the set S can be numbered with a natural number from 1 to N , Equation 94 can be re-written as a matrix equation:

$$\begin{bmatrix} \rho_{11} - 1 & \rho_{21} & \cdots & \rho_{N-1,1} & \rho_{N,1} \\ \rho_{12} & \rho_{22} - 1 & \cdots & \rho_{N-1,2} & \rho_{N,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \rho_{1,N-1} & \rho_{2,N-1} & \cdots & \rho_{N-1,N-1} - 1 & \rho_{N,N-1} \\ \rho_{1,N} & \rho_{2,N} & \cdots & \rho_{N-1,N} & \rho_{NN} - 1 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N-1} \\ P_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \quad (95)$$

Note that all columns of the matrix add up to zero, because of the property expressed in Equation 83. Therefore, the matrix is singular. Here is a small proof: Let \mathbf{A} be the matrix of Equation 95 (the one that is multiplying the vector of probabilities) and \mathbf{A}^T be the transpose of \mathbf{A} . Given that the columns of \mathbf{A} add up to zero, the rows of \mathbf{A}^T add up to zero as well. Therefore, \mathbf{A}^T has the eigenvalue 0 associated to the eigenvector $[1, 1, \dots, 1, 1]^T$. Given that the determinant is the product of the eigenvalues, the determinant of \mathbf{A}^T is zero. Therefore, \mathbf{A}^T is singular and \mathbf{A} is singular as well (since transposed matrices have the same determinant). If the matrix \mathbf{A} were not singular, then the only solution to Equation 95 would be $P_i = 0 \quad \forall i$.

To correct the redundancy of the equations in the matrix, it is possible to replace one of them with the known condition:

$$\sum_{k \in S} P_k = 1 \quad (96)$$

By replacing the last row of the matrix with the condition shown in Equation 96, the matrix equation becomes:

$$\begin{bmatrix} \rho_{11} - 1 & \rho_{21} & \cdots & \rho_{N-1,1} & \rho_{N,1} \\ \rho_{12} & \rho_{22} - 1 & \cdots & \rho_{N-1,2} & \rho_{N,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \rho_{1,N-1} & \rho_{2,N-1} & \cdots & \rho_{N-1,N-1} - 1 & \rho_{N,N-1} \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N-1} \\ P_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \quad (97)$$

Therefore, the steady-state probabilities can be obtained with:

$$\begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N-1} \\ P_N \end{bmatrix} = \begin{bmatrix} \rho_{11} - 1 & \rho_{21} & \cdots & \rho_{N-1,1} & \rho_{N,1} \\ \rho_{12} & \rho_{22} - 1 & \cdots & \rho_{N-1,2} & \rho_{N,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \rho_{1,N-1} & \rho_{2,N-1} & \cdots & \rho_{N-1,N-1} - 1 & \rho_{N,N-1} \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \quad (98)$$

The result would be the same regardless of which row of the matrix is replaced by the condition of Equation 96.

3.5. Heat pipe evacuated tube solar collectors

Solar collectors are devices meant to heat a liquid flow (in this case water) by absorbing radiation from the sun.

One of the most popular types of solar collectors is called “evacuated tube” solar collectors [50]. This name comes from the fact that the radiation-absorbing device is inside a vacuum environment provided by a glass tube. The transparency of the glass allows the radiative energy to reach the absorbing device, and the vacuum prevents heat losses produced by direct contact with the atmosphere (through conduction and convection).

A way of heating the water flow is by directly letting water pass through a radiation-absorbing tube inside the evacuated tube. Another way is by using a device called “heat pipe”. A heat pipe is a closed metal tube with a fluid restricted to its interior. The tube is usually made of copper and the inner fluid is usually a type of alcohol. The fluid receives the thermal energy through the walls of the tube and evaporates; then the vapor flows to a heat exchanger where it transfers heat to the water flow and condenses. The liquid then goes back to the heating part of the pipe and repeats the cycle. The movement of vapor and liquid is usually accomplished by putting the condenser in the highest part of the heat pipe; in this way, the movement of vapor and liquid is induced by gravity.

A popular design of evacuated tube solar collectors uses a double-walled glass tube; the vacuum is kept between the two walls. The outer face of the inner wall is covered with an absorbing coating, as Figure 7 shows. This coating is commonly known as “selective coating” and it must meet two basic requirements: high absorptance and low emissivity. This allows the device to receive high amounts of radiative energy and to lose a low amount of it due to emissions.

The heat pipe is located in the center of the double-walled tube. The thermal energy is transferred from the inner tube to the heat pipe by conduction through aluminum fins, which also fulfill the task of mechanically holding the heat pipe at the center of the tube. At the right part of Figure 7, the aluminum fins are drawn with grey lines. A small separation between the fins and the other parts has been left in the drawing in order to better differentiate the components, although in reality, the fins must be in direct contact with the inner tube and the heat pipe.

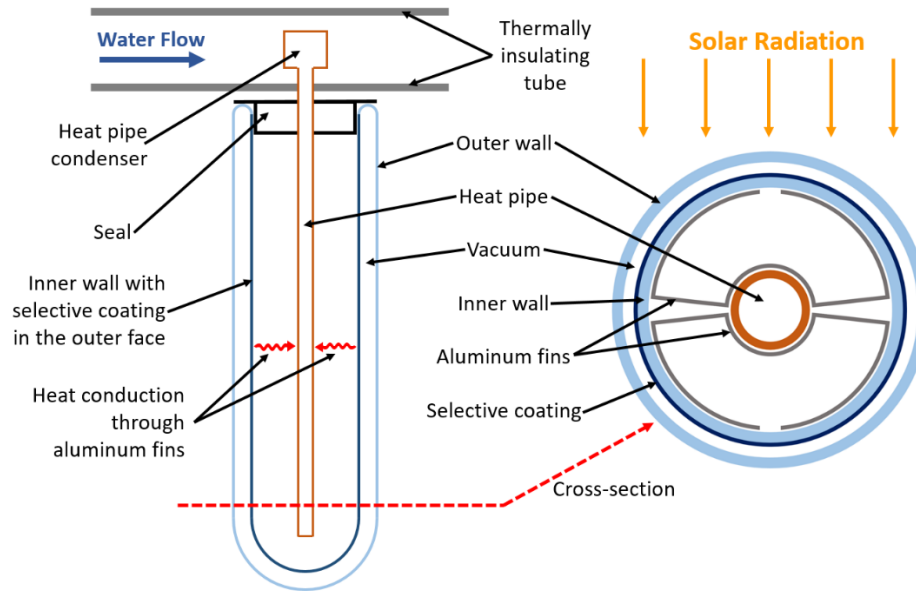


Figure 7: Basic components of a heat pipe evacuated tube solar collector.

In order to increase the absorbing area without needing to build a tube with a very large diameter, a solar collector is built by arranging several tubes in series, as shown at the left of Figure 8. Given that a fraction of the radiation will pass between the absorbing (internal) tubes, it is useful to install a reflector under the tube arrangement, in order to reflect the energy back into the tubes. This can increase the total amount of energy received by more than 25% [50].

As already discussed, the flow inside the heat pipe is normally induced by gravity. For this reason, and also to optimize the absorption of solar radiation, the solar collector is given a small inclination with respect to the ground, as shown in the right part of Figure 8.

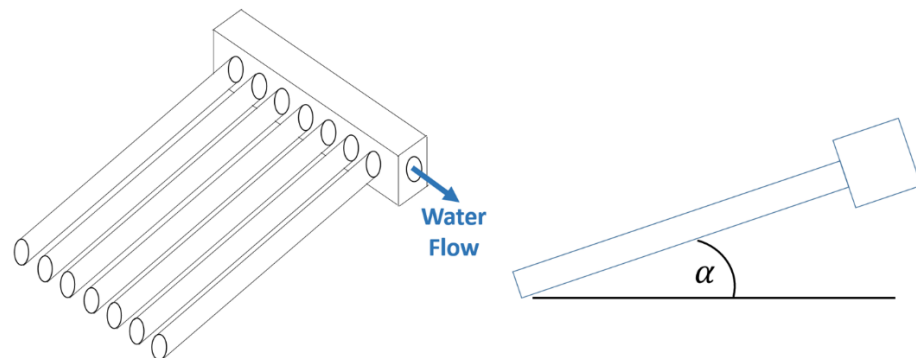


Figure 8. Left: Arrangement of tubes in a solar collector.
Right: Inclination of the collector with respect to the ground.

3.6. Heat pumps and refrigeration systems

Heat pumps and refrigeration systems work basically the same way; the only difference is the task they are meant to perform. Heat pumps are meant to deliver heat to a hot environment by extracting heat from a cold environment, whereas refrigeration systems are meant to cool a cold environment by transferring the extracted heat to a hotter environment. Thus, they do exactly the same, but receive different names depending on what the goal of the process is. Many air-conditioning systems are capable of fulfilling both tasks; i.e., they act as cooling machines during summer and as heating machines during winter.

To achieve this, a fluid goes through the following cycle (illustrated in Figure 9): in a heat exchanger called “evaporator”, the fluid extracts heat from the cold environment (or cold region) and evaporates. To achieve this, the fluid must be colder than the cold region. Then, a compressor compresses the gas with energy from an external source; this process increases the temperature of the gas. Due to this temperature increase, the gas is now able to transfer heat to the hot region; this process is carried out in a heat exchanger called a “condenser”, where the fluid releases heat by condensing. Finally, the fluid passes through an expansion valve back into the low-pressure zone. Due to this pressure drop, part of the fluid evaporates and the temperature of the fluid decreases; with this, the fluid is able to absorb heat from the cold region again. In Figure 9, \dot{Q}_{in} and \dot{Q}_{out} are the heat flow absorbed by the fluid from the cold region and the heat flow transferred by the fluid to the hot region, respectively. \dot{W}_{in} is the power (work) provided to the fluid by the compressor.

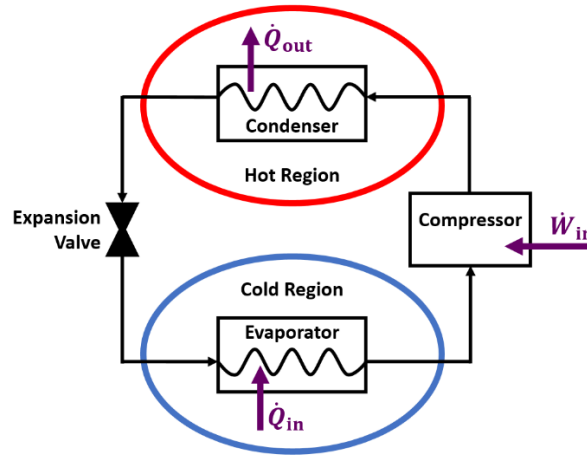


Figure 9: Illustration of the operation of refrigeration systems and heat pumps.

When steady state has been reached, the net energy exchange between the machine and the environment is zero, thus:

$$\dot{Q}_{in} + \dot{W}_{in} + \dot{Q}_{out} = 0 \quad (99)$$

In Equation 99, \dot{Q}_{in} and \dot{W}_{in} are considered to be positive, and \dot{Q}_{out} is considered to be negative.

The efficiency of the system is measured by an indicator called Coefficient of Performance (COP). This coefficient can assume two definitions, depending on the purpose of the equipment. If the device is used as a heating device, it is defined as the ratio between the heat provided to the hot region and the work consumed by the compressor. On the other hand, if the device is used as a cooling machine, it is defined as the ratio between the heat absorbed from the cold region and the work consumed by the compressor. In both cases, the COP is a measure of the ratio between the useful energy and the consumed energy.

$$COP_{heating} = \frac{|\dot{Q}_{out}|}{\dot{W}_{in}} \quad (100)$$

$$COP_{cooling} = \frac{\dot{Q}_{in}}{\dot{W}_{in}} \quad (101)$$

Because of Equation 99, it is easy to see that:

$$COP_{cooling} = COP_{heating} - 1 \quad (102)$$

This relation is not exactly true in actual equipments. For example, a fan might be used to force air to pass through a heat exchanger. This is power-consuming, and thus is considered as part of \dot{W}_{in} in Equations 100 and 101, but does not add work to the compression process of the gas, thus it is not part of \dot{W}_{in} of Equation 99 anymore. However, this ideal scenario is illustrative for understanding the thermodynamic principles of this type of devices.

Let T_H and T_C be the temperature in the hot region and in the cold region, respectively. Because of the fact that \dot{W}_{in} can be smaller than \dot{Q}_{in} and \dot{Q}_{out} (considering the absolute values of these energy flows), both indicators $COP_{heating}$ and $COP_{cooling}$ can be larger than one; this means that the “useful” energy is greater than the energy consumed. The theoretical maximum possible values for $COP_{heating}$ and $COP_{cooling}$ are given by the temperatures of the hot region and the cold region when reversibility is imposed (Moran et al. [51]):

$$COP_{heating,max} = \frac{T_H}{T_H - T_C} \quad (103)$$

$$COP_{cooling,max} = \frac{T_C}{T_H - T_C} \quad (104)$$

However, the actual values of these indicators are considerably lower in actual machines (see Moran et al. [51] for details).

3.7. TRNSYS

TRNSYS (Transient System Simulation Tool [15]) is a simulation software that allows modelling of transient systems. Although it is mostly used to simulate thermal energy systems, its main capability is the option to easily create new components depending on the needs of every user. This makes it extensible to a very wide range of areas. A simulation is created by adding and connecting the different components of the system (such as pumps, heat exchangers, storage tanks, heat pumps, and solar collector fields) and specifying their features. Climate data from many cities around the world are also available for the simulation of devices such as solar collectors and air-water heat pumps.

The “components” used to build simulations are called “types” by TRNSYS. For example, there are types to simulate different kinds of pumps (single-speed, variable-speed), heat pumps (air to water, water to water, etc.), solar collectors, heat exchangers, etc. Each type is identified by a number, e.g., Type 114 is a single-speed pump, and Type 110 is a variable-speed pump.

To better clarify how TRNSYS works and what it does, a very simple example is shown in Figure 10 and is explained below.

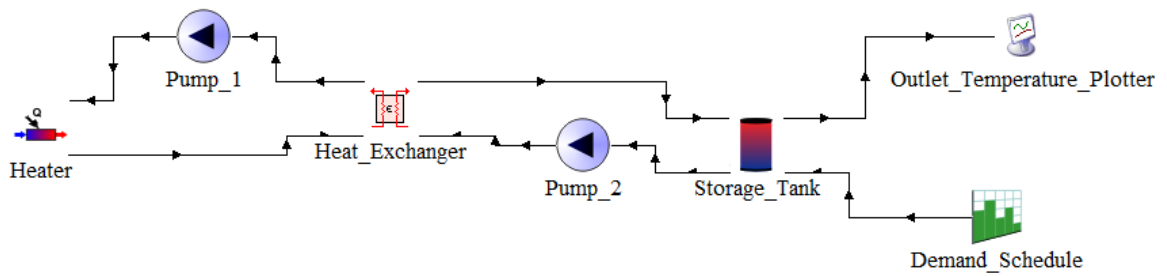


Figure 10: Example system implemented in TRNSYS

In the system shown in the figure, the heater is meant to heat the water stored in the storage tank. Pump 1 moves water through the heater, while Pump 2 recirculates the water contained in the tank. Both flows can exchange heat in the heat exchanger in the middle. The tank stores a constant volume of water and it has two inlets and two outlets: one inlet-outlet pair is used to move the water through the heat exchanger; the other inlet-outlet pair is used to receive water from the mains and to deliver it to the user. The “Demand Schedule” that can be seen at the right is simply a time-dependent function that imposes the flow entering the storage tank. The flow leaving the tank will be automatically equal to this imposed flow, thus it can be interpreted as the demanded water flow. The “Outlet Temperature Plotter” receives the temperature of the water that leaves the tank and plots it. Figure 11 shows the demand profile assumed and the resulting water temperature during one simulation day. The initial temperature of the water in the tank was set to 20°C, as well as the temperature of the mains water flow that enters the tank.

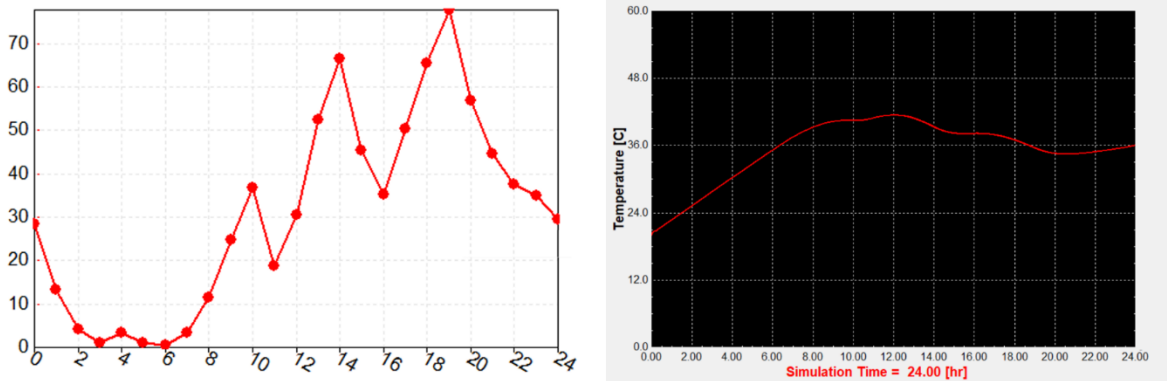


Figure 11. Left: Water demand profile in L/hr. Right: Temperature of the water leaving the tank.

To connect two elements of the system (e.g., the pump and the heater), the outputs of one of them must be connected to the inputs of the other. Inputs are variables that an element of the system receives from other elements, while outputs are variables that an element delivers to other elements. Figure 12 shows how the connection between Pump 1 and the heater is being established. Two outputs of the pump (the outlet fluid temperature and the outlet flow rate) are connected to two inputs of the heater (the inlet fluid temperature and the inlet flow rate). The inputs that are not connected to any output of any other element remain at a fixed value during the simulation.

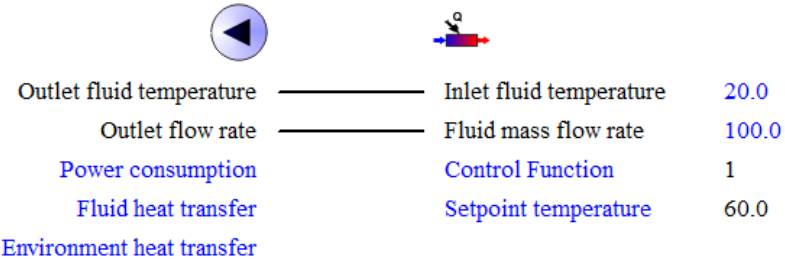


Figure 12: How to edit the connection between two elements by linking outputs to inputs

In addition to the inputs and outputs, for most types it is necessary to specify certain characteristics which are called “parameters”. Parameters cannot be passed to other elements as outputs, and they remain fixed during the simulation. Examples can be the volume of a tank, the power of a pump, the maximum power of a water heater, etc.

Chapter 4: Development of the training platform

4.1. System under study

The water heating system that the agents will seek to control corresponds to the installation that operates at the Faculty of Physical and Mathematical Sciences (FCFM) of the University of Chile, Santiago. The system delivers warm water to the dressing rooms in the sports area of the building, which is located in Beauchef 851, Santiago. Most of the data and the variables considered are based on previous work by Camila Correa [52] and Camila Correa et al. [32] on the same system.

The system is composed of three heating stages: a solar thermal energy stage, a heat recovery system from a water chiller, and four air-water heat pumps. The three stages heat the water flow in the mentioned order.

The solar stage consists of 44 heat-pipe evacuated tube solar collectors, which together have a total absorbing area of 105.6m^2 . Their brand and model code is “Hitek Solar NSC 58-30”. They are oriented towards north, with a tilt angle (the α angle shown in Figure 8) of 15° (Correa, 2019 [52]). The collectors are located on the roof of the building; their spatial distribution is shown in Figure 13. In the figure, North is indicated by an arrow. The image in the figure was taken from the thesis of Camila Correa [52] with her permission.

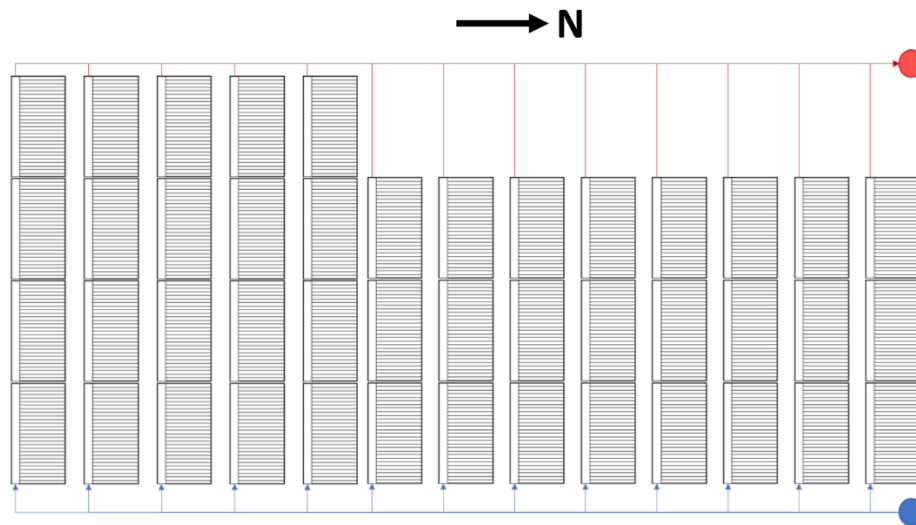


Figure 13: Spatial configuration of the solar collectors. Source: Camila Correa [52] “*Assessment of Deep Learning Techniques for Diagnosis in Thermal Systems through Anomaly Detection*”. 2019

As Figure 13 shows, the solar collectors are arranged in 13 parallel rows; in each row the collectors are arranged in series. Five rows have four collectors each, and eight rows have three collectors each.

The water chiller is a cooling machine, as discussed in Section 3.6, whose main purpose is to cool a water flow. The cooled water is then used to control the temperature or humidity inside the building. The heat rejected by the machine is passed to another water flow, and in regular systems, it can be released to the outside with a cooling tower. However, this heat can also be used instead of being dropped; this is the idea behind heat recovery. Thus, the second stage of the water heating system consists of using the heat rejected by the chiller to heat the water for the dressing rooms.

The third heating stage consists of four heat pumps, each of which heats the water contained in an individual storage tank. The heat pumps are intended to keep the water in the tanks at 60°C. They are automatically turned off if the temperature in the tanks reaches 62°C and are turned on again when the temperature drops to 55°C.

If the temperature of the water is above 45°C at the outlet of the third heating stage, it is mixed with a flow of mains water until it reaches 45°C, and then it is delivered to the dressing rooms. A diagram of the system is shown in Figure 14 and is explained below.

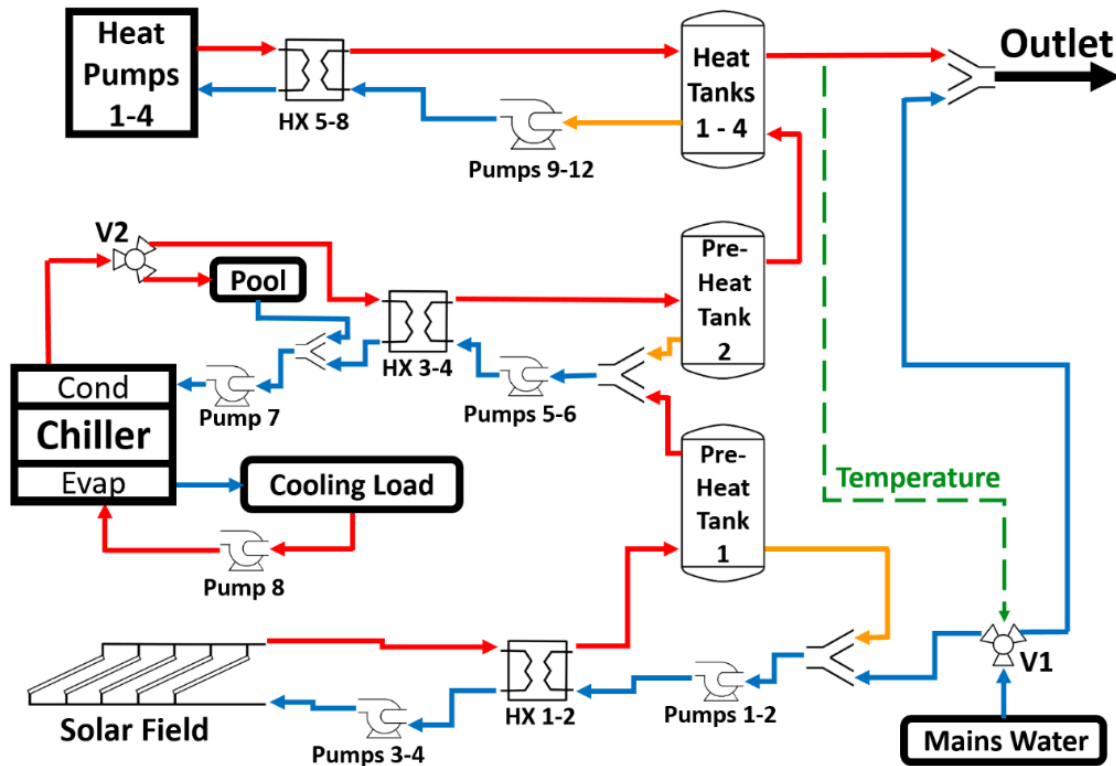


Figure 14: Diagram of the water heating system.

All explanations in the three next paragraphs are referencing the illustration shown in Figure 14.

The mains water flow enters the system through valve V1 (bottom right); this flow is equal to the warm water demand in the dressing rooms. Valve V1 receives the temperature at the outlet of the last heating stage (green dotted arrow) and splits the entering water flow, sending part of it directly to the outlet of the system, in order to maintain the delivered water at 45°C. If the water leaving the last heating stage is below 45°C, the entire flow is sent by valve V1 to the heating system. The volume of water stored in the system is constant, thus the flow entering it is always equal to the flow leaving it.

Preheating Tank 1 stores the water that is being heated by the solar collectors, and Preheating Tank 2 stores the water that is being heated by the heat recovery system of the chiller. In the last heating stage, each heating tank (1 through 4) receives heat from a heat pump. Pumps 1, 2, 5, 6 and 9 to 12 are permanently recirculating the water contained in their respective tanks in order to keep the heat exchange with the corresponding energy source regardless of the current warm water demand. The flows that are taken out of their tanks to be recirculated are shown with orange arrows.

The heat recovery system of the chiller is also used to warm the pool of the building. Valve V2 splits the flow leaving the condenser side of the chiller; 51% of the flow is used to heat the sanitary

water and the remaining 49% is used for the pool. The pool is not simulated in detail; instead, it is assumed that the water coming from the pool has a constant temperature of 44.5°C.

The demand for warm water in the dressing rooms is estimated at 24000 L/day. The assumed demand curve during each day is shown in Figure 15.

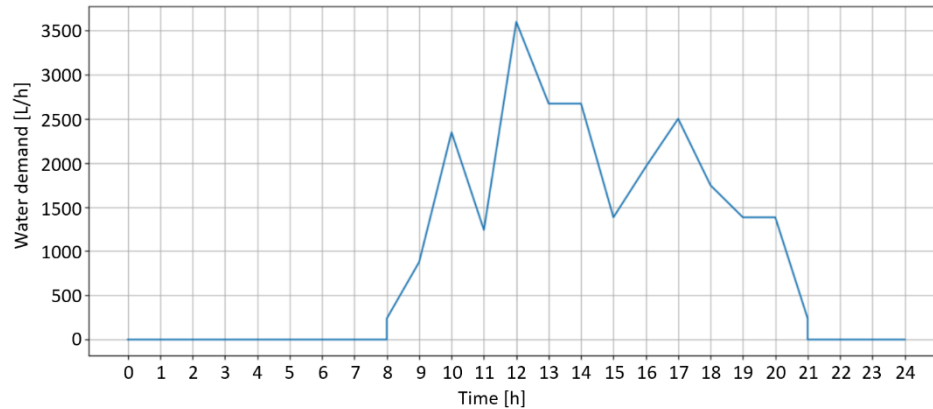


Figure 15: Estimated water demand during the day. Source: Camila Correa et al. [32]

The system is turned on at 7.00 AM and turned off at 9.00 PM (this includes all electrical devices).

In this section, only the essential details which are necessary to understand the study were given. Further details about the water heating system are given in Annexed A.

4.2. Actions

In this section, the agent-system interaction schedule and the possible actions are explained. The agent is allowed to execute 7 actions per simulation-day; these actions are executed at 8.00 AM, 10.00 AM, 12.00 PM, 2.00 PM, 4.00 PM, 6.00 PM and 8.00 PM. At every action instance, the agent decides which heating stages of the system (from the three heating stages discussed above) will be turned on and which will be turned off until a new action is required. States received by the agent at 10.00 PM are terminal states; this means that in this environment an “episode” is equivalent to one day.

In addition to the three heating stages, there is a fourth “degree of freedom” of the system that the agent is able to control. Here is why: it was discovered that at moments of high solar radiation and medium warm water demand, the temperature in Preheating Tank 1 (which receives heat from the solar collectors) can reach over 70°C; this is considerably higher than the target temperature in the last stage of the system (60°C). For safety reasons, a control system was implemented to automatically turn off Pumps 3 and 4 (see Figure 14) when the temperature in Preheating Tank 1 reaches 50°C, so that the tank stops heating up (the pumps are turned on again when the temperature in the tank drops to 45°C). Nevertheless, this leads to another problem which is overheating of the solar collectors, because thermal energy accumulates if the water is stagnant inside the collectors. This can lead the water in the collectors to reach temperatures over 100°C, according to the TRNSYS simulation. To solve this issue, the controlling agent has the option of activating an auxiliary flow in order to prevent Preheating Tank 1 from reaching the limit temperature of 50°C. This auxiliary flow is an extra flow of 3,000L/h that flows through the heating system and is not meant to be used in the dressing rooms, but only to extract excessive heat. This solution is not optimal because heat and water are being wasted, but it may be better than early failure of the solar collectors due to overheating.

Thus, the four independent systems that the agent is able to activate and deactivate are:

- The solar energy stage
- The heat recovery stage (chiller)
- The heat pump state
- The auxiliary flow to prevent the solar collectors from overheating

The reward of an action, whose formula will be detailed in the next section, is computed as a function of the performance of the water heating system in the timespan between the aforementioned action and the next action. In the case of the last action of the day, executed at 8.00 PM, its reward is computed at 10.00 PM, but no further actions are executed. This is considered to be a terminal state, as discussed in Section 3.2.1. At that time of day, the three heating stages are “turned on” and the auxiliary flow is turned off, but in practice everything is turned off since the whole heating system is turned off at 9.00 PM. On the next day, at 7.00 AM, the system is turned on, and therefore the three heating stages are turned on until the first action of the day is executed at 8.00 AM.

Figure 16 illustrates the daily schedule for the Deep Q-Learning method (see Section 3.2.5 and subsections for more details about the algorithm). At 8.00 AM no reward is computed, since there is no previous action to compute a reward for. At 10.00 PM, no action is executed, so the agent does not need to process the state of the system. Also at 10.00 PM, the neural network is trained by sampling a random set of experiences from the Replay Memory.

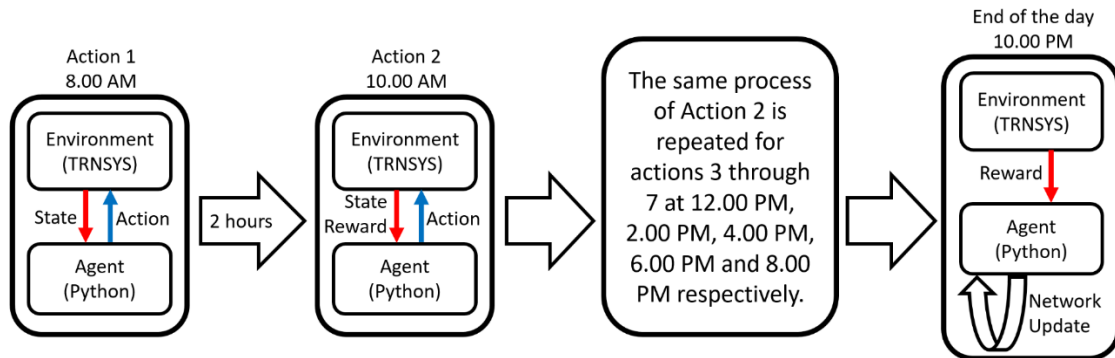


Figure 16: Interaction schedule between the agent and the water heating system (DQN algorithm)

In the case of the REINFORCE method (see Section 3.2.3), the process is very similar but the DNN is not updated every day. Instead, every certain number of days (episodes), the parameters are updated at 10.00 PM. On other days, the data is just stored for the next training iteration.

In the case of the Actor-Critic algorithm (see Section 3.2.4), the training iterations take place at every interaction instance except for the first interaction of the day at 8.00 AM. This is because a reward is needed to train the two deep neural networks. Therefore, the networks are trained at 10.00 AM, 12.00 PM, 2.00 PM, 4.00 PM, 6.00 PM, 8.00 PM and 10.00 PM.

Since the agent has four degrees of freedom with two options for each one, there are 16 possible actions to choose from, given by all possible combinations. All actions are shown in Table 1, where 1 means “on” and 0 means “off”.

Table 1: Possible actions of the agent.

Action	Solar field pumps	Chiller	Heat Pumps	Auxiliary Flow
0	1	1	1	0
1	1	1	0	0
2	1	0	1	0
3	0	1	1	0
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	0
8	1	1	1	1
9	1	1	0	1
10	1	0	1	1
11	0	1	1	1
12	1	0	0	1
13	0	1	0	1
14	0	0	1	1
15	0	0	0	1

4.3. Rewards

To calculate the reward of an action, the performance of the system is measured between that action and the next one. In the case of the last action of a day, executed at 8.00 PM, the reward is computed at 10.00 PM, but no action is executed at that time because the entire heating system is already off.

The indicators that are taken into account to compute the rewards are:

- Whether the system is delivering warm water as it is supposed to.
- Whether the solar collectors have reached an excessive internal temperature.
- Whether the auxiliary flow is being used.
- Whether the chiller is being used.
- The ratio between the heat delivered to the water flow and the electric consumption of the system.
- The ratio between the heat coming from clean sources and the total heat delivered. The concept of “clean sources” is discussed below.

Now the quantitative definition of these indicators will be discussed. In the following equations, α_1 through α_5 are parameters to define the importance of each indicator.

Comfort indicator

It is the prize for delivering warm water, and is defined as:

$$\text{comfort} = \int_{t_0}^{t_1} \mathbb{1}_{T_{hot} > 40^\circ\text{C}}(t) \cdot \left(1 + \alpha_2 \cdot \left(\frac{T_{hot}(t) - 40}{20} \right)^2 \right) dt \quad (105)$$

T_{hot} is the temperature of the water flow leaving the third heating stage before being mixed with the mains water flow to be delivered to the dressing rooms (see Figure 14). Therefore, this temperature can have a maximum value of 62°C (at that point the heat pumps are turned off). The function $\mathbb{1}$ is a Boolean indicator function as follows:

$$\mathbb{1}_{T_{hot} > 40^{\circ}C}(t) = \begin{cases} 1 & \text{if } T_{hot} > 40^{\circ}C \text{ at } t \\ 0 & \text{if } T_{hot} \leq 40^{\circ}C \text{ at } t \end{cases} \quad (106)$$

The comfort factor is computed as an integral between the instants t_0 and t_1 , which are the instant of the last action and the instant of the next action (or the end of the day), respectively. Note that the integrand will only be greater than zero if the delivered water's temperature is higher than 40°C. If the water's temperature is lower than 40°C during the whole time between an action and the next one, this indicator will be zero.

The α_2 parameter measures the importance of reaching temperatures remarkably higher than 40°C. If α_2 is zero, the comfort factor will only depend on whether the temperature was higher or lower than 40°C; if α_2 is greater than zero, then a higher temperature of the water leaving the heating system will yield a larger reward. The heat pumps in the last heating stage are designed to keep the water at 60°C, so the fraction multiplying α_2 will take values between 0 and 1 (or slightly greater than 1). Therefore, α_2 can be used as an "incentive" for the agent to use the heat pumps in order to deliver warmer water.

Degradation indicator

It is a penalization for overheating of the solar collectors. It is calculated as:

$$\text{degradation} = \alpha_3 \int_{t_0}^{t_1} \mathbb{1}_{T_{col} > 100^{\circ}C}(t) dt \quad (107)$$

T_{col} is the temperature at the outlet of the solar collector fields. Thus, the degradation factor is proportional to the time during which T_{col} exceeded 100°C, so the agent has to prevent this from happening.

Water use indicator

It is a penalization for using the auxiliary flow because of the water and energy waste. It is defined as follows:

$$\text{water use} = \begin{cases} \alpha_4 & \text{if the auxiliary flow is used} \\ 0 & \text{if the auxiliary flow is not used} \end{cases} \quad (108)$$

Clean heat indicator

The term "clean heat" will be used to refer to the heat coming from the solar collectors and the water chiller. These sources are considered to be clean because solar energy is renewable, and the heat rejection from the chiller is the heat that was extracted for another purpose, which is chilling another water flow. Therefore:

$$\text{Clean Heat} = (\text{Heat from Solar Collectors}) + (\text{Heat from Chiller}) \quad (109)$$

Reward functions

Two reward functions will be defined. Clearly, they cannot both be used at the same time to train an agent, since the agents only try to maximize *one* reward function.

The first reward function is defined as:

$$R_1 = \left(\alpha_1 \cdot \frac{\text{total heat}}{\text{electric consumption}} + (1 - \alpha_1) \cdot \frac{\text{clean heat}}{\text{total heat}} \right) \cdot \frac{\text{comfort}}{(1 + \text{degradation}) \cdot (1 + \text{water use})} \quad (110)$$

“Total heat” refers to the total amount of thermal energy delivered to the water flow. “Electric consumption” refers to the total consumption by all electric devices of the system. Thus, the fraction “total heat/electric consumption” is a prize for increasing the amount of heat delivered, while at the same time decreasing the consumption of electric energy. The fraction “clean heat/total heat” is a prize for increasing the heat coming from clean sources (solar collectors and chiller). α_1 is a parameter in the interval $[0, 1]$ to measure the importance given to each of these two fractions.

Note that the reward is defined so that the primary purpose of the agent is to keep a warm water supply in the dressing rooms. If it does not do it, the reward becomes automatically zero.

In reward function shown by Equation 110, a small value of α_1 means an “incentive” to use either the solar collectors or the chiller to heat the water flow. However, there is no direct incentive to activate the chiller, whereas in reality, it might be desirable to use the chiller for reasons which are external to the water heating system, like cooling the building, or heating and dehumidifying the pool (actually, these are the main purposes of the chiller). For this reason, a second reward function is proposed with a direct prize for using the chiller:

$$R_2 = \left(\frac{\text{total heat}}{\text{electric consumption}} + \alpha_5 \cdot (\text{Chiller Use}) \right) \cdot \frac{\text{comfort}}{(1 + \text{degradation}) \cdot (1 + \text{water use})} \quad (111)$$

Here, “Chiller Use” is defined as:

$$\text{Chiller Use} = \begin{cases} 1 & \text{if the Chiller was used} \\ 0 & \text{if the Chiller was NOT used} \end{cases} \quad (112)$$

α_5 is a parameter to measure the “prize” that is given to the agent for using the chiller. Clearly, this way of encouraging the agent to activate the chiller is not very realistic, because in the actual system, the need for using the chiller is due to the need for cooling the chilled water flow, which is being heated by sources that the simulation is not modeling in detail. For this reason, in order accurately train the agents to use the chiller for its main task, it would be necessary to add the cooling loads of the chiller to the simulation.

4.4. Environment state

The state is the information about the environment that the agent has access to. In this study, only the latest observation that the environment yields is used by the agent to make a decision. For this reason, from this section onwards, the term “state” or “environment state” will be used interchangeably with “observation”.

The state of the environment, i.e. the water heating system, is defined by 10 variables:

1. Time of day (from 0 to 24)
2. Time of year (from 0 to 8760)
3. Outside dry bulb temperature
4. Solar radiation
5. Preheating Tank 1 Temperature (solar stage)
6. Preheating Tank 2 Temperature (heat recovery stage)
7. Mean Temperature of the four Heating Tanks (3rd heating stage)
8. Mains water temperature
9. Inlet temperature in the evaporator side of the chiller (chilled water side)
10. Current warm water demand in the dressing rooms

All variables are normalized between 0 and 1. The hour of the year is also converted to a sine wave in order to avoid the discontinuity from 1 to 0 that would be produced when beginning a new year. This sine wave is equal to one at the beginning and at the end of the year, and equal to zero in the middle.

4.5. TRNSYS-Python connection

As discussed above, in order to make the training process possible, the TRNSYS software must be connected to the Python programming language, which is used to do all Machine Learning-related computations. In order to achieve this, Python has to receive certain results from TRNSYS as the simulation progresses, and it also has to be able to impose the decisions made by the agent, so that these decisions change the conditions of the simulation. Figure 17 shows a basic diagram of the operation of the platform. This figure can be considered as a more “detailed” version of the structure shown in Figure 3. It also considers the fact that only the latest observation is considered by the DNN in order to make decisions. The figure also shows that the Python Script should impose failures on the system; this is done only in Section 6.5 of this Thesis.

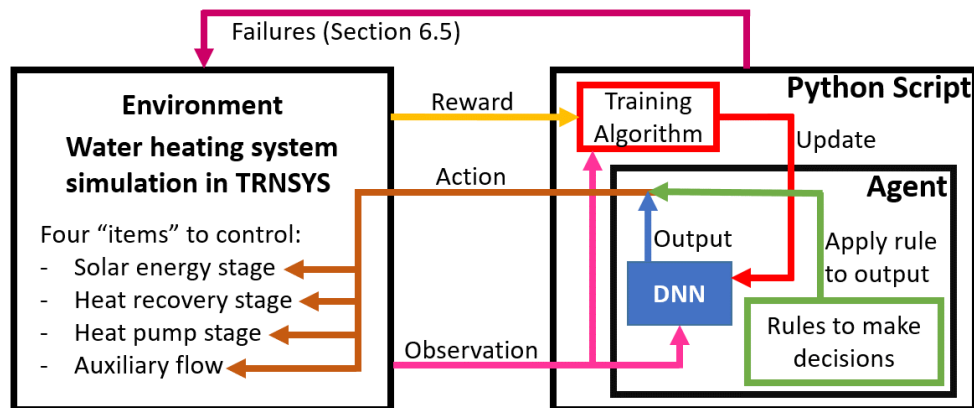


Figure 17: Structure of the DRL algorithm

TRNSYS has two “types” (i.e. elements which can be added to simulations) that make the interaction with Python possible. One of them is called Type 163 [53], and it works the following way: a data file (.dat), which contains all variables that will be passed to the Python script, is created by TRNSYS on every iteration of the simulation in the working directory. Then, Python imports the file with the outputs from TRNSYS and processes them following the code of the script. The results that Python yields are written in another data file, which is created in the working directory as well. This file is then imported by TRNSYS to continue the simulation (this is repeated on every *TRNSYS-iteration*; see “TRNSYS iterations” below). This method of connecting TRNSYS with Python has the advantage that it allows the use of Python libraries like Numpy [54], Scipy [55] and Tensorflow [56]. However, the process of writing the data files in the hard disk is very slow.

The other type that enables the TRNSYS-Python link, called Type 169 [53], is considerably faster. As a reference, a very simple Python code was tested with both types (163 and 169); Type 163 needed approximately 2.5 minutes to simulate a single day, whereas Type 169 made the same in less than 5 seconds. Type 169 achieves this by directly passing the variables to Python, instead of the input/output files described above. However, Type 169 only allows the use of basic Python libraries, i.e. the libraries that are included with the Python installation. Therefore, Numpy, Scipy and Deep Learning-specialized libraries cannot be used when Type 169 is being used to establish the connection. Considering the great difference in simulation speed when both options are compared, it was decided that it was unviable to use Type 163. Therefore, Type 169 was selected

and, for this reason, all the DNN and DRL algorithms, as well as the gradient calculations, had to be developed and implemented from scratch by using basic Python features like Lists [57] and the random [58] and bisect [59] libraries. In Section 3.1 and subsections, all details regarding DDNs and the backpropagation algorithm are detailed, and in Section 3.2 and subsections, the algorithms to carry out the training process through DRL are discussed.

Another issue is the fact that, under the connection mode just described, the whole Python script, from beginning to end, is run once every TRNSYS iteration. Therefore, Python cannot store the variables from previous time steps while the TRNSYS simulation progresses. The proposed solution is to store the parameters of the DNNs and other auxiliary data necessary for training in text (.txt) files and to import them when they are needed.

With all these considerations, a more detailed diagram of the structure of the training platform is shown in Figure 18 and discussed below.

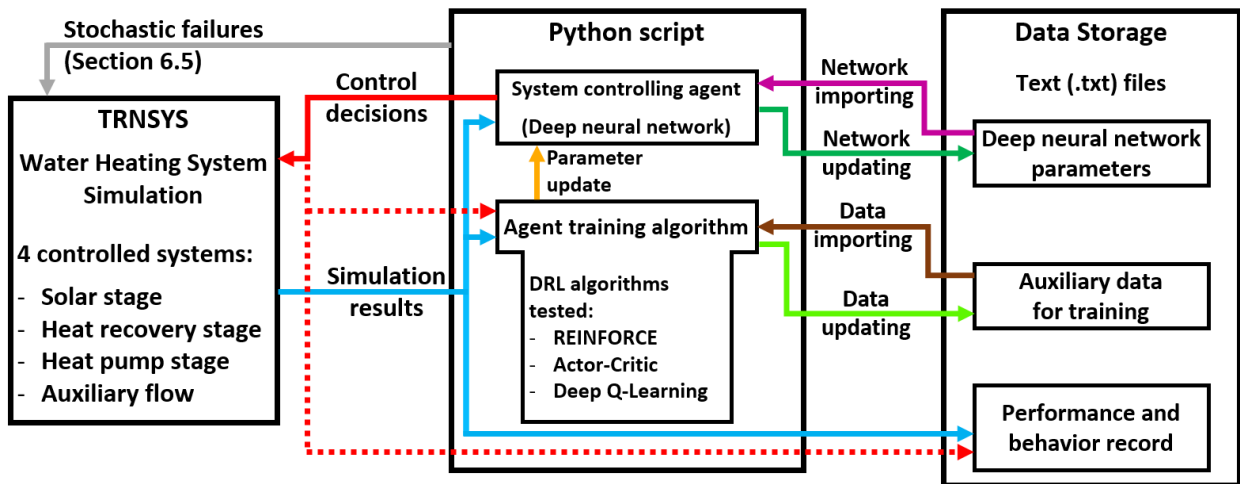


Figure 18: Structure of the training platform

Figure 18 shows the three main components of the platform: the simulation in the TRNSYS software, the Python script used to develop the controlling agent and the data stored in the hard disk. When a control decision has to be made, the network parameters are imported from a text file (purple arrow). The network evaluates the environment state, which is computed by the Python Script, based on the results delivered by the TRNSYS simulation (blue arrow at the top). With this information, a decision is made by the network and imposed on the TRNSYS simulation (red arrow). The chosen action (one of the 16 options shown in Table 1) is given as four values to TRNSYS; each of these values is either zero or one, and it tells the simulation if the corresponding system has to be activated (1) or deactivated (0).

In addition to this, the network has to be updated in order to improve the decisions made by the agent (orange arrow). The method to update the network will depend on the DRL algorithm that is being used (REINFORCE, Actor-Critic or Deep Q-Learning). All training algorithms depend on “auxiliary data” to update the DNN (in the case of the actor-critic algorithm, it depends on the auxiliary network that computes state values). This auxiliary data (shown in the figure as “Auxiliary data for training”) is imported from text files, as the brown arrow shows. When the network is updated, the new parameters are stored in the hard disk as the dark green arrow shows; this allows to import the new parameters of the DNN on future time steps. In addition to this, the training algorithm must update the auxiliary data with the new interaction data that is experienced by the agent (light green arrow). To achieve this, it registers the actions taken by the agent (red

dotted arrow at the top) as well as the simulation results (blue arrow in the middle). The simulation results delivered by TRNSYS include the quantities that must be computed as integrals over time, like the heat delivered to the water, the electric consumption, the “comfort” indicator (see Equation 105) and the “degradation” indicator (see Equation 107). These integrals are computed by a specialized TRNSYS component, called Type 24.

Besides, the script is used to record the agent’s performance and behavior in order to be analyzed once the simulation is over (this is shown by the blue arrow and the red dotted arrow at the bottom). This data is recorded in text files (.txt) as well.

Reinforcement Learning time steps and simulation time steps

Now it becomes necessary to make an important distinction. From the point of view of the RL process, a time step consists of a cycle of: a state, an action and a reward. From the point of view of TRNSYS, a time step is a short amount of time that the software uses to discretize the transient process that is being simulated. Therefore, a time step for the simulation software is much shorter than a time step for the RL algorithm, which in this case lasts two hours.

TRNSYS calls the Python script on every *simulation* time step, even though the script is only supposed to execute actions on the simulation every two hours. The solution to this is that, in the timespan between two actions, the Python script keeps imposing the same “order” to the simulation on every simulation time step until a new action is required. This also avoids having to import the DNN parameters on every simulation time step, which would make the process remarkably slower. This process is shown in Figure 19.

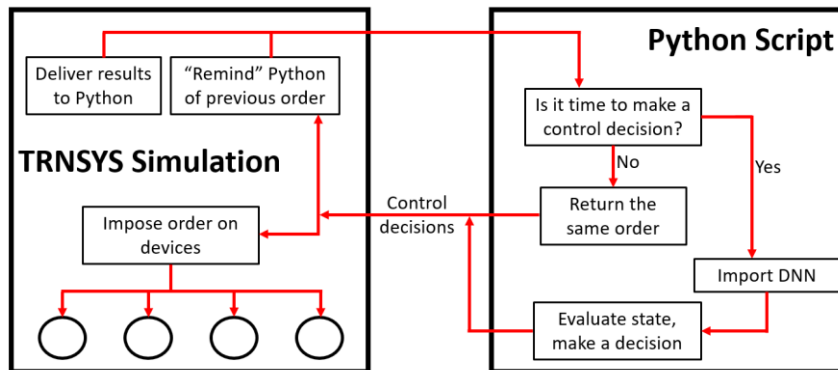


Figure 19: Process that must be repeated on every simulation time step

As already discussed, the Python script is run from the beginning on every simulation time step, thus it cannot “remember” what the action of the previous time step was. For this reason, the simulation has to “remind” the Python script of what order was imposed in the last simulation time step. This is also shown in Figure 19.

Although at 10.00 PM the agent does not have to execute an action on the system, at that time the Python script does not simply return the same action from the previous time step because there are other important things to do, e.g. training the DNN, storing the last reward of the day, etc., so a very similar process must be carried out at that time but no action is imposed on the system.

In order to achieve that TRNSYS returns the last received “order” to Python, a feature of TRNSYS that is called “equation” is used. An equation is an element that can be added to simulations in order to make calculations; it also provides basic programming features. Like any other “type” of TRNSYS, equations receive inputs from other elements of the simulation and yield outputs.

In Figure 20, the TRNSYS equation that is used to control the chiller is shown. The inputs of the equation are shown in the upper left part of the window; one of these inputs is the decision that the Python script made regarding the use of the chiller, and it is called “NN_order” in the equation window. Clearly, this input is received from the Type 169 that calls the Python script. The input “NN_order” can have two values: one or zero. If its value is one, the chiller is supposed to be used.

The outputs of the equation can be seen in the upper right part of the window; one of them is called “NN_order_chill_back”. As can be seen below, this output is defined as the same value of the “NN_order” input, i.e. the same value that was received from the Type 169 as the “order” of the Python script. This output is delivered to Type 169; in this way TRNSYS can “remind” Python of the last order that it delivered.

The other input of the equation, called “op”, is equal to one during the day and it becomes zero at night. The “chiller_control” output is defined as $op * NN_order$, and it is used to control the chiller. Therefore, if the value of the “NN_order” input is one and it is daytime, the chiller is used.

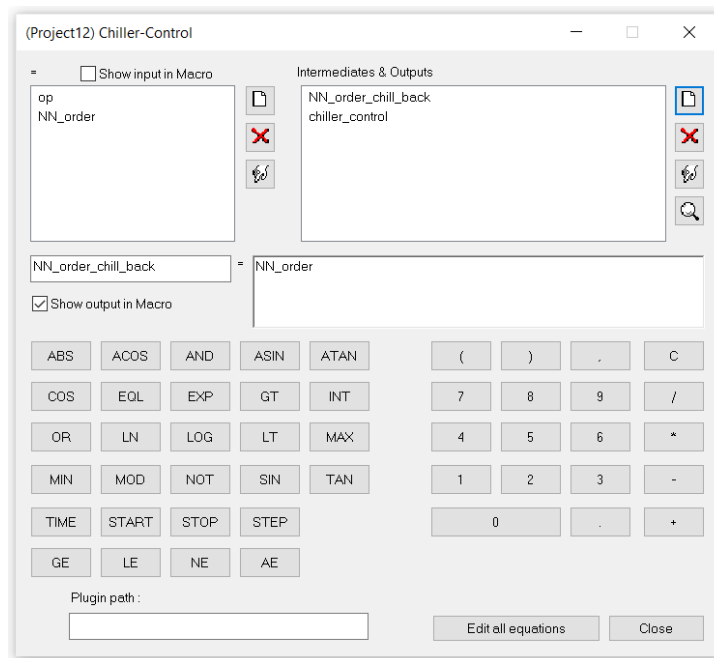


Figure 20: TRNSYS equation used to control the chiller

TRNSYS iterations

Up to this point, there is a very efficient way for the Python script to keep imposing the same decision in the time span between an action and the next one.

Nevertheless, there is still a problem that occurs at the moment of executing a new action on the system, i.e. at the times of day when the agent has to evaluate the state of the system and make a decision. The problem is that TRNSYS not only calls the Python script on every simulation time step, but also on every iteration. Iterations are calculations that the software makes several times per time step, until the conditions for convergence are met.

The problem that iterations generate is as follows: the condition to execute a new action on the system is that the current time of day must be one of the previously specified times: 8.00 AM, 10.00 AM, 12.00 PM, 2.00 PM, 4.00 PM, 6.00 PM, 8.00 PM, 10.00 PM (at 10.00 PM no action is executed, but there are other things that must be carried out by the Script). This condition will be met in all iterations of the corresponding time step. Therefore, the agent will impose a new order

on every TRNSYS iteration of the time step (provided that the time of day is one of the aforementioned ones). Since the actions are completely random at the beginning of the simulation, it is almost certain that the agent will impose different actions during the same time step, which causes problems with the convergence of the time step.

To solve this, the Python script has to impose a single action when the time to execute an action has come, and not an action for each iteration. To achieve this, a text file is created to store at what time the last decision was made. When it is time to make a new decision, the Python script imports this file; if the stored time is not equal to the current time, the Python script makes a new decision and updates the text file with the current time. In the next iteration of the same time step, the Python script imports the same file, but the stored time is now equal to the current time, so the Python script repeats the same decision that was decided on the previous iteration. With this method, the agent executes a new action only once at the corresponding time step. The process is illustrated in Figure 21, which is an extension of the process shown in Figure 19.

The process explained above implies that the decision is made on the first iteration of the time step; this is not optimal because the values of the environment state are still being calculated; however, they are expected not to be far from the final values.

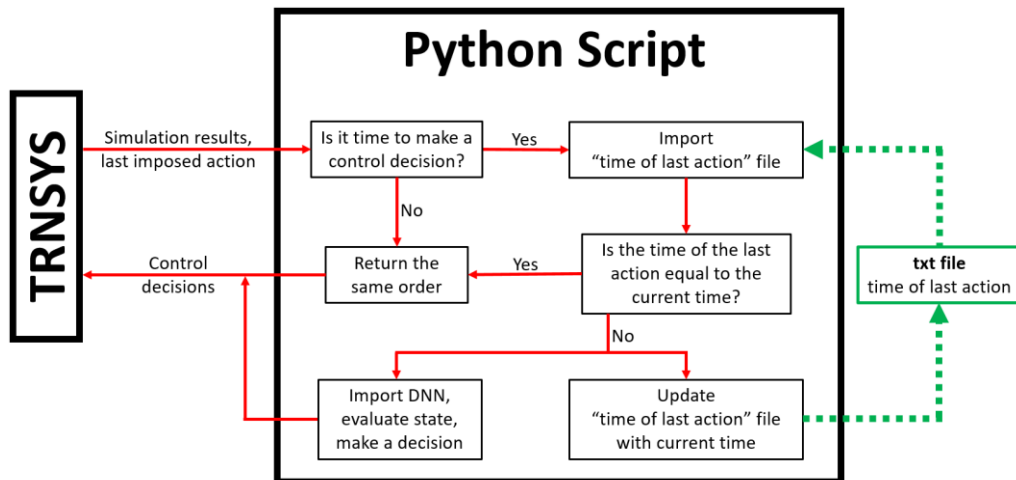


Figure 21: Process for every TRNSYS iteration.

At 10.00 PM no action is imposed on the system, but an analogous process must be carried out to store the last reward of the day, train the network, etc. At that time the “time of last action” file must be updated as well, in order to not repeat the process several times.

Pseudo-code versions of the Python codes that were used to initialize the agents and to train them are included in Section 4.7. But first, it is necessary to discuss the introduction of stochastic failures into the water heating system simulation.

4.6. Introducing stochastic failures

The goal of introducing stochastic failures is that the devices of the system go through temporary and unpredictable states of failure. The agent is expected to manage the system in such a way that the goal of delivering hot water to the dressing rooms is achieved. The failure states are temporary because all devices that can fail can also be repaired.

4.6.1. Failure rates of individual devices

The items in the system are considered to have constant failure rates. The failure rates are based on data of OREDA (Offshore Reliability Data Handbook [60]) (see Section 3.3 for details on reliability theory).

The repair rate is assumed to be constant as well; this means that, given that the device has failed, the chance of being put into operation in the “near future” is constant, regardless of the time span during which the device has been out of operation. These assumptions allow the use of discrete-time time-homogeneous Markov chains for the simulation of stochastic failures of the devices of the water-heating system. Repairs guarantee that all states are reachable regardless of the state where the Markov chain starts (see Section 3.4 for details on Markov chains).

Since in the OREDA book there is no specific data for heat pumps and chillers, these devices are split into their component parts. Since all components of the devices are necessary for the device to work, the total failure rate of the device is the sum of the failure rates of its components (see Section 3.3.3).

Each heat pump is considered to be composed of (see Section 3.6 for theoretical information on heat pumps):

- Three heat exchangers: two internal heat exchangers of the machine (evaporator and condenser) and one heat exchanger to transfer the heat to the water flow that goes to the dressing rooms (heat exchangers 5 to 8 in Figure 14).
- A compressor.
- Two water pumps: one pump moves the water from its respective heat pump to the heat exchanger (heat exchangers 5 to 8 in Figure 14); the other pump recirculates the water from its respective heating tank (pumps 9 to 12 in Figure 14) to extract heat from the heat pumps.
- An expansion valve.
- Three electric motors: one motor for the compressor and one motor for each water pump.

The chiller is considered to have exactly the same components as the heat pumps, but a simplification has to be made: in the case of the chiller, there are two heat exchangers to transfer heat to the water flow that goes to the dressing rooms (heat exchangers 3 and 4 in Figure 14). Therefore, if only one of these heat exchangers works, it is possible for the chiller to partially fulfill its task of heating that flow. However, the heat exchangers are not in parallel either, because if only one of them works, the system will not be equally efficient. Besides, if the heat exchangers were assumed to be in parallel, the system would not have a constant failure rate anymore, and the markovian model could not be used. For this reason, the chiller is assumed to have three heat exchangers: evaporator, condenser and the “external” heat exchanger that represents the two aforementioned heat exchangers. Assuming four heat exchangers in series is not correct either, because this model would have a lower reliability than the model with three heat exchangers, and the real system has at least the same reliability as the latter. Hence, the model with three heat exchangers in series will be used. With this, the chiller has the same components as one of the heat pumps. Actually, the assumption that is made regarding this topic is not very important, because the failure rate of the heat exchangers is relatively low in comparison to that of other components, so their contribution to the global failure rate is not high (this will be detailed below). Like the heat pumps, the chiller is assumed to have two associated pumps as well, but in the case of the chiller, one of these pumps is the one that moves the heated flow (pump 7 in Figure 14) and the other is the one that moves the cooled flow (pump 8 in Figure 14).

The solar stage is considered to be composed of two systems that can fail independently: each system is composed of a water pump (pumps 3 and 4 in Figure 14), the motor of the pump and a heat exchanger (heat exchangers 1 and 2 in Figure 14). Each of these “pump-motor-heat exchanger groups” is going to be called “pump-heat exchanger pair” in the coming sections (the motor of the pumps is considered to be part of the pump in the name, but from the point of view of the OREDA book, they are different components so their failure rates have to be summed). The probability of failure of the solar field is neglected because many rows of solar collectors operate in parallel (see Figure 13); therefore, if a collector fails, the solar field can still operate partially with the other rows. (From a reliability point of view, the collectors are not completely parallel because they do not operate equally well without a row of collectors, but this is being neglected). (Actually, failure of the solar field can be considered as a common-cause failure, which, as shall be discussed later, is considered as well. Nevertheless, the failure rate of the solar field is not calculated in detail).

The heat pumps can fail independently from each other, as well as the two pump-heat exchanger pairs of the solar stage (for the chiller there are no redundant items). For the heat pumps, it is considered that at least three of them must be operative in order for the third heating stage to operate (because with less heat pumps being operative, the remaining ones will require too much effort to keep their respective tanks at 60°C). In the case of the solar stage, at least one of the pump-heat exchanger pairs must be operative. The state (functional or not functional) of a heating stage does not influence the failure probability of the other two.

There is also the option of common-cause failures. Common-cause failures produce the immediate failure of a whole system; i.e., despite several redundant devices may be operative, a common-cause failure takes all of them out of operation. Each heating stage is independently subject to common-cause failures. In the case of the third heating stage (heat pump system), the rate of common-cause failures is 40% of the failure rate of a single heat pump; for the solar stage, the rate of common-cause failures is 40% of the failure rate of one pump-heat exchanger pair; and for the chiller, the rate of common-cause failures is 40% of the failure rate of the chiller.

The pumps that recirculate the water that is contained in the preheating tanks (i.e. Pumps 1, 2, 5 and 6 in Figure 14) are not subject to failures; here is why: these pumps play a fundamental role in supplying water to the dressing rooms. If these pumps fail, several inconsistencies occur in the simulation; besides, the goal of introducing failures is to test whether the controlling agent is capable of handling the system despite such failures. If the pumps that recirculate water fail, there is nothing that the agent can do to compensate that, because the water supply in the dressing rooms would be interrupted (see Figure 14).

The failure rates of the components considered to calculate the failure rates of the heat pumps, the chiller and the pump-heat exchanger pairs of the solar stage are summarized in Tables 2 and 3. As already discussed, these parameters were taken from OREDA [60].

Table 2: Failure parameters of the heat pumps and the chiller

Failure rate of heat exchangers	$16.50 \cdot 10^{-6} \text{ hr}^{-1}$
Number of heat exchangers per device	3
Failure rate of compressors	$268.58 \cdot 10^{-6} \text{ hr}^{-1}$
Number of compressors per device	1
Failure rate of pumps	$65.40 \cdot 10^{-6} \text{ hr}^{-1}$
Number of pumps per device	2
Failure rate of expansion valves	$25.97 \cdot 10^{-6} \text{ hr}^{-1}$
Number of expansion valves per device	1
Failure rate of motors	$32.75 \cdot 10^{-6} \text{ hr}^{-1}$
Number of motors per device	3
Failure rate of heat pumps and chiller (failure rates of all components summed)	$573.10 \cdot 10^{-6} \text{ hr}^{-1}$
Mean Time to Failure of Heat Pumps and Chiller (computed as shown in Equation 77)	1744.9 hr

Table 3: Failure parameters of the pump-heat exchanger pairs in the solar stage

Failure rate of heat exchangers	$16.50 \cdot 10^{-6} \text{ hr}^{-1}$
Number of heat exchangers per pair	1
Failure rate of pumps	$65.40 \cdot 10^{-6} \text{ hr}^{-1}$
Number of pumps per pair	1
Failure rate of motors	$32.75 \cdot 10^{-6} \text{ hr}^{-1}$
Number of motors per pair	1
Failure rate of one pump-heat exchanger pair (failure rates of all components summed)	$114.65 \cdot 10^{-6} \text{ hr}^{-1}$
Mean Time to Failure of one pump-heat exchanger pair (computed as shown in Equation 77)	8722.2 hr

4.6.2. Construction of the Markov chains

The failure rates that have just been calculated for the heat pumps, the chiller and the pump-heat exchanger pairs correspond to the constant hazard rates of these systems (see Section 3.3.1). Therefore, these values are useful in a mathematical model which is continuous in time. To build a discrete-time Markov chain with this parameters, Equation 74 can be used to estimate the failure probability between two consecutive instants. In other words, the failure rates just determined can be multiplied by the time span between two consecutive instants in the Markov chain, and this would yield the probability of failure in the time span between those instants. As expressed in Section 3.3.1, this is an approximation, but it is acceptable if the resulting probability is small.

As already discussed, the functional states of the different heating stages are independent; i.e. there must be three independent Markov chains being executed at the same time, each one to define the functional state of one heating stage.

The functional states of the devices will be determined by the same Python script that trains the DNNs. The time steps of the Markov chains will be executed at the same times of day at which the actions of the agents are executed (including at 10.00 PM; see Section 4.2 for details). Between consecutive “Markov time steps”, the Python script will keep imposing the same functional states, just as done with the actions of the agents (see Section 4.5 for details). Therefore, the time span between consecutive instants in the Markov chains is two hours. Here, an important simplification

is being made: the Markov chain is time-homogeneous; this means that the transition probabilities do not vary in time. Since no Markov time steps are executed at night, the transition probabilities between 10.00 PM and 8.00 AM of the next day must be the same as in the other time steps of two hours. The justification for this can be that the system is turned off at night, so the failure rates of the items of the system can be assumed to be lower. A more important simplification is the fact that the failure probability of an item does not change depending on whether the item is being used. If this were considered, it would be impossible to estimate the percentage of time that the system spends in each state, at least until after the training process.

To determine the repair probabilities between two consecutive instants, the following assumptions are made: a single device (heat pump, chiller or pump-heat exchanger pair) has a mean time to repair of two weeks. A common-cause failure has a mean time to repair of three weeks. Given that eight time steps are executed each day, the expected number of time steps until repair of a single device must be equal to $8 \cdot 14$, and the expected number of time steps until repair of a common-cause failure must be equal to $8 \cdot 21$. The random variable “time steps until repair” follows a geometric distribution (see Section 3.4.1); therefore, the probability of repair of a single device between two consecutive instants must be equal to $1/(8 \cdot 14) = 0.00893$, and the probability of repair of a common-cause failure must be equal to $1/(8 \cdot 21) = 0.00595$.

The failure rates and repair probabilities just defined are summarized in Table 4. In the same table, the failure rates are translated into failure probabilities between consecutive instants of the Markov chains, considering a time span of 2 hours between consecutive instants.

Table 4: Failure rates and failure and repair probabilities

Item	Value
Failure rate of a heat pump	$573.10 \cdot 10^{-6} \text{ hr}^{-1}$
Common-cause failure rate of the heat pump system	$229.24 \cdot 10^{-6} \text{ hr}^{-1}$
Failure rate of the chiller	$573.10 \cdot 10^{-6} \text{ hr}^{-1}$
Common-cause failure rate of the chiller	$229.24 \cdot 10^{-6} \text{ hr}^{-1}$
Failure rate of a pump-heat exchanger pair	$114.65 \cdot 10^{-6} \text{ hr}^{-1}$
Common-cause failure rate of the solar energy stage	$45.86 \cdot 10^{-6} \text{ hr}^{-1}$
Time span between two consecutive instants in the Markov chains	2 hours
Failure probability of a heat pump between two consecutive instants	$1.146 \cdot 10^{-3}$
Common-cause failure probability of the heat pump system between two consecutive instants	$4.585 \cdot 10^{-4}$
Failure probability of the chiller between two consecutive instants	$1.146 \cdot 10^{-3}$
Common-cause failure probability of the chiller between two consecutive instants	$4.585 \cdot 10^{-4}$
Failure probability of a pump-heat exchanger pair between two consecutive instants	$2.293 \cdot 10^{-4}$
Common-cause failure probability of the solar energy stage between two consecutive instants	$9.172 \cdot 10^{-5}$
Repair probability of individual items between two consecutive instants	$8.93 \cdot 10^{-3}$
Repair probability of common-cause failures between two consecutive instants	$5.95 \cdot 10^{-3}$

As already discussed, it is considered that at least three of the four heat pumps of the third heating stage must be operative in order for the third heating stage to operate. In the case of the solar stage, one of the two pump-heat exchanger pairs has to be operative.

As can be seen in Figures 22 to 24, the Markov chains of the solar stage and the heat pump stage are more complex than the Markov chain of the chiller. The three following assumptions concern only the heat pump system and the solar energy system:

1. When two items of the same heating stage are under failure state at the same time (state 3 in Figures 22 and 24), the repair rate will be equal to the repair rate of one of them. This assumption means that only one of the items is being repaired and not both at the same time.
2. When one of the heating stages is completely out of operation, the items which have not failed yet cannot fail before the corresponding heating stage is put into operation again. This is why in state 3 (Figures 22 and 24) it is impossible for a common-cause failure to occur; and in states 4 and 5 (Figures 22 and 24) it is impossible for a heat pump or a pump-heat exchanger pair to fail. Without this assumption the Markov chains would be far more complex. This also applies for the chiller in the sense that, when a common-cause failure has occurred, it is not possible for the chiller to fail as well, and vice versa.
3. If one of the devices has failed (state 2 in Figures 22 and 24) and then a common-cause failure occurs (state 5 in Figures 22 and 24), then the device which had failed before can be repaired while the system continues under common-cause failure (this would mean transiting from state 5 to state 4). However, there is also the possibility that the common-cause failure is repaired before the device (this would mean returning from state 5 to state 2).

Therefore, the Markov chains that will be used to model stochastic failures are as shown in Figures 22, 23 and 24. The arrows show the probabilities of changing states between consecutive instants; the values of the probabilities are better explained in Table 4. The probabilities of staying in the same state have not been written, but they can be easily calculated as one minus the total probability of leaving the corresponding state.

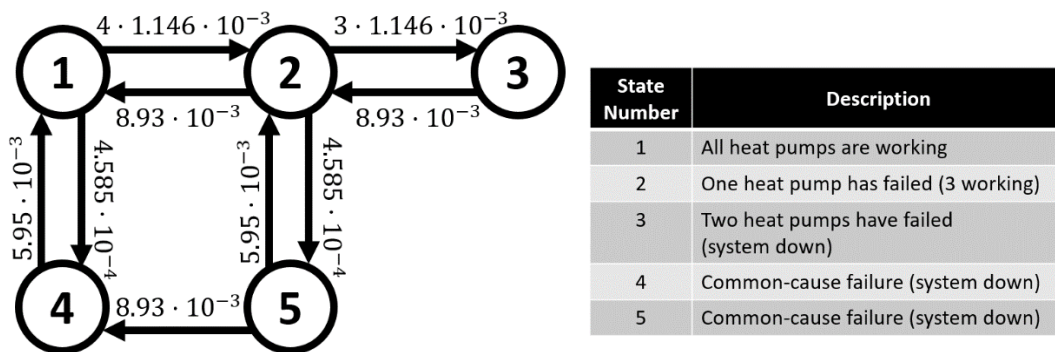


Figure 22: Markov chain of the third heating stage (heat pump system)

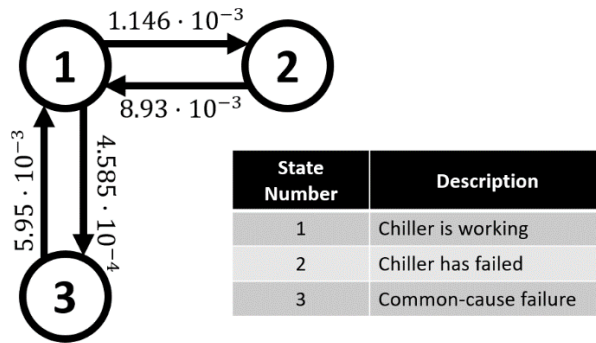


Figure 23: Markov chain of the chiller

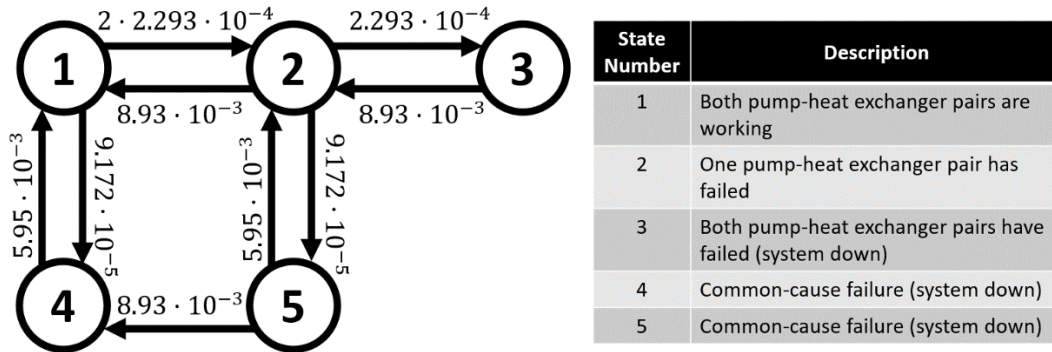


Figure 24: Markov chain of the solar stage

Tables 5, 6 and 7 show the steady-state probabilities for the Markov chains shown in Figures 22, 23 and 24. As already discussed in Section 3.4.2, those probabilities can be interpreted as the percentage of time that each markovian system will spend, on average, in each possible state.

Table 5: Steady-state regime probabilities of the heat pump system (Figure 22)

State	Probability [%]
1	55.4
2	27.6
3	10.6
4	5.54
5	0.850

Table 6: Steady-state regime probabilities of the chiller (Figure 23)

State	Probability [%]
1	83.0
2	10.6
3	6.39

Table 7: Steady-state regime probabilities of the solar energy system (Figure 24)

State	Probability [%]
1	93.6
2	4.78
3	0.123
4	1.49
5	0.0294

4.7. Pseudo-code versions of the Python Scripts

In this section, the codes that are necessary to run a training simulation are detailed. The codes include the use of Markov chains to impose failures on the system, and they use the Deep Q-Learning algorithm to train the network. With this characteristics, they could be interpreted as the codes used for Sections 6.5.1 or 6.5.3 of this thesis, depending on how the Markov chains and the set of possible actions are defined.

4.7.1. Initializer

The “initializer” is a code that must be run before beginning the TRNSYS simulation. The TRNSYS software does not interact with this code. It creates all text files that must be in the working directory in order for the simulation to work. An obvious example is the file that contains the parameters of the Deep Neural Network. The code is as follows:

```
## the term “text file” is used for files that are stored in the working directory and are read, created and
## modified by python
initialize neural network parameters ##weights and biases, according to the method of Section 3.1.5
initialize momentum, as a vector of zeroes
create “online network” text file
create “target network” text file
create “momentum” text file
store network parameters in “online network” text file
store network parameters in “target network” text file
store momentum in “momentum” text file
create “Replay Memory” text files (empty files)
create “Priority Numbers” text file (empty file)
create “time of last action” text file
write “0” on “time of last action” text file
create “previous Markov states” text file
write “1 1 1” on “previous Markov states” text file
create “days since last target network update” text file
write “0” on “days since last target network update” text file
```

4.7.2. Code to train the network

The code detailed here is the one that is “called” by TRNYS on every TRNSYS iteration. As discussed above, on most iterations it only delivers the same action and Markov states that were defined in the previous “action instance”. Some parts of the code have been split from the code shown in Section 4.7.2.1, and are detailed in Sections 4.7.2.2, 4.7.2.3 and 4.7.2.4.

4.7.2.1. Main code

```
## “import” is used for variables imported from TRNSYS
## “define” is used for variables specified by the user
## the term “text file” is used for files that are stored in the working directory and are read, created and
## modified by python
import time ## total simulation time in hours
import environment state variables
import integrated values ##energy, degradation time, warm-water-supply-time, comfort indicator, etc
import last action
import functional states of the heating stages
time_of_day = time%24
time_of_year = time%8760
if time_of_day in [8, 10, 12, 14, 16, 18, 20, 22]:
    open “time of last action” text file
    if time_of_last_action == time_of_day:
        enter = False
    else:
        enter = True
else:
    enter = False
define alpha_2 value ## parameter of the reward function, associated to comfort
if enter == True:
    define reward parameters (other than alpha_2)
    define training hyperparameters
    define transition probabilities of the Markov chains
    open “previous Markov states” text file
    compute new functional states of heating stages by executing a time step in each Markov chain
    open “online network” text file
    compute environment state
    if time_of_day == 8: ## first state of the day
        execute “interaction at 8.00 AM” ## see Section 4.7.2.2
    else:
        open “previous integrated values” text file
        compute variations of integrated values by subtracting previous ones from current ones
        compute reward with the variations of integrated values and the previous action
        if time_of_day == 22: ## terminal state
            execute “interaction at 10.00 PM” ## see Section 4.7.2.4
        else:
            execute “interaction from 10.00 AM to 8.00 PM” ## see Section 4.7.2.3
    create new “previous integrated values” text file
    store integrated values just received from TRNSYS in “previous integrated values” text file
    update “previous Markov states” text file
    update “time of last action” text file with current “time_of_day”
    impose selected action At on TRNSYS (as defined in Subsections 4.7.2.2, 4.7.2.3 and 4.7.2.4)
    impose new functional states of the three heating stages on TRNSYS
    impose alpha_2 (parameter of the reward function) on TRNSYS
else: ## no interaction between Python and TRNSYS is needed
    impose the same action just received from TRNSYS
    impose the same functional states of the heating stages just received from TRNSYS
    impose alpha_2 (parameter of the reward function) on TRNSYS
```

4.7.2.2. Interaction at 8.00 AM

```
compute epsilon ## (from eps-greedy) (calculation based on "time" imported from TRNSYS)
x = random value between zero and one
if x < epsilon:
    At = action selected randomly ## choose action to execute
else:
    evaluate environment state with online network
    At = action considered to be the best by the agent ## choose action to execute
create new "daily action record" text file
create new "daily environment state record" text file
store selected action At in "daily action record" text file
store environment state in "daily environment state record" text file
```

4.7.2.3. Interaction from 10.00 AM to 8.00 PM

```
compute epsilon ## (from eps-greedy) (calculation based on "time" imported from TRNSYS)
x = random value between zero and one
if x < epsilon:
    At = action selected randomly ## choose action to execute
else:
    evaluate environment state with online network
    At = action considered to be the best by the agent ## choose action to execute
update "daily action record" text file by adding the action At just selected
update "daily environment state record" text file by adding the current environment state
if time_of_day == 10:
    create new "daily reward record" text file
    store reward in "daily reward record" text file
else:
    update "daily reward record" text file by adding the reward just received
```

4.7.2.4. Interaction at 10.00 PM

```
## action to execute is always the same at 10 PM (see next line)
At = all three heating stages ON; auxiliary flow OFF ## choose action to execute
open "daily action record" text file
open "daily environment state record" text file
open "daily reward record" text file
update daily reward- and environment state-record with the last reward and state of the day
open "Replay Memory" text files
open "Priority Numbers" text file
update Replay Memory with experiences of the current day
update Priority Numbers by evaluating the new experiences with the online network
if length of "Replay Memory" and "Priority Numbers" is more than the limit:
    erase oldest experiences until reaching the maximum allowed length
if time-span before training is over: ## i.e. if the agent is being trained
    open "target network" text file
    open "momentum" text file
    select random set of experiences to form the batch with the "prioritized" method
    define target vector for each experience (Normal DQN or Double DQN method)
    compute gradient for each experience (online network)
    compute new priority number for each experience of the batch
    update Priority Numbers of experiences that were added to the batch
    average gradients and update momentum
    update online network with the momentum
    update "online network" text file
    update "momentum" text file
    open "days since last target network update" text file
    if days_since_last_target_network_update == target_network_update_period - 1:
        update target network by copying online network
        update "target network" text file
        days_since_last_target_network_update = 0
    else:
        days_since_last_target_network_update = days_since_last_target_network_update + 1
    update "days since last target network update" text file with the new value
update "Replay Memory" text files
update "Priority Numbers" text file
```

Chapter 5: Methodology

5.1. Stages of the study

The experiments can be divided into five main sections. In the four first parts, all experiments (simulations) are conducted with systems that are not subject to failures; i.e., they are completely operational 100% of the time.

In the first part, the three training algorithms (REINFORCE, Actor-Critic and Deep Reinforcement Learning; see Section 3.2 and subsections for details) are tested and compared.

In the second part, DRL-trained agents are compared to a baseline (not smart-controlled) simulation of the system. In the baseline, all heating stages of the system are permanently used and the auxiliary flow is not used. Rewards are used as metric to compare the baseline to the smart agents. This is consistent because, independently of how a reward is defined, it is by definition what the agent is meant to maximize (as discussed in Section 1.5, the higher the reward, the better). The goal of this part is to prove that the proposed method is effective to train smart agents that can perform better than the baseline strategy.

In the third part, the performance of the agents is analyzed as the training hyperparameters are changed. Different network architectures are also tested and compared. In this part, the main goal is to compare the rewards achieved by agents that were trained with different hyperparameters, and to determine the hyperparameters that maximize the performance of the agents.

In the fourth part, the behavior of the agents (i.e. the actions that they choose) is analyzed as the parameters of the reward function are changed. Modifying the parameters of the reward function is equivalent to changing the goal that the agent is meant to achieve. In this part, comparing the rewards of different agents does not make sense, because some agents will reach larger rewards than others only because of the reward definition that they were trained and tested with. What makes sense in this part is to analyze the actions that were chosen by the agents, which are expected to vary as the goal that the agent is supposed to seek is modified.

In the fifth part, the ability of the agents to operate the system under stochastic failures is analyzed. To do this, new agents are trained in an environment where the components can fail as well. Two types of agents are analyzed and compared: the first type of agent receives the same environment state that was defined in Section 4.4, with 10 variables. The second type of agent receives extra variables that define the functional state of the system; i.e. the agents of this second kind are being “told” which components of the system are operating and which not. In order for the agents of the first kind to be successful, they have to “infer” this information from the other variables of the system, like the temperatures of the water in the storage tanks.

5.2. Result Analysis

Depending on the goal of each specific test, different results need to be taken into account. One of the main metrics to compare different agents are the rewards received by them. To visualize the rewards, all rewards of each simulation day will be summed, and then they will be plotted with time in the horizontal axis. Hence, each point in the graph represents one simulation day and the total amount of rewards received on that day.

An example of the rewards that were received during a training process is shown in Figure 25. 12 years (4380 days) were simulated. It can be clearly seen that the rewards increase as the agent

learns to execute better actions on the system. There is a clear cyclical behavior in the rewards due to the fact that in summer there are many conditions that make it easier for the agent to get higher rewards: higher solar radiation availability, warmer temperature of the mains water flow and hotter air temperature, which increases the efficiency of the air-water heat pumps. The raw rewards, shown in blue, are very noisy, so for most comparisons a “moving average”, shown in orange, will be used to smooth the result. In the figure, a “time window” of 21 days is being used to compute the moving average corresponding to each day. The 21 days correspond to the ten previous days, the ten next days and the day itself for which the time window is being computed. This time window of 21 days will be used for the first four parts of the study (as defined in the previous section) with the exception of Figure 64 which uses a time window of 7 days. In the fifth part of the study, where failures are introduced, the time window will always be 7 days; i.e. the three previous days, the three next days and the day itself.

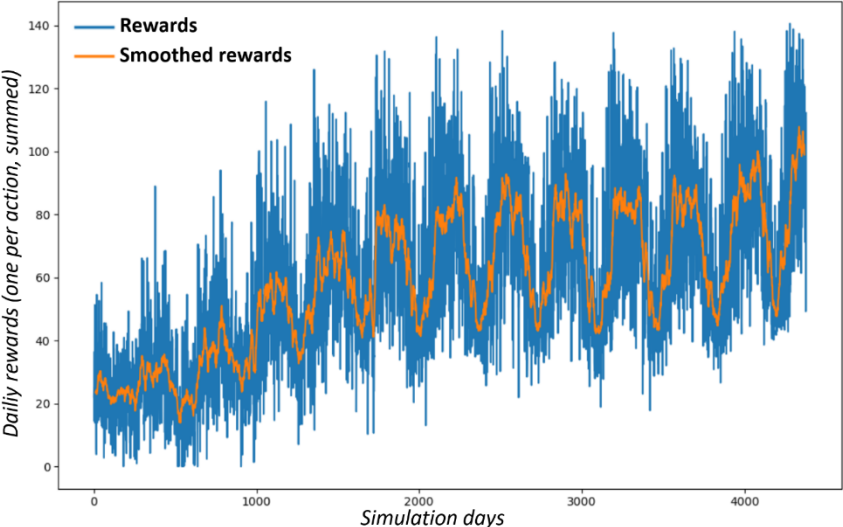


Figure 25: Rewards of each simulation day summed.

Another result that will be commonly shown is the actions that the agents took during one simulation year. This is particularly important when the parameters of the reward function are changed; in this case, it does not make sense to evaluate which agent got the highest rewards (since the rewards are defined in a different way for each agent), but it does make sense to evaluate how the actions of the agents change in order to achieve different goals defined by the reward parameters that are being modified. When the actions of the agents are evaluated, it is useful to take into account at what time of day and at what epoch of the year a particular action was taken. To do this, the number of times that a particular action (from the 16 possible actions that the agent can choose) was chosen at each time of day during each month of the year is counted. A visual way of visualizing this result is shown with an example in Figure 26. Each group of columns represents a moment of day at which an action has to be chosen. Within each group, each column represents a month (the columns are ordered from January to December). The different heights of the columns are due to the number days that each month has. Within each column, the colors represent the number of times during the corresponding month that a particular action was chosen at the corresponding time of day. At the right side of the figure, the actions of the agent are clarified by showing which systems are used. “H.P.” means “heat pumps” in the figures.

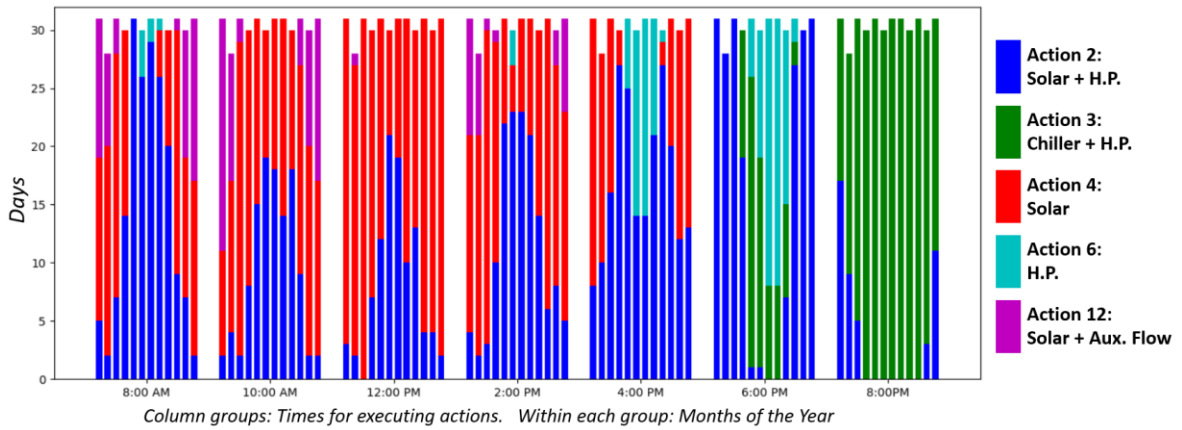


Figure 26: Visual illustration of the actions taken during one year

Another way of visualizing the same result is by grouping the columns that correspond to the same month. This is shown in Figure 27. Within each group, each column corresponds to a time of day at which an action has to be chosen (the columns are ordered from 8.00 AM to 20.00 PM). Figures 26 and 27 show exactly the same information but the information is presented in different ways in order to illustrate different features of the behavior of the agent.

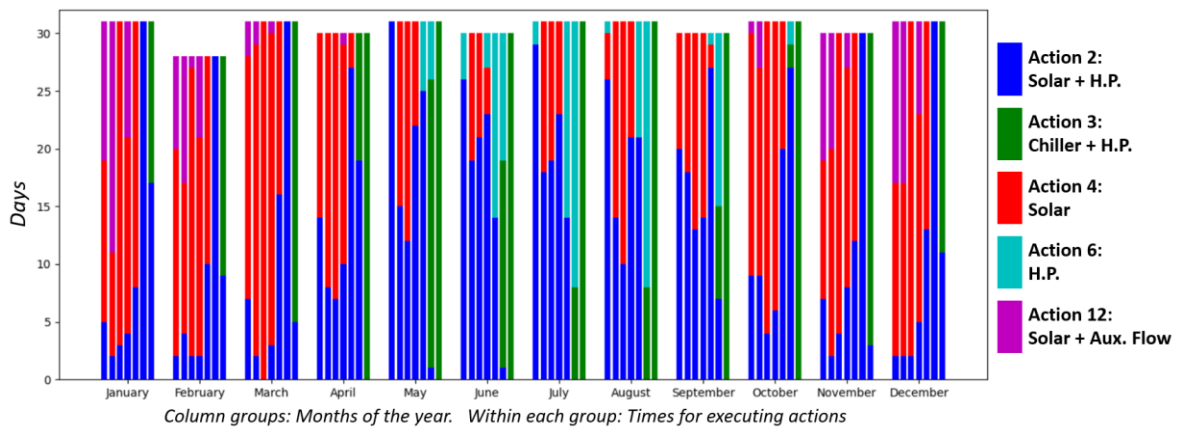


Figure 27: Alternative illustration of the actions taken during one year

A way of visualizing each individual action of an agent is shown in Figure 28. This method theoretically gives more information than the two previous figures, but the trends are harder to visualize. Figures 26, 27 and 28 are made from the same action record. This last method of analyzing the actions will be especially useful when failures of the components are introduced, because in that case, the change in the behavior of the agent can be visualized exactly at the moment when a failure occurs.

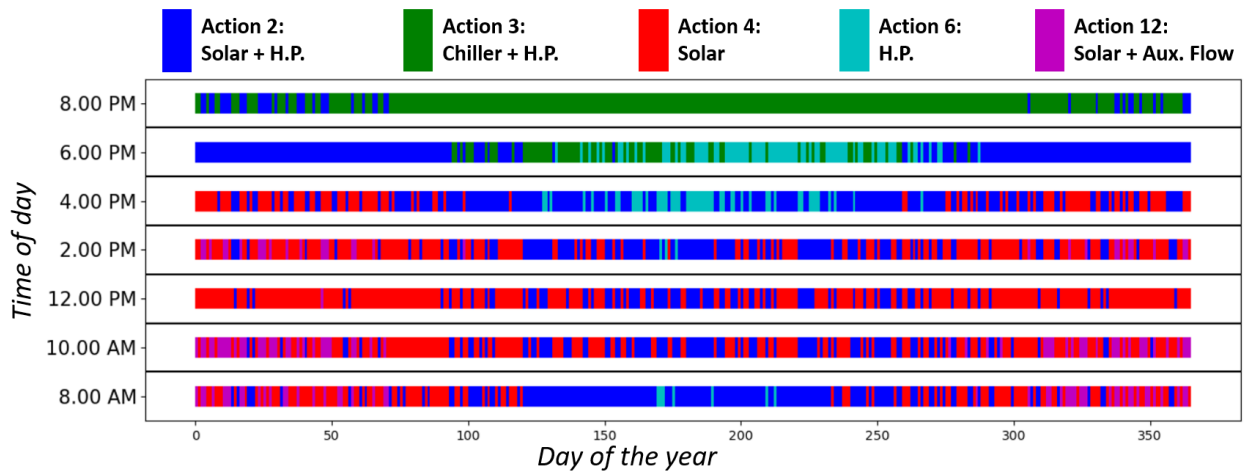


Figure 28: Complete description of the actions visualized in Figures 26 and 27.

Other indicators that will sometimes be considered are the ones that contribute to the calculation of the global reward function: the number of hours of a day during which the dressing rooms received water that was warmer than 40°C, the “total heat/electric consumption” indicator, the “clean heat/total heat” indicator, and the number of hours during which the solar collectors reached temperatures higher than 100°C.

Chapter 6: Results and Discussion

6.1. Comparison of DRL algorithms

In this section, the three training algorithms discussed in Section 3.2 and subsections are tested and compared. The training conditions, i.e. the architectures of the networks and other hyperparameters, are equal when it is possible to do so; however, each method has its own hyperparameters that have no analog in the other two, so it was not always possible to keep exactly the same conditions. For example, the Actor-Critic method has a second neural network that helps with training; this network can have its own architecture and its own learning rate.

To begin, two agents are trained with each method and the resulting rewards are compared. Only the architectures of the networks are changed between the two tests. The first reward function, R_1 , as defined in Equation 110, is used with the α_1 parameter being equal to one. This is equivalent to say that the second reward function R_2 (defined by Equation 111) is used with α_5 being equal to zero. The other parameters of the reward function are shown in Table 8 (see Section 4.3 for more details on the definition of the reward function).

Table 8: Parameters of the reward function (DRL algorithm comparison)

Parameter	Value
α_1	1
α_2	0.5
α_3	5
α_4	1

The architectures and other hyperparameters used for the REINFORCE, Actor-Critic and Deep Q-Learning algorithms are shown in Tables 9, 10 and 11 respectively. In all neural networks, all hidden layers use the Rectified Linear Unit (ReLU) as their activation function, and the output layer has either no activation function or the Softmax activation function, depending on the case. The ReLU function is defined as:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (113)$$

Table 9: Hyperparameters and architectures for the REINFORCE method

Total simulated time	20 years
Discount factor	0.4
Update period (How many episodes (days) are executed between the training steps?)	10
Learning rate	0.001
Momentum factor	0.8
Network architecture for Test 1	Input size: 10 Hidden layer sizes (in order): 32, 24 Output size: 16 Activation function of hidden layers: ReLU Activation function of output layer: Softmax
Network architecture for Test 2	Input size: 10 Hidden layer sizes (in order): 48, 32, 32, 24 Output size: 16 Activation function of hidden layers: ReLU Activation function of output layer: Softmax

Table 10: Hyperparameters and architectures for the Actor-Critic method

Total simulated time	20 years
Discount factor	0.4
Learning rate of the actor network	0.0002
Learning rate of the critic network	0.001
Momentum factor of the actor network	0.8
Momentum factor of the critic network	0.8
Actor network architecture for Test 1	Input size: 10 Hidden layer sizes (in order): 32, 24 Output size: 16 Activation function of hidden layers: ReLU Activation function of output layer: Softmax
Critic network architecture for Test 1	Input size: 10 Hidden layer sizes (in order): 24, 12 Output size: 1 Activation function of hidden layers: ReLU Activation function of output layer: None
Actor network architecture for Test 2	Input size: 10 Hidden layer sizes (in order): 48, 32, 32, 24 Output size: 16 Activation function of hidden layers: ReLU Activation function of output layer: Softmax
Critic network architecture for Test 2	Input size: 10 Hidden layer sizes (in order): 32, 24, 24, 8 Output size: 1 Activation function of hidden layers: ReLU Activation function of output layer: None

Table 11: Hyperparameters and architectures for the Deep Q-Learning method

Total simulated time	20 years
Discount factor	0.4
Type of training (Normal DQN or Double DQN)	Double DQN
Update period of the target network	10 days
Learning rate	0.001
Momentum factor	0.8
Network architecture for Test 1	Input size: 10 Hidden layer sizes (in order): 32, 24 Output size: 16 Activation function of hidden layers: ReLU Activation function of output layer: None
Network architecture for Test 2	Input size: 10 Hidden layer sizes (in order): 48, 32, 32, 24 Output size: 16 Activation function of hidden layers: ReLU Activation function of output layer: None
ϵ -greedy method	<ul style="list-style-type: none"> - The value of ϵ is equal to one at the beginning. It is maintained at this value for two years, and the network is not updated. This period is used to fill the replay memory. - After two years the value of ϵ starts to decline linearly; it reaches its minimum value of 0.2 at 12 simulation years. When the value of ϵ starts to decline, the network also begins to be updated once at the end of each day. - Between 12 years and 20 years of simulation, the value of ϵ remains constant at 0.2.
Length of the replay memory (in experiences)	6000
Batch size	100
Prioritized experience replay	<ul style="list-style-type: none"> - The proportional method is used to determine the priority number, as shown by Equation 56. - The value of α is set to 1. - The value of ϕ is set to 0.2. - The value of ψ is set to 0

Figures 29, 30 and 31 show the results of the REINFORCE algorithm, the Actor-Critic algorithm and the Deep Q-Learning algorithm respectively. In each figure, the graph at the top shows the result of the first test and the graph at the bottom shows the result of the second test, with different architectures as defined in Tables 9, 10 and 11.

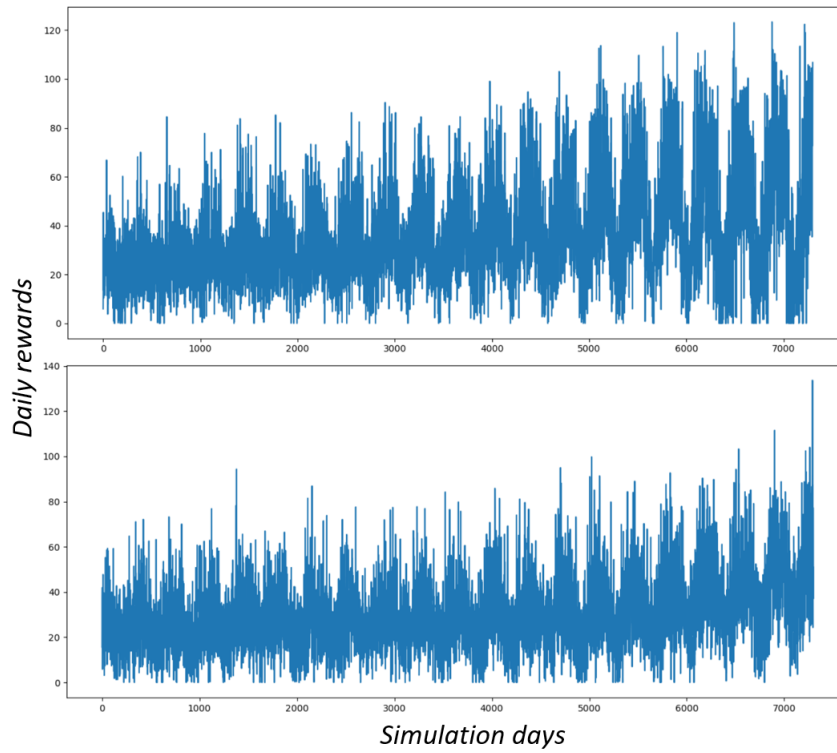


Figure 29: Results of the REINFORCE algorithm
 Top: Test 1. Bottom: Test 2

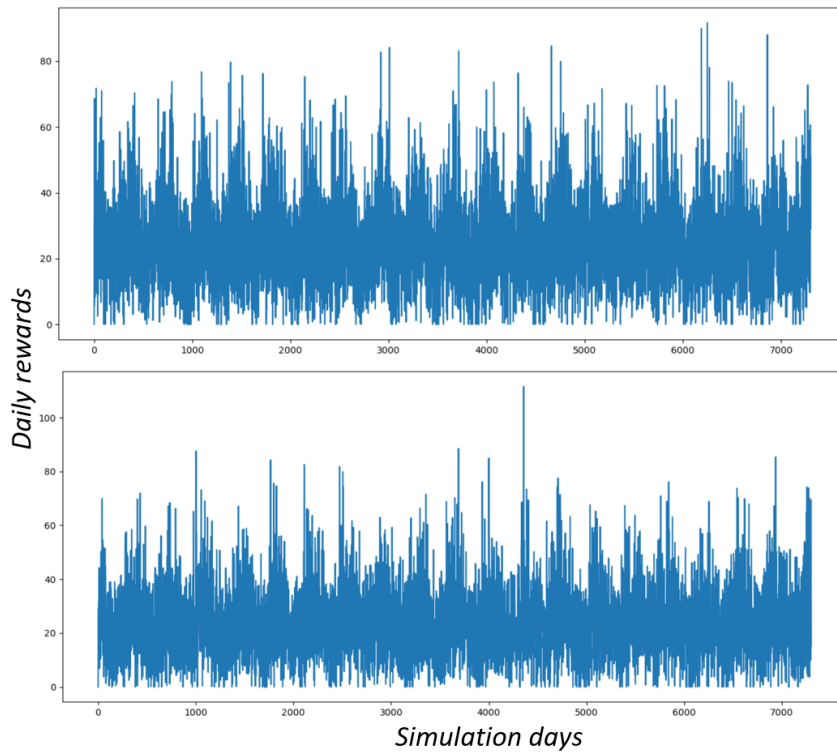


Figure 30: Results of the Actor-Critic algorithm
 Top: Test 1. Bottom: Test 2

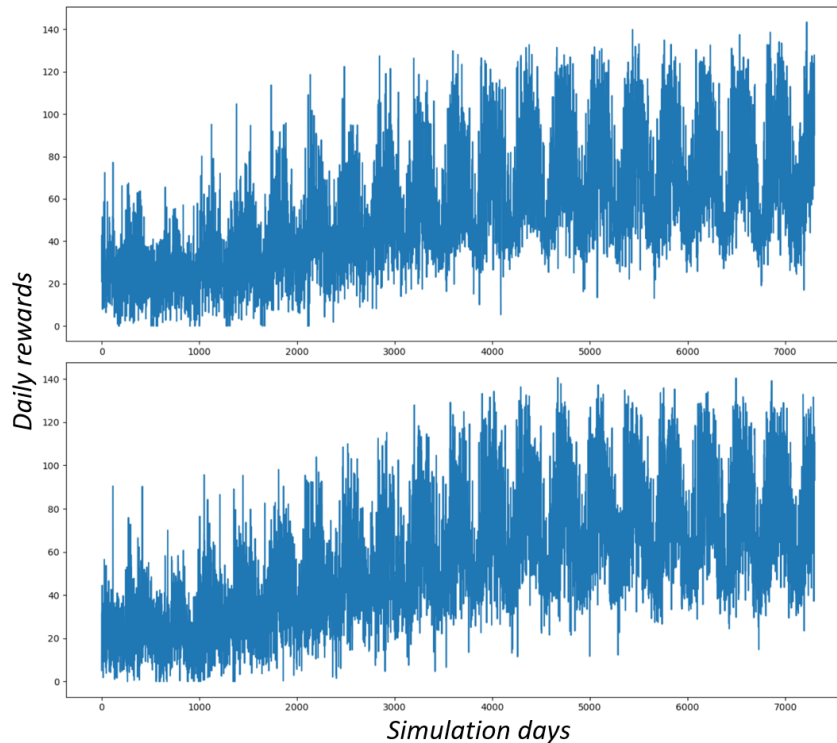


Figure 31: Results of the Deep Q-Learning algorithm
Top: Test 1. Bottom: Test 2

From the results shown in Figures 29, 30 and 31, it can be clearly seen that the Deep Q-Learning algorithm performs better than the other two algorithms. With the REINFORCE algorithm, the rewards seem to increase as the training process progresses, but this increase is not as remarkable as with the Deep Q-Learning method. In the first test with the REINFORCE algorithm, the rewards are quite high in summer, but they are also very low in winter, whereas with Deep Q-Learning the daily rewards rarely fall below 30 by the end of the training process. In the second test with the REINFORCE algorithm, it seems that the rewards start to increase consistently at the end of the training process. It seems that if this process is continued, the results could get better. With the Actor-Critic algorithm it does not seem that the rewards increase at all; instead, their behavior is similar during the whole training process. This could be because the hyperparameters are not appropriately tuned.

It was decided to extend the training process shown at the bottom of Figure 29 (REINFORCE algorithm) for 20 more years, in order to see how much the rewards increase if this process is continued. Besides, three more tests are carried out with the REINFORCE algorithm with different hyperparameters in order to check if the convergence of the algorithm could be achieved faster. In all three tests, Architecture 2, as defined in Table 9, was used. The hyperparameters are changed with respect to the first tests as follows: in the first test, the learning rate was increased to 0.002; in the second one, the learning rate was kept at 0.001 but the update period was decreased to 5 days; in the third test, the learning rate was increased to 0.003 and the update period was increased to 20 days.

The result of the extension of the training process shown at the bottom of Figure 29 is shown in Figure 32. Regarding the three extra tests, the third of them (learning rate equal to 0.003 and update period equal to 20 days) seemed to be converging at the end of the 20 years of training, just like the previous case. For this reason, it was extended to 40 years as well. The resulting rewards are shown in Figure 33.

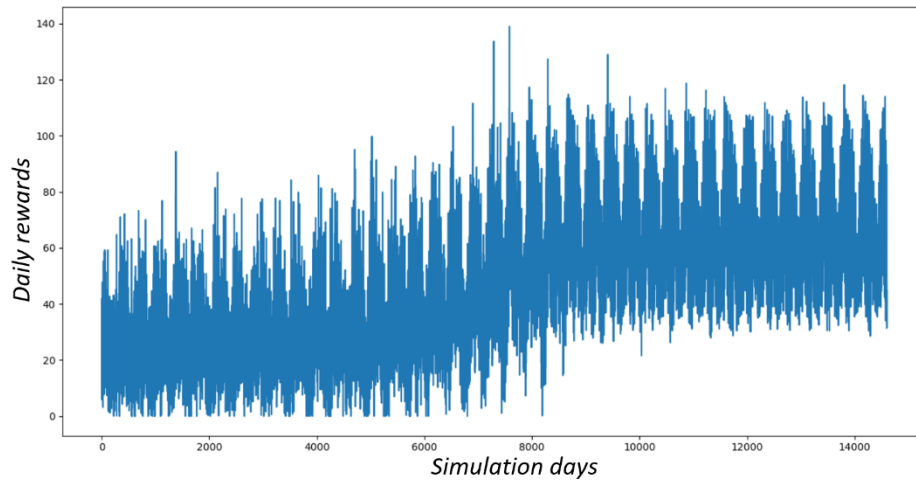


Figure 32: REINFORCE algorithm. Process shown at the bottom of Figure 29, extended to 40 years.

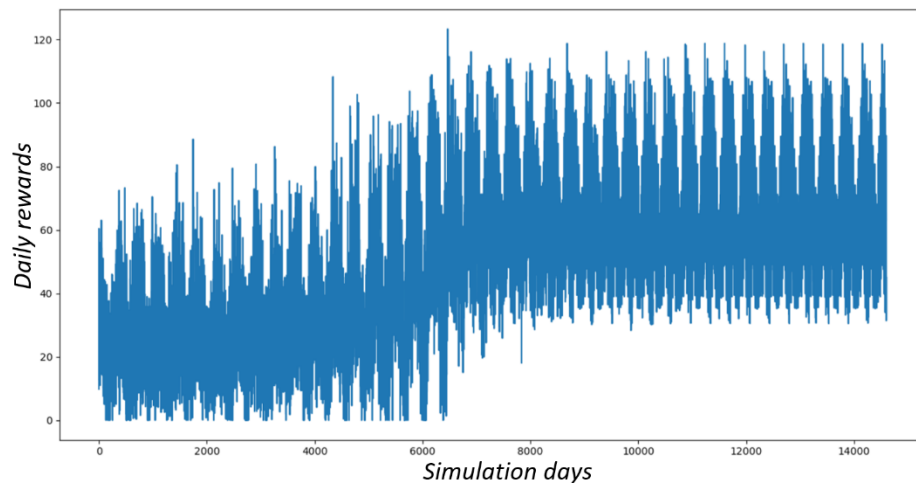


Figure 33: REINFORCE algorithm. Learning rate was set to 0.003; update period was set to 20 days.

As can be seen, both training processes have converged quite well, reaching acceptable and stable rewards. Regarding the Actor-Critic algorithm, more hyperparameter combinations were tested but no convergence was achieved. All results are similar to the ones shown in Figure 30.

Considering the results shown above, it was decided to continue the study with the Deep Q-Learning algorithm for the following reasons:

- The convergence of Deep Q-Learning seems to be less sensitive to hyperparameter tuning. One of the known difficulties of Deep Reinforcement Learning in general is that convergence is often very hard to achieve (Géron [36], page 633), but in this case, Deep Q-Learning seems to converge always and is not remarkably sensitive to hyperparameter variation.
- Convergence of Deep Q-Learning is faster than that of the REINFORCE algorithm. As will be discussed in coming sections, convergence of Deep Q-Learning can be achieved in less than 10 years if the value of ϵ (from the ϵ -greedy method, see Section 3.2.5.3 for details) is decreased faster. With REINFORCE, there is no direct control of the moment in which the training process converges, and in the case of the experiments that were conducted, convergence did not happen in less than approximately 20 simulation years.
- The rewards achieved by the Deep Q-Learning training processes are higher than those of the REINFORCE algorithm. This can be appreciated when they are compared with more

precision. In Figure 34, the rewards received during the last training year of each method are smoothed with a moving average of 21 days. At the beginning and at the end of the year, the smoothed rewards are remarkably higher with both agents trained with the Deep Q-Learning algorithm.

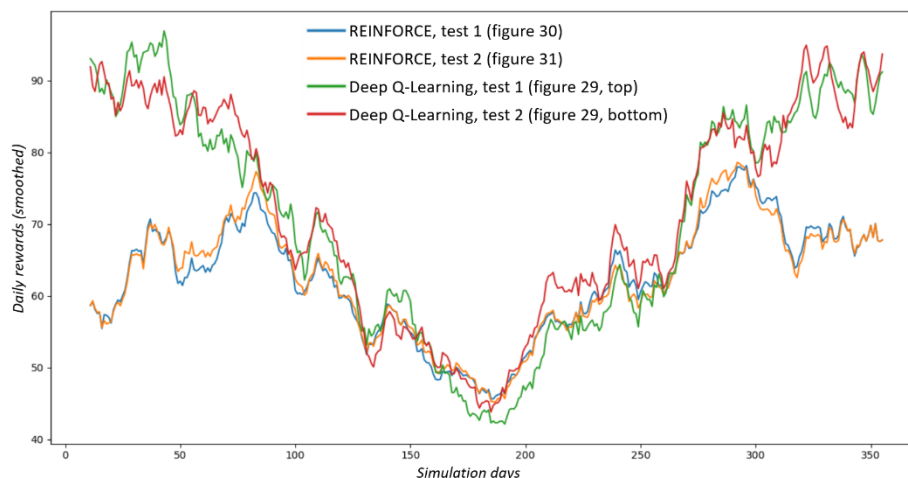


Figure 34: Smoothed rewards during the last simulation year. REINFORCE and Deep Q-Learning compared.

For the reasons just mentioned, all the training processes in the coming sections will be made with the Deep Q-Learning algorithm.

6.2. Comparison to a non-smart-controlled baseline

In this section, the Reinforcement Learning approach to the task of controlling the studied water heating system will be validated by comparing the performance of a DRL-trained agent with a non-controlled baseline.

For the baseline, it will be considered that all heating stages are permanently used, and the “auxiliary flow” is never used. To evaluate the performance of both systems, their rewards will be compared. However, there is a problem that must be solved in order for the comparison to be “fair”: when the “total heat/electric consumption” indicator is used for the reward function, an easy way for the controlling agent to increase this indicator is to turn off the chiller, because the chiller is by far the most energy-consuming device of the whole system. The reason why in the baseline the chiller is permanently used is that in reality, the chiller is needed for purposes which are external to the system that is being simulated here. These purposes are heating the pool, dehumidifying the building and cooling it. For this reason, the second reward function was defined in Equation 111 to take this “external motivation” to use the chiller into account. In this reward function, α_5 is a parameter to define a “prize” that is given to the agent for using the chiller. As already discussed in the rewards section (Section 4.3), this is not an accurate way of modeling an “external motivation” to use the chiller; the only way to do that would be to model the rest of the building in order to quantitatively measure the cooling load of the chiller. However, the α_5 parameter proposed here is a simple way to make the use of the chiller “desirable” in order to increase the rewards.

The parameters of the reward function and the hyperparameters of the training process are shown in Table 12.

Table 12: Reward and training parameters for the comparison to the baseline.
 R_2 as defined in Equation 111 was used

Reward Parameters	
Parameter	Value
α_2	0.5
α_3	5
α_4	1
α_5	{ 0 , 2 , 5 , 10 }
Training Hyperparameters	
Total training time	11 years
Discount factor	0.4
Type of training (Normal DQN or Double DQN)	Double
Update period of the target network	10 days
Learning rate	0.001
Momentum factor	0.8
Network architecture	Input size: 10 Hidden layer sizes (in order): 48, 32, 32, 24 Output size: 16 Activation function of hidden layers: ReLU Activation function of output layer: None
ϵ -greedy method	- The value of ϵ is equal to one at the beginning. It is maintained at this value for one year, and the network is not updated. This period is used to fill the replay memory. - After one year the value of ϵ starts to decline linearly; it reaches its minimum value of 0.2 at 5 simulation years. When the value of ϵ starts to decline, the network also begins to be updated once at the end of each day. - Between 5 years and 11 years of simulation, the value of ϵ remains constant at 0.2.
Length of the replay memory (in experiences)	6000
Batch size	100
Prioritized experience replay	- The proportional method is used to determine the priority number, as shown by Equation 56. - The value of α is set to 1. - The value of ϕ is set to 0.2. - The value of ψ is set to 0

6.2.1. Testing method

To compare the smart agents to the baseline, both will be tested under conditions that are different from those of the training process. During the test period, the smart agents are not trained (i.e. the parameters of the neural network are not updated) and no random actions are chosen (i.e. the actions which have the highest estimated Q-Value are always chosen). The rewards received by the smart agents will be compared to the rewards received by the baseline system during a time period of one year (from January 1 to December 31). It makes no sense to compare the rewards during a longer time span because the smart agents will repeat their decisions in the next years and therefore the

rewards will be very similar. Only the initial conditions of the simulation could create small differences between a year and the next one. To avoid the problem of the initial conditions, two years and 10 days will be simulated, and the second year will be considered for the results. The rewards will be smoothed, as discussed in Section 5.2, using a moving average of 21 days. That is the reason for simulating two years *and 10 days*: the last ten days are used by the moving average to smooth the rewards until the last day of the second year. The last ten days of the first simulation year are also necessary to compute the moving average on the first day of the second year. The same testing method will be used in the coming sections as well.

6.2.2. Results

Four different values of α_5 were considered: 0, 2, 5 and 10. Clearly, a different agent must be trained with each value of α_5 . Figure 35 shows the smoothed rewards of the smart agents and the baseline during the test period, for the each value of α_5 . The white and gray stripes at the background show the seasons of the year, starting and ending with summer. A detail worth mentioning is that the first trained agent is being shown in each case, so there is the option that better agents can be achieved if more attempts were executed.

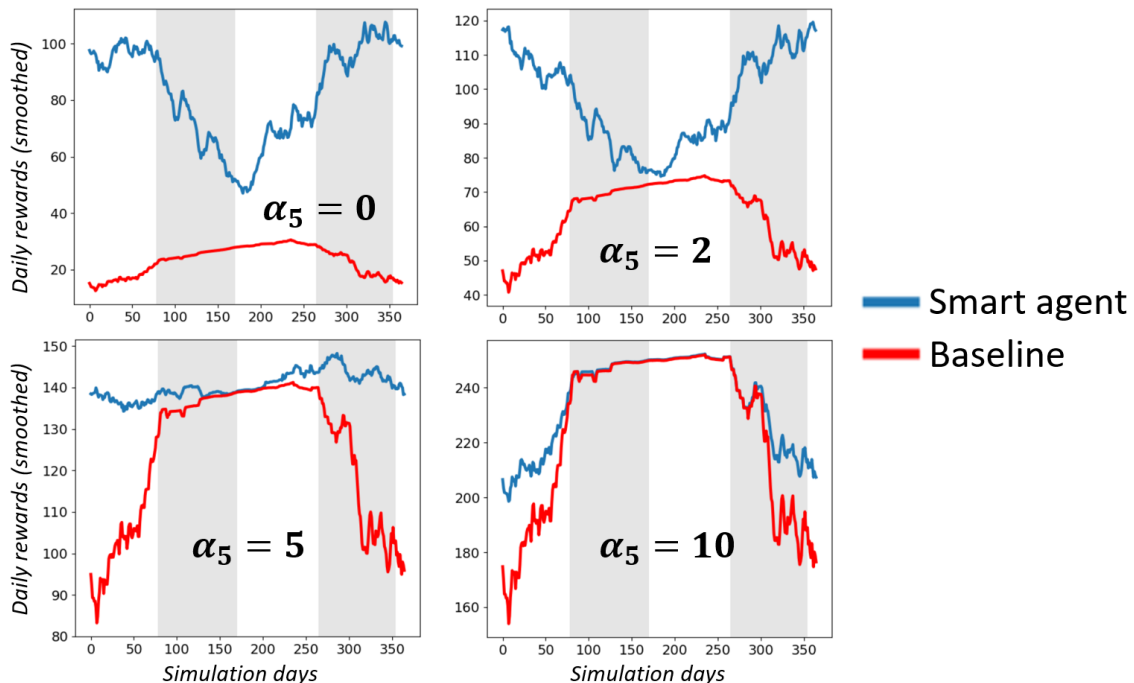


Figure 35: Results of the comparisons between the smart agents and the baseline

As Figure 35 shows, with the two lower values of α_5 , the rewards that the smart agents got are clearly higher than those of the baseline during the whole year. When α_5 was set to five, there is a small portion of the year during which the rewards of the baseline model are roughly equal to those of the smart agent. When α_5 was set to ten, the rewards of the smart agent are better in spring and in summer, and they are practically equal to the rewards received by the baseline during autumn and winter. This is because the best strategy that the smart agent found for the cold months with that value of α_5 is very similar to the strategy of the baseline. The general trend is that as the “prize” for using the chiller increases, the smart agents have a narrower margin to outcompete the baseline.

By comparing the actions chosen by agents that were trained with different values of α_5 , it can be seen how they chose to use the chiller with different frequencies. In Figure 36, which shows the actions chosen by the agent trained with α_5 equal to zero, the chiller was turned on in the actions

shown in blue and in red. This happened mainly during the central months of the year at 6.00 PM and at 8.00 PM. In Figure 37, which shows the actions chosen by the agent trained with α_5 equal to one, the chiller was turned on almost in every action, except for the actions that are shown in magenta and in red. From April to September, it can be seen that the agent almost always chose Action number 0, which is equivalent to the strategy that the baseline uses. Only at 8.00 PM the solar field is turned off.

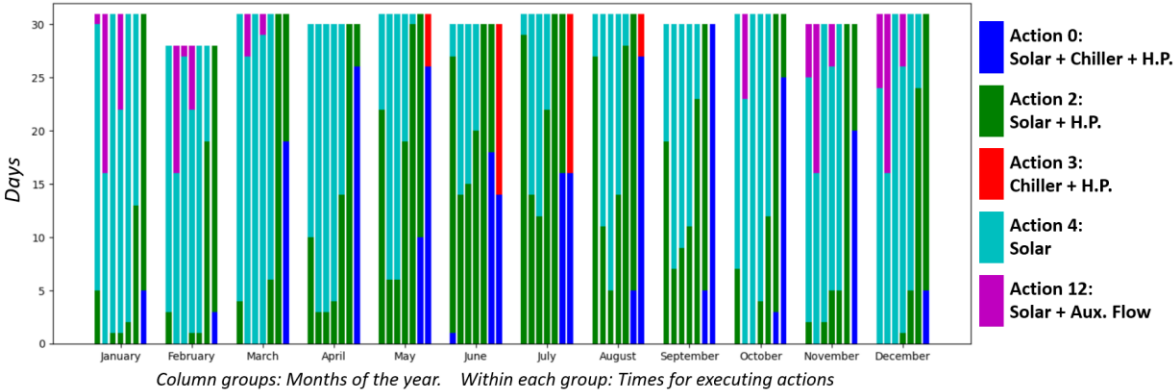


Figure 36: Actions taken by an agent trained with α_5 equal to zero

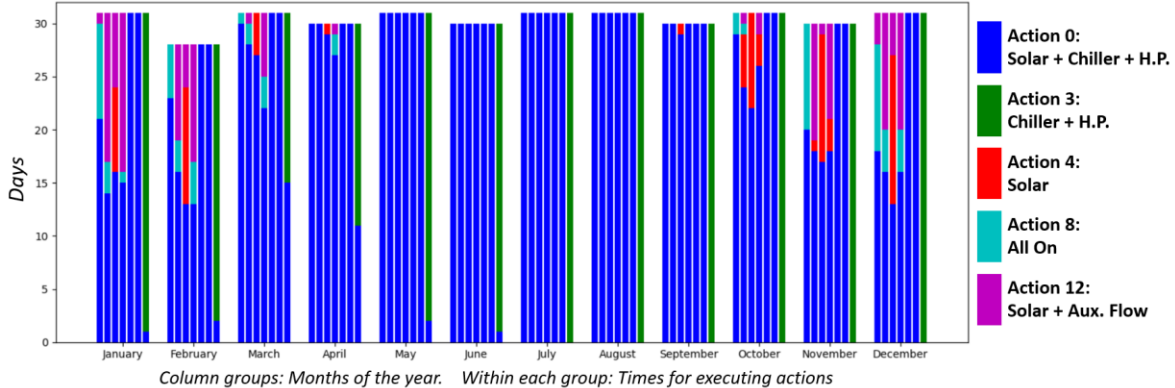


Figure 37: Actions taken by an agent trained with α_5 equal to ten

The α_5 parameter is clearly not a good method to “encourage” the agent to use the chiller, because when the value of that parameter is large, the agent uses the chiller more frequently in winter than in summer, which is not realistic. However, it has been shown that the smart agent can find the way to *at least* perform as good as the baseline strategy, even if it must adopt the same strategy because there is no possible improvement. The agent only uses that strategy when it is needed, while in warm months it finds a way to outcompete the baseline.

The reason why the rewards get smaller in summer with large values of α_5 for both methods, is that the “total heat/electric consumption” indicator loses its importance in the reward (see Equation 111). In this context of larger rewards, the “degradation” factor becomes important; this indicator becomes zero in winter, and in summer the smart agent makes more to avoid it (actually, the baseline strategy does nothing to avoid it); that is why the baseline performs worse than the smart agent in summer, but in winter both are equal.

6.3. Comparison of different network architectures and training hyperparameters

The main goal of this section is to optimize the training conditions by comparing different neural network architectures and training hyperparameters. Since the Deep Q-Learning algorithm is

defined by many hyperparameters, it is unfeasible to test all possible combinations with a “grid” exploration. Instead, only some of the important hyperparameters are modified: network architecture, discount factor, and normal DQN is compared to Double DQN. The importance of using momentum is also put to the test.

The first reward function R_1 , as defined in Equation 110, is used. The parameters of the reward function are detailed in Table 13 (see Section 4.3 for details on the reward function).

Table 13: Parameters of the reward function for Section 6.3 and subsections.
The function R_1 (Equation 110) is used.

Parameter	Value
α_1	1
α_2	0.5
α_3	5
α_4	1

Since many steps of the training process depend on random results, the agents resulting from different training processes under the same conditions will be, at least, slightly different. These “random steps” are the weight initialization of the neural network, the selection of experiences to form the batches, the decisions on whether to execute a random action or to choose the action that the agent considers to be the best, and the selected action when a random action is chosen. For this reason, the performance of a single agent does not give enough information about a specific combination of hyperparameters; the performance of other agents that were trained under the same conditions could be very different.

Because of this, to compare different values of the same hyperparameter (or different network architectures), several agents are trained with each value of the hyperparameter that is being varied, and then the performances of agents that were trained with the same hyperparameter value are averaged. Figure 38 shows an example of this. The curves shown in that figure were created randomly and do not reflect the performances of real agents. In the figure, each color represents one value of the hyperparameter that is being varied; three different agents were trained with each value. It is also useful to compute the standard deviation of the performances to check how variable the performance of agents that were trained with each hyperparameter value is.

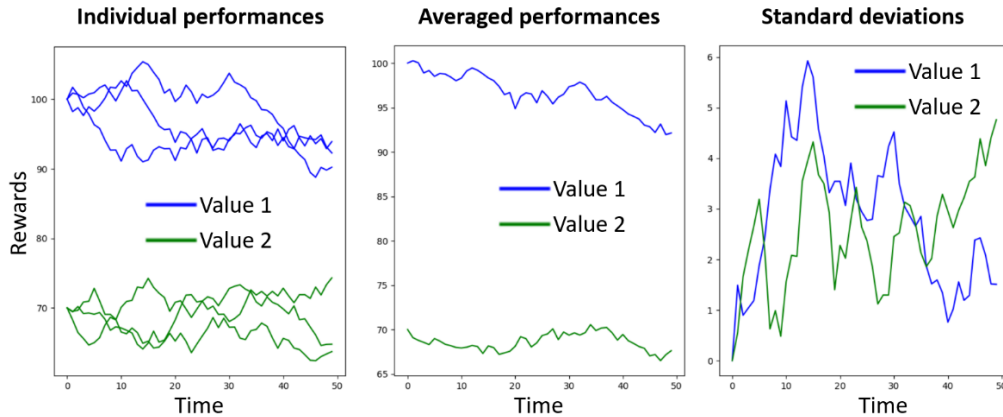


Figure 38: Process to compare different values of the same hyperparameter. Several agents are trained with each value (in this example 3 agents per value).

As already discussed in Section 6.2.1, the agents are tested during 2 years and 10 days. The rewards obtained by the agents during the second year are smoothed with a moving average of 21 days.

6.3.1. Comparing Different Architectures

In this Section, different layer sizes and network depths are compared. In all cases, the input dimension is 10 due to the environment state that was defined in Section 4.4, and the output layer has 16 neurons because the network produces a Q-Value approximation for each possible action (see Section 4.2 for details on the possible actions that the agent can choose). Figure 39 shows all architectures that are considered. Like in previous sections, the hidden layers use the ReLU activation function, which was defined in Section 6.1 (Equation 113). Table 14 shows the training hyperparameters.

Architecture 1			Architecture 2			Architecture 3			Architecture 4		
Layer	Size	Activation	Layer	Size	Activation	Layer	Size	Activation	Layer	Size	Activation
Input	10	-	Input	10	-	Input	10	-	Input	10	-
Hidden 1	72	ReLU	Hidden 1	48	ReLU	Hidden 1	128	ReLU	Hidden 1	72	ReLU
Hidden 2	48	ReLU	Hidden 2	32	ReLU	Hidden 2	72	ReLU	Hidden 2	48	ReLU
Output	16	-	Output	16	-	Output	16	-	Hidden 3	48	ReLU
									Output	16	-

Architecture 5			Architecture 6			Architecture 7		
Layer	Size	Activation	Layer	Size	Activation	Layer	Size	Activation
Input	10	-	Input	10	-	Input	10	-
Hidden 1	48	ReLU	Hidden 1	48	ReLU	Hidden 1	48	ReLU
Hidden 2	32	ReLU	Hidden 2	32	ReLU	Hidden 2	32	ReLU
Hidden 3	32	ReLU	Hidden 3	32	ReLU	Hidden 3	32	ReLU
Hidden 4	24	ReLU	Hidden 4	24	ReLU	Hidden 4	24	ReLU
Output	16	-	Hidden 5	24	ReLU	Hidden 5	10	ReLU
			Output	16	-	Output	16	-

Figure 39: Architectures tested.

Table 14: Hyperparameters of the Deep Q-Learning algorithm for Section 6.3.1

Total training time	11 years
Discount factor	0.6
Type of training (Normal DQN or Double DQN)	Double
Update period of the target network	10 days
Learning rate	0.001
Momentum factor	0.8
ϵ -greedy method	<ul style="list-style-type: none"> - The value of ϵ is equal to one at the beginning. It is maintained at this value for one year, and the network is not updated. This period is used to fill the replay memory. - After one year the value of ϵ starts to decline linearly; it reaches its minimum value of 0.2 at 5 simulation years. When the value of ϵ starts to decline, the network also begins to be updated once at the end of each day. - Between 5 years and 11 years of simulation, the value of ϵ remains constant at 0.2.
Length of the replay memory (in experiences)	6000
Batch size	100
Prioritized experience replay	<ul style="list-style-type: none"> - The proportional method is used to determine the priority number, as shown by Equation 56. - The value of α is set to 1. - The value of ϕ is set to 0.2. - The value of ψ is set to 0.

Each architecture is trained 5 times. As discussed above, the smoothed rewards of the agents that have the same network architecture are averaged. Figure 40 shows the averaged results, and Figure 41 shows the standard deviations.

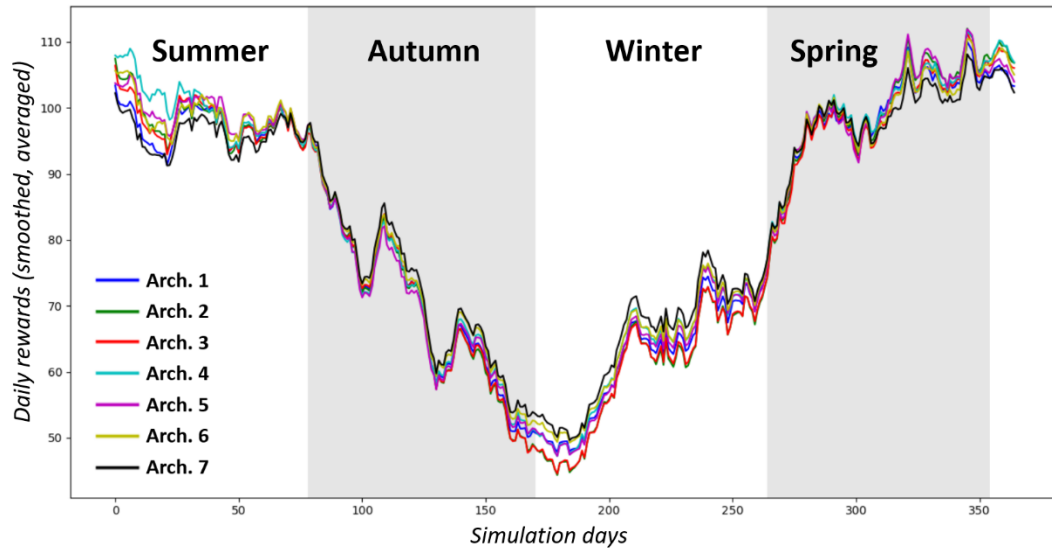


Figure 40: Mean smoothed rewards during the test year, considering 5 trainings for each architecture.

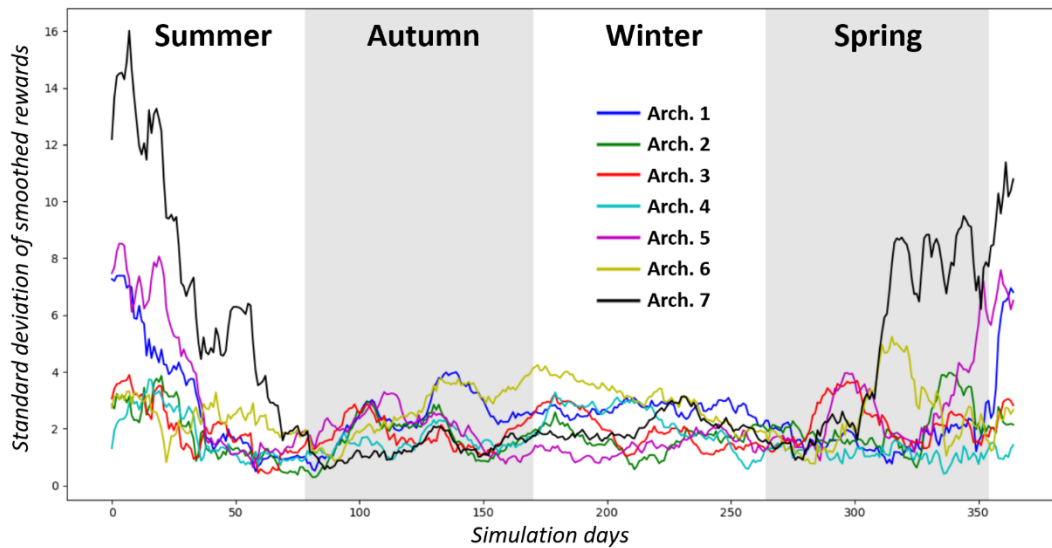


Figure 41: Standard deviation of smoothed rewards during the test year, considering 5 trainings for each architecture.

Upon observing the average results, all averaged rewards are quite similar. It is clear that the obtained rewards are considerably higher in summer (both sides of the graph) because it is easier for the agents to obtain more energy from the sun. This is consistent with the reward definition and it is the objective of the training process.

Now, it is possible to compute the mean values of the curves shown in both Figures 40 and 41, thus obtaining a “mean of means” and a “mean of standard deviations” for each architecture. This is shown in Table 15.

Table 15: Mean values of the curves shown in Figures 40 and 41

Architecture number	Mean reward	Mean standard deviation
1	81.98	2.51
2	81.84	1.79
3	81.58	1.97
4	83.21	1.65
5	82.68	2.49
6	83.17	2.61
7	82.77	3.91

It is remarkable that Architecture 7, which has the lowest performance in summer, also has the greatest rewards in winter, when the rewards reach their minimum. As can be seen in Figure 41 and in Table 15 as well, the results of this architecture vary significantly, thus not being a good option. Architecture 4 seems to be the best option since it has the highest mean rewards and also the lowest variability, as shown in Table 15.

6.3.2. Comparing Different Discount Factors

The training hyperparameters shown in Table 14 are kept in this section, with the exception of the discount factor, which is the parameter that will be analyzed. Architecture 4 of the previous section, defined by Figure 39, is used. The 5 values considered for the discount factor are: 0.2; 0.4; 0.6; 0.8 and 1.0. Five agents are trained with each discount factor value. The averaged performances of each value are shown in Figure 42.

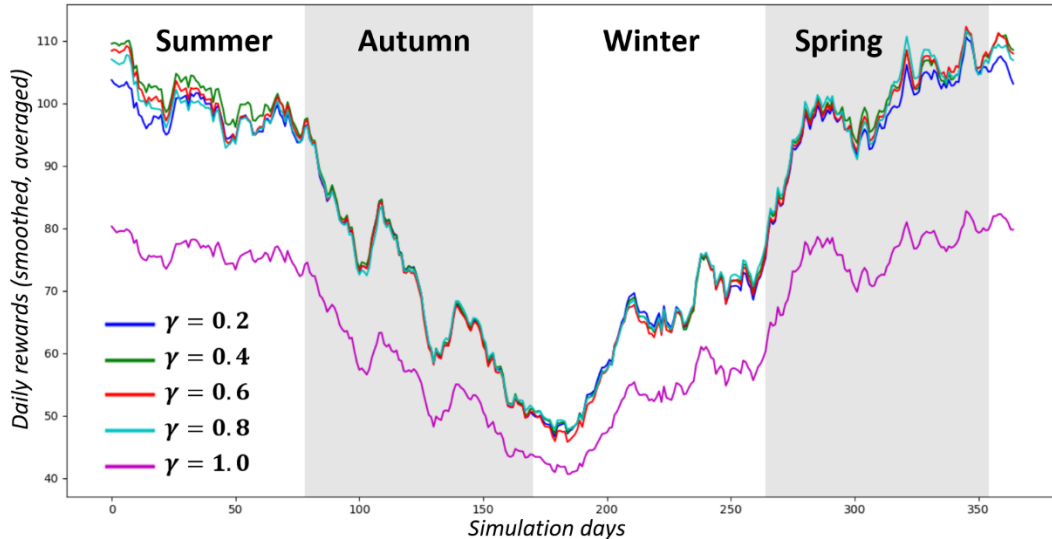


Figure 42: Averaged rewards when the discount factor is varied. 5 trainings for each value of γ

The most remarkable feature of the graph shown in Figure 42 is that γ equal to one got a clearly lower average performance. By visualizing the performance of the individual agents trained with that discount factor, it can be seen that this is because one of the training processes did not converge at all. Figure 43 shows the performances of the individual agents that were trained with γ equal to one.

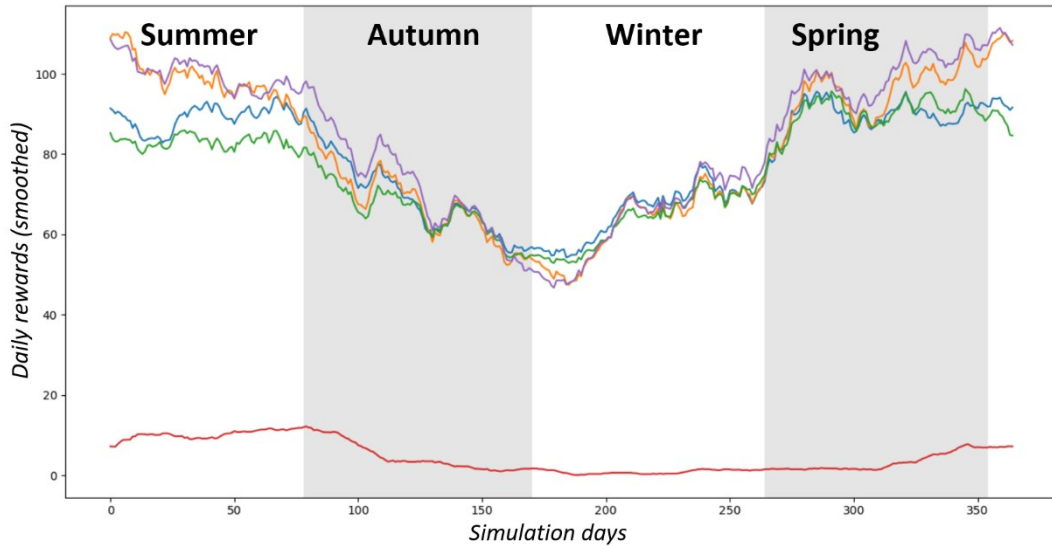


Figure 43: Performances of the individual agents that were trained with γ equal to one

Although the failed training process is a legitimate result that shows that there is the possibility of not achieving convergence with that discount factor, it would be interesting to answer the question: is 1 the best value for the discount factor if the failed training process is not considered? The answer is no, as shown in Figure 44. This figure shows the same results that are shown in Figure 42, but the worst result of γ equal to one is not being considered. During a small portion of the year, in winter, γ equal to one has the best average results, but for most of the year it is the worst of all discount factors.

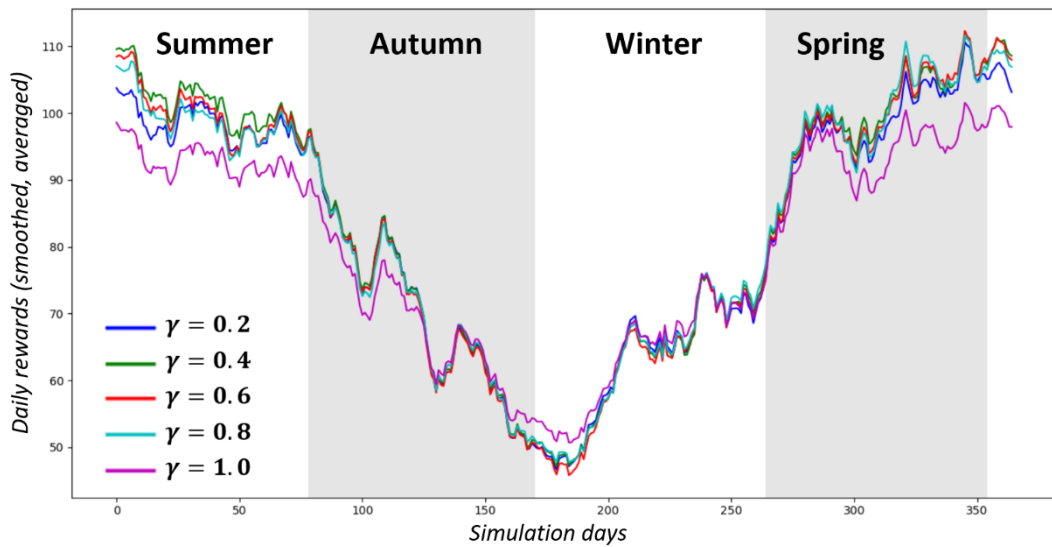


Figure 44: Average performances of the discount factors when the worst result of γ equal to one is not considered.

Figure 45 shows the standard deviations of the rewards. All results are being considered, even the worst result of γ equal to one. That is the reason why the standard deviation of that discount factor value is so high. Figure 46 shows the deviations of the other discount factors, i.e. when γ equal to one is not considered. The only reason to plot this is to better compare the deviations of the other discount factors.

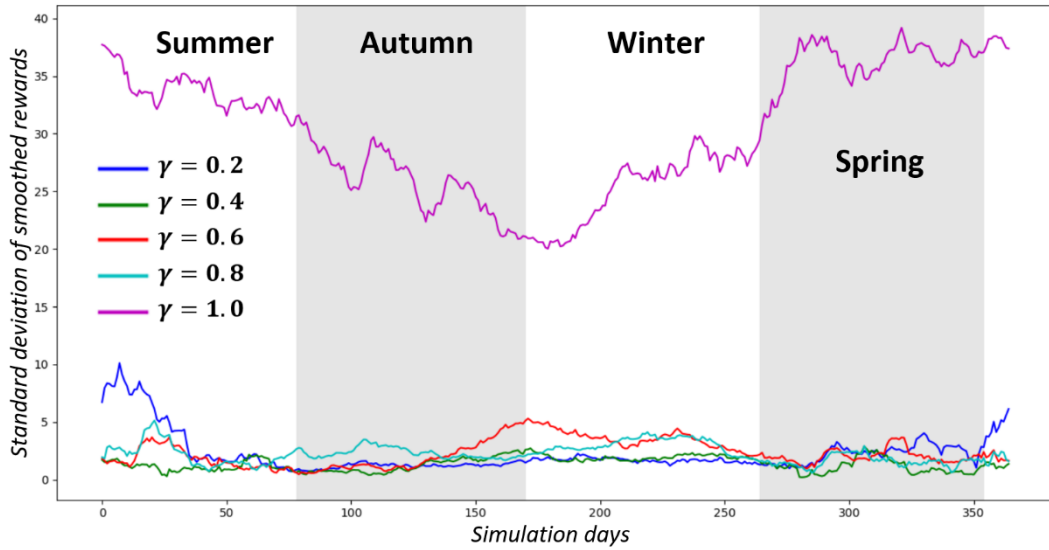


Figure 45: Standard deviations of rewards when the discount factor is varied. 5 trainings for each value of γ

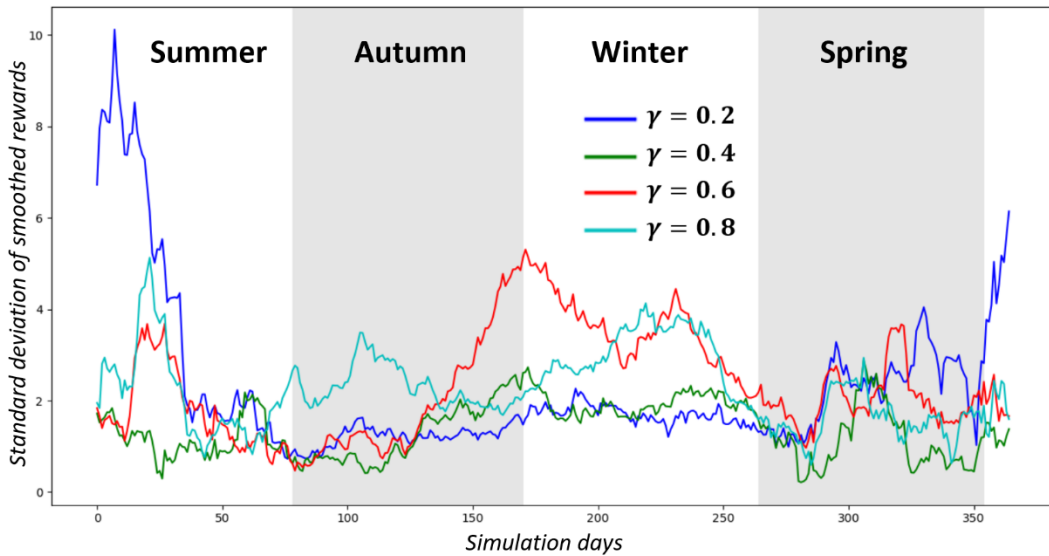


Figure 46: Standard deviations of rewards when the discount factor is varied. γ equal to one is not considered.

Table 16 shows the average values of the curves shown in Figures 42 and 45.

Table 16: Mean values of the curves shown in Figures 42 and 45.

Discount factor	Mean reward	Mean standard deviation
0.2	82.25	2.27
0.4	83.54	1.38
0.6	82.87	2.35
0.8	82.96	2.27
1.0	64.68	30.31

From Table 16, and also from Figures 42 and 46, it can be concluded that 0.4 is the best value for the discount factor, both because it maximizes the average rewards and because it minimizes the variability of the performance of the agents.

6.3.3. Comparing traditional DQN to Double DQN

The discount factor is now changed to 0.4, which is the best value discovered in the previous section. Architecture 4, as defined in Section 6.3.1, is still used in this section. All other parameters are the same as shown in Table 14.

The two modes of DQN, traditional and double, are compared. 10 agents are trained with each method. As before, the smoothed rewards of all tests are averaged and their standard deviations are calculated. Figures 47 and 48 show the average rewards and their standard deviation, respectively.

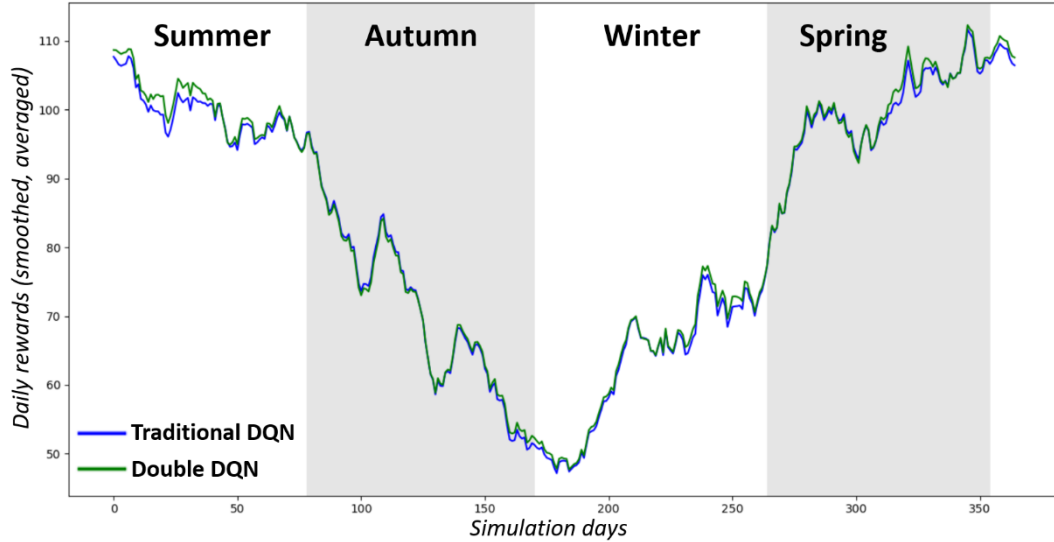


Figure 47: Traditional and Double DQN average rewards (ten agents were trained with each method).

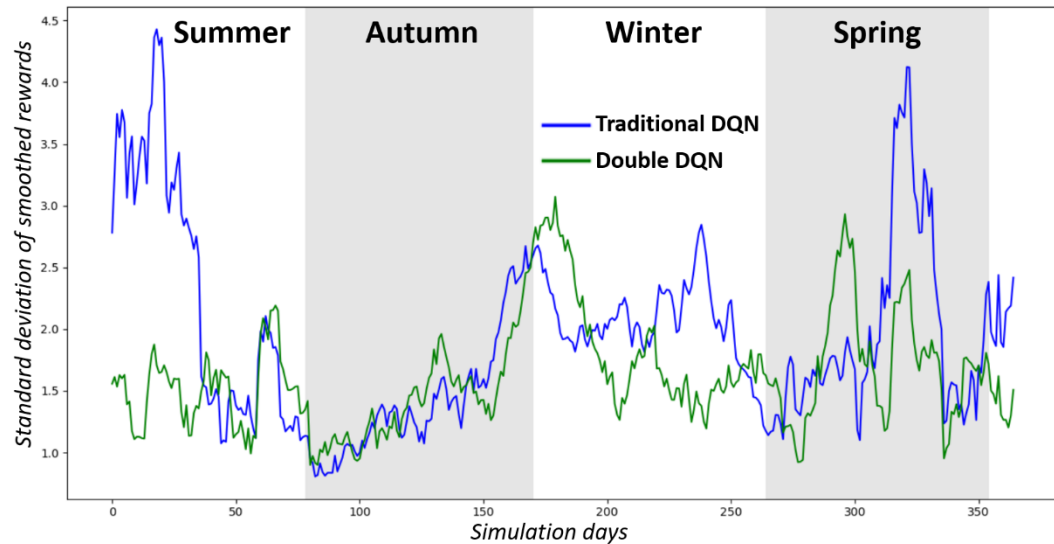


Figure 48: Traditional and Double DQN standard deviations (ten agents were trained with each method).

The average values of the curves shown in Figures 47 and 48 are shown in Table 17.

Table 17: Mean values of the curves shown in Figures 47 and 48.

Training Mode	Mean reward	Mean standard deviation
Traditional DQN	83.09	1.94
Double DQN	83.66	1.63

The averaged results of both DQN modes are very similar. Double DQN has a small advantage at some moments of the year, but this advantage is small in comparison to the standard deviations. Regarding the standard deviations of both algorithms, there are moments when traditional DQN has a smaller deviation, but it also reaches maximum standard deviations which are considerably larger than those of the Double DQN algorithm. This is reflected in the results shown in Table 17, where the mean standard deviation of the Double DQN algorithm is smaller than that of traditional DQN. However, these results do not show a very large difference between both methods. Does this mean that the Double DQN algorithm is not better than traditional DQN as the creators of this algorithm claimed? Definitely not; these Deep Reinforcement Learning methods were created and tested in environments which are far more complex than this one, and where the convergence of the methods is far more difficult to achieve [44]; in this context, preventing the overestimation of Q-Values becomes fundamental, and because of this, Double DQN has shown to have a clear advantage. However, in the environment that is being studied here, Double DQN seems to not make such a great difference.

6.3.4. Effect of momentum

As discussed in Section 3.1.2, the technique of “momentum” is a way for the updating process of the network to gain “velocity” in successive iterations. This can lead to a faster convergence and also allows the training process to avoid “local optima”.

In all previous tests, the momentum factor (here denoted as β) was set to 0.8. In this section, it was changed to 0.0; this means that momentum is not being used anymore. Ten agents were trained with Double DQN, momentum factor equal to 0.0 and all other hyperparameters of the previous section (Section 6.3.3).

In Figure 49, the smoothed rewards of the agents trained with β equal to zero are compared with the rewards of the ten agents that were trained with Double DQN in the previous section (Section 6.3.3). The rewards are not averaged; instead, the performances of the individual agents are plotted.

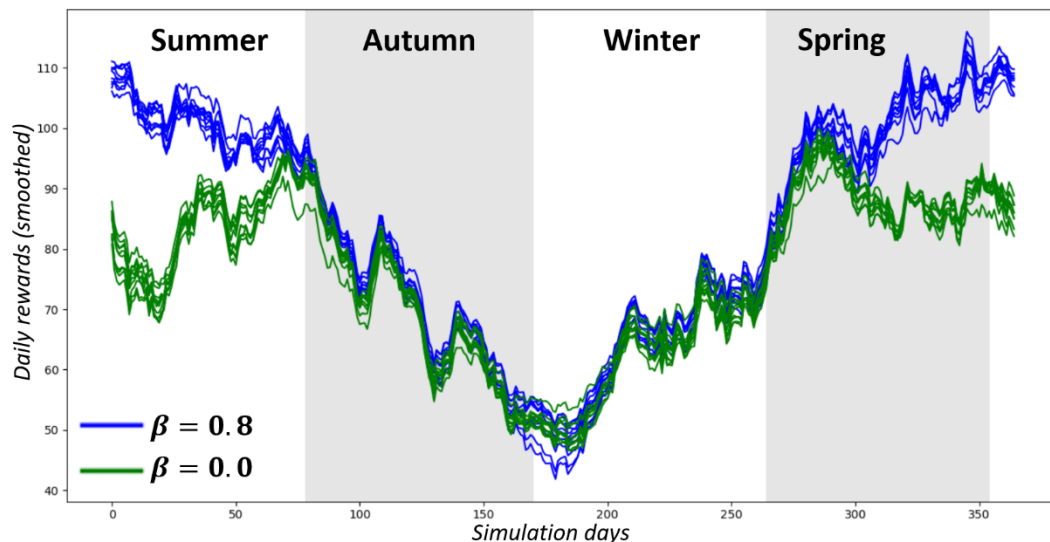


Figure 49: Performances when momentum is used and when it is not used

From Figure 49, it is clear that momentum plays a fundamental role in the training algorithm, since *all* agents that were trained without momentum had a considerably worse performance in spring and summer. This can be attributed to the fact that the agents got stuck in a local optimum when momentum was not used.

6.4. Behavior comparison under different reward parameters

In this section, the objective is not to look for the best parameters in order to optimize the performance of the agent, but to analyze how the behavior of the agent changes as the goal is changed, which takes place by changing the parameters of the reward function, as defined in Section 4.3. These parameters are meant to change the importance given to the indicators that are considered in the global reward function. In this context, it does not make much sense to compare the rewards that different agents get with different reward parameters, because when the parameters of the reward function are changed, then the rewards that the agents can expect to get change as well. Hence, it does not make sense to say “agent A got higher rewards than agent B” if agent A and agent B were trained and tested with different reward parameters. The only thing that agent A and agent B have in common is that they seek to maximize their respective reward function, and to achieve it they may take different decisions.

The reward function R_1 , as defined in Equation 110, is used.

Table 18 shows the training hyperparameters used for all tests in this section and subsections.

Table 18: Parameters of the Deep Q-Learning algorithm for Section 6.4 and subsections

Deep Neural Network Architecture	Architecture 4, as defined in Section 6.3.1.
Total training time	11 years
Total simulation time	12 years; during the last simulated year the network is not updated and the selected actions are always the ones that the agent considers to be the best.
Discount factor	0.4
Type of training (Normal DQN or Double DQN)	Double
Update period of the target network	10 days
Learning rate	0.001
Momentum factor	0.8
ϵ -greedy method	<ul style="list-style-type: none"> - The value of ϵ is equal to one at the beginning. It is maintained at this value for one year, and the network is not updated. This period is used to fill the replay memory. - After one year the value of ϵ starts to decline linearly; it reaches its minimum value of 0.2 at 5 simulation years. When the value of ϵ starts to decline, the network also begins to be updated once at the end of each day. - Between 5 years and 11 years of simulation, the value of ϵ remains constant at 0.2.
Length of the replay memory (in experiences)	6000
Batch size	100
Prioritized experience replay	<ul style="list-style-type: none"> - The proportional method is used to determine the priority number, as shown by Equation 56. - The value of α is set to 1. - The value of ϕ is set to 0.2. - The value of ψ is set to 0

6.4.1. Changing the value of α_1

The α_1 parameter measures the importance given to the indicators “total heat/electric consumption” and “clean heat/total heat”. When it is equal to one, only the “total heat/electric consumption” indicator is taken into account; when it is equal to zero, only the “clean heat/total heat” indicator is taken into account. As stated before, “clean heat” refers to the heat coming from the solar collectors and the chiller.

Table 19 shows the other parameters of the reward function that were constant while the value of α_1 was changed.

Table 19: Reward parameters (other than α_1) used in Section 6.4.1.

Parameter	Value
α_2	0.5
α_3	5
α_4	1

Figure 50 shows the “total heat/electric consumption” indicator of two agents during the 12 simulated years; one of those agents was trained with α_1 being equal to zero and the other one was trained with α_1 being equal to one. The first 11 years correspond to the training process; the last year could be considered as a “testing year”, as explained in Table 18. Figure 51 does the same, with the same two agents, but taking the “clean heat/total heat” indicator into account.

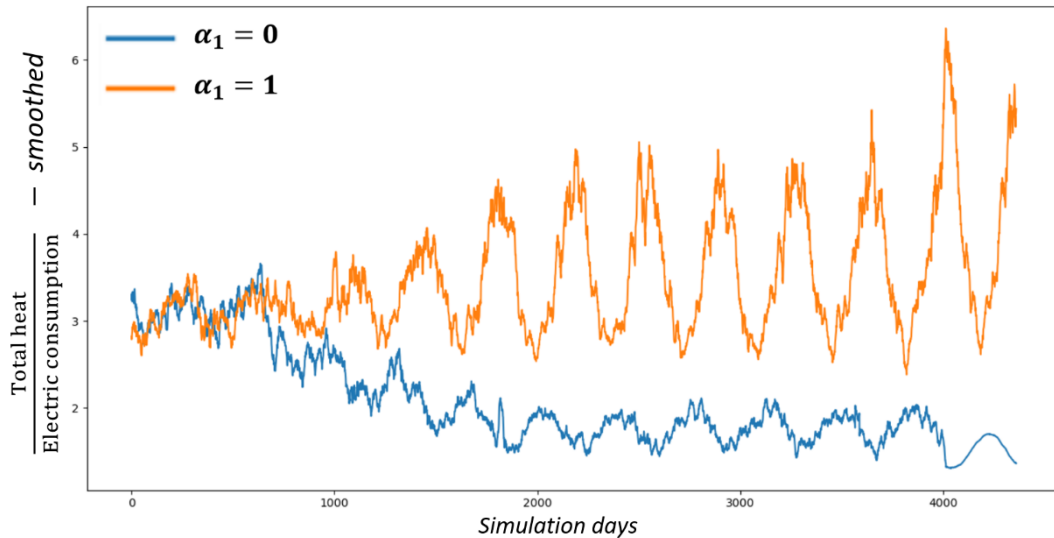


Figure 50: “Total heat/electric consumption” indicator. α_1 takes the values 0 and 1.

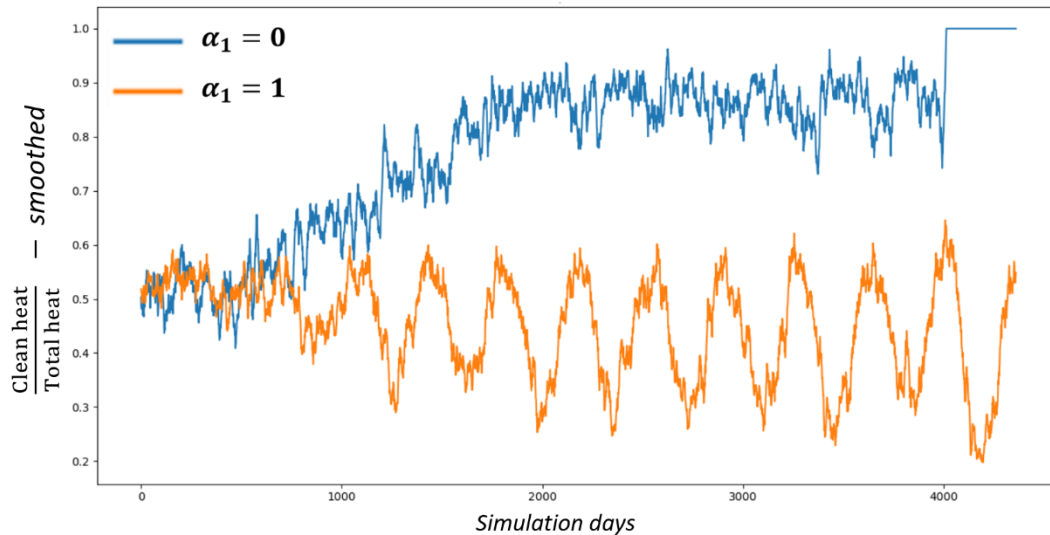


Figure 51: “Clean heat/total heat” indicator. α_1 takes the values 0 and 1.

It can be seen that, when training begins, both indicators are approximately equal when comparing the two agents. This is because, in the beginning, the network does not know anything about the environment yet, and all actions are random. When the probability of executing random actions decreases, it becomes appreciable that the agents privilege different indicators according to the reward that they are seeking. When α_1 is set to one, the agent only looks after the “total heat/electric consumption” indicator, and when α_1 is zero, the same happens for the “clean heat/total heat” indicator.

Note also that, when α_1 is zero, the “clean heat/total heat” indicator takes the value 1.0 during the whole last year (during no random actions are executed). This means that the water flow only receives heat from the solar field and the chiller; hence, the agent does not activate the heat pumps during the whole year, clearly seeking not to decrease the value of this indicator.

After the training processes shown in Figures 50 and 51, both agents were subjected to a testing period of two years and 10 days, as explained in Section 6.2.1. Figure 52 shows two other important indicators of the reward function that were recorded during that testing process: at the left, the daily hours during which the water coming out of the solar collectors reached temperatures higher than 100°C ; and at the right, the daily hours during which the water delivered to the dressing rooms was warmer than 40°C (this is considered to be the minimal comfortable temperature). In the case of the latter indicator, the expected value is 14 hours, which means that the water at the outlet of the system was warmer than 40°C from 8.00 AM to 10.00 PM. That indicator is not smoothed in Figure 52.

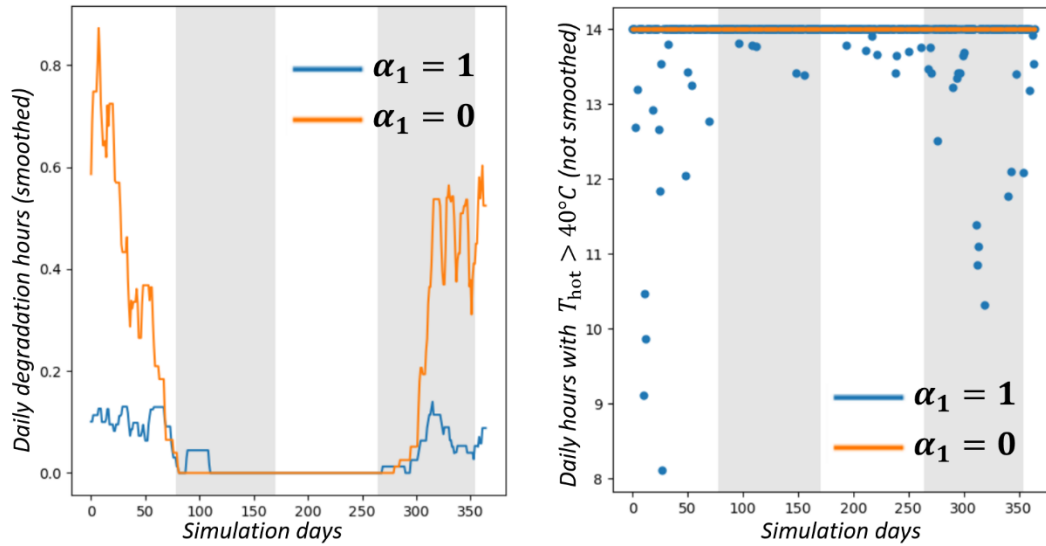


Figure 52: Degradation indicator and comfort indicator during a test year.
 Left: Daily hours during which the water temperature in the collectors reached more than 100°C
 Right: Daily hours during which the temperature of the water delivered by the system was higher than 40°C .

Although the α_3 parameter of the reward function, which measures the “punishment” for reaching high temperatures in the solar collectors, has been kept constant, the α_1 parameter seems to have an indirect effect on the amount of time that high temperatures are reached in the collectors, as can be seen at the left of Figure 52. α_1 equal to one seems to be better from this point of view. However, with that value of α_1 , there were days on which the water coming out of the heating system reached temperatures lower than 40°C ; this can be seen at the right of Figure 52. The agent delivered warm water 98.7% of the time of the whole year, but this is not a good reliability measure for the agent since it does not take into account the water demand at the moments when the water was not delivered at the minimum temperature of 40°C . A more accurate reliability indicator would be the percentage of the volume of demanded water that was delivered at temperatures higher than 40°C ; nevertheless, this was not recorded. The agent trained with α_1 equal to zero was able to always deliver water that was warmer than 40°C .

Figures 53 and 54 show the “total heat/electric consumption” and “clean heat/total heat” indicators of agents that were trained with intermediate values of α_1 . The values of α_1 that were used to train the agents whose results are plotted are: 0.0, 0.1, 0.2, 0.3, 0.5 and 1.0 (the values are closer to zero because the “total heat/electric consumption” indicator reaches values significantly larger than one, unlike the “clean heat/total heat” indicator). Each curve shows the result of only one agent (i.e. it is not the average of various agents). The results are taken from a testing period, as explained in Section 6.2.1.

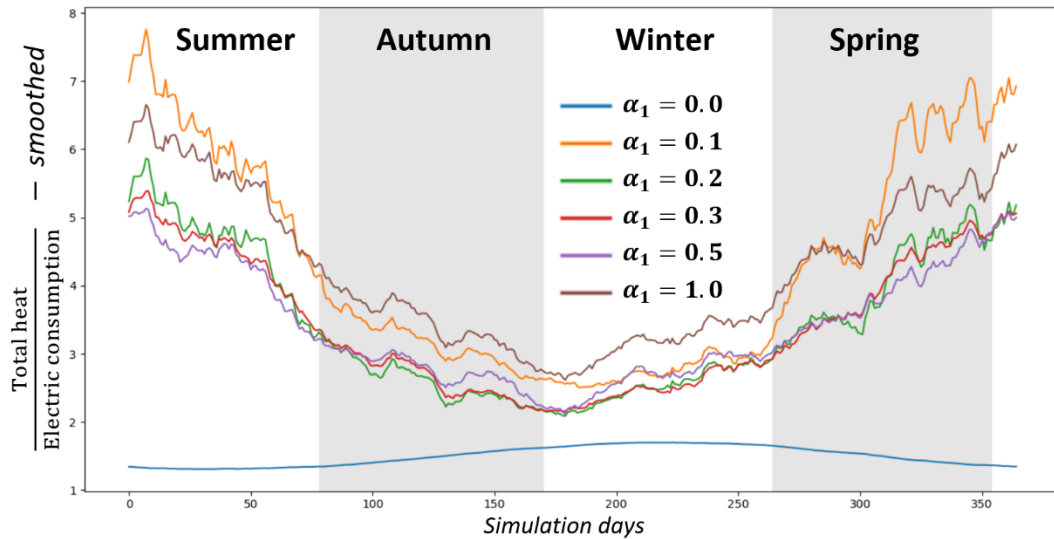


Figure 53: “Total heat/electric consumption” indicator during the testing period.

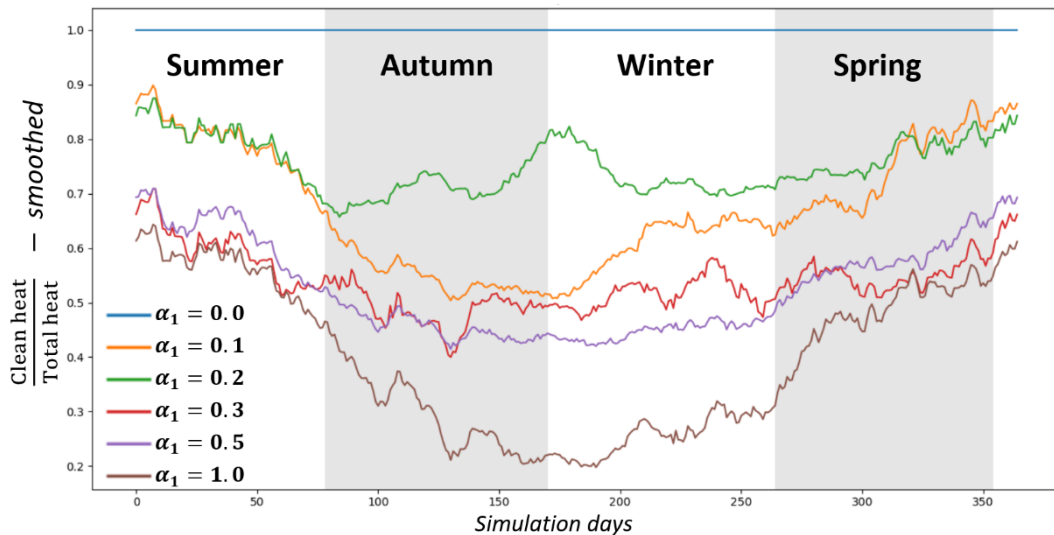


Figure 54: “Clean heat/total heat” indicator during the testing period.

In Figure 53, it seems that there is an abrupt change in the behavior of the “total heat/electric consumption” indicator when the α_1 parameter is changed from 0.0 to 0.1. On the other hand, the “clean heat/total heat” indicator (Figure 54) seems to have a more gradual change as the value of α_1 varies.

Figure 55 shows the behavior of the “total heat/electric consumption” indicator when α_1 takes values between 0.0 and 0.1. The goal is to discover “intermediate” behaviors, given the abrupt change that can be seen in Figure 53 between the values 0.0 and 0.1. The conclusion is that there are intermediate behaviors when α_1 takes the values 0.005, 0.010 and 0.020. With α_1 equal to 0.030, the indicator starts to behave like it does with the greater values of α_1 .

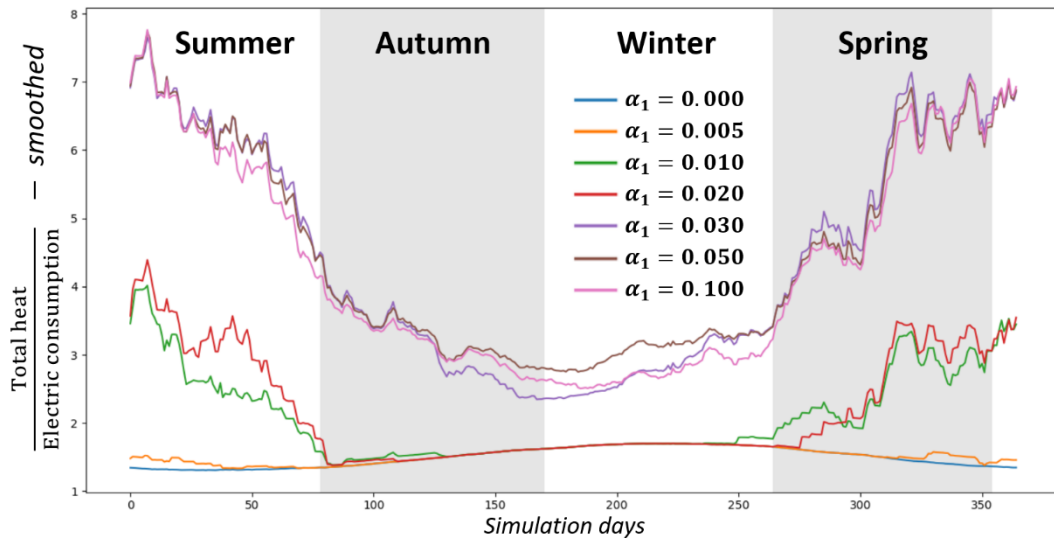


Figure 55: “Total heat/electric consumption” indicator during the testing period. α_1 between 0.0 and 0.1.

Figures 56 and 57 show the actions taken by the agents trained with $\alpha_1 = 0$ and $\alpha_1 = 1$, respectively. Regarding the value $\alpha_1 = 0$, it can only be said that the behavior was very simple: the agent always took the same action, which involved activating the chiller and the solar energy system. This is consistent with the result shown in Figures 51 and 54, which show that all the heat came from “clean sources”. Regarding the agent that was trained with $\alpha_1 = 1$, there are more features that can be mentioned: the solar collector field was used as the only energy source with more frequency in summer than in winter, and also with more frequency at midday than in the morning and in the afternoon. This is what is expected from the agent, since it is taking advantage of moments with high solar radiation to reduce the energy consumption. It can also be seen that the agent used the auxiliary flow in warm months and at times of high solar radiation. It is curious that at 12.00 PM the agent used the auxiliary flow with less frequency, but that can be explained because at that moment the water demand reaches its daily maximum; thus it is not necessary to use the auxiliary flow to avoid overheating of the solar collectors.

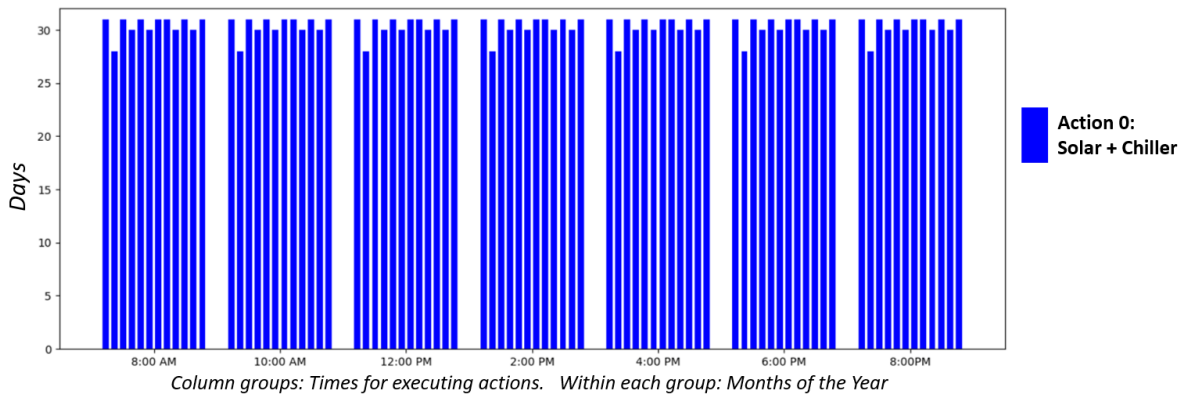


Figure 56: Actions taken by the agent trained with $\alpha_1 = 0$

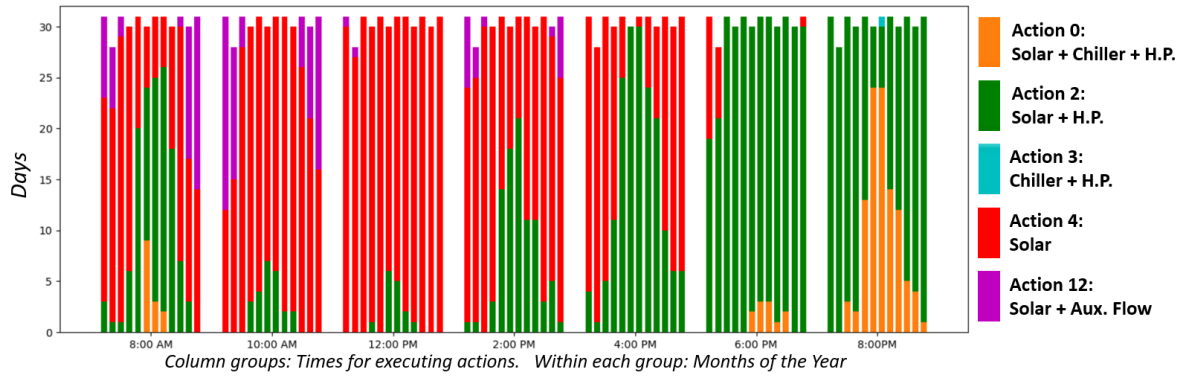


Figure 57: Actions taken by the agent trained with $\alpha_1 = 1$

After analyzing the behaviors shown in Figures 56 and 57, a question may arise: how “variable” are those behaviors if more agents are trained under the same conditions? To answer that question, Figure 58 shows the actions taken by two additional agents that were trained with $\alpha_1 = 0$ and Figure 59 shows the actions taken by two additional agents trained with $\alpha_1 = 1$. In other words, more agents were trained with exactly the same conditions in order to analyze how different the resulting actions are.

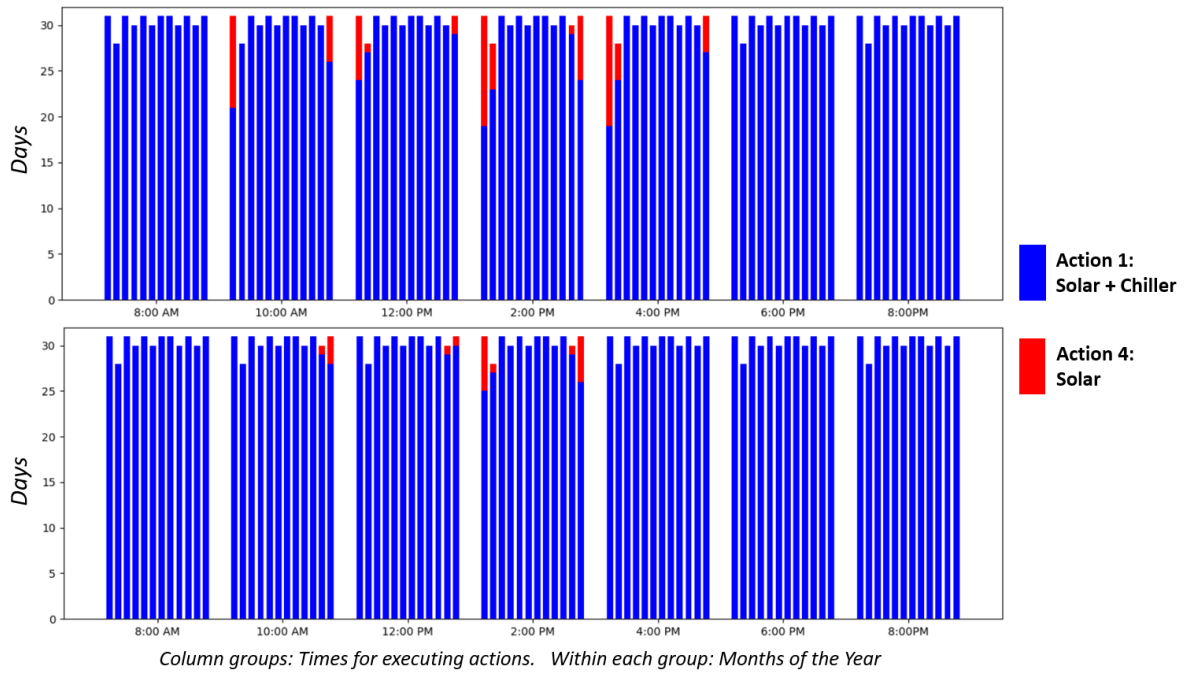


Figure 58: Two additional agents trained with $\alpha_1 = 0$

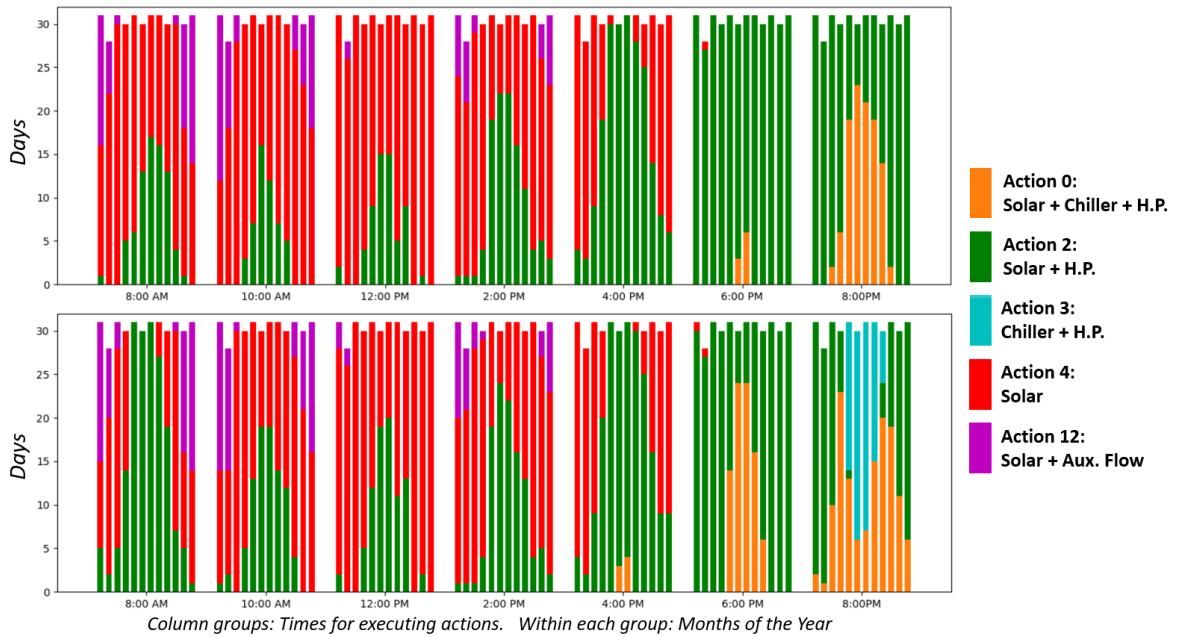


Figure 59: Two additional agents trained with $\alpha_1 = 1$

Unlike the agent presented in Figure 56, both additional agents that were trained with the same hyperparameters (which are shown in Figure 58) occasionally chose to exclusively use the solar collectors as energy source. As expected, this happened at warm months of the year and at times of high solar radiation.

Regarding the agents trained with $\alpha_1 = 1$, one of the additional agents did not use Action number 3 at all, and the other one used that action with much more frequency than the original agent (shown in Figure 57). The two additional agents did not use Action number 0 at 8.00 AM, but they did use it in the afternoon. The agent shown in Figure 57 used the solar collectors as the only energy source with more frequency at 10.00 AM and at 12.00 PM, also during winter.

A question that may arise now is: how different are the rewards of the agents that exhibit these different behaviors? This comparison will be done only with the agents that were trained with α_1 being equal to one, because the rewards of the agents that were trained with α_1 being equal to zero are practically equal. This comparison is shown in Figure 60. In the figure, “Agent 1” is the one shown in Figure 57, while “Agent 2” and “Agent 3” are the ones shown at the top and at the bottom of Figure 59, respectively.

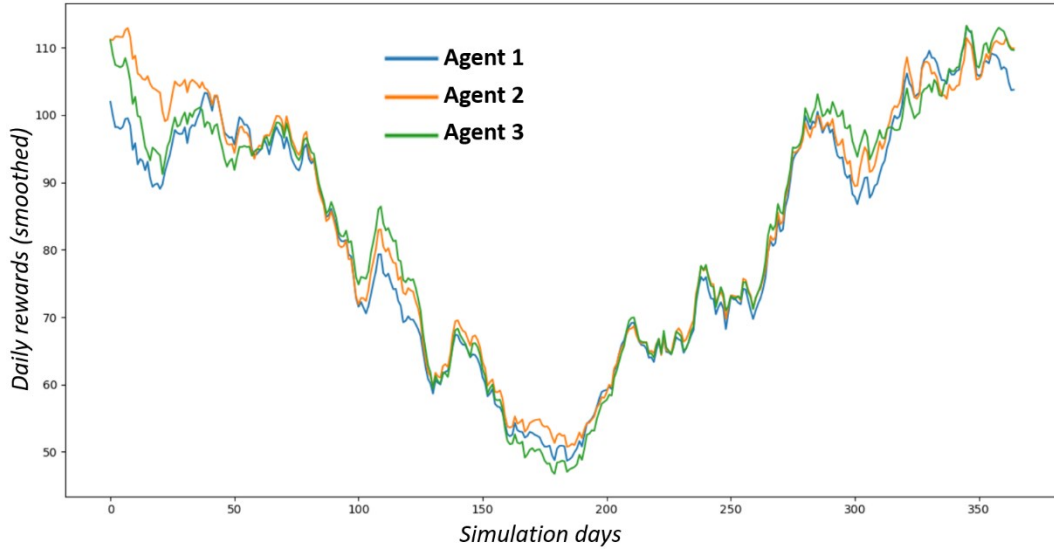


Figure 60: Reward comparison of agents trained with $\alpha_1 = 1$

The rewards are more similar than one would expect by seeing the differences between the behaviors of the agents. The first agent seems to use only the solar collectors at 10.00 AM and at 12.00 PM with much more frequency than the other two, during the whole year. However, the differences between their rewards are not so large. Only at the beginning of the year there seems to be a remarkable difference between their performances.

6.4.2. Changing the value of α_2

The α_2 parameter is part of the “comfort” factor of the reward function, as defined by Equation 105. It defines how much the reward grows if the water coming out from the last heating stage reaches temperatures remarkably higher than 40°C. When this factor is equal to zero, the reward does not depend on the temperature of the water coming out of the heating system (only on whether its temperature is higher than 40°C).

The values considered for α_2 were 0, 1 and 4. These three values were combined with α_1 equal to zero and one, so six reward definitions were tested. The other reward parameters are shown in Table 20.

Table 20: Reward parameters for Section 6.4.2

Parameter	Value(s)
α_1	{ 0, 1 }
α_2	{ 0, 1, 4 }
α_3	5
α_4	1

Figure 61 shows the actions that were executed by the three agents trained with α_1 equal to zero.

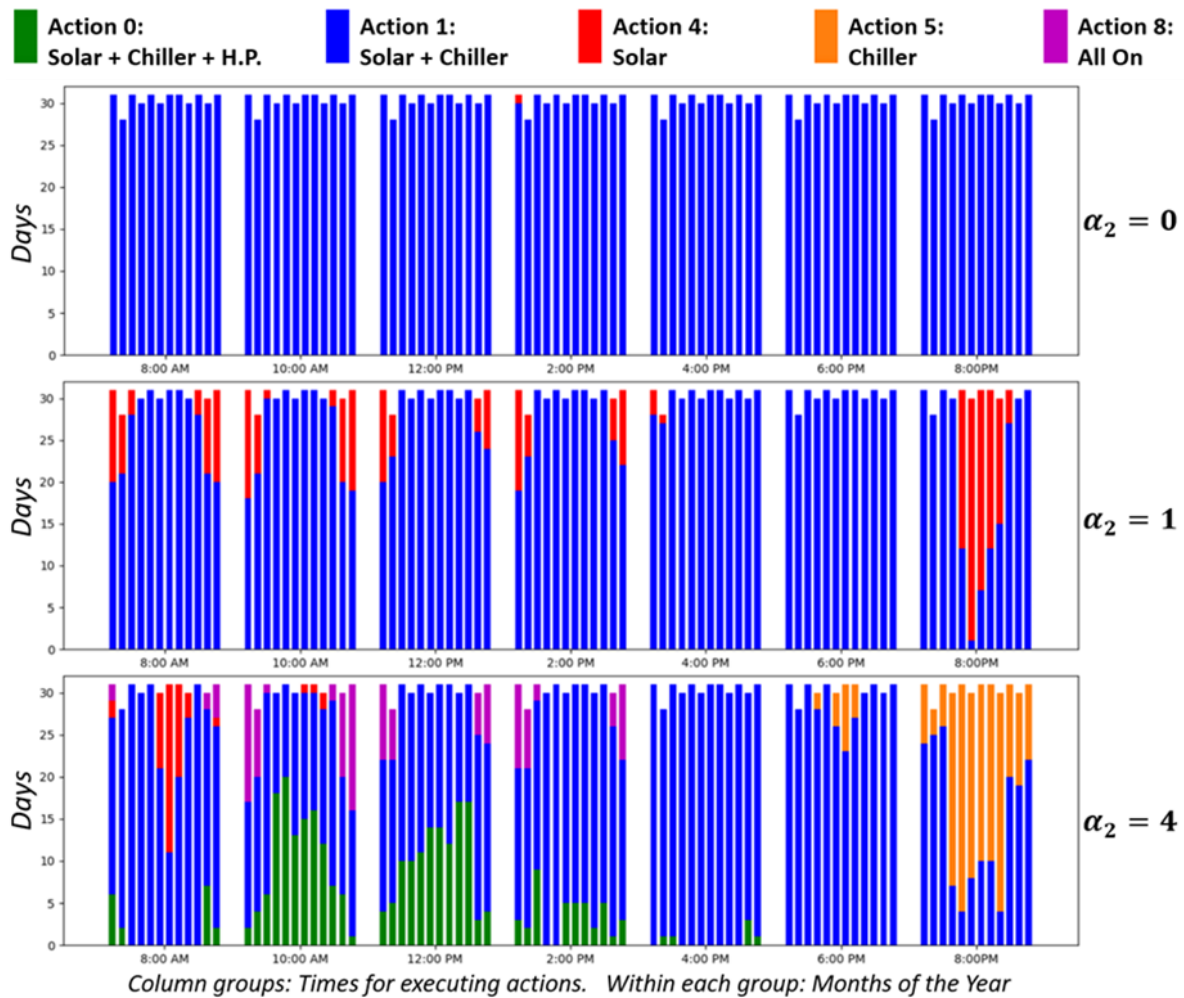


Figure 61: Actions taken by the agents trained with α_1 equal to zero.

From Figure 61, it is remarkable that with the two lower values of α_2 , the agents did not use the heat pumps in the entire year, just like in Section 6.4.1 when α_1 was set to zero. But when α_2 was set to four, the agent did use the heat pumps several times (all actions marked with green and purple involve using the heat pumps). This has a clear explanation: all three agents shown in Figure 61 were trained with α_1 being equal to zero; this means that the agent seeks to increase the “clean heat/total heat” indicator. To do this, the best is to only use the chiller and/or the solar collectors. Nevertheless, larger values of α_2 entail larger rewards if higher temperatures are reached at the outlet of the last heating stage of the system. For this reason, when the value of α_2 is large enough, the agent uses the heat pumps even though this means reducing the value of the “clean heat/total heat” indicator. Figure 62 shows the temperature of the water flow leaving the last heating stage (before being mixed with mains water to be delivered to the dressing rooms) during a whole year when only the chiller and the solar collectors are used every time (i.e. the same strategy shown in Figure 56). Considering that the heat pumps are meant to keep the water in their respective storage tanks at around 60°C, there is great potential of increasing the water temperature by activating the heat pumps.

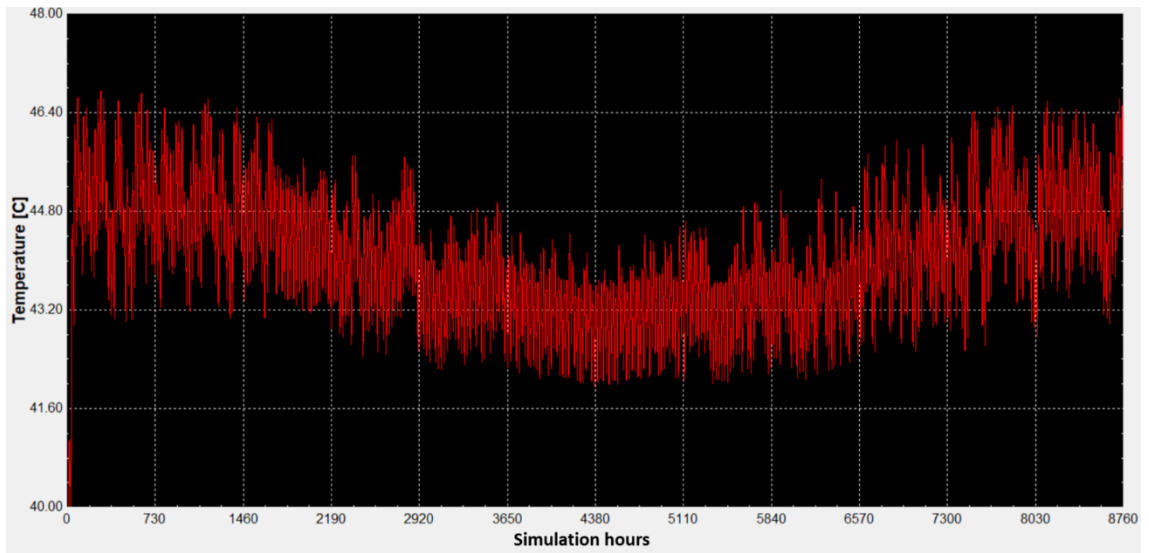


Figure 62: Temperatures reached when only the chiller and the solar collectors are used every time

Figure 63 shows the actions taken by the three agents that were trained with α_1 equal to one.

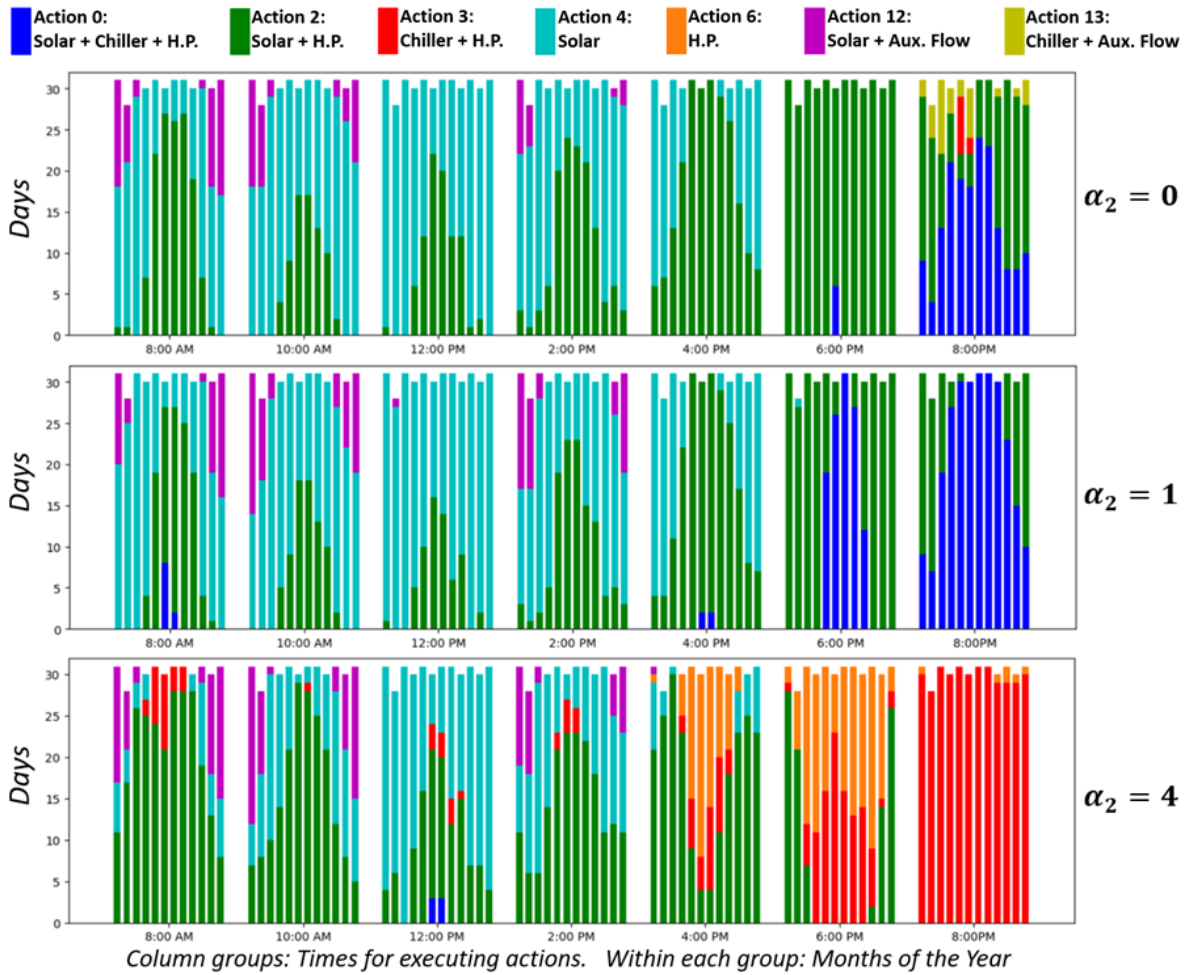


Figure 63: Actions taken by the agents trained with α_1 equal to one.

Regarding the agents shown in Figure 63, it is striking that one of them chose to use Action number 13 sometimes; this action involves using the chiller and the auxiliary flow. The auxiliary flow was

proposed as a method to decrease the “penalization” that the agent receives when the solar collectors reach excessive temperatures; however, in order for this goal to be achieved, the solar collectors need to be used along with the auxiliary flow. Besides, the “excessive temperatures” are most commonly reached during midday and not at 8.00 PM. A possible explanation for this behavior is that the agent is trying to use the auxiliary flow to increase the “total heat/electric consumption” indicator. Indeed, if the cold water flow entering the system increases, it can extract more heat, and the chiller also consumes less energy because it gives off heat to a colder water flow; both factors contribute to increase the aforementioned indicator. This is obviously an undesired behavior, and that is the reason to penalize the use of the auxiliary flow with the α_4 parameter of the reward function, so that it is only used when it is needed. Another possible explanation for the use of Action 13 is that the agent is simply acting in a non-optimal way. If the first proposed explanation is correct, an increase in the α_4 parameter would be necessary in order to avoid that behavior.

A way of partially answering that question is to repeat the same test, with the same sequence of actions, but replacing all executions of Action 13 with Action 0 or Action 2, which are the two most common actions at that time of day. This result is shown in Figure 64, where the original rewards are compared with two new reward graphs: in one of them all “Actions 13” were replaced by “Actions 2”; in the other, the same was done but using Action 0 instead of Action 13. The rewards were so similar that a close-up to some regions of the graph where the differences are larger was made. The result shows that sometimes Action 13 was better than the other two options, but not always. It has been said above that this method “partially” answers the question because there is still the possibility that another strategy (a combination of Action 0, Action 2 and maybe other actions) does always better than the original agent. However, it would be very hard to find that strategy (otherwise the use of DNNs would not be justified). An important detail is that in Figure 64, the time window to compute the moving average has been reduced to 7 days in order to better appreciate the details of the rewards.

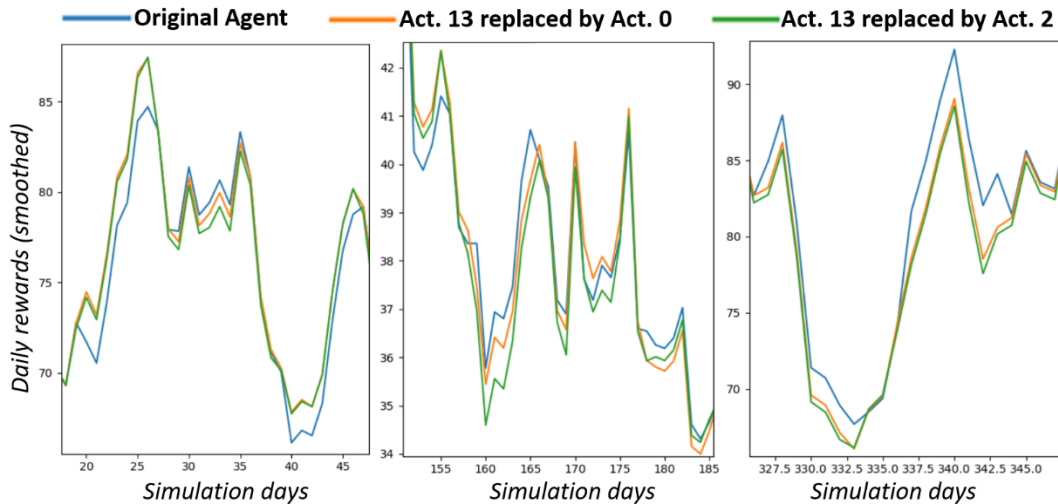


Figure 64: Agent shown at the top of Figure 63, compared with two alternative strategies

6.4.3. Changing the value of α_3

The α_3 parameter defines the importance given to the degradation factor, as expressed in Equation 107. Thus, it defines how much the agent will be penalized for excessive temperatures in the solar collectors. The values considered for α_3 in this section are: 0, 1, 3, 10 and 30. For α_1 , the values 0 and 1 are considered; hence, ten agents must be trained for this section. The other reward parameters are shown in Table 21.

Table 21: Parameters of the reward function in Section 6.4.3

Parameter	Value
α_1	$\{0, 1\}$
α_2	0.5
α_3	$\{0, 1, 3, 10, 30\}$
α_4	1

The results obtained with both values of α_1 are shown in Figure 65.

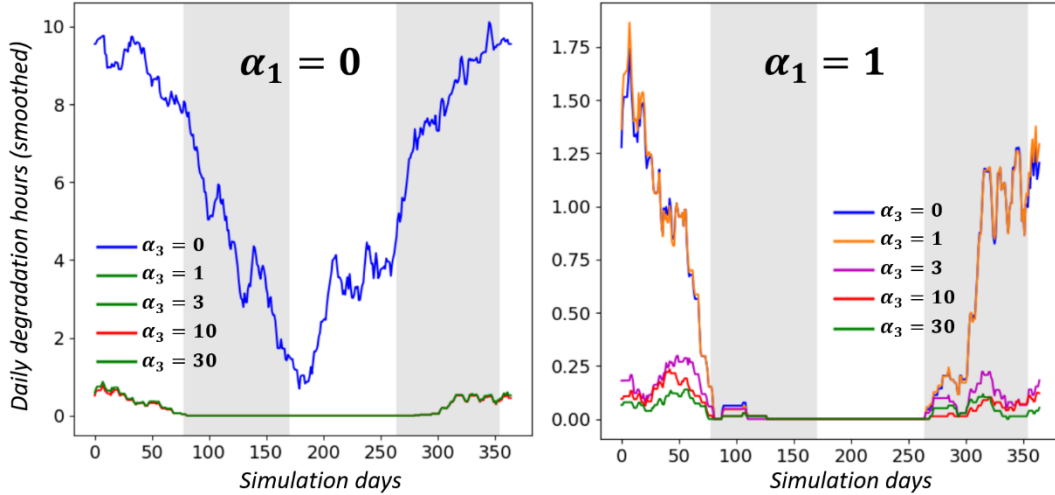


Figure 65: Daily hours with water temperature in the solar collectors higher than 100°C. Left: α_1 was set to zero. Right: α_1 was set to one.

The results obtained with α_1 equal to zero can be interpreted the following way: the blue curve shows that the collector suffered from degradation (i.e. the water in their interior reached more than 100°C) during the whole year, with more daily hours in summer. This is because the agent only used the chiller to heat the water during the whole year. Because of this, the water in the collectors was always stagnant and reached high temperatures even on days with relatively low solar radiation. The three other agents used another strategy: they used the chiller and the collectors almost in every action; only sometimes they chose to use only the collectors. The curves of the agents that were trained with α_3 equal to 1, 3 and 30 are so similar that they cannot be distinguished, even if the blue curve is taken out of the graph. That is why they are all plotted with the same color.

Regarding the results obtained with α_1 equal to one, the two lower values of α_3 (0 and 1) have quite similar results. With both of these values, the number of daily degradation hours is relatively large in comparison to the other values of α_3 . When α_3 is changed from 1 to 3, an abrupt change occurs, and the daily degradation hours decrease notoriously. With even larger values of α_3 (10 and 30), the agents seem to care even more about degradation, but the change is more subtle.

6.4.4. Effect of the α_4 parameter

As already discussed, the α_4 parameter is a penalization for using the auxiliary flow. In all previous sections, this parameter was set to 1; this means that using the auxiliary flow halves the reward. Thus, the auxiliary flow is only useful if the “gain” of using it is more than the aforementioned penalization. Why would the auxiliary flow be useful? To avoid excessive temperatures in the solar collectors, which also yield penalizations. As already discussed, penalizing the use of the auxiliary flow is necessary in order to avoid indiscriminate use of it. In Figure 66, the actions taken by two

agents that were trained with α_4 being equal to zero are shown. The other reward parameters are specified in Table 22. Both agents were trained under exactly the same conditions.

Table 22: Reward parameters for Section 6.4.4

Parameter	Value
α_1	1
α_2	0.5
α_3	5
α_4	0

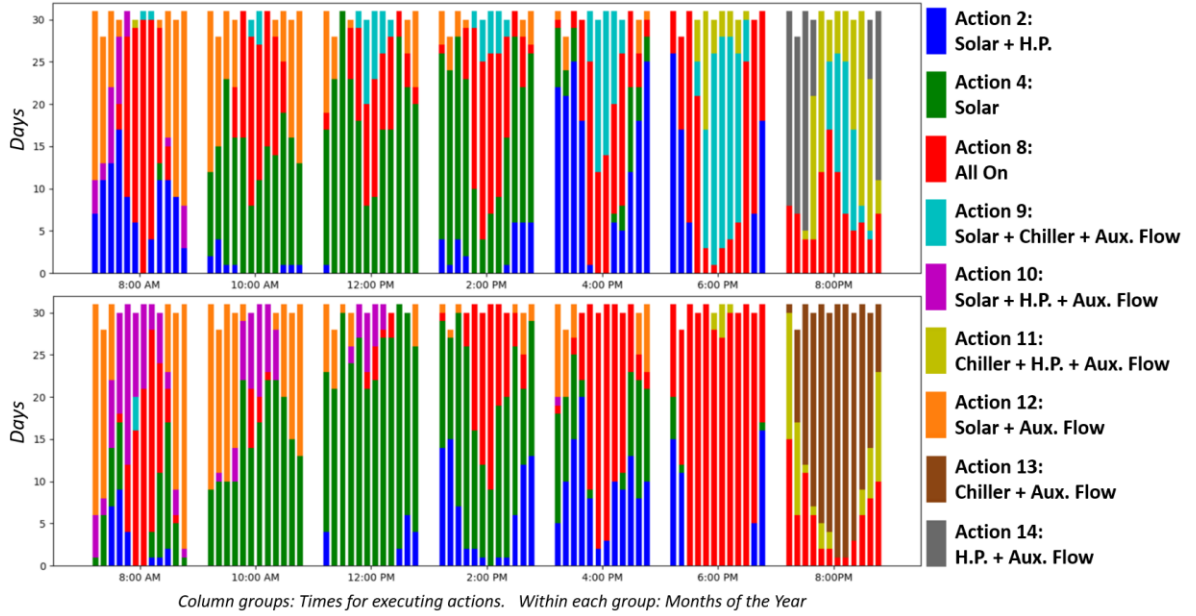


Figure 66: Actions taken by two agents that were trained with α_4 equal to zero.

From all the actions that are shown in Figure 66, only Action number 2 (blue) and Action number 4 (green) do not involve using the auxiliary flow. As can be seen, these agents used the auxiliary flow much more frequently than all previous agents whose actions have been visualized. This is because the α_1 parameter has been set to one, which means that the agents seek to increase the “total heat/electric consumption” indicator. An easy way of achieving this is by introducing the auxiliary flow, because a larger water flow inside the system has more capacity of extracting heat from it. Obviously, this is not the expected use of the auxiliary flow, and that is the reason why the α_4 parameter is necessary in order that the agents behave in a desirable way.

6.5. System subject to failures

In this part, agents are trained and tested in environments where the heating devices of the system can fail and be repaired. The main goal is to achieve that, when confronted with the failure of a component, the agent is able to manage the system in order to achieve an acceptable performance.

The reward function R_1 , as defined in Equation 110, will be used during the whole section, and the value of the α_1 parameter will be set to one. Here is why: it has been discovered in previous sections that when only the “total heat/electric consumption” indicator of the reward function is taken into account (i.e. when the α_1 parameter of Equation 110 is equal to one), the agents privilege the use of the solar field and the heat pumps to heat the water flow. This makes sense because the chiller is far more energy-consuming than the other devices. It was also discovered in Section 4.6, while the Markov chains were being formulated, that the heat pumps have far more probability of failing

than the solar energy system. Therefore, the main goal for an agent that is trained to privilege the “total heat/electric consumption” indicator (with α_1 of Equation 110 equal to one) is that it learns to replace the heat pumps with the chiller when the former fail. This is especially important during winter, because the solar energy available is not enough for the water to reach the minimum desirable temperature of 40°C. Therefore, the rewards would be reduced to zero if the behavior of the agent does not change when the heat pumps fail. The other parameters of the reward function are specified in Table 23.

Table 23: Parameters of the reward function for Section 6.5

Parameter	Value
α_1	1
α_2	0.5
α_3	5
α_4	1

Two kinds of agents are going to be trained: the first kind of agent will receive the same environment state that was defined in Section 4.4, with 10 variables. The second kind of agent will receive five extra variables that contain information about the functional states of the three heating stages of the system (from now on, the term “functional state” will be used quite often to describe the operation/failure of the devices of the system, in contrast to the “environment state” that is evaluated by the DNNs). These five extra variables work as follows:

- Two variables for the heat pump stage:
 - [1,1] if the heat pump stage is completely functional.
 - [1,0] if one of the heat pumps has failed.
 - [0,0] if the heat pump stage is out of operation.
- One variable for the chiller:
 - [1] if the chiller is working.
 - [0] if the chiller has failed.
- Two variables for the solar energy system:
 - [1,1] if the system is completely functional.
 - [1,0] if one of the pump-heat exchanger pairs has failed.
 - [0,0] if the solar energy system is out of operation.

With these extra variables, the agents can “know” what the functional state of the heating system is. The agents that do not receive this information have to “infer” it from the other variables that are given to them, like the temperatures in the storage tanks.

Given that the case presented now is more complex than the case without failures, a few changes in the training hyperparameters are made with respect to what is shown in Table 18. Now the training hyperparameters are as shown in Table 24.

Table 24: Training hyperparameters for Section 6.5

Training time and simulation time	12 years
Discount factor	0.4
Type of training (Normal DQN or Double DQN)	Double
Update period of the target network	10 days
Learning rate	0.001
Momentum factor	0.8
ϵ -greedy method	<ul style="list-style-type: none"> - The value of ϵ is equal to one at the beginning. It is maintained at this value for two years, and the network is not updated. This period is used to fill the replay memory. - After two years the value of ϵ starts to decline linearly; it reaches its minimum value of 0.2 at 12 simulation years. When the value of ϵ starts to decline, the network also begins to be updated once at the end of each day. - ϵ reaches its minimum value of 0.2 at the same time that the simulation finishes.
Length of the replay memory (in experiences)	10500
Batch size	100
Prioritized experience replay	<ul style="list-style-type: none"> - The method (Proportional/Rank-based) is going to be varied, as well as the value of α. - The value of ϕ is set to 0.2. - The value of ψ is set to 0

In summary, the changes with respect to Table 18 are:

- Now the whole simulation time is used to train the agent, with no time left at the end as a “test period”.
- The ϵ -greedy method is now carried out slower, so that the agent has more time to “explore” the actions and find the best ones.
- The Replay Memory now stores 10500 experiences, which is more than before. This number was decided so that the last four years of interaction experiences could be stored in the memory.
- The prioritizing method is going to be varied, unlike previous sections where it was constant.

In all subsections of Section 6.5, the results shown are being selected from a larger set of tested agents. This clearly produces a bias, so the results must not be interpreted as statistically accurate; they are shown only as illustrative examples of interesting results. In Annexed B, all results from which the selected results have been taken are shown.

6.5.1. Training is carried out with the Markov chains of the real system

In this part, failures will be introduced into the training process by using the same Markov chains that were defined in Section 4.6. This means that, on average, the agents will be exposed to the different functional states of each heating stage during the time percentages that are shown in Tables 5, 6 and 7. Given that the Markov chains are independent from each other, it can happen that failures of different heating stages coincide at the same moment. If this happens with the heat pumps and the chiller in winter, there is nothing that the agents can do to compensate this, because

the solar field is not able to provide enough energy to the water flow alone. Nevertheless, this will happen during a relatively low percentage of the time. Both the heat pumps and the chiller are expected to be under failure 17% of the time, so their failures are expected to coincide $(0.17)^2 = 2.89\%$ of the time.

Due to the higher level of complexity of this new environment, a new “architecture exploration” is carried out. In other words, new DNN architectures are proposed, trained and tested in the failure-subject environment. The rationale behind this is that, because of the fact that the agents have to learn to control the system under various distinct conditions, deeper DNNs may be needed in order to find good solutions. Architecture 4, as defined in Figure 39, was the most successful in previous sections, so it is tested in this section as well, and five new architectures are proposed. In addition to this, the hyperparameter α (from Prioritized Experience Replay; see Section 3.2.5.5) is given different values, in contrast to previous sections where it was always equal to 1. This is done because the main goal of this section is to train agents that are capable of handling failures, which are relatively rare events in the training process. Because of this, it could be the case that a larger prioritization of rare experiences does help the agents to find better solutions.

The new proposed architectures are numbered from eight onwards, in order not to cause confusion with the architectures defined in Figure 39. In Figure 67 the architectures that are tested in this part are shown.

Architecture 4		
Layer	Size	Activation
Input	10 or 15	-
Hidden 1	72	ReLU
Hidden 2	48	ReLU
Hidden 3	48	ReLU
Output	16	-

Architecture 8		
Layer	Size	Activation
Input	10 or 15	-
Hidden 1	72	ReLU
Hidden 2	48	ReLU
Hidden 3	48	ReLU
Hidden 4	32	ReLU
Output	16	-

Architecture 9		
Layer	Size	Activation
Input	10 or 15	-
Hidden 1	80	ReLU
Hidden 2	72	ReLU
Hidden 3	48	ReLU
Hidden 4	48	ReLU
Hidden 5	32	ReLU
Output	16	-

Architecture 10		
Layer	Size	Activation
Input	10 or 15	-
Hidden 1	80	ReLU
Hidden 2	72	ReLU
Hidden 3	48	ReLU
Hidden 4	48	ReLU
Output	16	-

Architecture 11		
Layer	Size	Activation
Input	10 or 15	-
Hidden 1	72	ReLU
Hidden 2	48	ReLU
Hidden 3	32	ReLU
Hidden 4	32	ReLU
Hidden 5	24	ReLU
Output	16	-

Architecture 12		
Layer	Size	Activation
Input	10 or 15	-
Hidden 1	72	ReLU
Hidden 2	48	ReLU
Hidden 3	40	ReLU
Hidden 4	32	ReLU
Hidden 5	32	ReLU
Hidden 6	24	ReLU
Hidden 7	20	ReLU
Output	16	-

Figure 67: Architectures considered for Section 6.5.1.

The first big discovery was that not all architectures converge to desirable results, and even the ones that do, do not always achieve it.

This is shown in Figure 68, where eight agents that were trained with the same hyperparameters are shown, and only one of them (agent number 3, represented with the green curve) has achieved the goal of handling a failure of the heat pumps. All the agents in that figure have Architecture 11, as defined in Figure 67. They are compared by their daily rewards. To understand the figure, a few clarifications have to be made: the agents have been tested in an environment where failures occur according to the Markov chains defined in Section 4.6. The moments of failure are the same for all the agents. The graph shows a time span of one year that corresponds to the second simulation

year, just as in the testing method that was defined in Section 6.2.1. The three lines at the top, with green, yellow and red sections, show the functional states of the three heating stages at each moment. Green means that the corresponding heating stage is completely functional; yellow means a degraded state (in the case of the heat pumps, one of them has failed; in the case of the solar collectors, one of the pump-heat exchanger pairs has failed); and red means that the corresponding heating stage is completely out of operation (in the case of the chiller, the only possible colors are green and red).

In the case of Figure 68, the agents were trained with an environment state of 15 variables, which means that the agents receive the functional states of the heating stages as information. The proportional method was used to prioritize experiences, with the parameter α being equal to 2. Also, the time window to smooth the rewards was reduced to 7 days; this was done to better appreciate the abrupt changes of the rewards when failures occur. The time window of 7 days will be maintained in the coming sections.

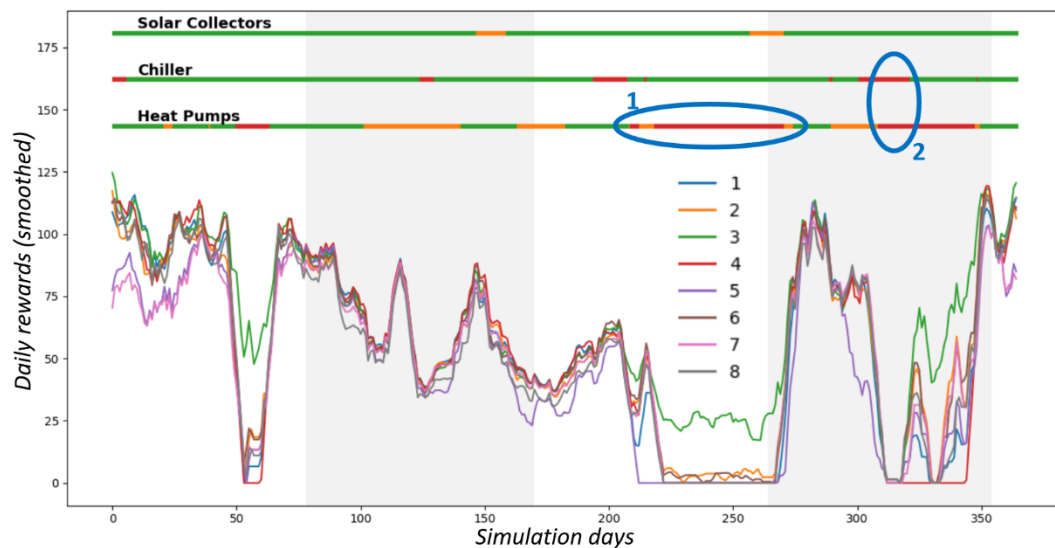


Figure 68: Eight agents trained with the same hyperparameters. Only agent number 3 has learnt to handle a failure of the heat pumps. The agents receive the functional states of the heating stages as information. A time window of 7 days was used to compute the moving average (i.e. to smooth the rewards).

In Figure 68, two interesting moments are being shown by the blue ellipses. The first moment is a long failure of the heat pumps; the rewards of most agents fall close to zero at that moment. Nevertheless, one agent (number 3; green curve) has rewards which are remarkably better. The second moment (ellipse two) is a moment at which failures of the chiller and the heat pumps coincide; this makes the rewards of all agents fall to zero, because there is no possibility of heating up the water up to 40°C. What is remarkable about Agent 3 is that it also performs quiet well in moments when the system has no failures. At the beginning and at the end of the year, it reaches rewards above 100, which is similar to what the agents got in previous sections.

Figure 69 shows the actions taken by Agent 3 of Figure 68. The graph shows that the behavior of the agent drastically changes when failures of the heat pumps occur. The agent starts to use actions 0 and 3. The former involves using all three heating stages; the latter involves using the chiller and the heat pumps. Given that the heat pumps have failed, Action 0 is equivalent to Action 1 (Solar + Chiller) and Action 3 is equivalent to Action 5 (only the Chiller is used). The fact that the agent “prefers” actions 0 and 3 over actions 1 and 5 could be because it is “expecting” the heat pumps to be repaired, so it permanently tries to activate them. However, this agent receives the functional

state of the system as information, thus it should have no trouble at finding out when the functional state of the system changes. In the case of an agent that does not receive the functional state of the system as input, this hypothesis makes complete sense. Other reason why this specific agent chooses actions 0 and 3 could be simply because it converged to that solution, and it could have found the solution of using actions 1 and 5 with equal probability.

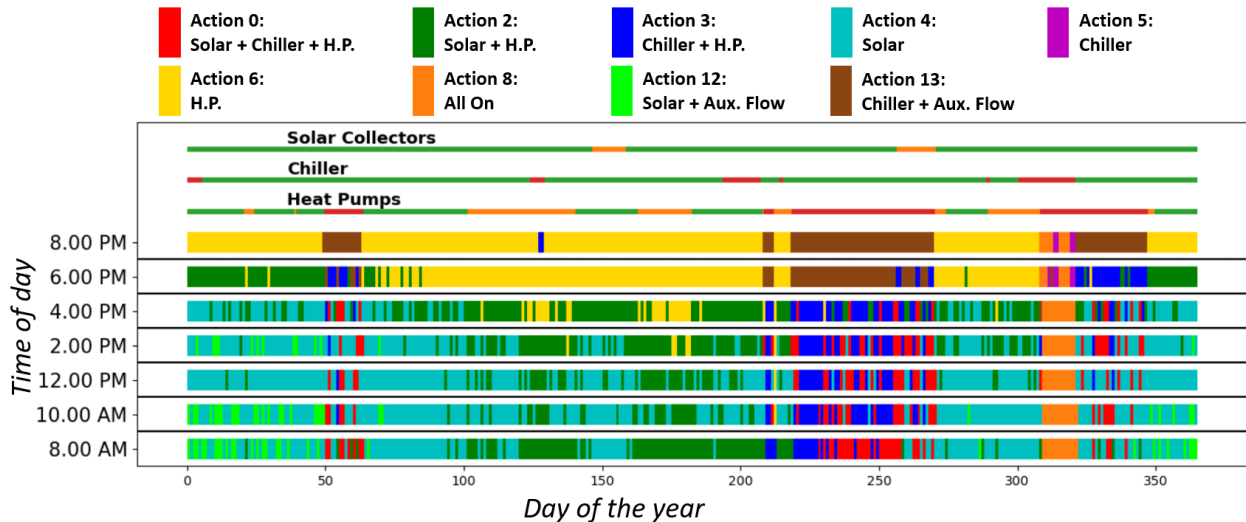


Figure 69: Actions executed by agent number 3 of Figure 68

There are two remarkable curiosities regarding the behavior of the agent in Figure 69 and the use of the auxiliary flow:

1. The agent uses Action 8 (which involves using all available systems, including the auxiliary flow) when the heat pumps and the chiller fail at the same time. At that moment, only the solar collectors are operating, and the rewards are reduced to zero; this is clearly because the water flow does not reach 40°C (see Equation 110). A question worth asking is: would the water flow reach temperatures higher than 40°C if the agent did not introduce the extra flow? If this were the case, the agent would receive larger rewards by not using the auxiliary flow, thus its behavior is not being optimal (i.e. the best possible). This would be easily explained by the fact that this combination of failures only occurs 2.89% of the time, thus the agent has no opportunities of learning to deal with it (the agent is struggling to learn to deal with a failure that occurs 17% of the time, so no wonder). Another option is that the agent would get zero rewards by using only the solar fields as well (because the temperature does not reach 40°C either), so it is using Action 8 just by accident, because no option is better. (Given that α_4 is equal to one, the reward gets cut in half if the auxiliary flow is used, but if all options yield zero reward, then using the auxiliary flow does not make it worse).
2. Another curiosity is the use of Action 13 in the late afternoon while the heat pumps have failed. That action involves using the chiller and the auxiliary flow. Given that the α_4 parameter of the reward function is equal to 1, the reward is cut in half if the auxiliary flow is used. This action is clearly not used to reduce the temperature in the solar collectors, since they are not being used. Two possible explanations are: 1. this action is simply not the best option, and the agent is acting sub-optimally; and 2. the water flow extracts more heat from the chiller, and the chiller also reduces its consumption when the auxiliary flow is used, thus the “total heat/electric consumption” indicator is increased enough so that the penalization for using the auxiliary flow is compensated.

The two questions above can be answered by comparing the rewards received by the agent just mentioned with the hypothetical rewards that it would have got if it used Action 0 instead of Action 8 and Action 5 instead of Action 13. This is shown in Figure 70. The “new strategy” (with actions 0 and 5) clearly surpasses the original agent during a small portion of the year. However, the performances are almost the same during most of the year. When both the heat pumps and the chiller fail, the rewards virtually do not increase by using Action 0; this confirms that any action would yield the same result at that moment (actually, the rewards do increase marginally by using Action 0, but the change is considered to be too small). It is also remarkable that replacing Action 13 by Action 5 barely changes the reward, although in this case, the action of taking out the auxiliary flow (if all other variables remained the same) would double the reward. This appears to show that using the auxiliary flow changes the other variables of the reward function in such a way that the penalization for the use of the flow is almost perfectly compensated.

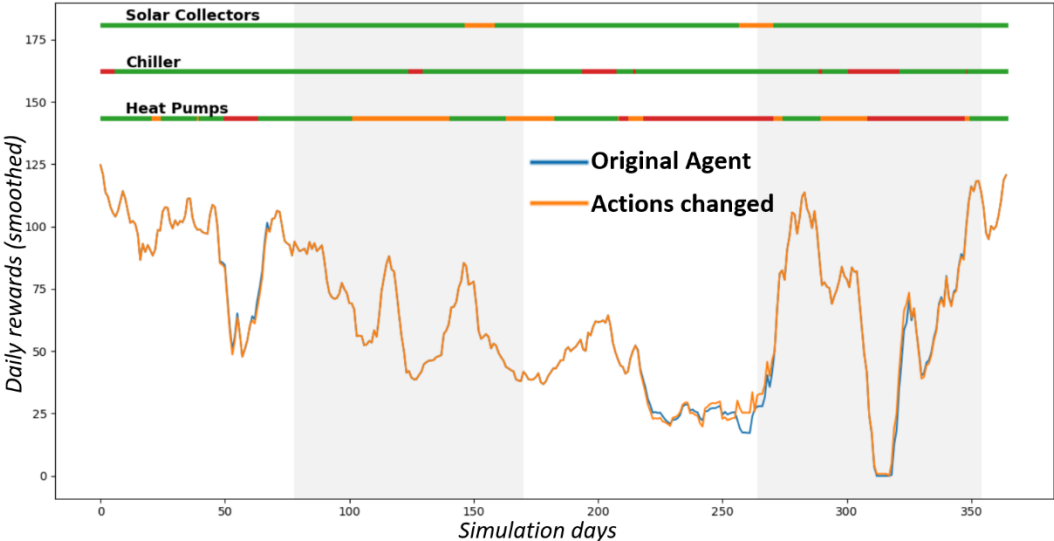


Figure 70: Original agent vs the result of replacing actions 8 and 13 with actions 0 and 5

The same sequence of failures is going to be used for future tests. At the end, some agents will be selected and tested with different moments of failure of the items.

A fairly good result with an environment state of ten variables (i.e. without telling the agent which components are working) is shown in Figure 71. Again, eight agents were trained but one of them (number 2, shown with the yellow curve) was able to manage the failures of the heat pumps without the rewards falling to zero. The architecture of the agents is the same as in Figure 68, i.e. Architecture 11, only with a difference in the size of the input. Again, the proportional prioritizing method with $\alpha = 2$ was used.

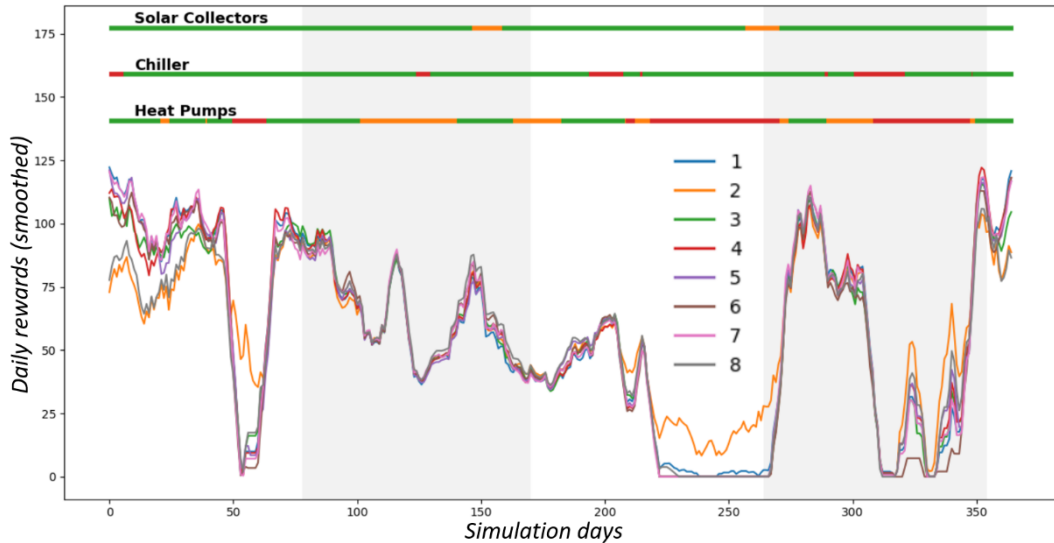


Figure 71: Eight agents with Architecture 11, with an environment state of 10 variables as input

The result shown in Figure 71 is clearly not as good as the best result shown in Figure 68. The agent with the yellow curve in Figure 71 also has smaller rewards at moments when it should not be affected by failures of the system, such as at the beginning of the year. This is not positive since the agents should be able to manage the failures without sacrificing the rewards when the system is operating well.

Regarding the other architectures, Architecture 4 (the same that was used in previous sections) was quite successful when trained with an environment state of 15 variables. 6 agents were initially trained with that architecture, but after detecting that the results with the proportional prioritizing method and $\alpha = 1$ were quite good, 6 more agents were trained with the same conditions to see if better results were achieved. The results of the 12 agents are shown in Figure 72.

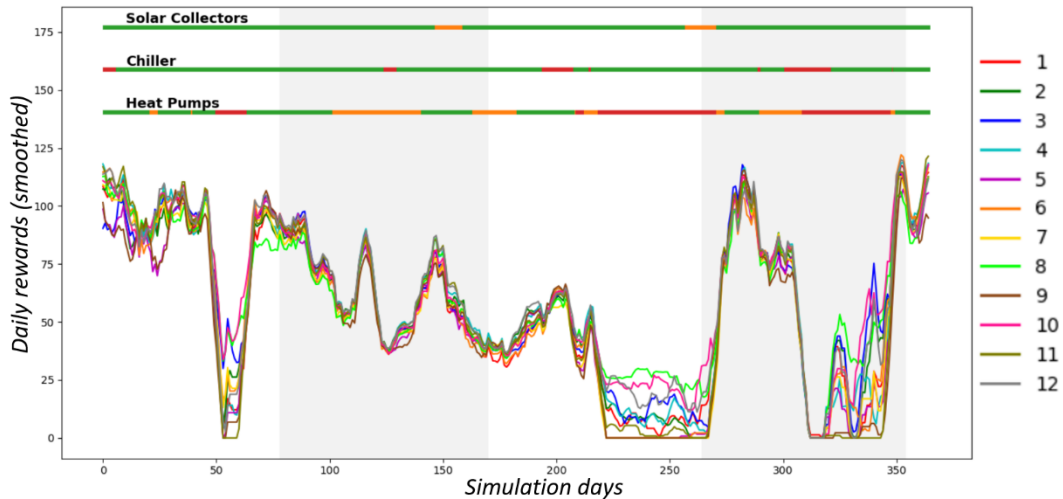


Figure 72: 12 agents with Architecture 4, trained with the proportional prioritizing method and $\alpha = 1$

Architecture 4 seems to produce more “stable” results than Architecture 11 in the following sense: in Figure 68, one agent greatly outperforms the other ones; in Figure 72, more agents have reached “intermediate” performances.

Architectures 8 and 9, as defined in Figure 67, have remarkably worse results; examples of these results are shown in Figures 73 (Architecture 8) and 74 (Architecture 9). In both images, the agents

were trained with an environment state of 15 variables (i.e. they are told the functional state of the system) and the proportional prioritization method with $\alpha = 1$. More trainings were executed with $\alpha = 2$ and with an environment state of 10 variables, but the results are very similar to those shown here.

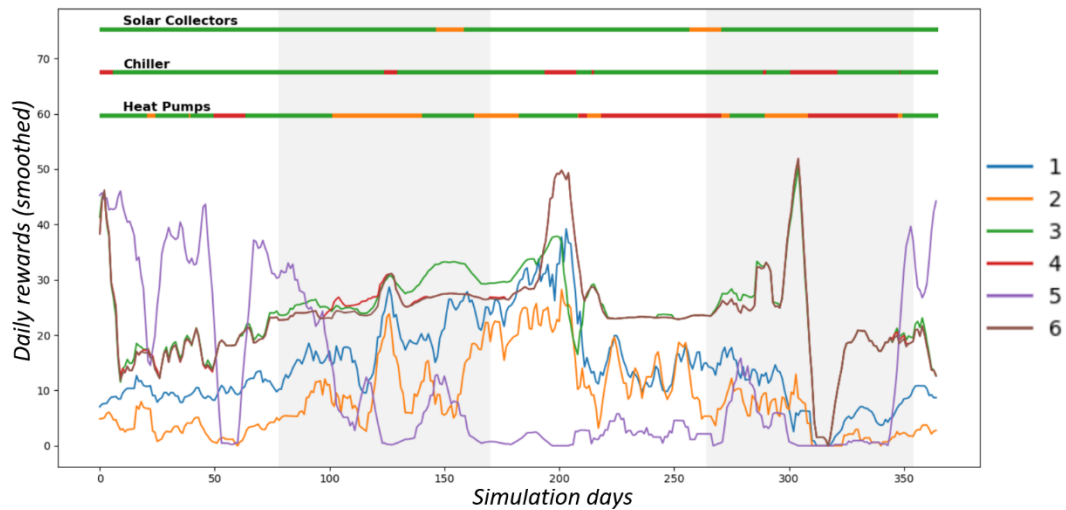


Figure 73: Six agents with Architecture 8, as defined in Figure 67

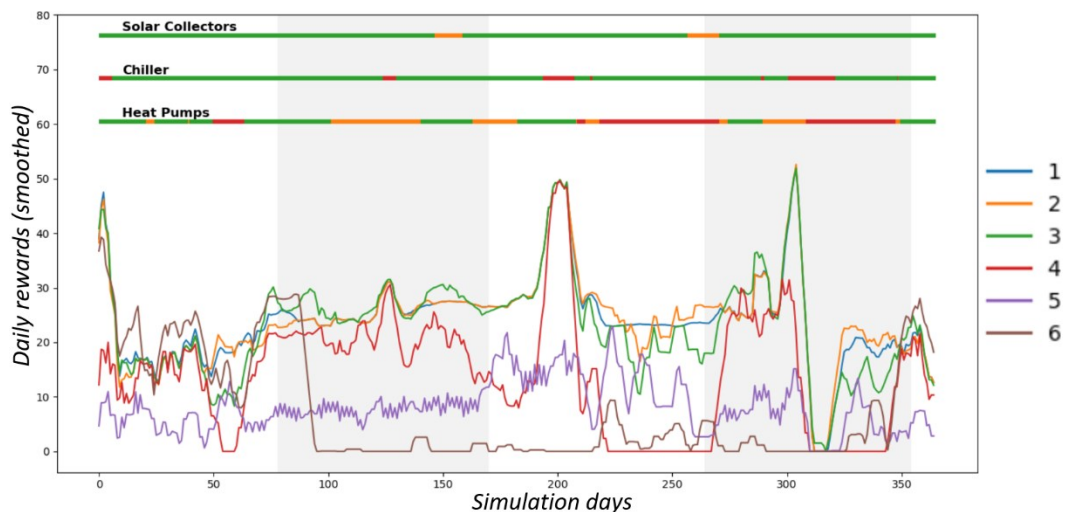


Figure 74: Six agents with Architecture 9, as defined in Figure 67

The results shown in Figures 73 and 74 can be explained as follows: some of the agents did not converge at all (agents 4, 5 and 6 in Figure 74) and others converged to a strategy that involves permanently using the chiller, which prevents them from getting larger rewards when the chiller is not needed (agents 1, 2 and 3 in Figure 74). That is the reason why, when the chiller fails, the rewards of the latter agents remarkably increase: at that moment the chiller stops consuming electricity and thus the “total heat/electric consumption” indicator of the reward function increases.

Although the agents just shown do not represent good results, they confirm the importance of the architecture of the DNN. In the case without failures, although there were differences between architectures, there was much less variability than in the case shown here (see Figures 40 and 41, where the mean rewards and the variabilities of each architecture are shown for an environment without failures).

Regarding Architectures 10 and 12, both of them achieved acceptable results as well. Figure 75 shows 6 agents trained with Architecture 10; Figure 76 does the same with Architecture 12. In the case of Figure 75, all agents were trained with proportional prioritization and $\alpha = 1$. In Figure 76, the agents were trained with proportional prioritization and $\alpha = 2$. In both cases the agents receive 15-variable environment states.

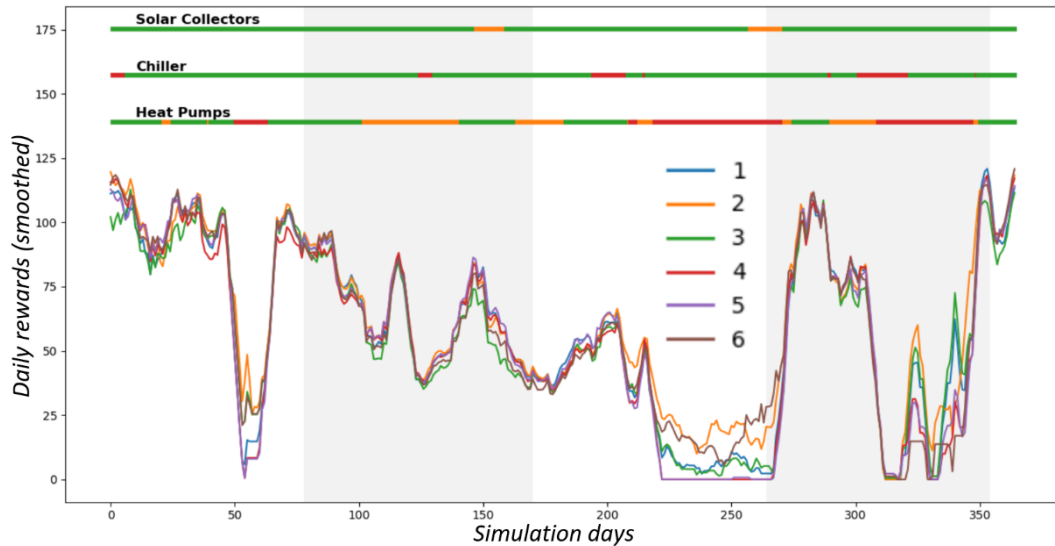


Figure 75: Six agents with Architecture 10, as defined in Figure 67

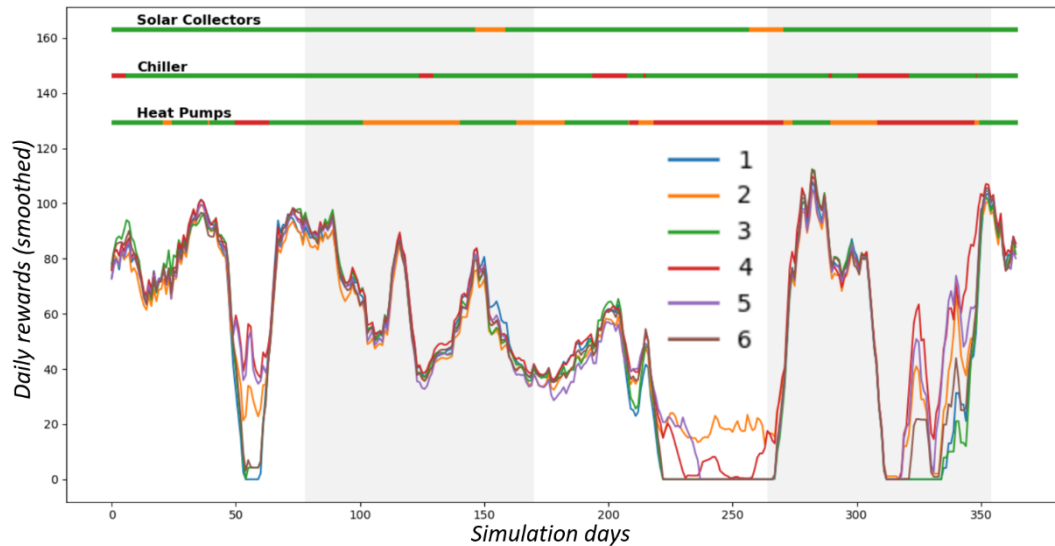


Figure 76: Six agents with Architecture 12, as defined in Figure 67

6.5.2. Training is carried out with planned failure cycles

It was discovered in Section 6.5.1 that some of the tested architectures have the potential of achieving very good results, but from many training processes carried out with the same conditions, only few of them converged to a desirable behavior.

In this section, the following changes to the training method are introduced in order to make the convergence of the training processes easier:

1. The set of possible actions is reduced by taking out actions that clearly do not make sense to use. The rationale of this is that the agents have to discover the “best actions” for the

different states of the system by executing random actions and, only by pure chance, discovering the best option after several trials. If the set of actions is reduced, it becomes easier for the agents to find the “good actions” by chance. The actions that will be taken out are all actions that involve using the auxiliary flow without using the solar collectors (actions 11, 13, 14 and 15 in Table 1) and the action that involves turning off all devices of the system (action 7 in Table 1). With this, the action set is reduced to 11 possible actions, which are shown in Table 25. The same architectures shown in 66 will be used, with the only change that their output layers will be reduced from 16 neurons to 11 (because the set of possible actions has been reduced).

2. Instead of training the agents with the same Markov chains with which the tests are carried out, the agents will be trained with failures of a *fixed* duration. This means that the state of the system during training is not stochastic anymore; however, the testing will be made with the same Markov chains as before. The cycles are shown in Figure 77. For each training process, only one of the cycles must be chosen so that the agent experiences that cycle during training. In Figure 77, the word “degradation” means that one of the heat pumps have failed or that one of the pump-heat exchanger pairs of the solar field has failed (as already discussed, the solar field itself does not fail, but the solar-heat exchanger pairs do). Cycle 1 takes into account that the agent must privilege dealing with failures of the heat pumps.

Table 25: New set of possible actions of the agent.

Action	Solar field pumps	Chiller	Heat Pumps	Auxiliary Flow
0	1	1	1	0
1	1	1	0	0
2	1	0	1	0
3	0	1	1	0
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	1	1	1	1
8	1	1	0	1
9	1	0	1	1
10	1	0	0	1

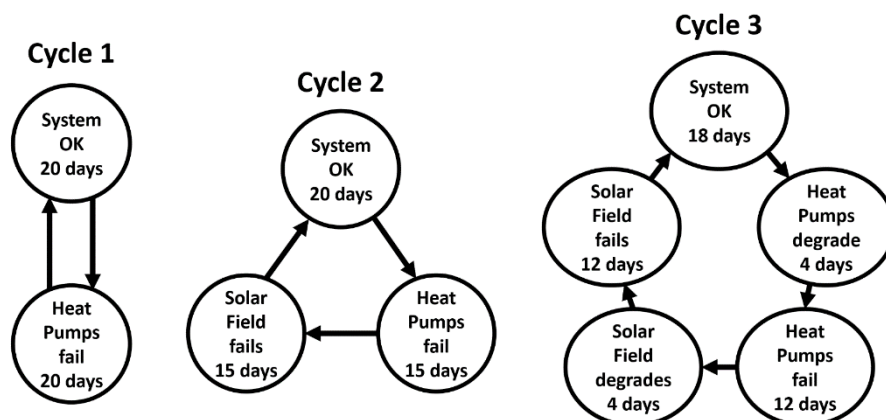


Figure 77: Cycles of failures considered

Architectures 11 and 12 were tested in this section with the proportional prioritization method and α taking the values 1 and 2. The results that are not shown here are shown in Annexed B.

The results seem to show that the cycles tend to benefit the performance of agents that receive environment states of 10 variables, because they are exposed more time to the states of the system that they have to diagnose by themselves.

However, the agents that receive 15 variables get “confused” when they experience a failure that they have never seen, or a combination of failures that did not overlap in the training process (with the cycles no overlaps are produced between failures or degradation of different heating stages) (see the definition of the 5 extra variables for the environment state in the introduction of Section 6.5). An example may be more clarifying: if the agent has always estimated the Q-Values of a given environment state by assuming that certain input variable is 1, the result could change dramatically when that input is changed to zero (this happens when a device fails). This may happen even if the device that has failed is not necessary to deliver warm water. However, this was not always the case; some agents reached good results despite the problems just mentioned.

Maybe, this problem of the agents that receive 15-variable states could be solved by using the $l1$ -regularizer or Lasso regression (see Géron [36], page 155) since this method sets the less important weights of the DNN to zero. The weights that always receive the same value of some variable during training would therefore be set to zero, and when this variable changes its value during the testing process, the predicted Q-Values would not be affected. This was not tested during this study; instead, a reformulation of the Markov processes is proposed in the next section. The solution of using the regularizer is left for future work on this topic.

Some remarkable results obtained with this “failure-cycle” method are shown in Figures 78 to 84. Table 26 shows the architecture, the failure cycle (1, 2 or 3), the number of environment state variables (10 or 15) and the experience prioritizing method used in each result shown. In all cases, the figures show 6 agents that were trained under exactly the same conditions; some of them achieved good results and others did not.

Table 26: Training conditions

Figure number	Cycle number	Environment state variables	Architecture number	Prioritizing method
78	1	10	11	Proportional prioritization with $\alpha = 1$
79	1	10	12	Proportional prioritization with $\alpha = 1$
80	1	15	11	Proportional prioritization with $\alpha = 1$
81	1	15	12	Proportional prioritization with $\alpha = 1$
82	2	15	12	Proportional prioritization with $\alpha = 1$
83	3	10	11	Proportional prioritization with $\alpha = 2$
84	3	15	11	Proportional prioritization with $\alpha = 1$

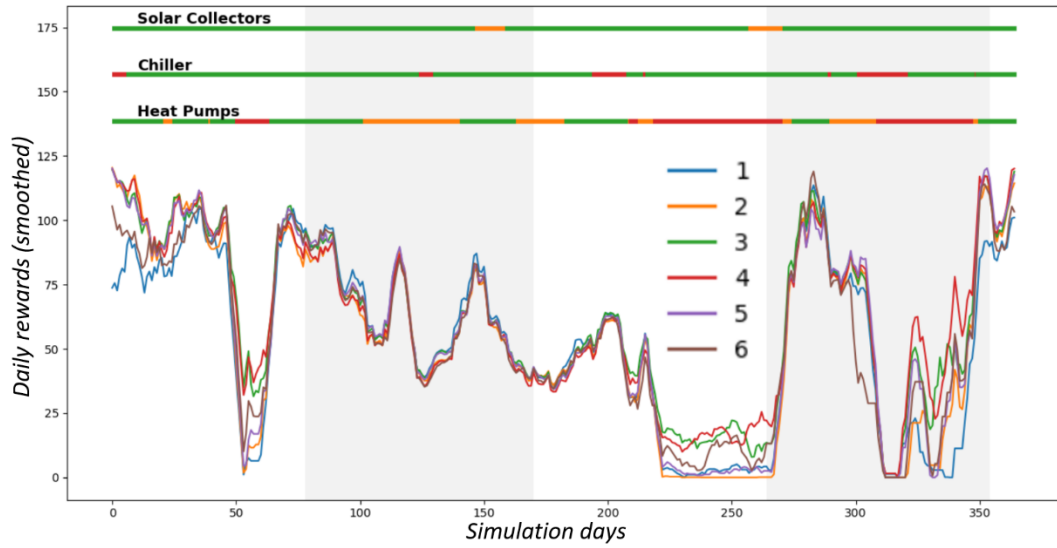


Figure 78: Arch. 11; Cycle 1; environment state has 10 variables.

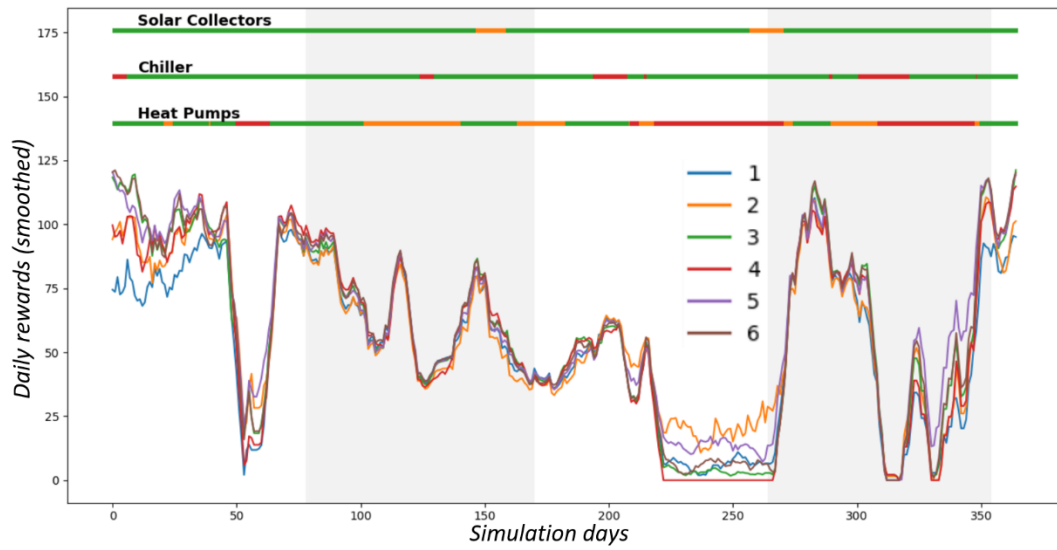


Figure 79: Arch. 12; Cycle 1; environment state has 10 variables.

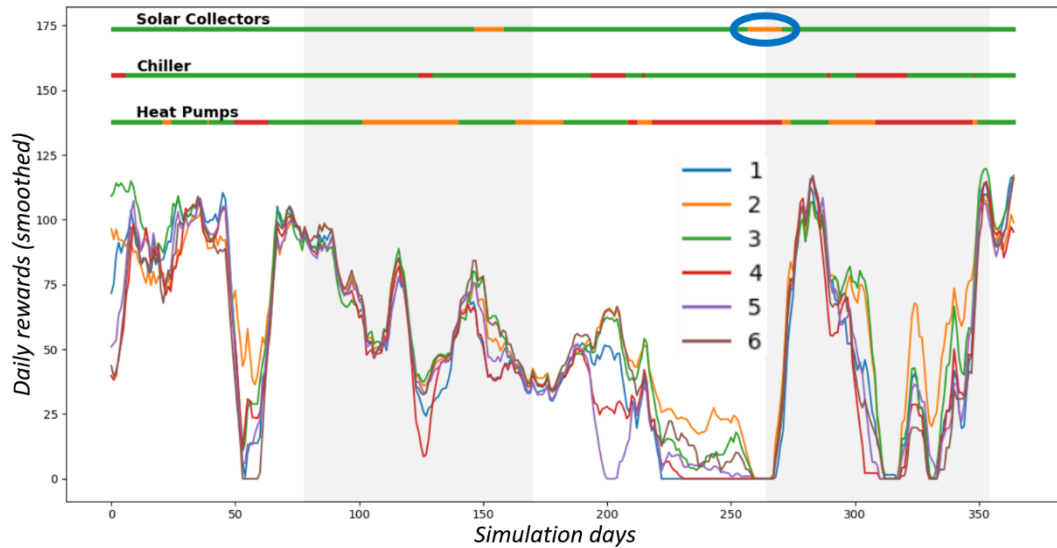


Figure 80: Arch. 11; Cycle 1; environment state has 15 variables.

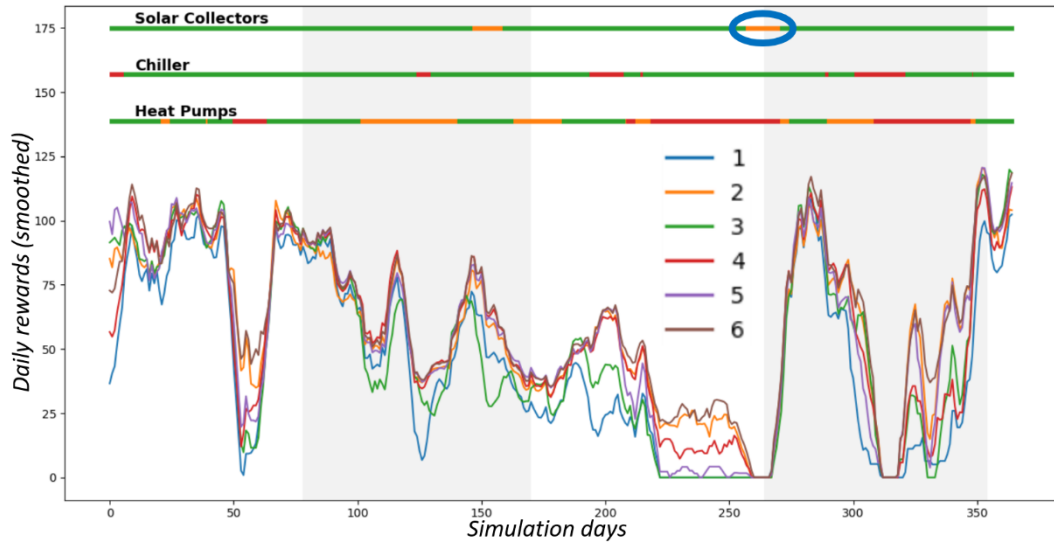


Figure 81: Arch. 12; Cycle 1; environment state has 15 variables.

Figures 78 to 81 show quite clearly what was discussed regarding the benefits for the agents that receive 10 variables and the problems experienced by the agents that receive 15 variables. All agents in the four figures were trained by using Cycle 1; i.e. the system is completely functional 50% of the time, and the heat pumps fail during the remaining 50%. The agents that receive 10 variables, shown in Figures 78 and 79, have clearly improved their performance in comparison with the results of the previous section, where from dozens of agents, only one of them seems to have learnt something about handling failures (the agent 2 in Figure 71).

On the other hand, the agents that receive 15 variables seem to be handling the failure of the heat pumps quite well until one of the pump-heat exchanger pairs of the solar field fails. That moment is highlighted with a blue ellipse in Figures 80 and 81. Figures 68 and 70 show that that moment can theoretically be handled without major problems (in the sense that it is possible to continue receiving rewards while the solar field is degraded). However, the rewards of *all* agents in Figures 80 and 81 drop to zero at that moment. This is most likely because an environment state variable that did not change during the whole training process is changing at that moment.

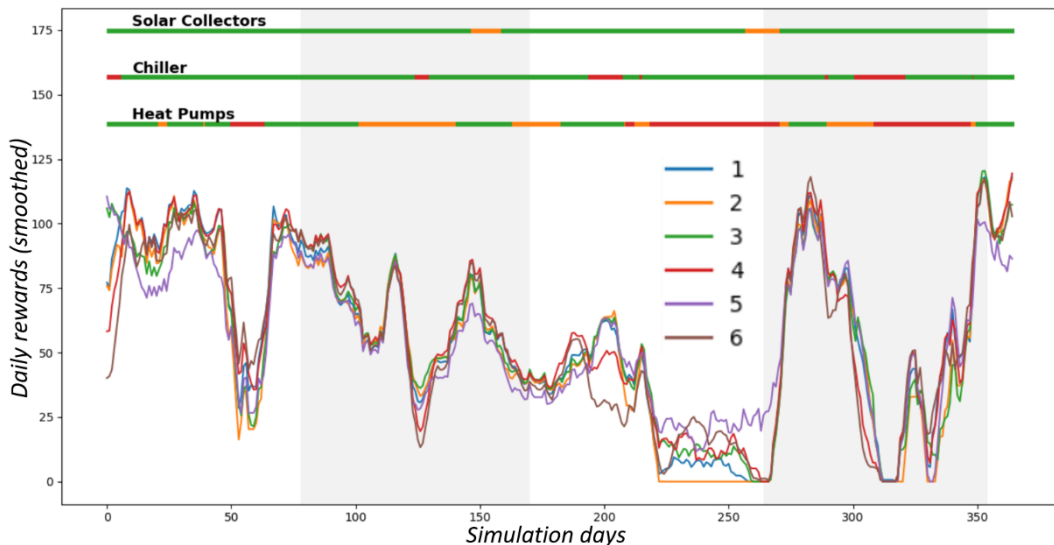


Figure 82: Arch. 12; Cycle 2; environment state has 15 variables.

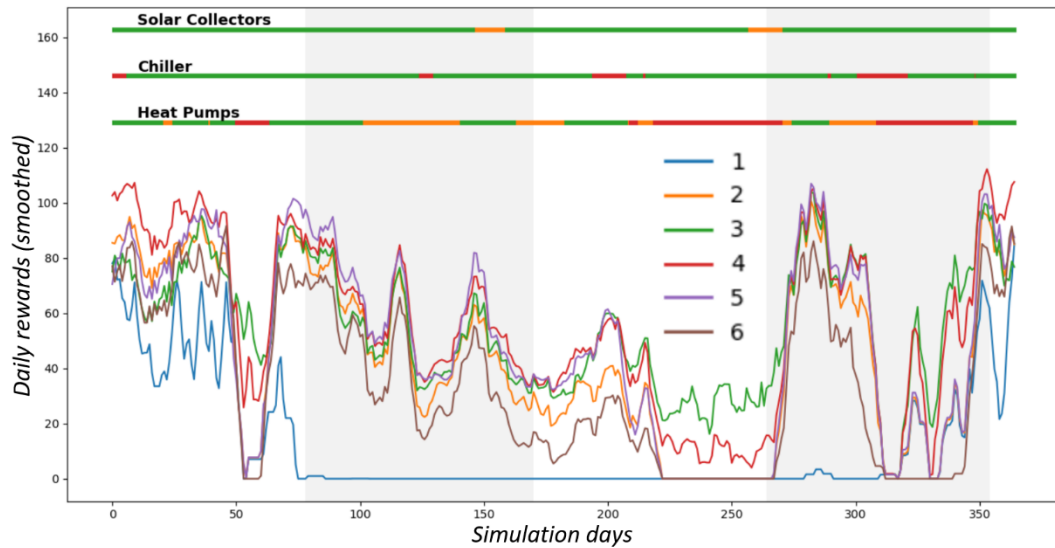


Figure 83: Arch. 11; Cycle 3; environment state has 10 variables.

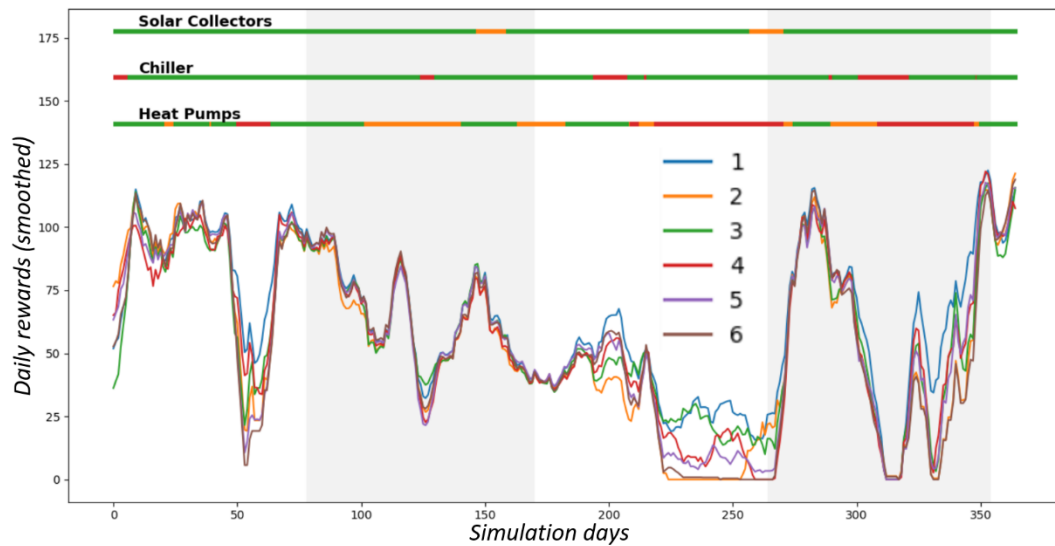


Figure 84: Arch. 11; Cycle 3; environment state has 15 variables.

Figure 82 shows that, by using Cycle 2, most agents that receive 15 variables suffer the same problem that is explained above; however, one of them (agent number 5) continues receiving rewards when the solar field degrades. This may happen because the exposition to failures of the solar field indeed does help the agent to operate under that condition, even when no combination of failures occurs during training; or it may happen because, only by chance, that particular agent does not depend on the variable that becomes zero when the solar field degrades, and it continues choosing the right actions. In the case of Figure 82, all agents were trained with proportional prioritization and $\alpha = 1$.

Figure 83 is interesting because agent number 3 receives remarkably large rewards (considering that it receives environment states of 10 variables) while the heat pumps fail; however, the rewards of that agent are not as large as they could be at other moments of the year. At the beginning of the year, for example, its daily rewards are approximately 80, while other agents receive rewards above 100 at that same time. In the same figure, agent 1 received zero rewards most of the year, which could indicate that this combination of hyperparameters yields very variable results. Maybe, this could be attributed to the value of α (2) in that figure.

In Figure 84, the agents are trained with Cycle 3; this cycle involves degradation and failures of the heat pumps and the solar field. This cycle seems to help some agents to overcome the combination of the failure of the heat pumps and the degradation of the solar field; however, the rewards of agent 4 drop to zero like in previous cases; this is probably because these two conditions (failure of the heat pumps and degradation of the solar field) never overlapped during training; hence, the agent learns to deal with the conditions separately, but not when they are combined. In that figure, the proportional prioritization is used with $\alpha = 1$.

6.5.3. Alternative Markov chains

Good and bad things can be concluded from the method proposed in the previous section (Section 6.5.2). Figures 80 and 81 show that the agents actually *do* perform better than the agents trained with Markov chains while facing the failure of the heat pumps (in the sense that more agents achieve good and intermediate results) until the degradation of the solar field occurs. At that moment, as already discussed, the previously remarkable rewards of the agents drop to zero. From this, two things can be concluded: 1. reducing the number of possible actions and increasing the amount of time that the agents face failures of the items can improve their performance; and 2. the Markov processes have the clear advantage that they automatically let combinations of failures happen.

For this reason, this section will once more make use of Markov chains for the training processes. However, the Markov chains that will be used for training will not be the same that are used for testing. For the testing process, the same Markov chains that represent the “real” system will be used (the ones created in Section 4.6.2); for the training process, other Markov chains will be proposed in order that more of the trained agents actually learn to cope with failures. The “reduced action space” of 11 possible actions will be maintained in this section.

The Markov chains that will be used in this section are shown in Figure 85. The same Markov chain is used for the heat pumps and the solar energy stage; nevertheless, the states of these two heating stages is still independent; this means that each heating stage follows its own Markov chain, although the transition probabilities of both Markov chains are identical. The arrows between the states show the transition probabilities between consecutive instants (the probabilities of staying in the same state are not shown but they can be determined as one minus the total probability of leaving the corresponding state). For the heat pumps and the solar field, three states are considered. These states are: “functional”, “degraded” and “failure”. In the case of the solar field, the “degraded” state means that a pump-heat exchanger pair has failed; in the case of the heat pumps, the “degraded” state means that one of the four heat pumps has failed. For the chiller there are only two possible states: “functional” and “failure”. The rationale behind the transition probabilities is explained below.

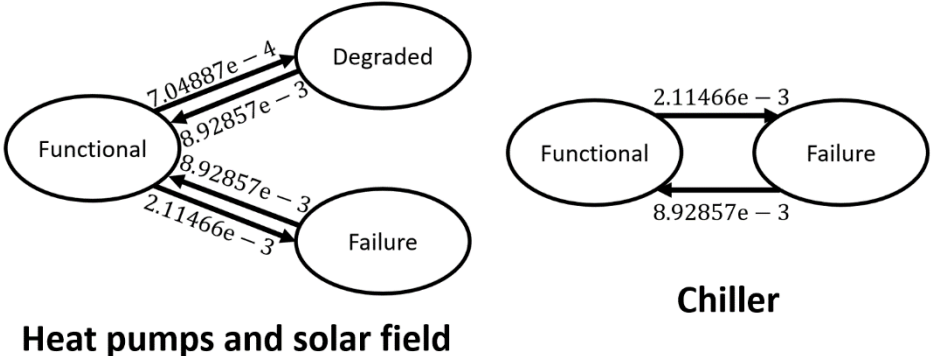


Figure 85: Markov chains used in Section 6.5.3.

In the Markov chain at the left side of Figure 85, the transition probabilities are imposed as follows: first it is imposed that the mean time of permanence in the “degraded” and “failure” states is equal to two weeks; by using the geometric distribution (Section 3.4.1), it is possible to determine that the probability of leaving those states must be equal to $1/(8 \cdot 14) = 8.92857e - 3$ (also, the fact that 8 time steps are executed each day is being taken into account as well). In the number just mentioned, “e” is used to indicate the exponent of 10 when the number is written in scientific notation; e.g. $1.5e - 3 = 1.5 \cdot 10^{-3}$.

The second assumption is that the system has to spend 18% of the time in the “failure” state and 6% of the time in the “degraded” state. By knowing the probability of getting out of those states, it is possible to determine the probabilities of entering them by imposing the ratios of their mean times of permanence. By using the method to determine the steady-state probabilities discussed in Section 3.4.2, it can be verified that the probabilities for the “functional”, “degraded” and “failure” states are 76%, 6% and 18% respectively.

For the Markov chain at the right side of Figure 85, corresponding to the chiller, the same transition probabilities between the “functional” state and the “failure” state of the Markov chain at the left are imposed. With this, the steady-state probabilities for the Markov chain at the right are 80.9% for the “functional” state and 19.1% for the “failure” state.

This section is focused on agents which receive 15-variable states, because the method proposed in Section 6.5.2 seems to have worked well with agents that receive 10-variable states. Architectures 4, 11 and 12 are tested with both proportional and ranked-based prioritization methods and several values of α . All results are included in Annexed B.

In Figures 86, 87 and 88, some remarkable results of each of the tested architectures are shown. Something worth mentioning is that all of these results come from training processes where the proportional prioritization method was used. The rank-based method was tested with α -values ranging from 0.0 to 0.8 but none of the results obtained was so good (actually, $\alpha = 0$ implies that no prioritization is used so the proportional and rank-based method are equivalent with that value of α).

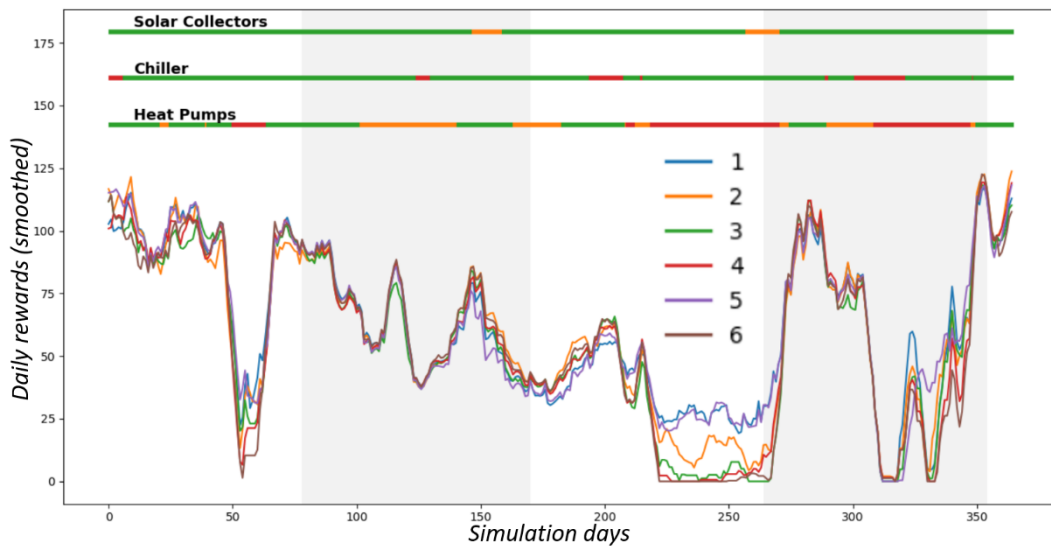


Figure 86: Architecture 4; proportional prioritization; $\alpha = 1.0$

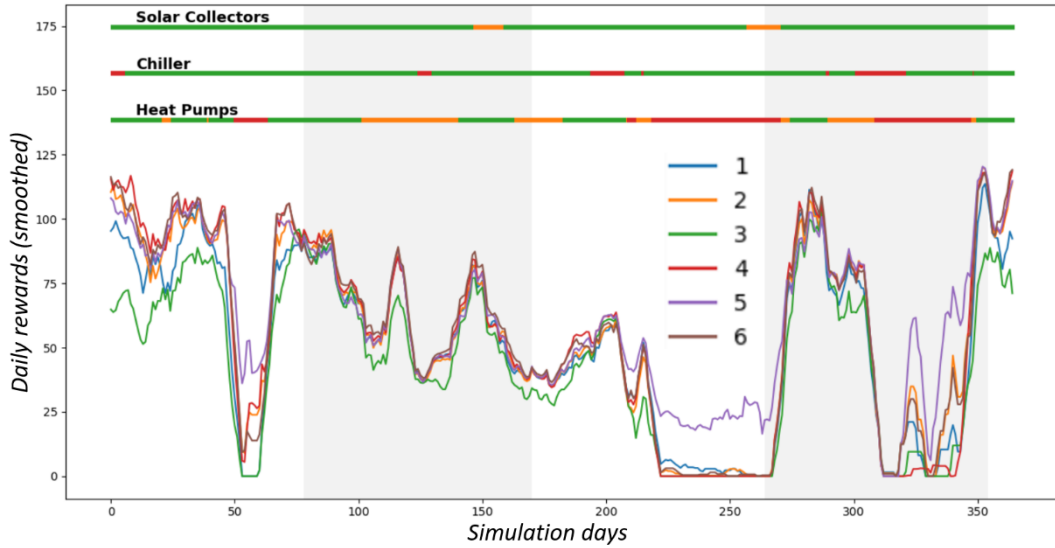


Figure 87: Architecture 11; proportional prioritization; $\alpha = 2.0$

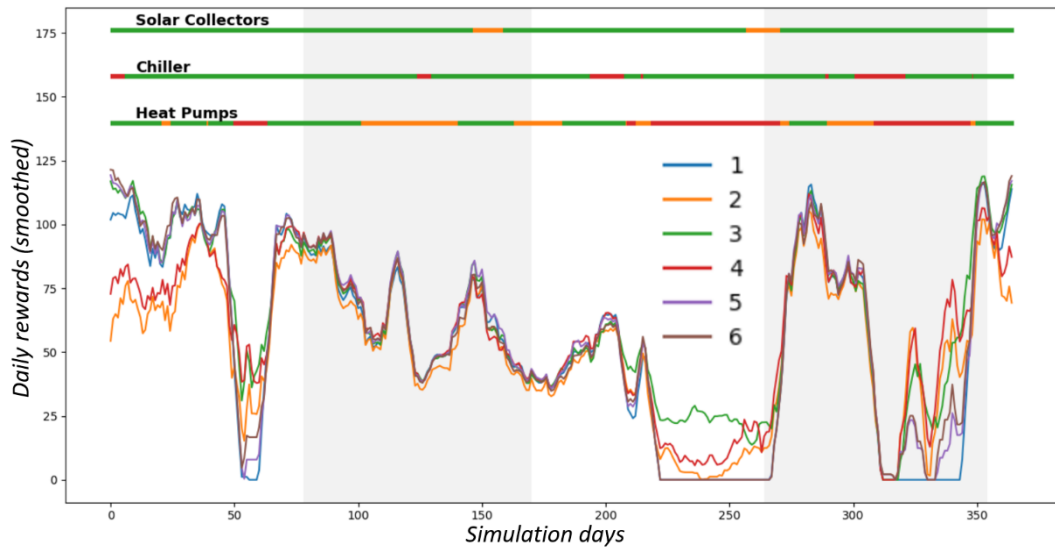


Figure 88: Architecture 12; proportional prioritization; $\alpha = 1.0$

Another important conclusion is that the rank-based prioritization method has to be implemented with lower values of α than the proportional prioritization method. As shown in this section and in previous ones, many architectures reached their best results with the proportional prioritization method and with α being equal to 2. $\alpha = 3$ was also tested, as shown in Annexed B, but with that value the agents did not converge to good solutions at all. With the rank-based method, this problem of not achieving convergence occurred with α being equal to 0.8; in other words, the same problem occurred with a remarkably lower value for the rank-based method than for the proportional method. This has a clear explanation: with a Replay Memory of 10500 experiences and the rank-based prioritization method, $\alpha = 1$ implies that the experience with the largest priority number has 10.16% probability of being selected (see Equations 57 and 58 in Section 3.2.5.5); $\alpha = 1.5$ implies that the experience with the largest priority number has 38.57% probability of being selected; and $\alpha = 2$ implies that the experience with the largest priority number has 60.80% probability of being selected. In the latter case, the probability of selecting any of the last 9500 experiences in the ranking is equal to $5.4976e - 4$; i.e. 1000 from the 10500 experiences of the Replay Memory (9.524%) are being used 99.945% of the time. Something that could compensate this huge inequality of probabilities is the fact that the order in the ranking varies in time; in this way, an

experience that is used many times to train the network will be lower in the ranking in the future. Nevertheless, it is to be expected that this “compensation” does not work for arbitrarily large values of α , because as α approaches infinity, the algorithm will tend to select a single experience for each training step.

Figure 89 shows the results of 6 agents with Architecture 12 that were trained with rank-based prioritization and $\alpha = 0.8$. This is an example of the effect of the α parameter explained above. Four of the six agents are shown with the same color because their curves cover each other, because they obtained exactly the same rewards.

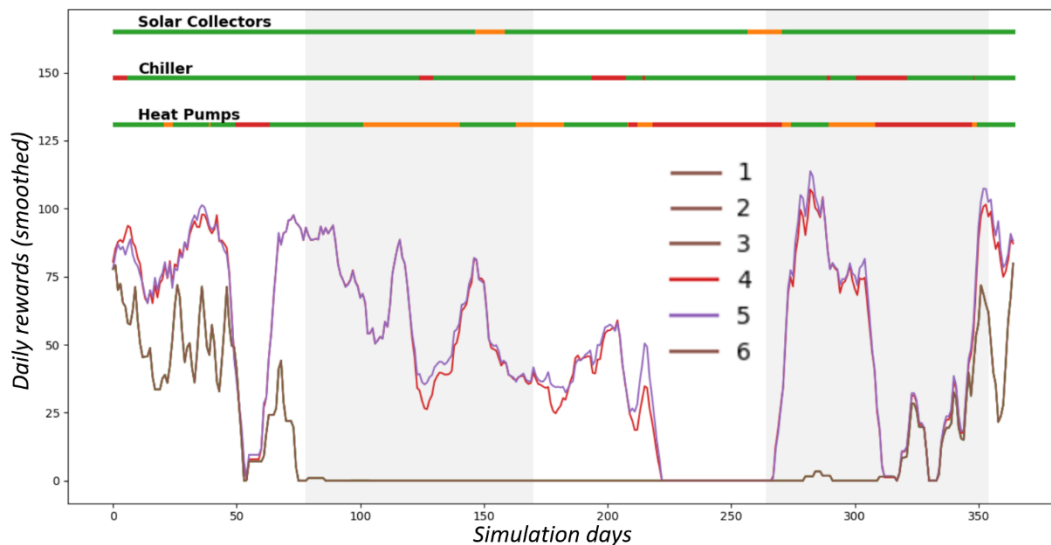


Figure 89: Six agents with Architecture 12, rank-based prioritization and $\alpha = 0.8$

6.5.4. Effect of momentum for failure-subject agents

It was shown in Section 6.3.4 that momentum (see Section 3.1.2 for details on the algorithm) has an important role to play in the training process of the agents that are not subject to failures. In this section, a possible improvement of the training hyperparameters regarding the use of momentum will be discussed, in this case for the agents that are trained to cope with failures of the system.

All the previous results of Section 6.5 and its subsections were produced with the momentum factor (β) being equal to 0.8. Figure 90 shows the results of six agents with Architecture 12 (as defined in Figure 67) when the momentum factor was increased from 0.8 to 0.9. The agents in shown Figure 90 were trained with the conditions of Section 6.5.1; i.e. they were trained with the Markov chains of the real system (as defined in Section 4.6) and have an action space of 16 possible actions, unlike the agents of Sections 6.5.2 and 6.5.3 which have an action space of 11 actions. Moreover, the agents in Figure 90 receive 15-variable environment states (i.e. they receive the functional state of the system as information) and were trained with the proportional prioritization method and $\alpha = 1$.

One of the agents in Figure 90 (Agent 5) performs quite well; in fact, it performs better than all the previous results of the same architecture (Arch. 12). Why is this so important? Because in previous sections a selection of the best results was made after multiple training processes; in the case shown here, only twelve agents were trained with β being equal to 0.9: six agents were trained with $\alpha = 1$ and six more were trained with $\alpha = 2$. The results with $\alpha = 2$ are not good, as shown in Annexed B. The fact that an agent outperforms all previous results of the same architecture with so few

attempts could be showing that the new value of β considerably increases the probability of getting good results.

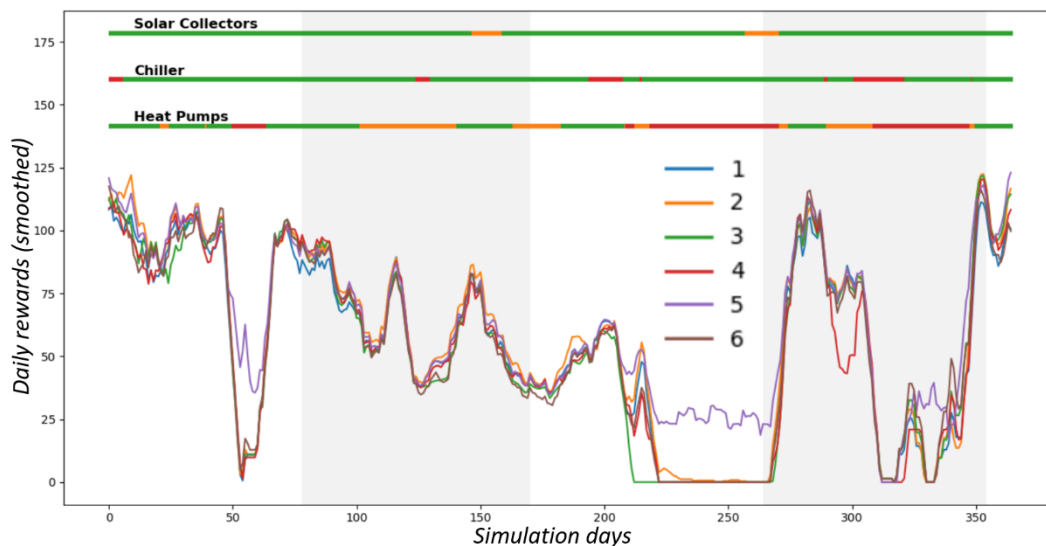


Figure 90: Architecture 12 being trained with momentum factor equal to 0.9

To support the claim made above that Agent 5 of Figure 90 outperforms all previous agents with the same architecture, Table 27 makes a comparison with agents shown in previous figures considering two indicators: the mean smoothed rewards of the whole year (i.e. averaging all values of the smoothed reward curves) and the number of days of the year that each agent has the maximum smoothed daily rewards among the four agents considered for the comparison. Agent 5 of Figure 90 reaches the maximum value for both indicators.

Table 27: Comparison of the best results of Architecture 12

Agent	Mean smoothed daily rewards	Number of days of the year having the maximum smoothed daily rewards among the four agents
Agent 2 of Figure 76	53.19	9
Agent 5 of Figure 82	55.21	33
Agent 3 of Figure 88	62.00	151
Agent 5 of Figure 90	62.37	172

After obtaining the aforementioned results, Architecture 12 was maintained and the β hyperparameter was set to 0.9 while using the method presented in Section 6.5.3, i.e. training the agent with the Markov chains shown in Figure 85 and with an action space of 11 actions. The proportional prioritization method was used with α taking the values 0.5, 0.8 and 1.0. Figure 91 shows the six agents that were trained with $\alpha = 0.8$.

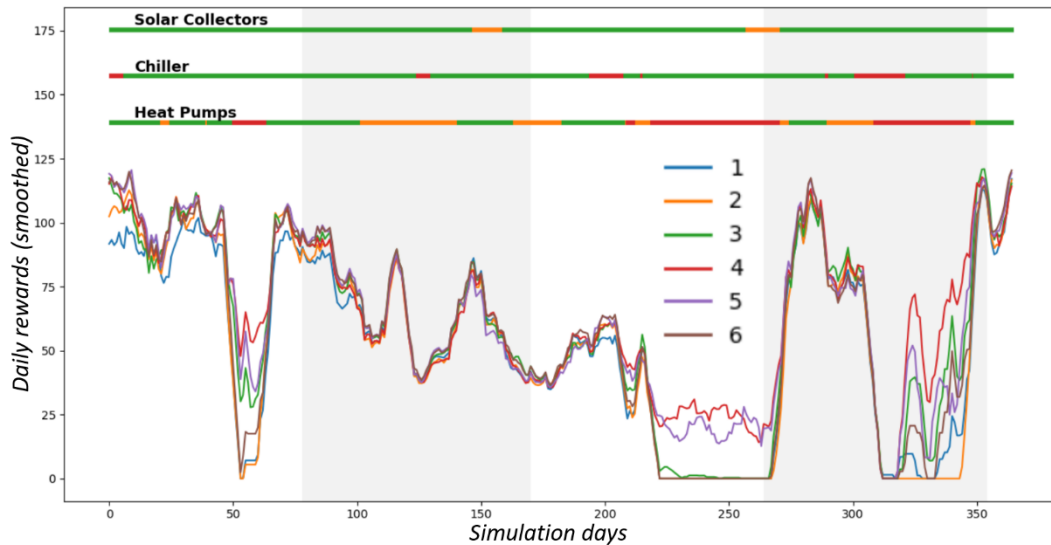


Figure 91: Arch.12; $\beta = 0.9$; $\alpha = 0.8$

6.5.5. Agent selection and final testing

In this section, agents which performed well in the previous subsections of Section 6.5 are selected and tested in a new sequence of failures. This is important because all agents that have been trained to handle failures have been tested with the same failure sequence; this can obviously create a bias towards selecting agents that are good for that particular failure sequence, but they might perform poorly in other conditions.

Table 28 shows the agents that were selected to be tested in this section. The first column of the table shows the new number by which each agent will be referred to from now on; the second and the third column show the figure that originally shows the performance of the agent and the number of the agent in that figure, respectively (recall that each figure shows several agents that have been trained under the same conditions but they usually have quite different performances). The fourth column shows the method by which the agent has been trained. “Real” refers to the method presented in Section 6.5.1, where the agents are trained with the Markov chains of the real system; “Cycle” refers to the method presented in Section 6.5.2, where that agents are trained with planned failure cycles; and “Alter.” refers to the method presented in Section 6.5.3, where the agents are trained with alternative Markov chains. The fifth column shows the number of possible actions that the agent can choose; these “action spaces” can have either 16 or 11 actions. The sixth column shows the architecture of the agent; the seventh column shows the number of state variables that the agent receives; the eighth column shows the value of α from the experience prioritization method (all selected agents were trained with the proportional prioritization method) and the ninth column shows the value of the momentum factor. For most agents, the momentum factor is 0.8, except for the last two that were trained with momentum factor equal to 0.9.

Table 28: Selected agents

New Agent Number	Figure Number	Agent Number in the Figure	Training Regime	Action space size	Arch. Number	State Variables	Value of α	Momentum Factor
1	68	3	Real	16	11	15	2	0.8
2	72	8	Real	16	4	15	1	0.8
3	75	2	Real	16	10	15	1	0.8
4	82	5	Cycle 2	11	12	15	1	0.8
5	83	3	Cycle 3	11	11	10	2	0.8
6	84	1	Cycle 3	11	11	15	1	0.8
7	86	1	Alter.	11	4	15	1	0.8
8	86	5	Alter.	11	4	15	1	0.8
9	87	5	Alter.	11	11	15	2	0.8
10	88	3	Alter.	11	12	15	1	0.8
11	90	5	Real	16	12	15	1	0.9
12	91	4	Alter.	11	12	15	0.8	0.9

In order to test the agents in the most objective way possible, a 12-year-long test will be established with the Markov chains of the real system, i.e. the Markov chains that are shown in Figures 22, 23 and 24. “Establishing a test” means that the sequence of failures will be produced before testing the agents, and then all agents will be tested under the same failure sequence. The duration of the test is increased to 12 years so that the percentage of the time that the system spends in each state of the Markov chains approaches the expected percentages shown in Tables 5, 6 and 7. In this way, there is no “bias” towards selecting a particular failure sequence because it looks “convenient” for the agents.

Table 29 shows the 12 agents ranked according to two indicators: their rewards and the daily hours during which they supplied warm water. Both indicators have been smoothed with a moving average of 7 days and then the values of the smoothed indicators have been averaged considering the whole 12-year-long test.

Table 29: Agents ranked according to the results of the whole 12-year-long test

Ranking	Smoothed rewards		Smoothed daily hours with $T_{hot} > 40^{\circ}C$	
	Agent	Mean Value	Agent	Mean Value
1	Agent 1	72.322	Agent 2	13.247
2	Agent 12	72.316	Agent 12	13.231
3	Agent 10	71.667	Agent 1	13.201
4	Agent 6	71.596	Agent 5	13.128
5	Agent 11	71.521	Agent 10	12.965
6	Agent 3	70.500	Agent 11	12.833
7	Agent 8	69.316	Agent 9	12.803
8	Agent 2	69.210	Agent 7	12.800
9	Agent 9	68.841	Agent 8	12.730
10	Agent 7	68.662	Agent 6	12.657
11	Agent 4	62.645	Agent 4	12.465
12	Agent 5	60.477	Agent 3	12.388

The results shown in Table 29 correspond to the average results of the 12-year testing process. Apart from this, it is useful to analyze particular moments when the differences between the performances of distinct agents become more appreciable; this normally happens at moments when failures of the devices of the system occur. As an example, Figure 92 shows a time-span during which the capacity of the agents to deliver hot water is not largely affected by failures. On the other hand, Figure 93 shows a shorter time-span when the “warm-water-supply-time” of several agents is greatly affected by a failure of the heat pumps. In that figure, only some of the agents, which were selected according to the results presented in Table 30, are shown.

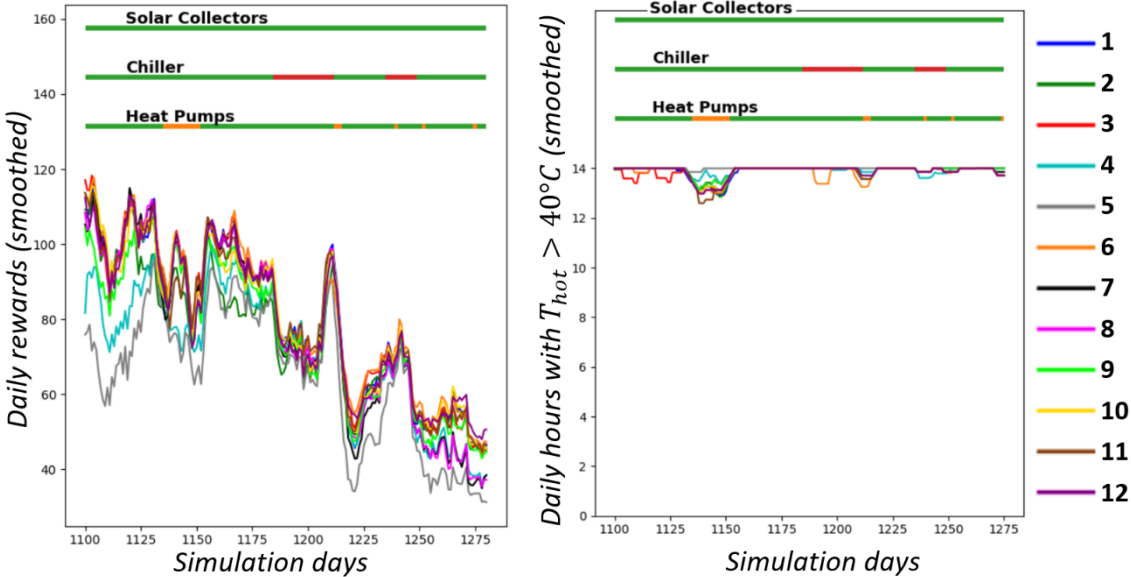


Figure 92: A period without major problems due to failures

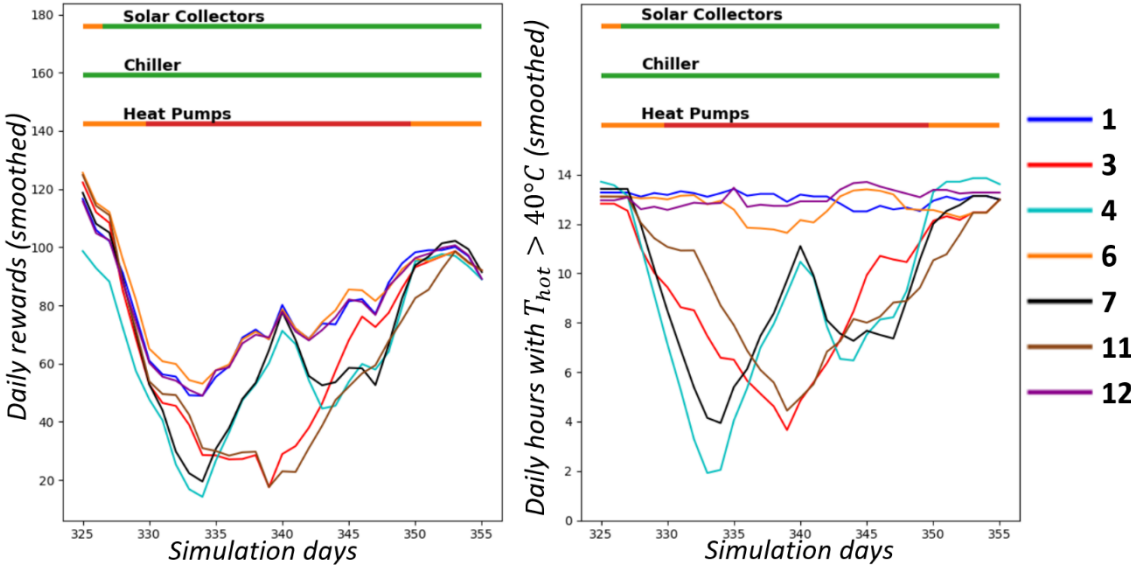


Figure 93: A moment that shows a drop of the rewards of some agents.

Table 30 has the same format as Table 29, but it only considers the time-lapse from day 325 to day 355. The agents presented in Figure 93 were selected because they were among the best three or the worst three according to some of the indicators shown in Table 30.

Table 30: Agent comparison from day 325 to day 355

Ranking	Smoothed rewards		Smoothed daily hours with $T_{hot} > 40^{\circ}C$	
	Agent	Mean Value	Agent	Mean Value
1	Agent 6	81.44	Agent 12	13.06
2	Agent 1	79.35	Agent 1	13.04
3	Agent 12	78.73	Agent 6	12.70
4	Agent 5	74.69	Agent 2	12.29
5	Agent 8	73.56	Agent 5	12.18
6	Agent 9	72.52	Agent 8	11.62
7	Agent 7	67.93	Agent 9	10.58
8	Agent 10	67.34	Agent 10	10.41
9	Agent 2	66.56	Agent 11	9.42
10	Agent 3	64.22	Agent 7	9.38
11	Agent 4	62.69	Agent 3	9.21
12	Agent 11	60.99	Agent 4	9.08

In Figure 93 and in Table 30, the three agents that have the largest rewards are also the ones that supplied warm water most time, but they are not ranked in the same order. This is not surprising because the agents can privilege other factors that contribute to the reward function, thus decreasing the time of warm water supply while actually increasing the reward function. This is not a desirable behavior, but in order to fix it, the reward function would have to be modified. The new reward function must depend on the energy efficiency as well; otherwise, the agents would make no efforts to reduce the energy consumption which is one of the important goals of the study. What is important about the reward function is that there must be no way for the agents to increase it by reducing the time they supply warm water. In other words: the reward function must be such that the agent which has the largest rewards also supplies warm water more time.

Table 31 shows the results of another time-span of the same 12-year-long test, from day 2500 to 2580. In that case, Agent 5 presents a more extreme case of the problem mentioned above about the reward function: it is the best agent when considering the mean daily hours with warm water supply, but it is also the worst one according to the rewards. From a strict “Reinforcement Learning” point of view, this would mean that Agent 5 is the worst of all, but common sense would say that it is not actually that bad because it has the maximum mean value of the most important indicator of the reward function. This is telling that, although the reward function has worked acceptably well in order to train the agents, it can be improved. This is confirmed by Figure 94, which shows the (smoothed) rewards and daily hours with warm water supply of the same time-lapse presented in Table 31, for Agents 1, 2, 5, and 7 (because they are the best according to Table 31). Agent 5 is clearly the best at the right and the worst at the left. Another thing worth mentioning about Agent 5 is that it is the only “selected agent” (from the list shown in Table 28) which receives 10-variable environment states; i.e. it does not receive the functional state of the system as direct information. It was trained with “Cycle 3” as defined in Section 6.5.2, where planned failure cycles are used (see Table 28 for details); this shows that the cycles *do* improve the performance of agents that receive 10-variable states (though only in comparison with other agents that receive 10 variables, because in comparison with the other selected agents, the result is only good from a subjective, not RL-centered point of view).

Table 31: Agent comparison from day 2500 to day 2580

Ranking	Smoothed rewards		Smoothed daily hours with $T_{hot} > 40^{\circ}C$	
	Agent	Mean Value	Agent	Mean Value
1	Agent 2	100.66	Agent 5	13.83
2	Agent 1	100.38	Agent 7	13.52
3	Agent 10	100.01	Agent 4	13.45
4	Agent 7	99.85	Agent 8	13.42
5	Agent 11	99.55	Agent 1	13.41
6	Agent 12	99.41	Agent 2	13.39
7	Agent 8	98.63	Agent 9	13.35
8	Agent 6	98.18	Agent 12	13.34
9	Agent 3	97.80	Agent 10	12.97
10	Agent 9	96.90	Agent 11	12.88
11	Agent 4	86.39	Agent 6	12.81
12	Agent 5	80.02	Agent 3	12.78

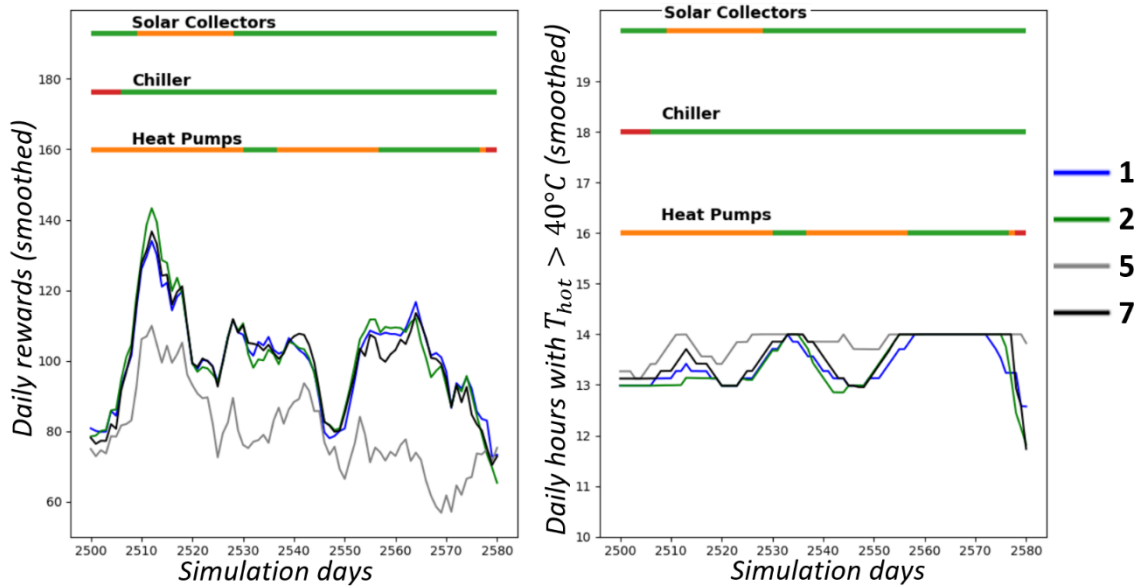


Figure 94: Performance of 4 agents (best two of both indicators) in the time-span shown in Table 31

Another interesting moment is presented in Figure 95, when a failure of the heat pumps overlaps with a degradation of the solar energy system. In the time-span presented, from day 385 to day 410, it was clearly agent 10 that supplied warm water more time than the rest. It also has the maximum rewards some of the time, yet not all of it. In this case Agent 1 draws attention because in the previous comparisons it was always one of the best agents; in this case, both its rewards and its “warm-water-supply-time” are remarkably low. Recall that the agents receive the functional states of the heating stages as five extra variables in the environment state (except for agent 5), thus, as already discussed, combinations of these variables that the agents did not experience enough during training could make the agents “get confused”. This could be what is happening to Agent 1. Agent 5, which does not receive the functional states of the heating stages, is affected by the combination of failures as well, yet not as much as Agent 1.

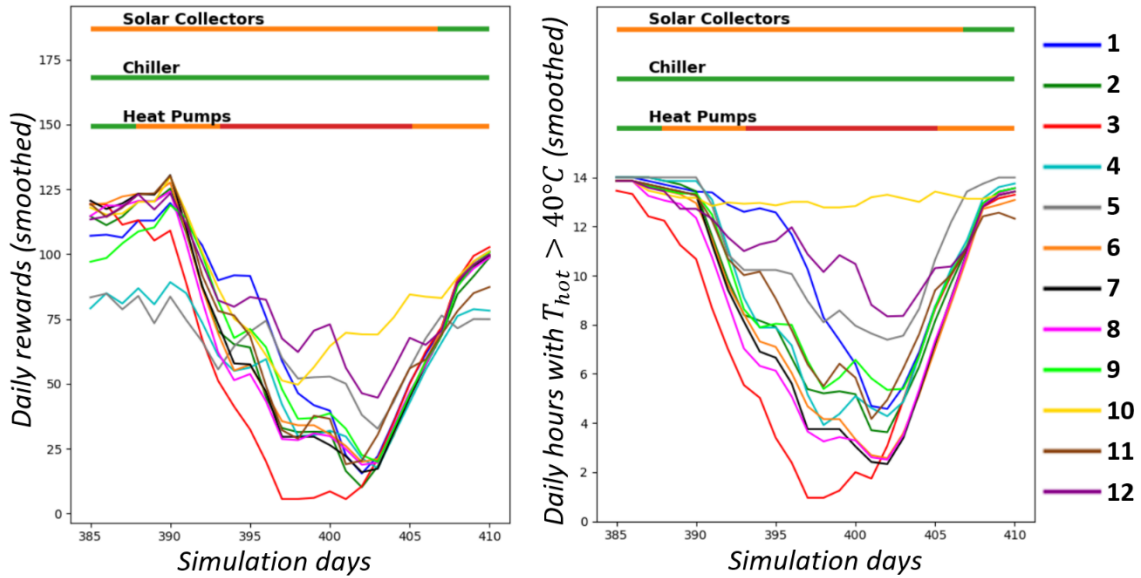


Figure 95: Rewards and warm water supply time from day 385 to day 410

Another “bad moment” for Agent 1 is shown in Figure 96. This moment is interesting because it shows something that rarely happens according to the Markov chains shown in Figures 22, 23 and 24 and in Tables 5, 6 and 7: a complete failure of the solar energy system. At that moment, Agent 1 is one of the “bad ones” according to the rewards and definitely the worst one according to the “warm-water-supply-time”. In fact, 10 of the 12 agents did not reduce their “warm-water-supply-time” at all due to the failure of the solar collectors. This could be because Agent 1 was trained with the Markov chains of the real system (the ones shown in Figures 22, 23 and 24), and in these Markov chains failures of the solar energy system happen only during 1.64% of the time.

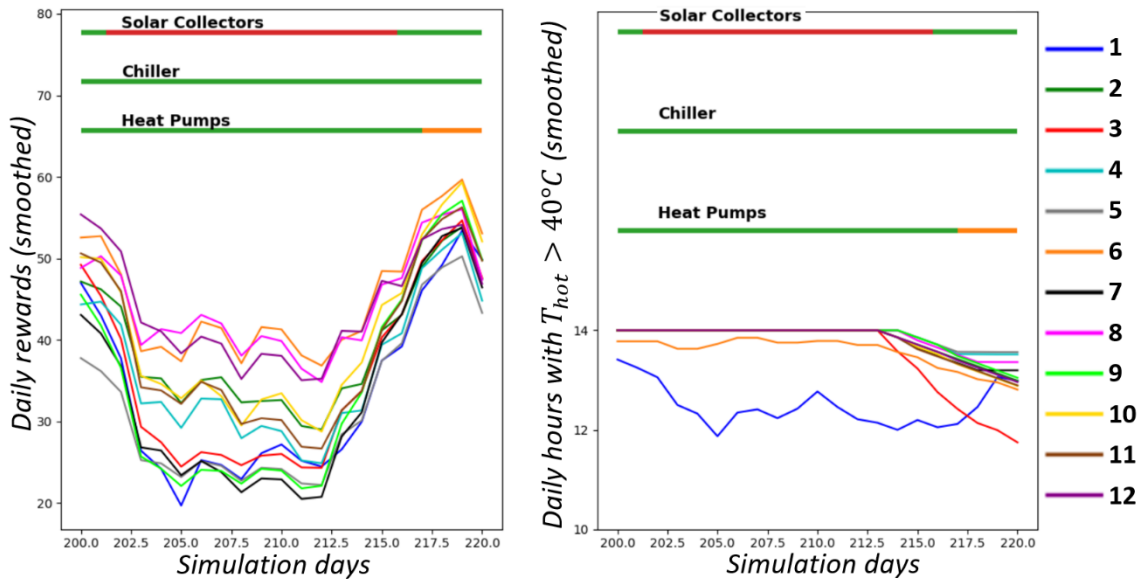


Figure 96: A failure of the solar energy system

Figure 96 shows a failure of the solar energy stage that occurs in winter (mid-July); this is the moment when the solar collectors are least important for the heating process. More interesting would be to analyze a failure of the solar collectors that occurs in summer. This did not happen in the 12-year test without overlapping with failures or degradations of the other heating stages.

Therefore, the result shown in Figure 97 was taken from a test like the one used for Sections 6.2.1 through 6.5.4, i.e. two years and ten days long. The result shown in the figure corresponds to the beginning of the second test year. In this case, more agents delivered warm water during less than 14 hours daily; Agent 1 is still the minimum according to that indicator. A close-up to the rewards in Figure 98 shows that it is one of the agents with low rewards as well. Moreover, the three agents with the worst results in terms of their “warm-water-supply-time” were all trained with the Markov chains of Section 6.5.1 (i.e. with failures of the solar energy system happening only 1.64% of the time). Most agents have no trouble to continue providing warm water in the same time-span.

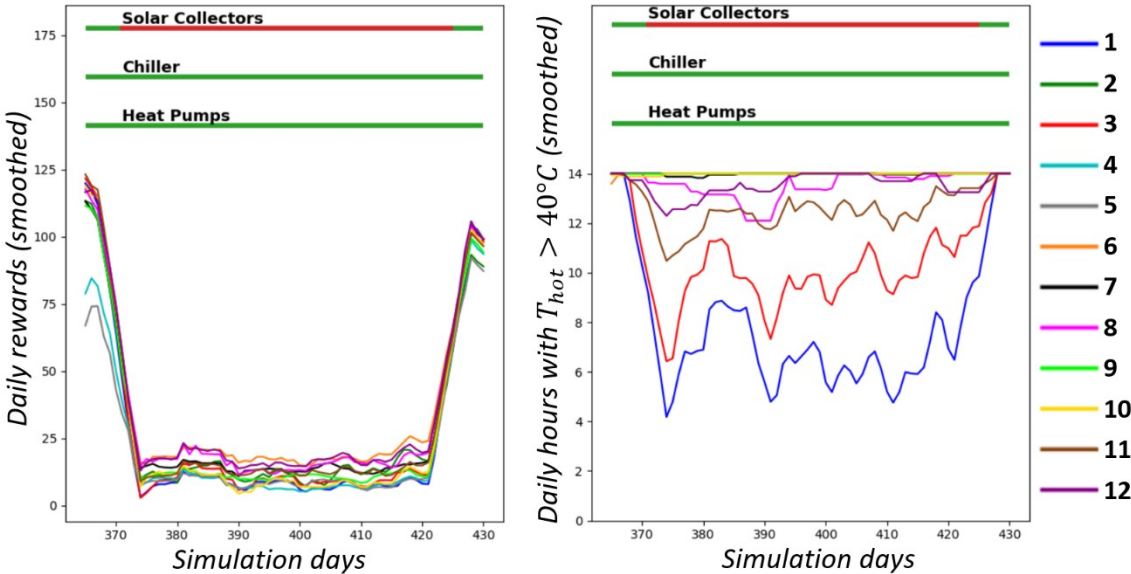


Figure 97: Rewards and “warm-water-supply-time” under a failure of the solar energy system in summer

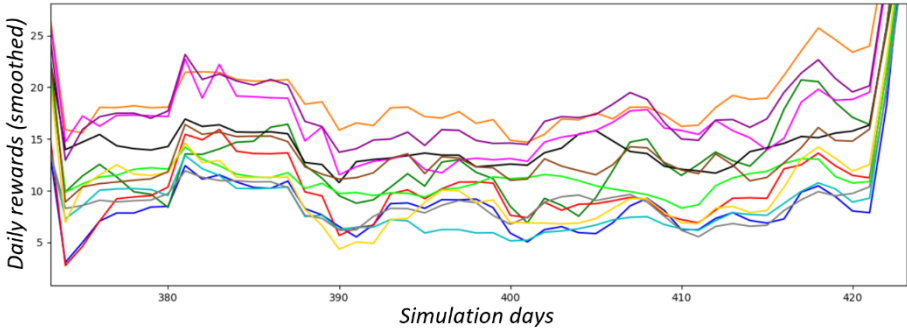


Figure 98: Close-up to the rewards of Figure 97

It would be possible to extract more “interesting” moments from the 12-year-long test; however, the conclusions would probably be the same that can be inferred from the current results: not all agents perform equally well when confronting failures of the system, and some agents that perform quite well with some failures perform worse with other failures or combinations of failures.

Also, the reward function can be improved because the agents do not always have what would be, from a common-sense point of view, the best possible behavior as they try to maximize it. An example of this, as already discussed, is shown in Figure 94.

Chapter 7: Conclusions

7.1. Accomplishment of objectives

In the following paragraphs, the same specific objectives that were set on Section 1.6.2 are repeated and analyzed considering the results of the study (the specific objectives are written in italics).

1. *Establishing a connection between the TRNSYS software and the Python programming language. The Python code has to be able to transmit decisions to the TRNSYS simulation, regarding which devices are used. In addition to this, the code must receive results from the simulation, use them to make decisions and impose these decisions on the simulation.*

This objective was accomplished as detailed in Section 4.5. In order to achieve it, a standard TRNSYS feature that makes it possible to connect this software to Python (i.e. Type 169) was used. As already discussed, this mode of connection made it necessary to store information of previous moments in text files, and also to develop the training algorithm without the use of specialized Deep Learning libraries.

2. *Showing that an effective training process of the DNNs can be achieved in a basic programming language, without the use of specialized Deep Learning libraries.*

By using the theoretical concepts discussed in Sections 3.1 and 3.2, it was possible to develop the training platform without Numpy and other specialized Deep Learning libraries. The entire DNN-training algorithm was developed by using custom functions mainly based on Python lists. This, together with the ability of Python to create and read text files, made it possible for the Python code to interact with the TRNSYS simulation and to train smart controlling agents. As shown in Section 6.2, these agents clearly learn to outperform a baseline strategy, or at least to do as good the baseline when no margin of improvement seems to be possible. This result is encouraging because it shows that the same agent-training techniques can be applied to environments (real or simulated) where access to Deep Learning platforms is not possible.

3. *Defining a reward function that fulfills the condition of producing a desirable behavior of the smart agents as they try to maximize it.*

A reward function has been developed considering the most basic indicators that can be extracted from the simulation during the time-span after the previous action, such as energy- and temperature-related indicators, and the previous action itself. In Section 4.3, these indicators are introduced in a “global” reward function by using different parameters to set their contributions. Although this reward function seems to be effective to encourage the agents to deliver warm water while sparing energy, it has a big flaw: it is possible for the agents to increase the total rewards while delivering warm water less time. Therefore, the reward function does not completely meet a condition that is assumed in RL: “the higher the reward, the better”. This was discussed in Section 6.5.5.

4. *Analyzing the training results when the hyperparameters of the training algorithm are changed. Different neural network architectures are tested as well.*

As shown in Section 6.3, some hyperparameters do not seem to produce a highly significant performance variation, like for example, the use of Double DQN (this was not tested in a failure-subject environment, so the results may vary there). On the other hand, some hyperparameters produce highly different performances as they are varied, like the use of momentum. This was

discussed in Section 6.3.4, showing that momentum is highly beneficial for the performance of the agents. This also leads to think that better results could be obtained with better optimization algorithms (such as Adam, RMSprop, etc.).

5. *Analyzing how the behavior of the controlling agents changes as the reward function is modified. Modifying the reward function is equivalent to changing what the agents are supposed to achieve.*

As shown in Section 6.4, the behavior of the agents seems to change as expected when the parameters of the reward function are changed, considering how these parameters are justified in Section 4.3. This means that the rationale behind the definition of the reward function was right (despite the problem just mentioned in Objective 3); therefore, the reward function presented here could be adapted to other environments where Reinforcement Learning is being applied, considering the specific goals of the training process in those environments.

6. *Training agents to fulfill their task when the devices of the system are subject to random failures.*

Section 4.5 proposes three methods of training agents that are resilient to certain failures of the system. Although not all results are positive, it has been proven that it is possible to train a single neural network that can handle the system under different scenarios. An obvious and very simple alternative way to address the failure-resilience issue would be to train a different DNN for each possible functional state of the system, i.e. exposing each DNN to a unique functional state of the system during training, and then using each DNN only when the state of the system is the one for which that DNN has been trained. This method would clearly outperform all agents presented here (at least from the point of view of the rewards), but part of the motivation of the study presented here was to have a single DNN that could handle all possible states of the system. Moreover, the system presented here has 18 possible functional states (considering all combinations of states of the heating stages), thus 18 DNNs would have to be trained and used (some of them can be discarded because the agents would get zero rewards anyway, e.g. the combination of all heating stages having failed).

7.2. Future work

Regarding the possibility of putting this project into practice, it is clear that the simulation used to train the agents has many simplifications with respect to the actual water heating system, located in the building of Beauchef 851, Santiago. An obvious example is the fact that in the simulation the warm water demand profile is the same every day. Because of this, the agents developed here are prepared for a system that is much simpler than the actual system; this could lead them to perform poorly were they put into practice. Therefore, the obvious next step of this project would be to improve the simulated system by making more things stochastic (like the warm water demand) and to enhance many simplifications that the simulation has (like not simulating the pool). As already discussed in Section 6.2, there is a clear simplification being made in this study: the main task of the chiller is to cool a cooling load that is not being modeled in detail. Therefore, it is hard to consider the cooling function of the chiller for the computation of the reward function. This motivation for using the chiller was rudimentarily introduced into the reward function in the same section, but the method proposed would not allow the agents to be put into practice. In order to really take the cooling load of the chiller into account, it would have to be simulated as part of the system.

Introducing all these extra complexities into the simulation would clearly make the training process much more difficult, and the agents would have a hard time even in an environment without

failures. This can be concluded from the fact that, even with a simple complexification of the system like adding failures to the devices, the agents struggle to find a good solution. Thus, one of the first things to do is to make more powerful neural networks. The introduction of recurrent neural networks would be the obvious next step, as well as introducing regularization methods (e.g. l_1 regularization, l_2 regularization and dropout). Adding predictions of the future (like weather-, demand- and failure predictions which are subject to errors) would also make the result more interesting and more likely to be put into practice.

Another change which could improve the results, even with more complexities being considered in the simulation, could be reducing the time-span between consecutive actions of the agent. The following example will clarify why: if the water demand in the dressing rooms were stochastic instead of a fixed function that is the same every day, then the agent would have to choose which devices to turn on without actually knowing how large the future demand will be. In this context, it is possible that the agent makes the mistake of turning on devices that are enough to supply warm water for a low demand, and then the demand turns out to be quite high. If the time-span between the actions of the agent were shorter, then the agent might have time to rectify its error in the next action before the temperature of the water drops below the comfort threshold.

One must also keep in mind that in Reinforcement Learning good results are never easy to get. Getting good results is hard even in environments that one would consider “simple”. That is why this area of Machine Learning took off only a few years ago. However, for the same reason there are new techniques permanently being discovered and published. For future developments of this platform, it may be necessary to add recently discovered methods in order to improve the convergence of the algorithms.

8. Glossary

CH ₄	methane
CO ₂	carbon dioxide
DWH	domestic water heating
DNN	deep neural network (in this case Dense unless specified)
DQN	Deep Q-Network, method to train a DNN to predict Q-Values (see 3.2.5)
DRL	Deep Reinforcement Learning
FCFM	Faculty of Physical and Mathematical Sciences of the University of Chile
GHG	greenhouse gas
H ₂ O	water molecule
HVAC	heating, ventilation, and air conditioning
KPI	key performance indicator
NASA	National Aeronautics and Space Administration
ODE	ordinary differential equation
PDF	probability density function
RL	Reinforcement Learning
SWH	solar water heating
TES	thermal energy storage
TRNSYS	Transient System Simulation Tool

9. Bibliography

- [1] National Ocean Service, National Oceanic and Atmospheric Administration: *What is the Carbon Cycle?* February 26, 2021. <https://oceanservice.noaa.gov/facts/carbon-cycle.html>
- [2] Melissa Denchak: *Greenhouse Effect 101*, Natural Resources Defense Council. July 16, 2019 <https://www.nrdc.org/stories/greenhouse-effect-101>
- [3] NASA: *Graphic: The relentless rise of carbon dioxide*. August 2, 2021. Data from: Luthi, D. et al. 2008; Etheridge, D.M. et al. 2010; Vostok ice core data/J.R. Petit et al.; NOAA Mauna Loa CO2 record. https://climate.nasa.gov/climate_resources/24/graphic-the-relentless-rise-of-carbon-dioxide/
- [4] NASA Scientific Visualization Studio: *Global Temperature Anomalies from 1880 to 2020*. January 14, 2021. Visualization by Lori Perkins; data provided by Robert B. Schmunk <https://svs.gsfc.nasa.gov/cgi-bin/details.cgi?aid=4882>
- [5] Climate Reality Project: *How feedback loops are making the climate crisis worse*. January 7, 2020 <https://www.climaterealityproject.org/blog/how-feedback-loops-are-making-climate-crisis-worse>
- [6] Brian C. O'Neill; Elmar Kriegler; Kristie L. Ebi; Eric Kemp-Benedict, Keywan Riahi; Dale S. Rothman; Bas J. van Ruijven; Detlef P. van Vuuren; Joern Birkmann; Kasper Kok; Marc Levy; William Solecki: *The roads ahead: Narratives for shared socioeconomic pathways describing world futures in the 21st century*. Global Environmental Change, Vol. 40, Pages 169-180. 2017
- [7] Jeff Tollefson: *How hot will Earth get by 2100?* Nature 580, 443-445 (2020)
- [8] Ngan Le: *The impact of fast fashion on the environment*. Princeton Student Climate Initiative. July 20, 2020. <https://psci.princeton.edu/tips/2020/7/20/the-impact-of-fast-fashion-on-the-environment>
- [9] The World Bank: *Life expectancy at birth, total (years)*. <https://data.worldbank.org/indicator/SP.DYN.LE00.IN>
- [10] Junhyuk Oh; Matteo Hessel; Wojciech M. Czarnecki; Zhongwen Xu; Hado van Hasselt; Satinder Sigh; David Silver: *Discovering Reinforcement Learning Algorithms*. DeepMind, January 5, 2021
- [11] Pierre Ménard; Omer Darwiche Domingues; Anders Jonsson; Emilie Kaufmann; Edouard Leurent; Michal Valko: *Fast active learning for pure exploration in reinforcement learning*. DeepMind, October 10, 2020
- [12] Markus Wulfmeier; Dushyant Rao; Roland Hafner; Thomas Lampe; Abbas Abdolmaleki; Tim Hertweck; Michael Neunert; Dhruva Tirumala; Noah Siegel; Nicolas Heess; Martin Riedmiller: *Data-efficient Hindsight Off-policy Option Learning*. DeepMind, June 15, 2021
- [13] Roland Hafner; Tim Hertweck; Philipp Klöppner; Michael Bloesch; Michael Neunert Markus Wulfmeier; Saran Tunyasuvunakool; Nicolas Heess; Martin Riedmiller: *Towards General and Autonomous Learning of Core Skills: A Case Study in Locomotion*. DeepMind, August 6, 2020

- [14] Volodymyr Mnih; Koray Kavukcuoglu; David Silver; Andrei A. Rusu; Joel Veness; Marc G. Bellemare; Alex Graves; Martin Riedmiller; Andreas K. Fidjeland; Georg Ostrovski; Stig Petersen; Charles Beattie; Amir Sadik; Ioannis Antonoglou; Helen King; Dharshan Kumaran; Daan Wierstra; Shane Legg; Demis Hassabis: *Human-level control through deep reinforcement learning*. Nature 518, 529-533 (2015).
- [15] *What is TRNSYS?* <http://www.trnsys.com>
- [16] Hannah Ritchie and Max Roser (2020): *Energy mix*. Published online at OurWorldInData.org. Retrieved from: <https://ourworldindata.org/energy-mix>
- [17] Célia Artur; Diana Neves; Boaventura C. Cuamba; António J. Leão: *Domestic hot water technology transition for solar thermal systems: An assessment for the urban areas of Maputo city, Mozambique*. Journal of Cleaner production 260 (2020) 121043
- [18] Carolina Aguilar; D. J. White; David L. Ryan: *Domestic Water Heating and Water Heater Energy Consumption in Canada*. CBEEDAC, 2005
- [19] Brett Dolter; Nicholas Rivers. *The cost of decarbonizing the Canadian electricity system*. Energy Policy 113 (2018) 135-148
- [20] Mehdi Jahangiri; Akbar Alidadi Shamsabadi; Hamed Saghaei: *Comprehensive Evaluation of Using Solar Water Heater on a Household Scale in Canada*. Journal of Renewable Energy and Environment, Vol. 5, No. 1 (Winter 2018) 35-42
- [21] E. Saloux; J.A. Candanedo. *Optimal rule-based control for the management of thermal energy storage in a Canadian solar district heating system*. Solar Energy 207 (2020) 1191-1201
- [22] Zhiyong Tian; Shicong Zhang; Jie Deng; Jianhua Fan; Junpeng Huang; Weiqiang Kong; Bengt Perers; Simon Furbo: *Large-scale solar district heating plants in Danish smart thermal grid: Developments and recent trends*. Energy Conversion and Management 189 (2019) 67-80
- [23] Abhiram Mullapudi; Matthew J. Lewis; Cyndee L. Gruden; Branko Kerkez: *Deep reinforcement learning for the real time control of stormwater systems*. Advances in Water Resources 140 (2020) 103600
- [24] J.J. Yang; M. Yang; M.X. Wang; P.J. Du; Y.X. Yu: *A deep reinforcement learning method for managing wind farm uncertainties through energy storage system control and external reserve purchasing*. Electrical Power and Energy Systems 119 (2020) 105928
- [25] Taha Abdelhalim Nakabi; Pekka Toivanen: *Deep reinforcement learning for energy management in a microgrid with flexible demand*. Sustainable Energy, Grids and Networks 25 (2021) 100413
- [26] Xiaozhen Lu; Xingyu Xiao; Liang Xiao; Canhuang Dai; Mugen Peng; H. Vincent Poor: *Reinforcement Learning-Based Microgrid Energy Trading With a Reduced Power Plant Schedule*. IEEE Internet of Things Journal, Vol 6, No. 6, December 2019
- [27] Yan Du; Fangxing Li; Jeffrey Munk; Kuldeep Kurte; Olivera Kotevska; Kadir Amasyali; Helia Zandi: *Multi-task deep reinforcement learning for intelligent multi-zone residential HVAC control*. Electric Power Systems Research 192 (2021) 106959

- [28] Anchal Gupta; Youakim Badr; Ashkan Negahban; Robin G. Qiu: *Energy-efficient heating control for smart buildings with deep reinforcement learning*. Journal of Building Engineering 34 (2021) 101739
- [29] Silvio Brandi; Marco Savino Piscitelli; Marco Martellacci; Alfonso Capozzoli: *Deep reinforcement learning to optimise indoor temperature control and heating energy consumption in buildings*. Energy & Buildings 224 (2020) 110225
- [30] Paulo Lissa; Conor Deane; Michael Schukat; Federico Seri; Marcus Keane; Enda Barrett: *Deep reinforcement learning for home energy management system control*. Energy and AI 3 (2021) 100043
- [31] Guanyu Gao; Jie Li; Yonggang Wen: *Energy-Efficient Thermal Comfort Control in Smart Buildings via Deep Reinforcement Learning*. IEEE Internet of Things Journal, Volume 7, Issue 9. 2019
- [32] Camila Correa-Jullian; Enrique López Droguett; José Miguel Cardemil: *Operation scheduling in a solar thermal system: A reinforcement learning-based framework*. Applied Energy vol. 268 (June 15, 2020) 114943
- [33] Amir Ramezani Dooraki; Deok-Jin Lee: *Reinforcement learning based flight controller capable of controlling a quadcopter with four, three and two working motors*. 20th International Conference on Control Automation and Systems (ICCAS) (2020). Busan, Korea.
- [34] Daniel L. K. Yamins; James J. DiCarlo: *Using goal-driven deep learning models to understand sensory cortex*. Nature Neuroscience 19, 356-365 (2016).
- [35] Edgar Y. Walker; Fabian H. Sinz; Erick Cobos; Taliah Muhammad; Emmanouil Froudarakis; Paul G. Fahey; Alexander S. Ecker; Jacob Reimer; Xaq Pitkow; Andreas S. Tolias: *Inception loops discover what excites neurons most using deep predictive models*. Nature Neuroscience 22, 2060-2065 (2019).
- [36] Aurélien Géron: *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow. Concepts, Tools and Techniques to Build Intelligent Systems*. 2nd Edition. O'Reilly Media, 2019
- [37] Michael A. Nielsen: *Neural Networks and Deep Learning*, Determination Press, 2015. Chapter 2: *How the backpropagation algorithm works*
- [38] David E. Rumelhart; Geoffrey E. Hinton; Ronald J. Williams: *Learning representations by back-propagating errors*. Nature 323, 533-536 (1986).
- [39] Kian Katanforoosh; Daniel Kunin, *Initializing neural networks*, deeplearning.ai, 2019.
- [40] Keras: *Dense layer*. https://keras.io/api/layers/core_layers/dense/
- [41] Xavier Glorot; Joshua Bengio: *Understanding the difficulty of training deep feedforward neural networks*. 2010
- [42] *RL Course by David Silver - Lecture 1: Introduction to Reinforcement Learning*. Lecture in DeepMind's Youtube Channel. <https://www.youtube.com/watch?v=2pWv7GOvuf0>
- [43] Richard S. Sutton; Andrew G. Barto: *Reinforcement Learning: An introduction*. Second edition. MIT Press, 2018

- [44] Hado van Hasselt; Arthur Guez; David Silver (Google DeepMind): *Deep Reinforcement Learning with Double Q-Learning*. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. February 2016
- [45] Tom Schaul; John Quan; Ioannis Antonoglou; David Silver (Deep Mind): *Prioritized Experience Replay*. International Conference on Learning Representations, May 2016
- [46] Mohammad Modarres; Mark P. Kaminskiy; Vasiliy Krivtsov: *Reliability Engineering and Risk Analysis; a Practical Guide*. Third Edition, 2016
- [47] Oliver C. Ibe: *Markov Processes for Stochastic Modeling*. Second Edition (2013), Chapter 4
- [48] *Andrey Andreyevich Markov*. Encyclopaedia Britannica
<https://www.britannica.com/biography/Andrey-Andreyevich-Markov>
- [49] Alexander Holmes; Barbara Illowsky; Susan Dean. *Introductory Business Statistics*, 2017
<https://openstax.org/details/books/introductory-business-statistics>
- [50] Francesco Asdrubali; Umberto Desideri: *Handbook of Energy Efficiency in Buildings: A Life Cycle Approach* (2018). Page 516
- [51] Michael J. Moran; Howard N. Shapiro; Daisie D. Boettner; Margaret B. Bailey: *Fundamentals of Engineering Thermodynamics*. Eighth Edition (2014)
- [52] Camila Asunción Correa Jullian: *Assessment of Deep Learning Techniques for Diagnosis in Thermal Systems through Anomaly Detection*. Thesis, University of Chile, 2019
- [53] *TRNSYS 18: A Transient System Simulation Program*. Chapter 4: *Mathematical Reference*
- [54] *What is NumPy?* <https://numpy.org/doc/stable/user/whatisnumpy.html>
- [55] *SciPy library*. <https://www.scipy.org/scipylib/index.html>
- [56] *Introduction to Tensorflow*. <https://www.tensorflow.org/learn>
- [57] *The Python Tutorial*, Section 3.1.3: *Lists*
<https://docs.python.org/3.7/tutorial/introduction.html#lists>
- [58] *random* — *Generate pseudo-random numbers*
<https://docs.python.org/3/library/random.html>
- [59] *bisect* — *Array bisection algorithm*. <https://docs.python.org/3/library/bisect.html>
- [60] ENI S.p.A./AGIP Exploration & Production; BP Exploration Operating Company Ltd; ExxonMobil International Ltd.; Norsk Hydro ASA; Phillips Petroleum Company Norway; Statoil ASA; Shell Exploration & Production; TotalFinalElf: *Oreda. Offshore Reliability Data Handbook*. 4th Edition. 2002
- [61] *Meteonorm Software – Worldwide irradiation data*. <https://meteonorm.com/en/>
- [62] Jay Burch; Craig Christensen: *Towards Development of an Algorithm for Mains Water Temperature*.

Annexes

Annexed A. Further details about the water heating system simulation

Details about the heating system which were considered not to be essential for understanding the study were left out of the main part of the thesis. Here, those details are explained, so that a similar model of the same water heating system can be created by the reader. The model is based on the previous work of Camila Correa [49] and Camila Correa et al. [32].

The simulations were carried out in a Toshiba notebook which was acquired at the end of the year 2013 and has the following characteristics:

- Notebook model: Toshiba Satellite P55t – ASP5303SL
- CPU: Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz
- RAM: 12.0 GB
- Disc: HGST HTS541010A9E680 (Hard Disc, 1 TB capacity)

The time steps of the simulation are one minute long.

Annexed A.1. Detailed flow diagrams

Here, some items which have been grouped in Figure 14 will be detailed. Figures 99 through 101 correspond to the solar stage, the chiller and the heat pump stage respectively. It is necessary to understand Figure 14 in order to understand the three figures below.

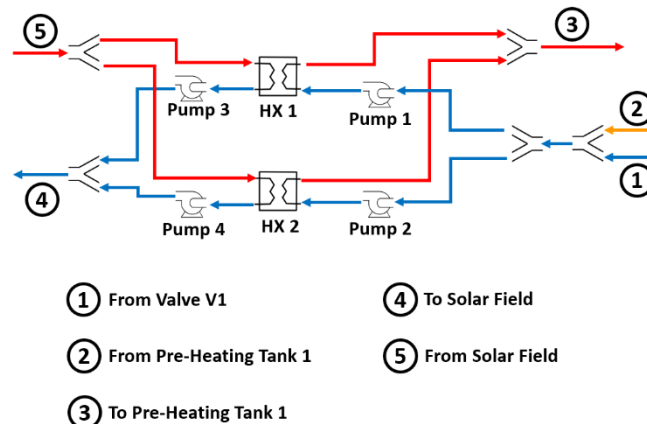


Figure 99: Detailed flow diagram of the solar stage

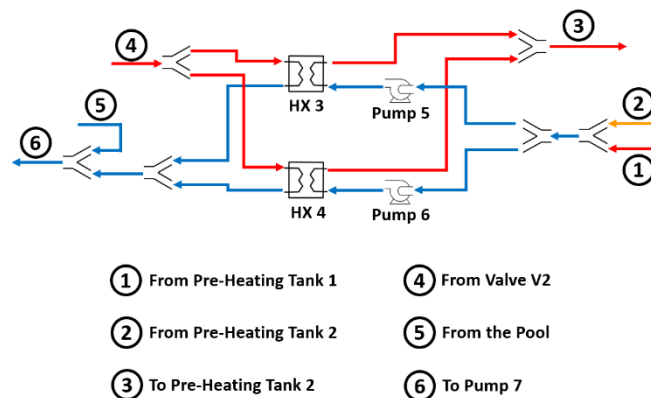


Figure 100: Detailed flow diagram of the heat recovery stage (chiller)

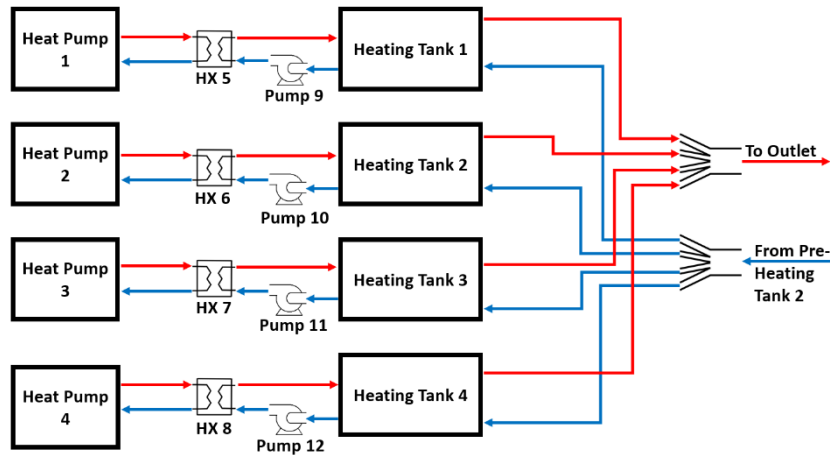


Figure 101: Detailed flow diagram of the heat pump system. It is considered that each heat pump has an integrated water pump in order to move the water flow to the heat exchangers.

Annexed A.2. Parameters of the elements in the system simulation (Types)

Heat exchangers

All heat exchangers in the system are modeled using the Type 91 (Constant Effectiveness Heat Exchanger) and they have the same parameters, which are detailed in Table 32.

Table 32: Parameters of the heat exchangers

Parameter	Value
Heat exchanger effectiveness	0.6
Specific heat of source side fluid	4.19 kJ/kg.K
Specific heat of load size fluid	4.19 kJ/kg.K

Water pumps

All pumps are single-speed pumps (Type 114); in all of them, the “Motor heat loss fraction” is set to zero, and the “Fluid specific heat” is set to 4.19 kJ/kg.K. The other parameters are specified in Table 33. The “internal” pumps of the heat pumps are separate items in the TRNSYS software, so they are specified in the last row of the table.

Table 33: Parameters of the water pumps

Pump Number	Rated Flow Rate	Rated Power
1 and 2	5300 kg/hr	2520 kJ/hr
3 and 4	5300 kg/hr	3960 kJ/hr
5 and 6	13500 kg/hr	2520 kJ/hr
7	52390.8 kg/hr	13422.82 kJ/hr
8	29937.6 kg/hr	8053.5 kJ/hr
9 to 12	5300 kg/hr	1440 kJ/hr
Pumps of the heat pumps 1 to 4	5300 kg/hr	1260 kJ/hr

Weather Data

Weather data is necessary to provide the solar radiation to the solar collectors and the external dry bulb temperature to the heat pumps of the third heating stage. Weather data from many cities around the world is available from various sources; in the case of this study, weather data of Meteonorm

[61] which is included in TRNSYS was used to get the weather data of Santiago. In order to do this, Type 15-3 was used.

Storage tanks

Preheating Tanks 1 and 2, which are part of the first and second heating stage respectively, are modeled with the Type 39. The “Excess Flow” is recirculated through the respective heat exchanger to permanently extract heat from the respective source, as shown in Figure 14. The “Flow Rate To Load”, which is an input to “tell” the tank the demanded water flow, is by definition the same flow that is entering in the system; in this way, the volume inside these tanks never varies. Both tanks have exactly the same parameters, which are specified in Table 34. The difference between them is, clearly, the other elements (types) in the simulation with which they are connected.

Table 34: Parameters of Preheating Tanks 1 and 2

Parameter	Value
Tank operation mode	1
Overall tank volume	2 m ³
Minimum fluid volume	1.0 m ³
Maximum fluid volume	2 m ³
Tank circumference	4.89 m
Cross-sectional area	1.9 m ²
Wetted loss coefficient	6.0 kJ/hr.m ² .K
Dry loss coefficient	4.0 kJ/hr.m ² .K
Fluid specific heat	4.19 kJ/kg.K
Fluid density	1000 kg/m ³
Initial fluid temperature	35 °C
Initial fluid volume	2 m ³

The storage tanks of the third heating stage, i.e. the Heating Tanks 1 to 4, are modeled by using Type 534. The inlet-outlet pair number 1 is used to receive the water flow from the previous heating stages and to deliver water to the outlet of the system, while the inlet-outlet pair number 2 is used to recirculate the fluid inside the tanks to extract heat from the heat pumps. All the tanks have the same parameters, which are listed in Table 35.

Table 35: Parameters of Heating Tanks 1 to 4

Parameter	Value
LU for Data File	-1
Number of Tank Nodes	10
Number of Ports	2
Number of Immersed Heat Exchangers	0
Number of Miscellaneous Heat Flows	0
Tank Volume	4 m ³
Tank Height	2.1 m
Tank Fluid	0
Fluid Specific Heat	4.19 kJ/kg.K
Fluid Density	1000 kg/m ³
Fluid Thermal Conductivity	2.14 kJ/hr.m.K
Fluid Viscosity	3.21 kg/m.hr
Fluid Thermal Expansion Coefficient	0.00026 /K
Top Loss Coefficient	5.0 kJ/hr.m ² .K
Edge Loss Coefficient for all Nodes	5.0 kJ/hr.m ² .K
Bottom Loss Coefficient	5.0 kJ/hr.m ² .K
Inlet Flow Mode-1	1
Entry Node-1	10
Exit Node-1	1
Inlet Flow Mode-2	1
Entry Node-2	1
Exit Node-2	10
Flue Overall Loss Coefficient for all Nodes	3.0 kJ/hr.K

Valve V1

As Figure 14 shows, valve V1 deviates part of the entering water directly to the outlet of the system. It does this so that the water leaving the system does not surpass a temperature of 45°C. To accomplish this task, the valve V1 has to receive the temperature of the water that is leaving the third heating stage, so that it can calculate the amount of water that it has to send directly to the outlet.

To do this in TRNSYS, valve V1 has to be modeled with Type 11b (Tempering Valve). The “Setpoint Temperature”, which is an input and not a parameter, is given a constant value of 45°C. The flow mixer in the outlet, which mixes the flow coming out of the third heating stage with the mains water flow in order to keep the water at 45°C, is modeled with a normal flow mixer (Type 11h). The flow mixer at the outlet of the heat pumps, which mixes the flows coming out of the four heat pumps and is illustrated in Figure 101, sends its outlet temperature to valve V1 (Type 11b) in order to regulate the temperature of the water that leaves the system, as already discussed.

Valve V2 and Pool

Valve V2 is configured to always send 51% of the flow coming out of the chiller to the heat exchangers where it gives off heat to the water for the dressing rooms. The remaining 49% is sent to the pool. The pool is not being modeled in the simulation; it is simply assumed that the water flow coming out of the pool has a temperature of 44.5°C. This is assumed following the original design of the thermal system of the building.

Automatic Control Systems

The heat pumps and the pumps of the solar field (pumps 3 and 4 in Figure 14) have automatic control systems that turn them on and off depending on the temperature of the water in their respective storage tanks. This is only done when the smart controlling agent (neural network) has previously decided to turn on the corresponding stage of the heating system. Type 911 (Differential Controller with Lock-Outs) is used to control de devices. Table 36 shows the parameters and inputs of the controllers. In the case of the heat pumps, each heat pump has its own controller which bases its decision on the temperature of the storage tank of the corresponding heat pump.

Table 36: Parameters and inputs of the automatic controllers

Parameter or input	Controller of the pumps of the solar field	Controllers of the heat pumps (each heat pump has its individual controller)
# of Oscillations	5	5
Minimum Run-Time	0.25	0.5
Minimum Reset Time	0.25	0.5
Upper input temperature Th	47	60
Lower input Temperature Tl	Equal to the temperature of Preheating Tank 1	Equal to the average temperature of the corresponding storage tank
Monitoring Temperature Tin	60	60
High Limit Cut-Out	75	75
Upper Dead Band dT	2	5
Lower Dead Band dT	-3	-2
Lock-Out Control Signal	0	0

Annexed A.3. Elements (Types) with external files

Solar fields

In the simulation, the solar field is divided in two Types 71, one of them representing the part of the filed where there are four collectors per row, and the other representing the part of the field where there are three collectors per row (see Figure 13).

The parameters of both types are shown in Table 37.

Table 37: Parameters of the solar fields

Parameter	Solar Field 1	Solar Field 2
Number in series	4	3
Collector area	48 m ²	58 m ²
Fluid specific heat	4.19 kJ/kg.K	4.19 kJ/kg.K
Efficiency mode	2	2
Flow rate at test conditions	68.4 kg/hr.m ²	68.4 kg/hr.m ²
Intercept efficiency	0.618	0.618
Negative of first order efficiency coefficient	1.3767 W/m ² .K	1.3767 W/m ² .K
Negative of second order efficiency coefficient	0.0184 W/m ² .K ²	0.0184 W/m ² .K ²
Logical unit of file containing biaxial IAM data	31	32
Number of longitudinal angles for which IAMs are provided	10	10
Number of transverse angles for which IAMs are provided	10	10

The solar fields are associated to a data file that provides the incidence angle modifier (IAM) for each combination of transverse and longitudinal angles of the sun. The data is shown in Tables 38 and 39. It was obtained from the previous work by Camila Correa [52].

Table 38: IAM values of the solar collectors; transverse angles from 0° to 40°

Longitudinal Angle	Transverse Angle				
	0°	10°	20°	30°	40°
0°	1.0000	1.0200	1.0800	1.1800	1.3700
10°	1.0000	1.0090	1.0180	1.0549	1.1498
20°	0.9900	1.0039	1.0129	1.0497	1.1441
30°	0.9800	1.0282	1.0373	1.0750	1.1717
40°	0.9600	0.9837	0.9925	1.0285	1.1211
50°	0.9300	0.9615	0.9701	1.0053	1.0958
60°	0.8700	0.9221	0.9303	0.9641	1.0509
70°	0.7400	0.8413	0.8488	0.8796	0.9588
80°	0.3800	0.3876	0.4104	0.4484	0.5206
90°	0.0000	0.0000	0.0000	0.0000	0.0000

Table 39: IAM values of the solar collectors; transverse angles from 50° to 90°

Longitudinal Angle	Transverse Angle				
	50°	60°	70°	80°	90°
0°	1.4000	1.3400	1.2400	0.9500	0.0000
10°	1.4505	1.4605	1.2597	0.9500	0.0000
20°	1.4433	1.4532	1.2534	0.9405	0.0000
30°	1.4781	1.4883	1.2837	0.9310	0.0000
40°	1.4142	1.4240	1.2282	0.9120	0.0000
50°	1.3823	1.3918	1.2005	0.8835	0.0000
60°	1.3257	1.3348	1.1513	0.8265	0.0000
70°	1.2095	1.2178	1.0504	0.7030	0.0000
80°	0.5320	0.5092	0.4712	0.3610	0.0000
90°	0.0000	0.0000	0.0000	0.0000	0.0000

Chiller

The chiller is modeled with Type 666. The parameters and the chilled water set point, which is actually an input but is left constant, are shown in Table 40.

Table 40: Parameters and chilled water set point of the chiller

Parameter or input	Value
Rated Capacity	1080000 kJ/hr
Rated C.O.P.	4.5
Logical Unit – Performance Data	46
Logical Unit – PLR Data	47
CHW Fluid Specific Heat	4.19 kJ/kg.K
CW Fluid Specific Heat	4.19 kJ/kg.K
Number of CW Points	6
Number of CHW Points	6
Number of PLRs	5
CHW Set Point Temperature	10 C

The chiller uses two external files to specify its performance. One of the files specifies the cooling capacity and the C.O.P. of the machine operating at full load, as functions of the temperature of the chilled water leaving the machine and the temperature of the cooling water entering the machine. In that file, the capacity and the COP have to be specified as ratios of the rated capacity and rated C.O.P. which were specified in Table 40. The performance data is detailed in Tables 41 and 42; it is based on the work of Camila Correa [52].

Table 41: Capacity ratios of the chiller at full load

Cooling water inlet temperature	Chilled water leaving temperature					
	5°C	6°C	7°C	8°C	9°C	10°C
30°C	1.031	1.063	1.096	1.130	1.165	1.200
35°C	0.978	1.009	1.041	1.073	1.106	1.140
40°C	0.922	0.952	0.981	1.012	1.043	1.075
45°C	0.863	0.890	0.918	0.947	0.976	1.007
50°C	0.803	0.828	0.854	0.881	0.909	0.937
55°C	0.738	0.761	0.786	0.811	0.836	0.862

Table 42: COP ratios of the chiller at full load

Cooling water inlet temperature	Chilled water leaving temperature					
	5°C	6°C	7°C	8°C	9°C	10°C
30°C	1.204	1.238	1.271	1.309	1.342	1.380
35°C	1.040	1.071	1.102	1.131	1.164	1.196
40°C	0.889	0.918	0.944	0.971	1.000	1.027
45°C	0.753	0.776	0.800	0.822	0.847	0.871
50°C	0.631	0.651	0.671	0.691	0.711	0.733
55°C	0.522	0.538	0.556	0.571	0.589	0.607

The second file specifies how the performance of the chiller is modified when it is operating at part-load. This can happen because the chiller is meant to cool the chilled water flow up to a certain temperature. If the chiller does not have to use all its capacity to do it, it will operate at part-load.

Although there is the option of providing a customized part-load data file, in this case the “standard” file that is provided by TRNSYS is used.

Heat Pumps

The air-water heat pumps were the only component of the system that had to be added during the course of this study. All other elements were part of the previous studies by Correa [52] and Correa et al. [32] (in those works the heat pumps were modeled as traditional electric heaters). The heat pumps are modeled with Type 941. The performance data was obtained from the datasheet shown in Figure 102. The heat pumps of the system correspond to Model 117 shown in the table of the figure. However, the maximum external air temperature in the data was 10°C (15°C is also in the table, but the values are wrong). Thus, a linear regression was made to estimate the performance at temperatures over 30°C which are commonly reached in Santiago during summer.


HEATING SYSTEM		SIRIO				LINEA RESIDENTIAL AND LIGHT COMMERCIAL					
PRESTAZIONI IN MODALITA' POMPA DI CALORE (RISCALDAMENTO / A.C.S)											
HEAT PUMP PERFORMANCE (HEATING /H.S.W.)											
TAGLIA SIZE	Tain / Twoutc	35 °C		40 °C		45 °C		55 °C		65 °C	
		Pt	Pa	Pt	Pa	Pt	Pa	Pt	Pa	Pt	Pa
18	-20	5,21	2,23	5,29	2,40	5,29	2,57	-	-	-	-
	-10	6,87	2,23	6,90	2,51	6,90	2,68	6,94	2,98	-	-
	-5	7,71	2,31	7,71	2,57	7,71	2,74	7,78	3,34	7,78	4,11
	0	8,84	2,27	8,84	2,64	8,81	2,81	8,79	3,41	8,76	4,18
	7	10,43	2,21	10,43	2,74	10,35	2,91	10,20	3,51	10,12	4,28
	10	11,18	2,48	11,18	2,74	11,11	3,00	10,95	3,59	10,80	4,37
113	15	6,52	1,45	6,52	1,60	6,48	1,75	6,39	2,10	6,30	2,55
	-20	7,20	2,75	7,31	2,96	7,31	3,17	-	-	-	-
	-10	9,50	2,75	9,53	3,10	9,53	3,31	9,59	3,68	-	-
	-5	10,65	2,85	10,65	3,17	10,65	3,38	10,75	4,12	10,75	5,07
	0	12,21	2,80	12,21	3,26	12,17	3,47	12,14	4,21	12,10	5,16
	7	14,40	2,73	14,40	3,38	14,30	3,59	14,09	4,33	13,99	5,28
117	10	15,45	3,06	15,45	3,38	15,34	3,70	15,14	4,43	14,93	5,39
	15	9,01	1,79	9,01	1,97	8,95	2,16	8,83	2,59	8,71	3,14
	-20	10,60	3,89	10,75	4,19	10,75	4,49	-	-	-	-
	-10	13,98	3,89	14,03	4,39	14,03	4,69	14,10	5,21	-	-
	-5	15,66	4,04	15,66	4,49	15,66	4,79	15,82	5,84	15,82	7,19
	0	17,97	3,97	17,97	4,62	17,90	4,92	17,87	5,96	17,80	7,31
117	7	21,19	3,87	21,19	4,79	21,04	5,09	20,73	6,14	20,58	7,49
	10	22,73	4,34	22,73	4,79	22,58	5,24	22,27	6,29	21,96	7,64
	15	13,26	2,53	13,26	2,79	13,17	3,06	12,99	3,67	12,81	4,45

Figure 102: Performance data of the heat pumps. Model 117 is the one used for the study

The parameters used for Type 941 are shown in Table 43. Many of the parameters are meant to define the cooling performance of the heat pumps. They will be specified although they have no importance for the simulation, because the heat pumps are always used in “heating mode”.

Table 43: Parameters of the heat pumps

Parameter	Value
Humidity Mode	2
Logical Unit for Cooling Data	49
Logical Unit for Heating Data	50
Number of Water Temperatures – Cooling	6
Number of Water Temperatures – Heating	5
Number of Dry Bulb Temperatures – Cooling	6
Number of Dry Bulb Temperatures – Heating	5
Specific Heat of Liquid Stream	4.19 kJ/kg.K
Specific Heat of DHW Stream	4.19 kJ/kg.K
Blower Power	662 kJ/hr
Total Air Flowrate	1500 L/s
Rated Cooling Capacity	40301kJ/hr
Rated Cooling Power	7722 kJ/hr
Rated Heating Capacity	72000 kJ/hr
Rated Heating Power	18000 kJ/hr
Capacity of Auxiliary	0.0 kJ/hr

The performance data that is needed for the heat pumps is very similar to the data of the chiller. In the case of the heat pumps, the normalized capacity and normalized consumption (power) are needed as functions of the temperatures of the water and the outside air that are entering the heat pump (“normalized” with respect to the “rated” values of Table 43). Unlike the chiller, the heat pumps only need one performance data file because they always operate at full-load. The file is as follows:

Table 44: Normalized capacity of each heat pump

Entering Air Temperature	Entering Water Temperature				
	35°C	40°C	45°C	55°C	65°C
-5°C	0.7819	0.7819	0.7814	0.7887	0.7894
5°C	1.0162	1.0162	1.0106	1.0011	0.9926
15°C	1.2506	1.2506	1.2397	1.2136	1.1958
25°C	1.4849	1.4849	1.4688	1.4260	1.3990
35°C	1.7193	1.7193	1.6980	1.6385	1.6022

Table 45: Normalized consumption (power) of each heat pump

Entering Air Temperature	Entering Water Temperature				
	35°C	40°C	45°C	55°C	65°C
-5°C	0.7921	0.9008	0.9561	1.1652	1.4352
5°C	0.8157	0.9429	1.0135	1.2231	1.4931
15°C	0.8393	0.9851	1.0709	1.2809	1.5509
25°C	0.8630	1.0273	1.1283	1.3387	1.6087
35°C	0.8866	1.0695	1.1857	1.3965	1.6665

Annexed A.4. Imposed Temperatures

Two temperatures are imposed to the simulation by data-files: the temperature coming from the cooling load of the chiller (i.e. the temperature of the chilled water flow when it is entering the chiller) and the mains water temperature.

The temperature coming from the cooling load of the chiller repeats itself every 24 hours; it is shown in Figure 103. The chiller is meant to cool this water to 10°C.

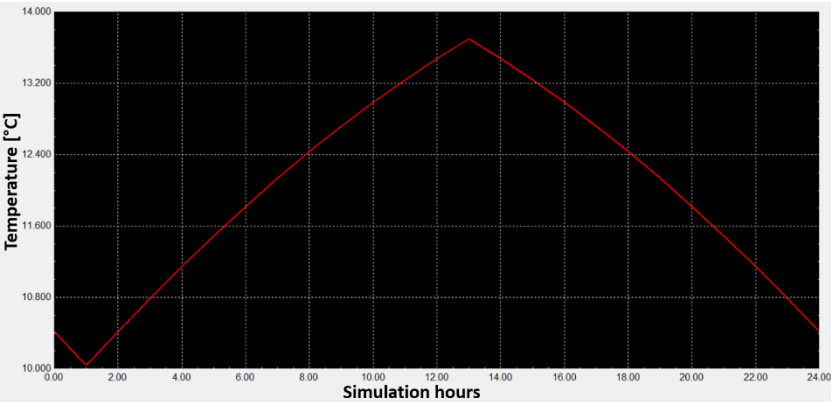


Figure 103: Daily temperature of the chilled water flow when entering the chiller

The mains water temperature was calculated by Correa et al. [32] based on the method presented by Burch and Christensen [62]. This temperature repeats itself each year; it is shown in Figure 104.

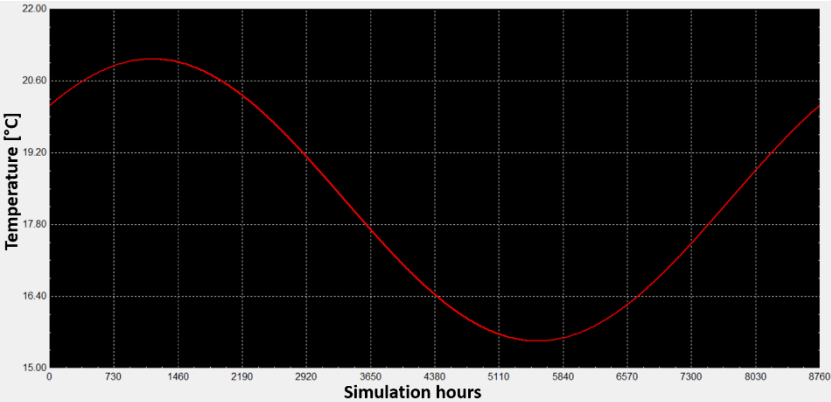


Figure 104: Yearly temperature of the mains water

Annexed B. All results of Section 6.5.

The results shown in Figures 68 and 71 were taken from a parameter exploration that was done only by using Architecture 11, as defined in figure 67. Only the proportional prioritization method was used, and the value of α was varied as shown in Figures 105 and 106. Figure 105 shows the results of the agents that receive 10-variable environment states, and Figure 106 shows the results of the agents that receive 15-variable environment states. The agents shown in the same graph were trained with equal conditions; eight were trained with each hyperparameter combination.

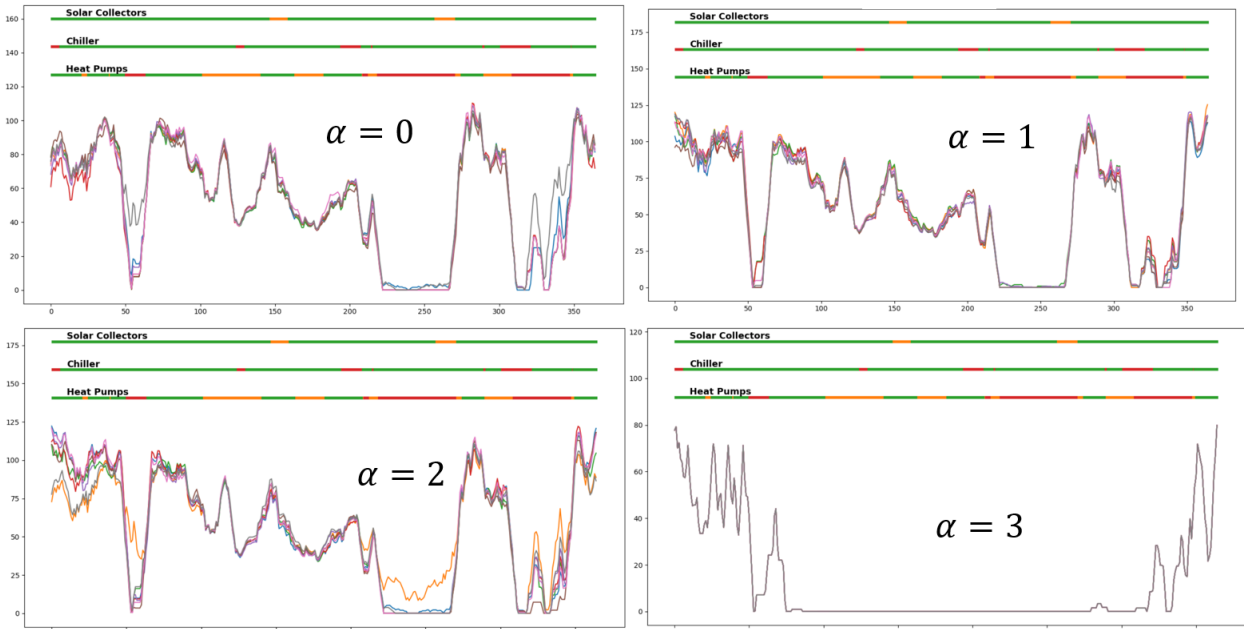


Figure 105: 10-variable states

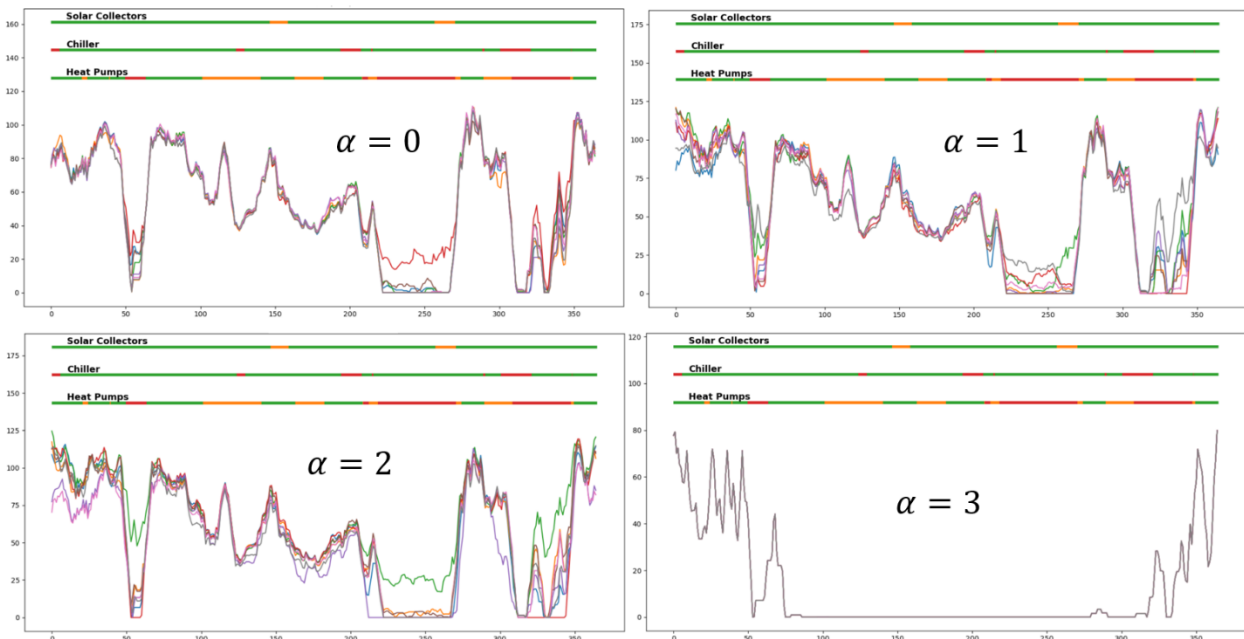


Figure 106: 15-variable states

Figures 72, 73, 74, 75 and 76 were taken from an exploration that was done with Architectures 4, 8, 9, 10 and 12. Six agents were trained with each hyperparameter combination. In the case of Figure 72, only that hyperparameter combination was extended to 12 agents, as shown in that figure. Here, 6 agents are shown in all cases (in the case of Figure 72 the six agents that were originally trained are shown here). Figures 107 and 108 show the results of the exploration.

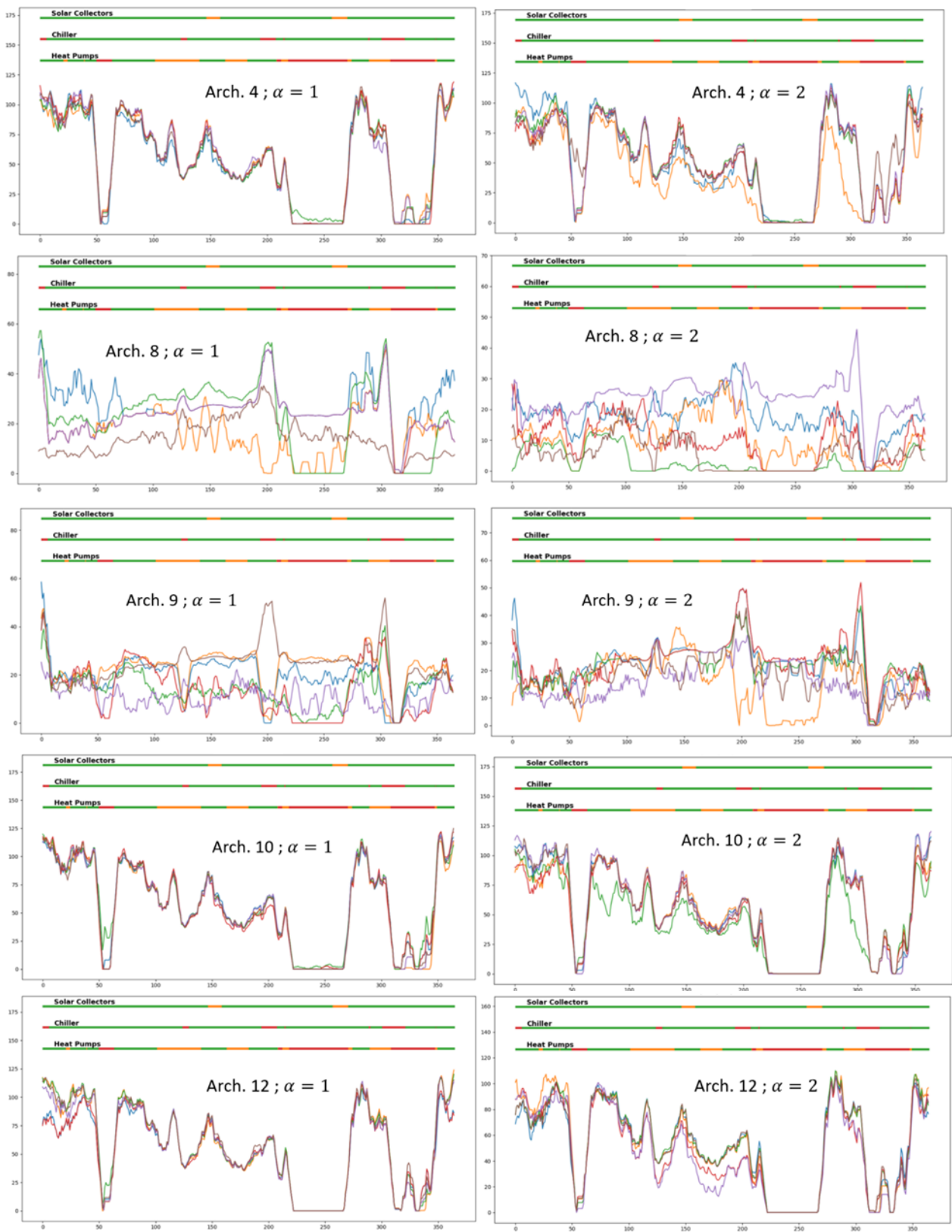


Figure 107: 10-variable states

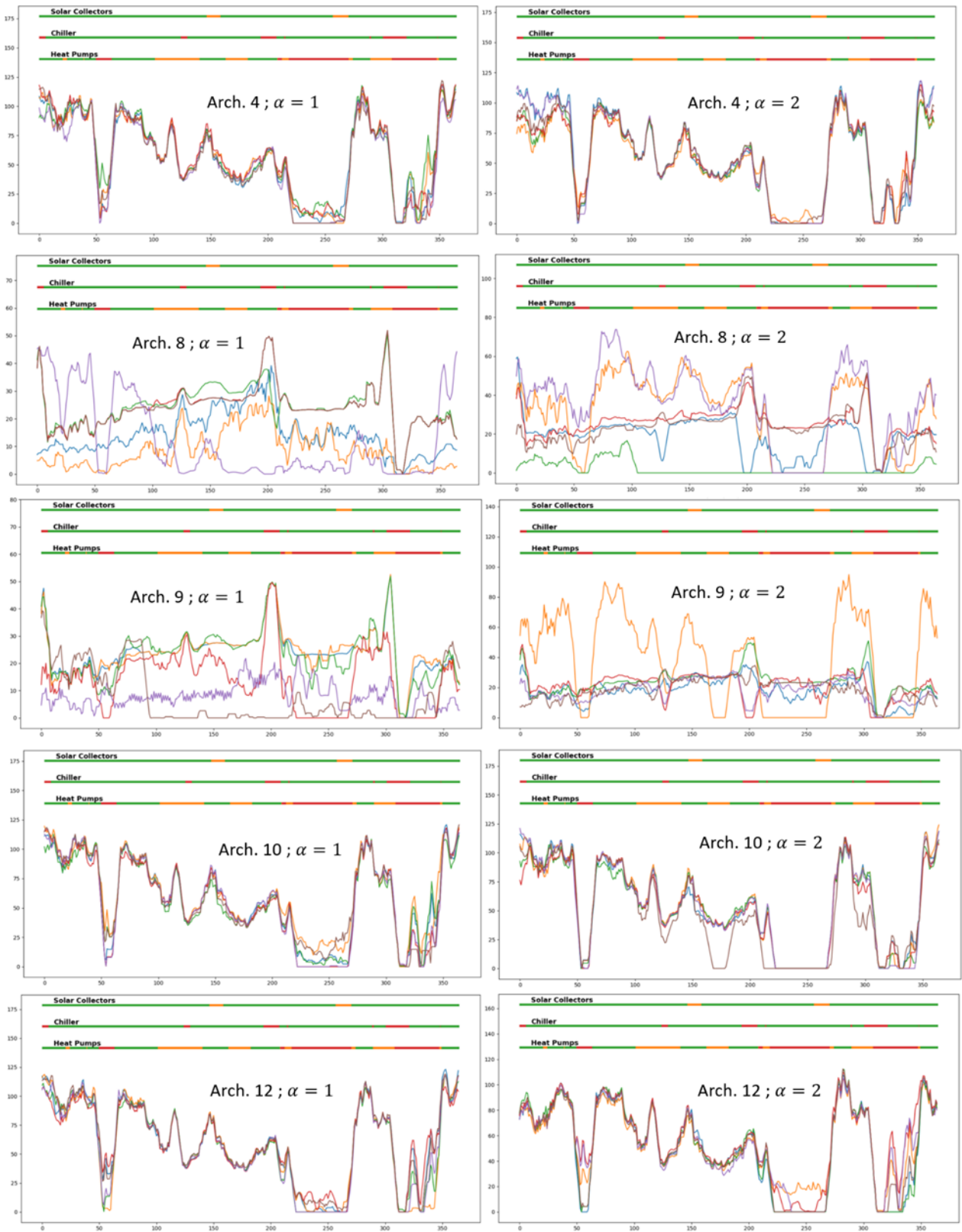


Figure 108: 15-variable states

Figures 78 through 84 were taken from an exploration that was done only with Architectures 11 and 12. The proportional prioritization method was used with α taking the values 1 and 2. 10-variables states as well as 15-variable states were tested. Six agents were trained with each

hyperparameter combination. Figures 109 through 114 show all the results of the exploration process.

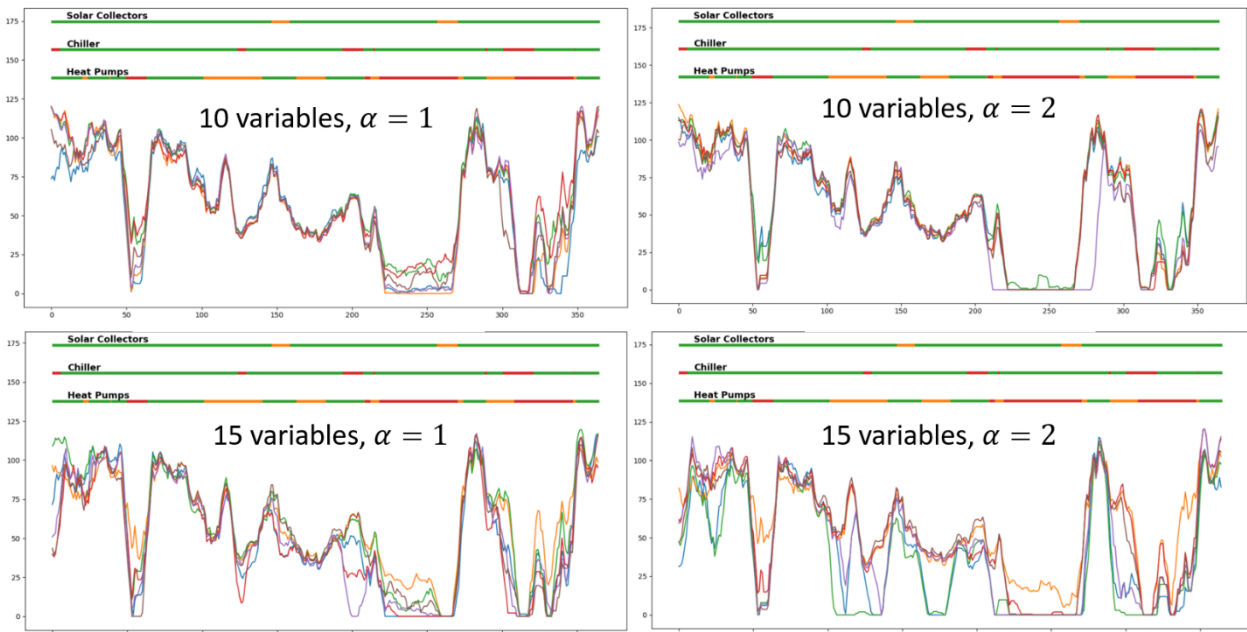


Figure 109: Cycle 1; Architecture 11.

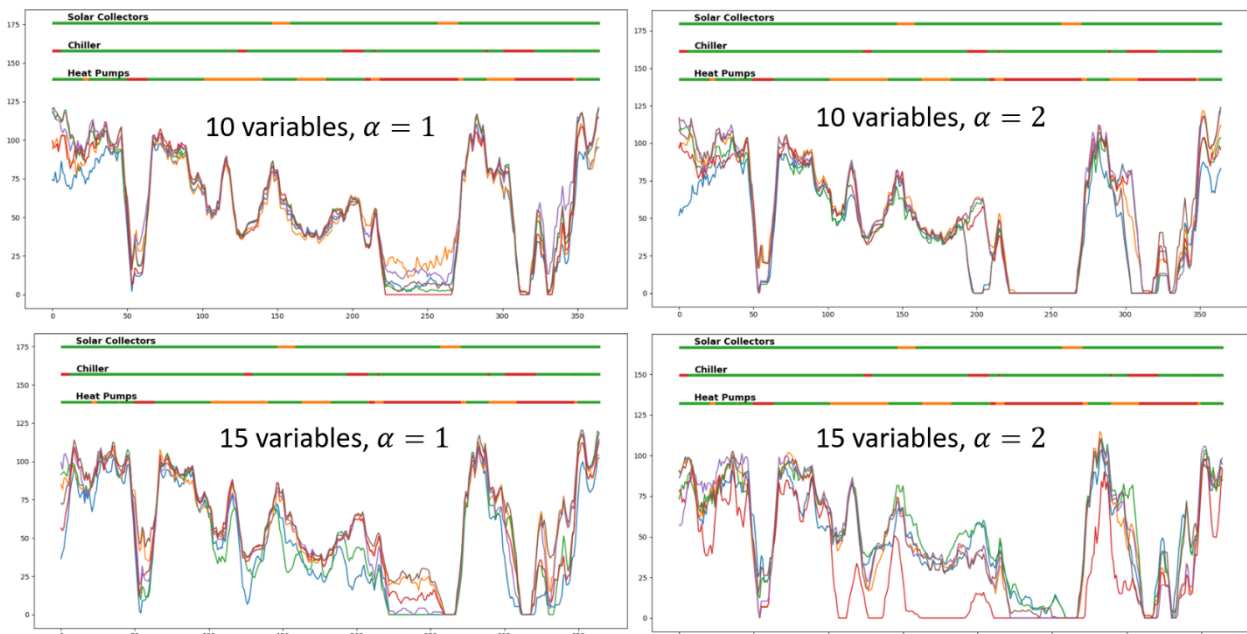


Figure 110: Cycle 1; Architecture 12.

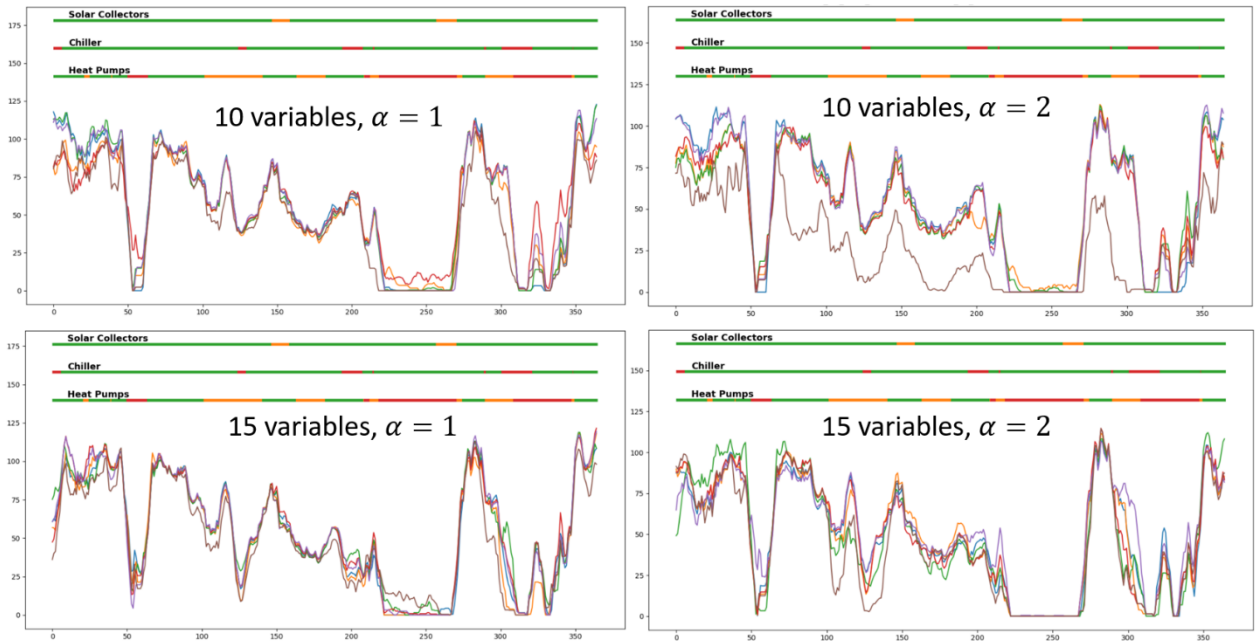


Figure 111: Cycle 2; Architecture 11.

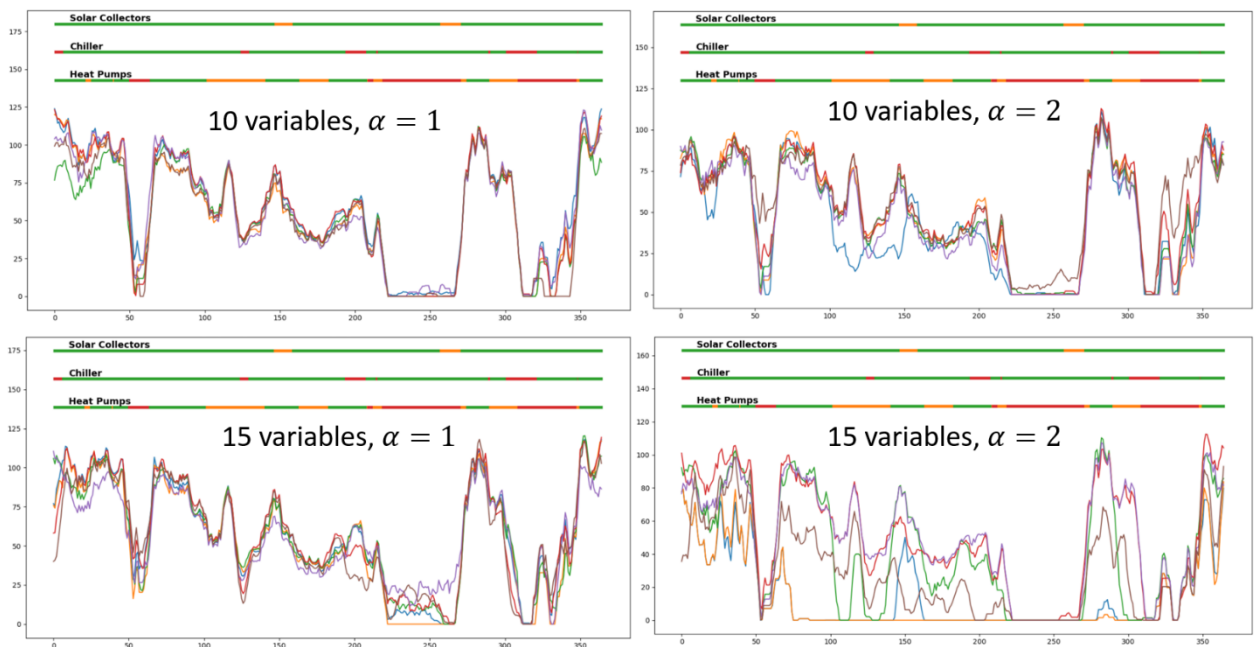


Figure 112: Cycle 2; Architecture 12.

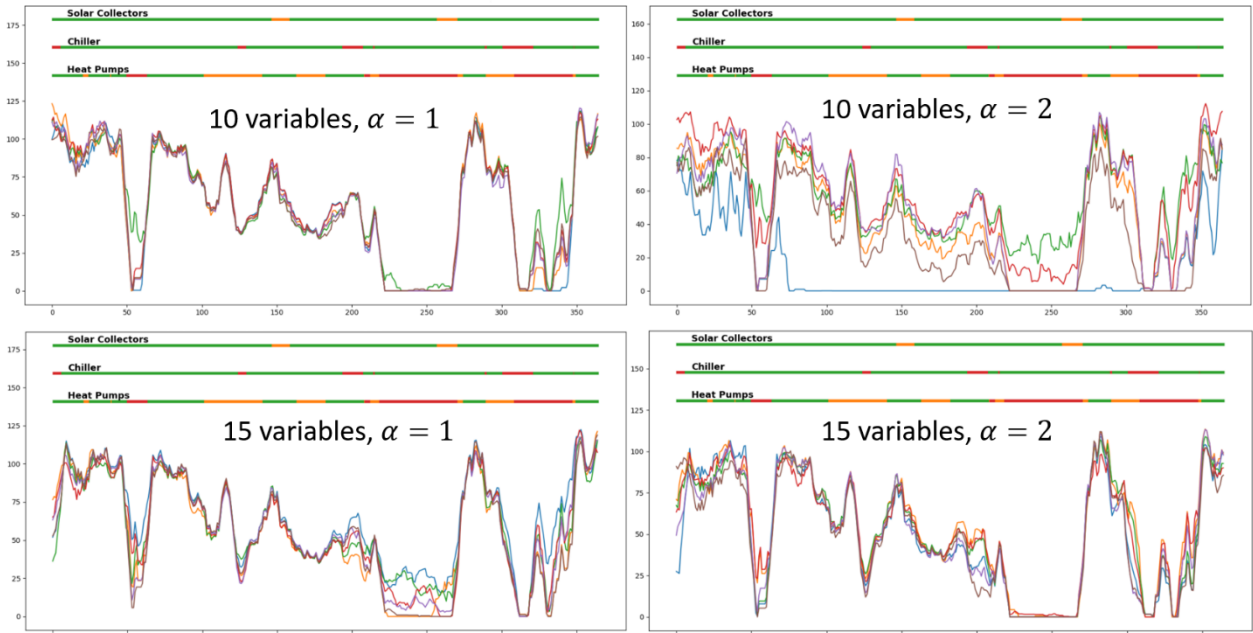


Figure 113: Cycle 3; Architecture 11.

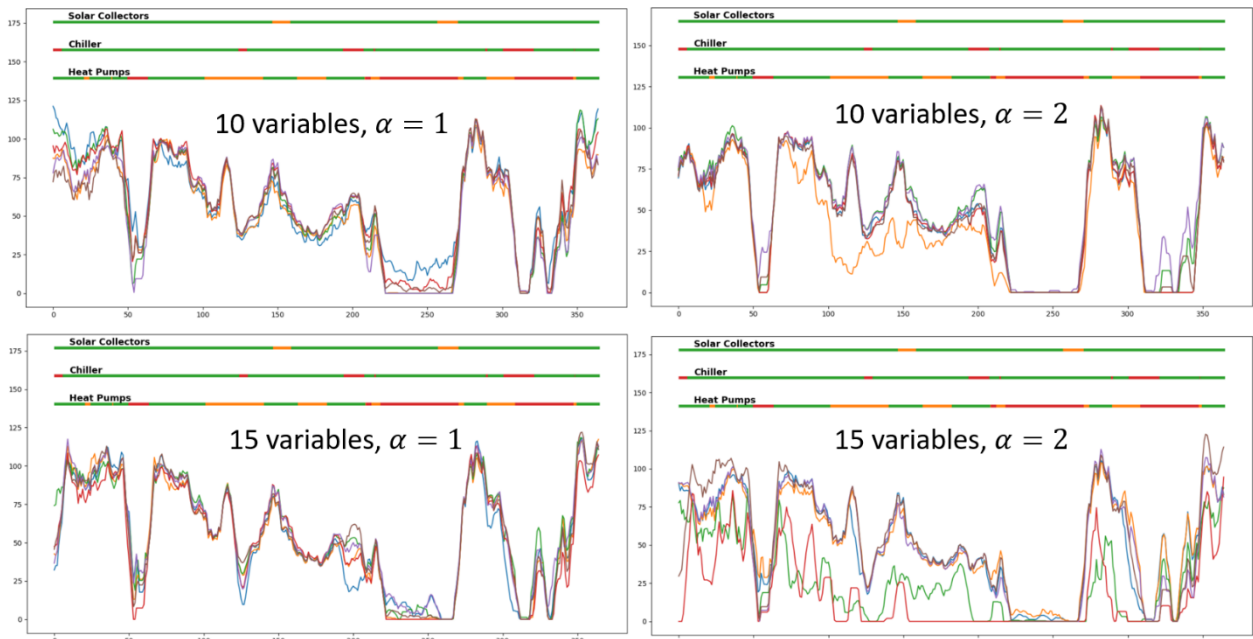


Figure 114: Cycle 3; Architecture 12.

In Section 6.5.3 the alternative Markov chains were proposed and used to train the agents. Architectures 4, 11 and 12 were tested. Both proportional and ranked-based prioritization methods were used. For the proportional method, α took the values 0.2, 0.5, 0.8, 1.0, 1.5 and 2.0. With the rank-based method, α took the values 0.2, 0.5 and 0.8. The value $\alpha = 0$ was tested as well; this implies not using any prioritization method. Six agents were trained with each hyperparameter combination. Figures 115 through 117 show all the results of Section 6.5.3.

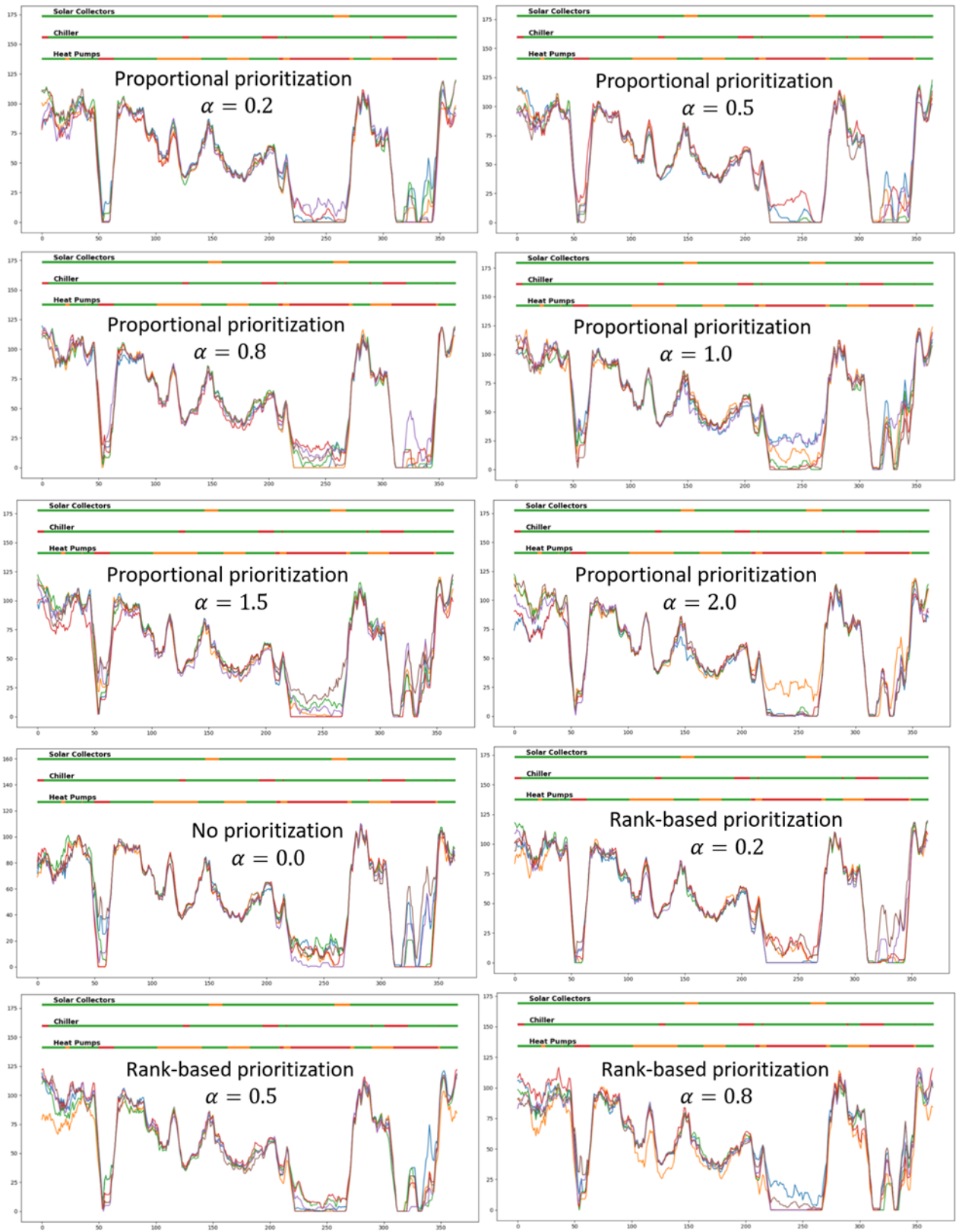


Figure 115: Architecture 4

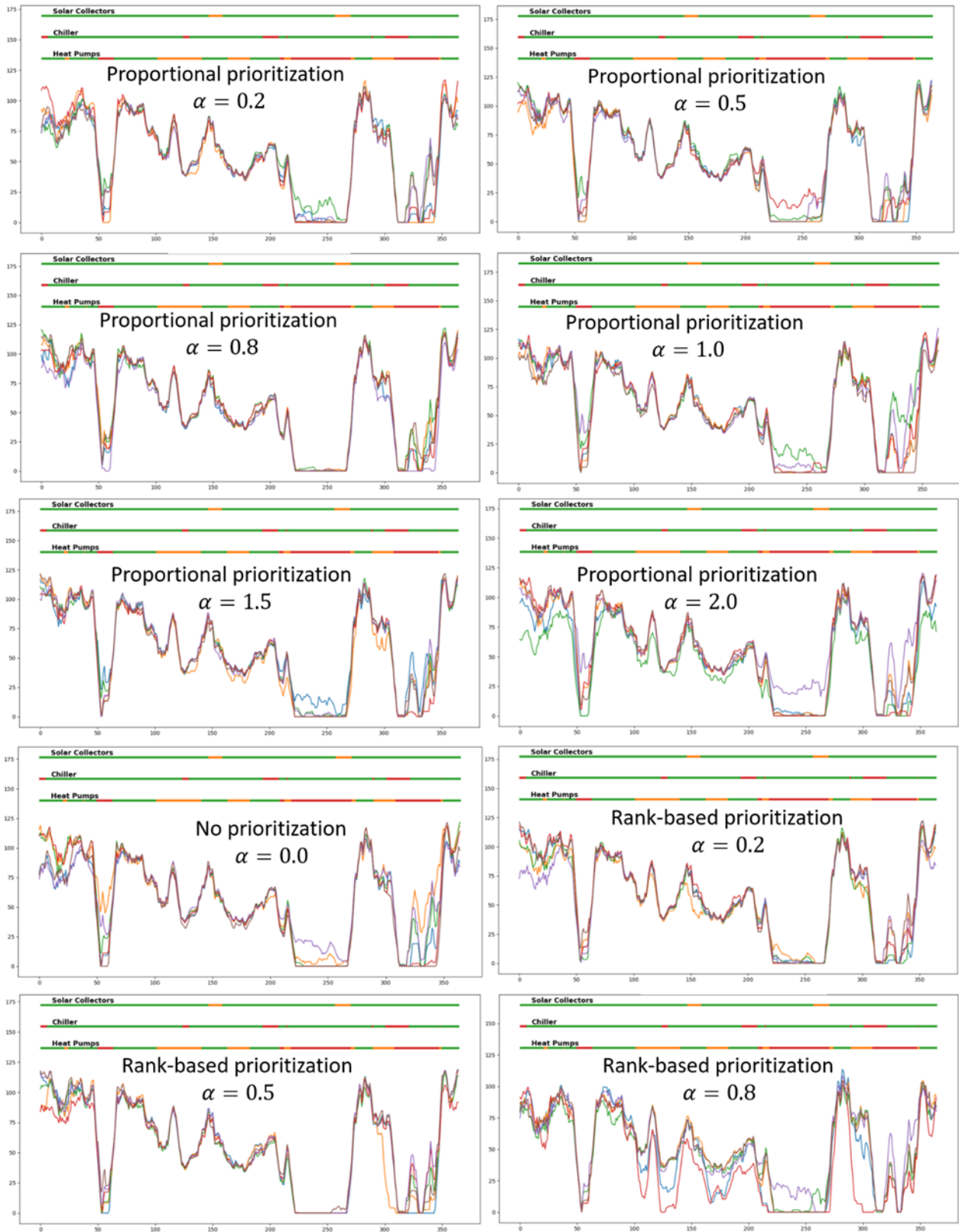


Figure 116: Architecture 11

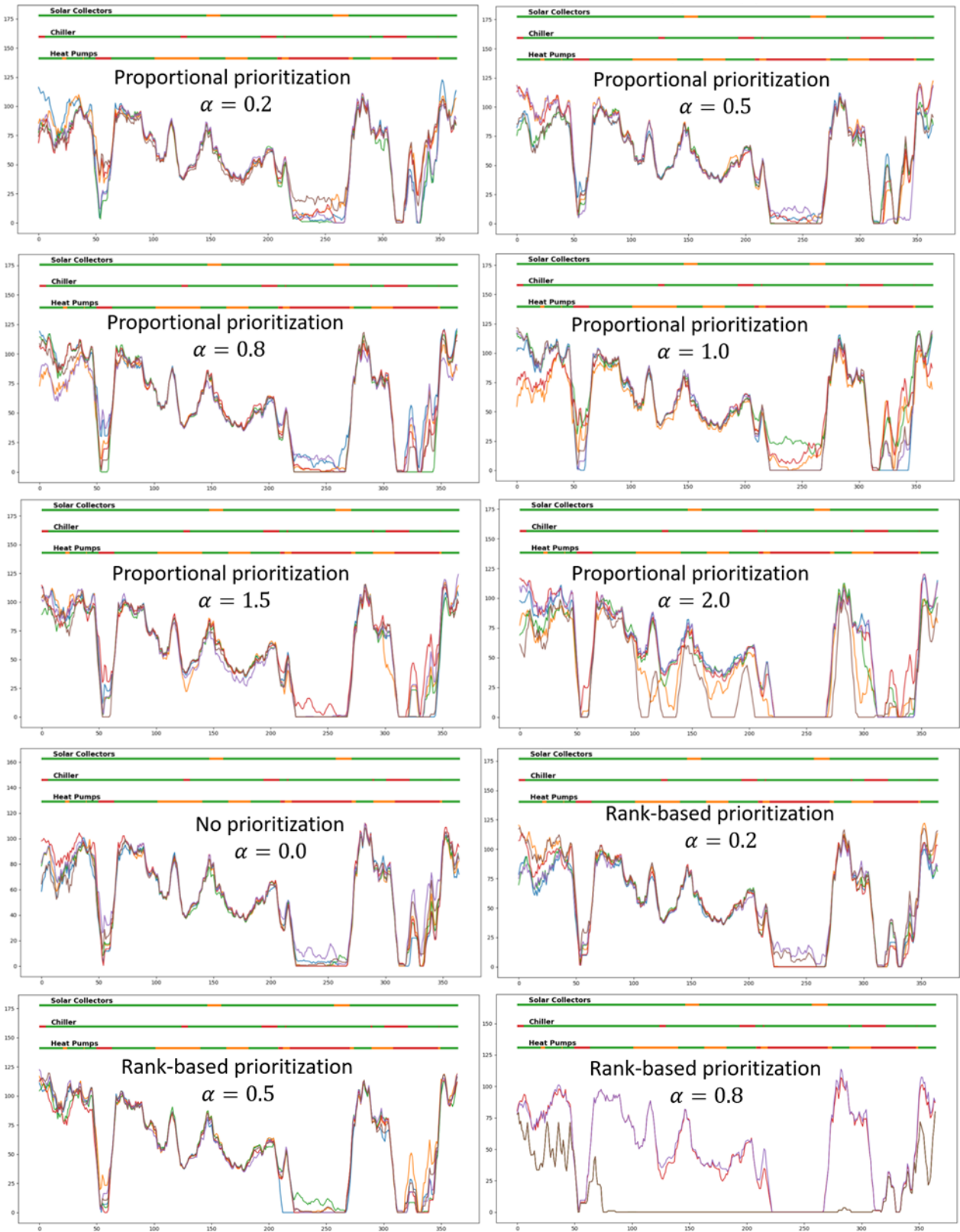


Figure 117: Architecture 12

In section 6.5.4, as already discussed, the momentum factor was increased to 0.9 while using Architecture 12. The experiment of decreasing the learning rate to 0.0005 was also carried out, but the results were not good like in the case of the momentum factor. In both cases, the training method

used in Section 6.5.1 was used (i.e. with the Markov chains of the real system and with 16 possible actions). Figures 118 and 119 show the results.

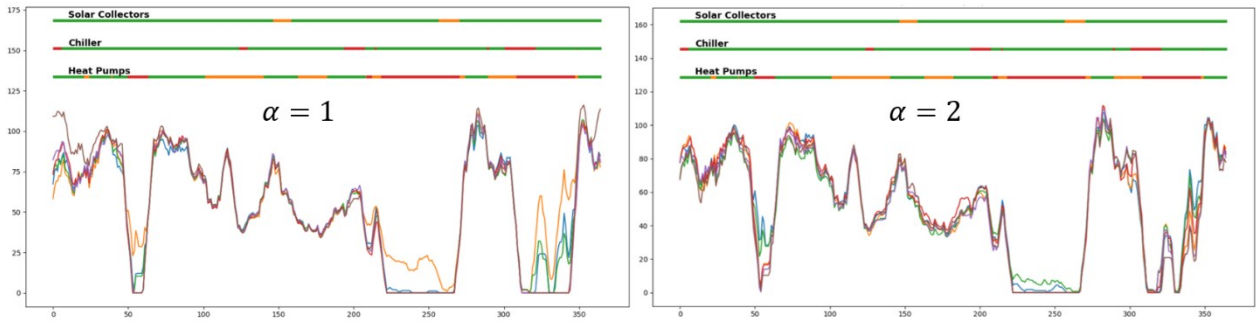


Figure 118: Learning rate = 0.0005; Momentum factor = 0.8

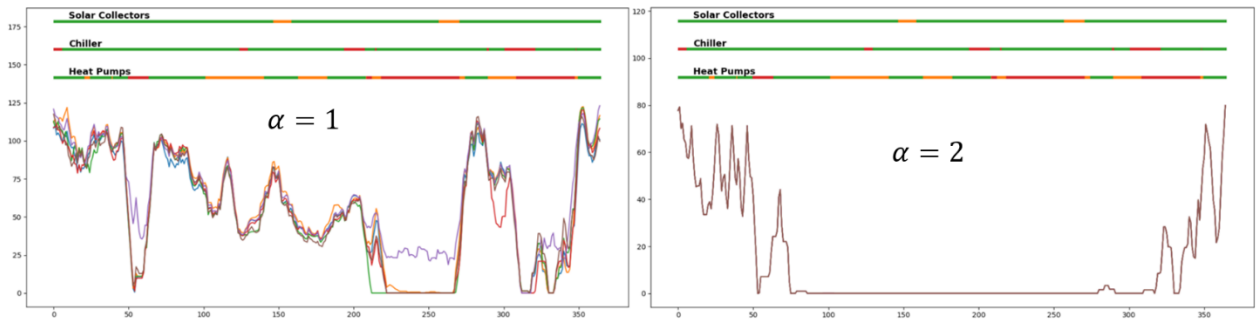


Figure 119: Learning rate = 0.001; Momentum factor = 0.9