Norwegian University
of Life Sciences

Master's Thesis 2019 30 ECTS
Faculty of Science and Technology

Reinforcement learning for grid control in an electric distribution system

Vegard Ulriksen Solberg
Environmental Physics and Renewable Energy

Preface

This thesis marks the end of my studies at the Norwegian University of Life Sciences. I have learned and experienced a lot during my 5 years in Ås, and I can look back at many great memories. The combination of motivated lecturers, hardworking students and kind friends have been essential for making my time here as good as it was.

I would like to thank my supervisors Oliver Tomic and Kristian Liland for valuable discussions and input in the writing process of this thesis. I would also like to show my gratitude to my supervisors at Statnett, Boye Annfelt Høverstad and Leif Warland, for helpful and constructive guidance, and for giving me the idea of using reinforcement learning in an electric power system. I would also thank my friends and family for the support during the thesis writing. Especially, I would like to thank Helene for always being there throughout the ups and downs this semester.

Vegard Ulriksen Solberg

Ås, 13.05.2019

Abstract

The increasing amount of variable renewable energy (VRE) sources such as solar and wind power in the power mix brings new challenges to existing power system infrastructure. A fundamental property of an electric power system is that the production of power at all times must be consumed somewhere in the grid. Therefore, excess power from VRE must be exported and consumed elsewhere, which can break the power capacity in distribution lines and damage the voltage quality in an electric grid. A workaround this problem is that consumers shift their power consumption pattern such that more solar power is consumed locally during daytime. Methods for achieving a change in consumption patterns are called demand response programs.

The purpose of this thesis is to make a Python implementation of an automatic and simplified demand response program by using reinforcement learning (RL), a subcategory of machine learning. The RL algorithm is allowed to increase or decrease the power consumption every hour in an electric grid that has a high amount of local solar production and high peak demand. Decreasing the power consumption can for instance correspond to a collection of electric vehicles postponing the charging to later in the day. Once the RL algorithm has modified the power consumption, the resulting line currents and voltages in grid are calculated. The goal for the algorithm is to learn a behaviour that reduces the number of current and voltage violations in the grid.

The trained RL algorithm is found to reduce the number of safety violations in the grid by 14 % in a test simulation. However, investigating the results reveals that the RL algorithm only avoids safety violations in hours of peak demand, and that it actually produces more violations during hours of peak solar production. The algorithm is better overall because more violations occur during the afternoon. Further investigation is needed to fine-tune the algorithm such that it behaves well in an entire day.

Sammendrag

Økt andel uregulerbar fornybar kraftproduksjon som sol- og vindenergi i energimiksen gir utfordringer for eksisterende infrastruktur i et elektrisk kraftnett. En grunnleggende egenskap i det elektriske kraftsystemet er at all produsert kraft alltid i sanntid må forbrukes et sted i kraftnettet. Overskudsenergi fra solproduksjon må derfor transporteres ut på nettet, som i verste fall kan bryte kapasiteten i kraftledningene og ødelegge spenningskvaliteten i kraftnettet. En løsning på dette problemet er at forbrukere forskyver forbruksmønsteret sitt slik at mer solkraft forbrukes lokalt på dagtid, slik at kraften ikke må eksporteres til kraftnettet. Metoder som har som mål å endre forbruksmønsteret kalles *program for forbrukerfleksibilitet*.

Hovedmålet i denne masteroppgaven er å lage en Python-implementasjon av et automatisk og forenklet program for forbrukerfleksibilitet ved hjelp av forsterkende læring (FL), en underkategori av maskinlæring. FL-algoritmen får lov til å øke eller minke kraftforbruket hver time i et kraftnett med høy lokal produksjon av solkraft og høyt forbruk på ettermidagen. Et eksempel på å minke kraftforbruket kan være å utsette ladningen av flere elbiler til senere. Når FL-algoritmen har modifisert kraftforbruket i nettet, så kalkuleres de påfølgende verdiene for strøm og spenning. Målet til algoritmen er å lære seg en strategi som reduserer antall ganger verdiene for strøm og spenning går utenfor sine respektive sikkerhetsmarginer.

Den trente FL-algoritmen reduserer antall sikkerhetsavvik i kraftnettet med 14 % i test-simuleringen. Det viser seg imidlertid at algoritmen kun klarer å redusere antallet sikkerhetsavvik sent på ettermiddagen, når forbruket er på sitt høyeste. Algoritmen gjør situasjonen verre i timer med høy kraftproduksjon fra solceller. Totalt sett er den trente algoritmen bedre siden det er flere sikkerhetsavvik i timer med høyt forbruk. Videre undersøkelser trengs for å justere algoritmen slik at den lærer seg en strategi som fungerer hele døgnet.

Contents

Chapter 1

Introduction

The electric transmission system is an infrastructure that is vital for the modern society as virtually everything depends on a reliable and secure supply of electric power. Statnett is the transmission system operator in Norway and is responsible for stable supply of electric power and the maintenance of 11 000 kilometers of high voltage transmission lines. The growing share of variable renewable energy sources is changing the dynamics in the power system and offers both new challenges and opportunities. A fundamental property of an electric power grid is that the electric supply and demand must equal each other at all times. In other words, the electric power generated from all power plants, wind parks and solar farms must in real-time be consumed somewhere in the grid, either by consumers or in the form of losses. The power balance is a self-fulfilling fact that always holds, but the power flow can damage the grid and electric equipment if it is not controlled appropriately. If too much power is produced, the voltages in the grid will increase in such a way that electric devices will draw more power from the grid. It is Statnett's task to balance the production and supply at all times. Norway has a somewhat easier job in terms of system control compared to other nations because over 94 % of the electric energy production comes from hydroelectric power plants [1]. Hydropower plants are flexible and can change their power production faster and cheaper than thermal power plants running on nuclear, coal and gas. As renewable energy increases, the flexibility and stability in the power system decreases. Thermal and hydroelectric power plants have large spinning masses (the turbines and generators) that gives inertia to the power system. They naturally resist changes in frequency of the voltage in the grid, which is a very convenient property. This is unfortunately not a feature solar power or wind power have. This is one of the reasons that the job of the transmission system operator becomes more complicated as uncontrollable renewable energy grows.

Another problem with distributed energy resources such as solar power production is that peak production normally occurs around noon, when the sun is at its highest, while peak power consumption in residential areas is in the afternoon. In some cases, the consequence of this is that power during daytime must be exported out to the central grid due to excess power from solar production, and later it must be imported to meet the peak power demand. As a result, there are two periods of the day where residential consumers are highly dependent on the electrical grid. As solar production becomes more prevalent in residential areas, the amount of power that must be exported out during daytime increases, possibly challenging the capacity of distribution lines. Similarly, the amount of power that must be imported in the afternoon can grow over time due to city growth in urban areas. A possible consequence is that the existing grid must be upgraded to support higher power flow, which is an investment that is very costly and in some sense

unnecessary. The consumers could be self-sustained in terms of energy in day, but they must rely on the grid because production and consumption at all times must equal each other.

A way to cope with the export/import problem is to consume energy in periods with high solar power production, instead of in the afternoon. In other word, one could shift the consumption pattern in a normal day. Naturally, all consumption is not flexible and cannot be shifted to noon, but devices such as electric vehicles, dishwashers and washing machines do not necessarily need to consume power in the afternoon. The methods for impacting the demand pattern of consumers are called *demand response programs*. The effect of a successful demand response program is that less power must be exported and imported during a normal day, possibly avoiding a costly upgrade of distribution system infrastructure. In addition, renewable energy production can grow faster as the existing power system infrastructure is no longer a bottleneck. This is especially important considering the urgent climate crisis we experience, where emissions of carbon-dioxide must be reduced quickly. Although demand response programs have great benefits, it is not clear how they should be controlled.

Recent advances in reinforcement learning, a subcategory of machine learning, have showed that reinforcement learning algorithms can master complex control tasks such as learning to play games only from pixel inputs and learning chess solely through self-play [2][3]. It is natural to wonder if the advances in reinforcement learning also can applied to electric power systems and demand response.

The scope of this thesis is to explore the use of a reinforcement learning algorithm to control an electric power system through a simplified and ideal program of demand response. The reinforcement learning algorithm used is called deep deterministic policy gradient (DDPG), and the goal of this thesis is to answer the following research question (RQ):

> **RQ**: Is the DDPG algorithm able to reduce the number of safety violations in a grid with high peak solar power production and high peak demand by the means of demand response?

First, a theoretic background for electric power systems and reinforcement learning is given in chapter 2 and 3 respectively. Chapter 4 presents a detailed problem description, before an overview of the state of the art for reinforcement learning and demand response is given in chapter 5. Chapter 6 describes how make a Python implementation of a reinforcement learning algorithm that controls an electric grid. Lastly, the DDPG algorithm is applied on a test case and the results are presented in chapter 7 and discussed in chapter 8. The most relevant parts of the Python code can be found in appendix B.

Chapter 2

Power system

The reinforcement agent is operating within an electrical power system and it is therefore necessary to give an introduction to the electrical grid and relevant quantities describing it. The theory presented in this chapter and more can be found in Alexandra Von Meier's book *Electric power systems: a conceptual introduction* [4].

2.1 Electric circuit theory

Voltage, current and power

An electric circuit is a model that describes how electrical power is transferred from an electrical source unit to a load. An example of a source is a power socket on the wall and a load can be a light bulb or a vacuum cleaner. A model of a simple electric circuit is shown in figure 2.1, where U is placed next to the electrical source and R next to the load. The electric transmission system is analogous to this configuration, except it with several electrical sources (power plants) and loads (cities) connected.

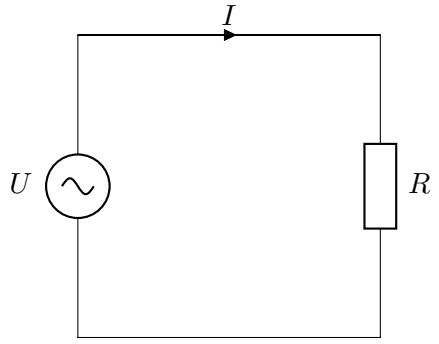


Figure 2.1: Simple electric circuit with voltage source U, current I and resistance R.

U is the voltage in the circuit and is a measure of the potential energy between charges at each terminal of the voltage source. Volt (V) is the unit for voltage, which is equivalent to joule per coulomb. The current I flowing in the wire is a measure for the amount of charges passing through a cross section of the circuit wire per second. The unit for current is ampere (A) or coulomb per second. The resistance R is a measure for how much an electric load, such as a light bulb, resists the flow of electric charges. The unit for resistance is ohm, denoted by Ω. The magnitudes of the voltage, current and

resistance are governed by Ohm's law

$$U = RI \tag{2.1}$$

where U is the voltage, R is the resistance and I is the current flowing. Another version of this is equation is found by introducing the admittance Y which is defined as the inverse of the resistance R, i.e. $Y = 1/R$. The admittance Y is a measure for how a load allows the flow charges in a circuit. The unit for admittance is siemens (S). By using the admittance Y, Ohm's law can be expressed as

$$I = UY \tag{2.2}$$

The power P an electric load consumes can be found easily by multiplying the current I and voltage drop U over the load

$$P = UI \tag{2.3}$$

The unit for power is watt (W) or joule per second.

Kirchoff's laws

Figure 2.2 shows a circuit with several branches and loads

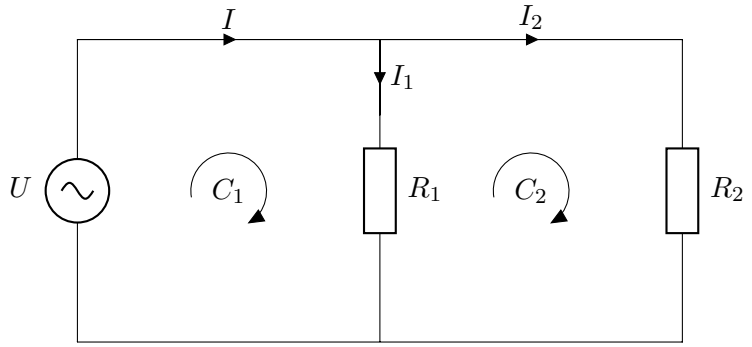


Figure 2.2: Simple electric circuit with several branches and resistors

The current I will split when it reaches and intersection, such that the total current flowing into the node equal the total current flowing out from it. Referring to figure 2.2 we have that $I = I_1 + I_2$. This conservation of current is called Kirchoff's 1st law or simply Kirchhoff current law. With the introduction of branches in a circuit, there will also be closed loops. In figure 2.2 there are two closed loops C_1 and C_2. Kirchoff's 2nd law, also known as Kirchoffs's voltage law, states that the voltage U over the components in a closed loop C is equal to zero.

$$\sum_i U_i = 0, \;\; U_i \in C \tag{2.4}$$

For the two loops C_1, C_2 in figure 2.2 we have

$$\begin{aligned} U + I_1 R_1 &= 0 \\ I_1 R_1 - I_2 R_2 &= 0 \end{aligned} \tag{2.5}$$

2.2 Reactive components

There are electrical loads that cause a phase shift between the current and voltage in an electric circuit. For instance, the circuit shown in figure 2.3 has a capacitor as load.

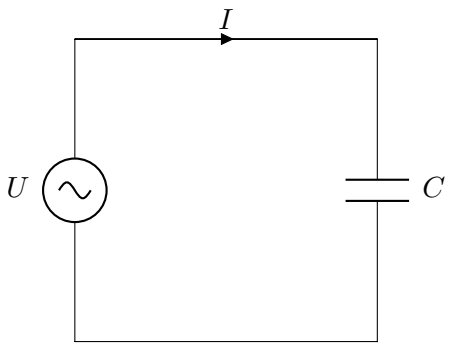


Figure 2.3: Circuit with a capacitor as load

A capacitor is a component that can store an electrical charge Q when a direct current (DC) voltage source U is applied over it. A capacitor is characterised by the charge Q it can hold for a given DC voltage U. This quantity is called the capacitance C of the capacitor and is given by

$$C = \frac{Q}{U} \tag{2.6}$$

where Q is the charge stored by the capacitor and U is the applied DC voltage. The units for capacitance is farad (F). Capacitors are relevant because they appear in the electric modelling of a transmission line. Applying Kirchoff's voltage law to this circuit when the source is a sinusoidal signal $U\sin(\omega t)$ and using equation (2.6) gives

$$U\sin(\omega t) - \frac{Q(t)}{C} = 0, \quad Q(0) = 0 \tag{2.7}$$

Recognising that the current I is the derivative of Q with respect to time reveals that the current is given as

$$I(t) = U\omega C\sin(\omega t + \pi/2) \tag{2.8}$$

The solution shows that the current I is phase shifted 90 degrees ahead of the voltage U. It is convenient to introduce the capacitive reactance X_c as

$$X_c = \frac{1}{\omega C} \tag{2.9}$$

where ω is the angular frequency of the signal and C is the capacitance of the capacitor. The circuit current can now be expressed on the same form as Ohm's law

$$I(t) = \frac{U}{X_c}\sin(\omega t + \pi/2) \tag{2.10}$$

The circuit shown in figure 2.4 is another example of a reactive component that phase shifts the current. The load is an electromagnetic coil, also called an inductor, and appears in a wide range of electric components. For instance, an electric transmission line is mainly modelled as an inductor. The voltage across an electromagnetic coil is proportional to the time derivative of the current flowing through it due to Faraday's law of induction.

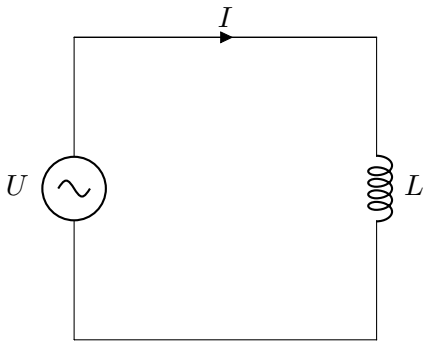


Figure 2.4: Circuit with an electromagnetic coil as load

The proportional constant is called the inductance L of the coil and it is given in henry H. Similarly as with the capacitor, Kirchoff's voltage law gives rise to a differential equation. Let the voltage source be $U\sin(\omega t)$

$$U\sin(\omega t) - LI'(t) = 0, \quad I(0) = 0 \tag{2.11}$$

The solution of equation (2.11) is

$$I(t) = \frac{U}{\omega L}\sin(\omega t - \pi/2) \tag{2.12}$$

The current is shifted 90 degrees behind the voltage in this case. We see that both inductors and capacitor shift the current by 90 degrees, but in different direction. The inductive reactance X_c is defined as

$$X_c = \omega L \tag{2.13}$$

where ω is the angular frequency of the voltage and L is the inductance of the coil.

2.3 Reactive power

For a purely resistive load with no phase shift between current and voltage, the instantaneous power transferred to the load as given by equation (2.3) is always positive, as shown in figure 2.5. This power is called active power, and is measured in watt (W). This is not the case for reactive loads. The phase shift between current and voltage results in a pulsating power between the source and the load, as shown in figure 2.5. The pulsating power resulting from reactive loads is called reactive power Q and has unit var, to distinguish it from unidirectional power flow. Figure 2.5 shows that the instantaneous power resulting from an inductive and capacitive load are opposite of each other. As a result, a circuit with equal inductive and capacitive reactance connected in parallel will draw zero instantaneous power from the source. By convention, a capacitive load is defined to supply reactive power while an inductive load is a reactive power consumer. Consequently, overhead transmission lines are considered reactive consumers, because they are mainly inductive.

2.4 Voltage, current and power as complex numbers

Current, voltage, impedance and power are all expressed as complex numbers in an AC electric power system. Consider the circuit shown in figure 2.6. The resistor will draw

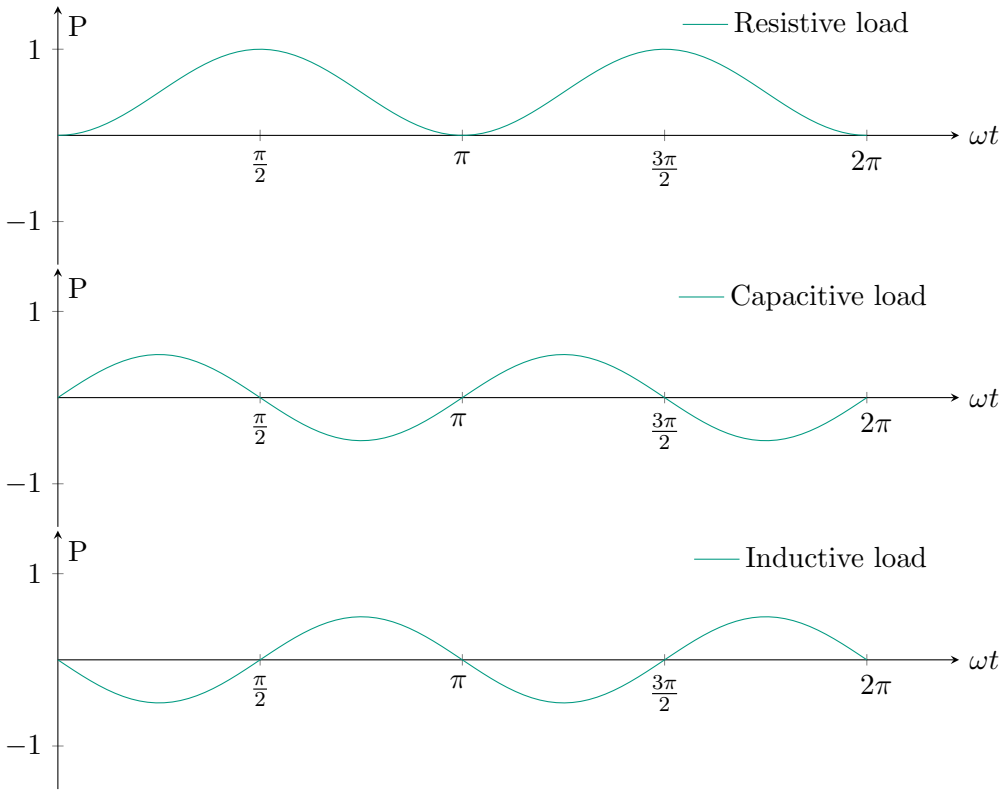


Figure 2.5: Instantaneous power transferred to the load in a circuit with a pure resistive, inductive and capacitive load that are equal in size

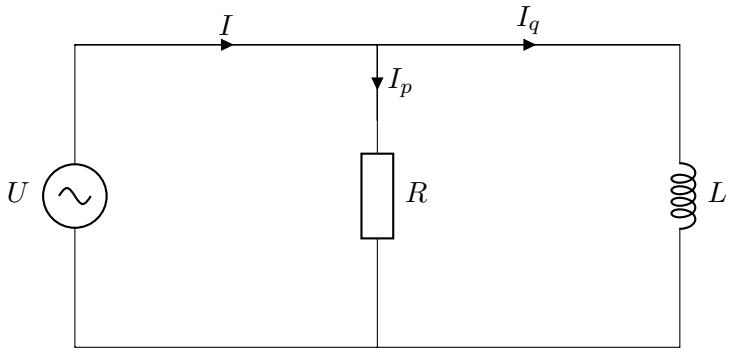


Figure 2.6: Circuit with a resistor and an inductance connected i parallel

a current I_p that is in phase with the source voltage. The inductor will draw a current I_q that lags the voltage by 90 degrees. The resultant current drawn from the source will therefore be a linear combination of two phase shifted sinusoidal signals. The equations for summing phase-shifted sinusoidal signals are complicated, and not easy to work with. This motivates the introduction of complex numbers. Euler's formula states that a complex number A can be expressed by

$$A = Re^{j\omega t} = R\cos(\omega t) + jR\sin(\omega t) \tag{2.14}$$

where e is the base of the natural logarithm, j is the imaginary unit and R is the magnitude of A. The currents I_p and I_q can therefore be expressed as the imaginary part of two complex numbers A_1, A_2. Treating the currents as complex numbers makes it

easier to sum them because they form a right-angled triangle in the complex plane, as shown in figure 2.7. One can always convert back to the sinusoidal current by taking the imaginary part of the complex resultant current I.

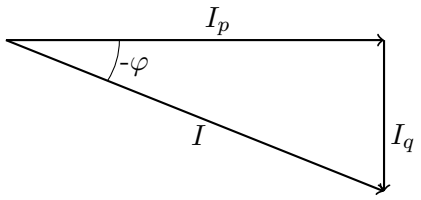


Figure 2.7: Current as complex numbers. I_q is the current drawn by an inductor, and lags the voltage by 90 degrees. I_p is the current drawn by a resistive load and is in phase with the voltage. The current I is commonly defined to be lagging, so that $I = |I|e^{-j\varphi}$. By this definition, φ is a positive real number when the current is lagging the voltage.

The resultant current I drawn from the voltage source can now be expressed as

$$I = |I|e^{-j\varphi} = I_p - jI_q \tag{2.15}$$

where $|I|$ and φ respectively are the magnitude and phase shift of the current. The current I_q is 90^o behind I_p and is therefore multiplied by $-j$, which corresponds to a 90^o clockwise rotation in the complex plane. Comparing equation (2.14) and (2.15) shows that the angular frequency ω is removed, and that the current is simply considered a complex constant. This has do with the fact that the sum of two synchronous sinusoidal signals (same ω) inherits the same frequency. One can therefore consider the current at some arbitrary time, say t=0, and consider it a constant because the phase shift φ is independent of time. The resultant complex current can also be expressed as $|I|\angle -\varphi$. The current magnitude $|I|$ is given by the Pythagorean theorem.

$$|I| = \sqrt{I_p^2 + I_q^2} \tag{2.16}$$

The phase shift φ is described by the trigonometric relation

$$\tan\varphi = -\frac{I_q}{I_p} \tag{2.17}$$

Inductive and capacitive reactances can also be expressed as complex numbers. A coil phase shifts the current 90 degrees behind the voltage. Considering the current I_q as a complex number in Ohm's law gives that the inductive reactance can be expressed as jX_L, because multiplication by j corresponds to a 90 degree anticlockwise rotation in the complex plane

$$U = I \cdot jX_L \tag{2.18}$$

Similarly, a capacitive reactance X_c phase shifts the current 90 degrees a head of the current. Therefore, it can be expressed as $-jX_c$

$$U = I \cdot (-jX_c) \tag{2.19}$$

The complex notation also works for a circuit with resistive, inductive and capacitive components connected in series, as shown in figure 2.8.

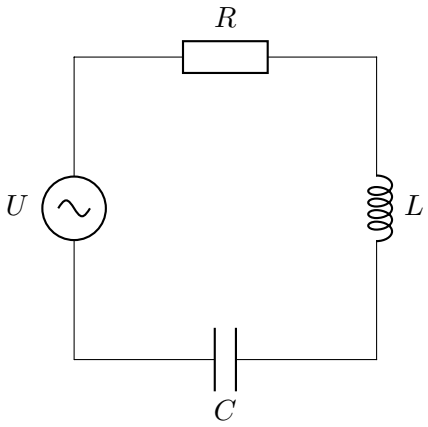


Figure 2.8: AC circuit with a resistor, capacitor and coil connected in series

The sum of the resistance and reactances is called the impedance Z of the circuit and is given as

$$Z = R + jX_L - jX_c \tag{2.20}$$

Using the complex impedance Z in Ohm's law describes both the resultant magnitude $|I|$ and phase angle of the current φ in an AC system.

The active and reactive power flowing in a line are also expressed as a complex number. The apparent power S flowing in a line is defined to be

$$S = UI^* = P + jQ = |S|e^{j\varphi} \tag{2.21}$$

Where U is the voltage, I^* is the complex conjugate of the current, P and Q are respectively the active and reactive power supplied by the voltage source. The conjugation of the current is a convenience to make the reactive power Q be a positive number when the current is lagging the voltage, as is the case for an inductor. According to this definition, an inductor consumes reactive power while a capacitor is a reactive source. The magnitude of the apparent power $|S|$ is

$$|S| = \sqrt{P^2 + Q^2} \tag{2.22}$$

where P and Q are the active and reactive power respectively. The angle of the apparent power S is the same as the phase angle of the current I.

2.5 Three-phase electric power

A conventional electrical power line transfers power in three conductors that have equal voltage magnitude and are phase-shifted 120 degrees with respect to each other. An electrical circuit of a three-phase power system is shown is figure 2.9.

The three conductors are not drawn in illustrations of a power system infrastructure, but replaced by a one-line diagram. A one-line diagram is well suited for illustrating power flow, but it should be noted that there in reality is a three-phase system with three conductors. The voltage magnitude given in a one-line diagram can be expressed either by the phase voltage $|U_{ph}|$ which is the voltage relative to ground, or the voltage between the lines $|U_{LL}|$. The relation between them in a balanced three-phase system is

$$|U_{LL}| = \sqrt{3}|U_{ph}| \tag{2.23}$$

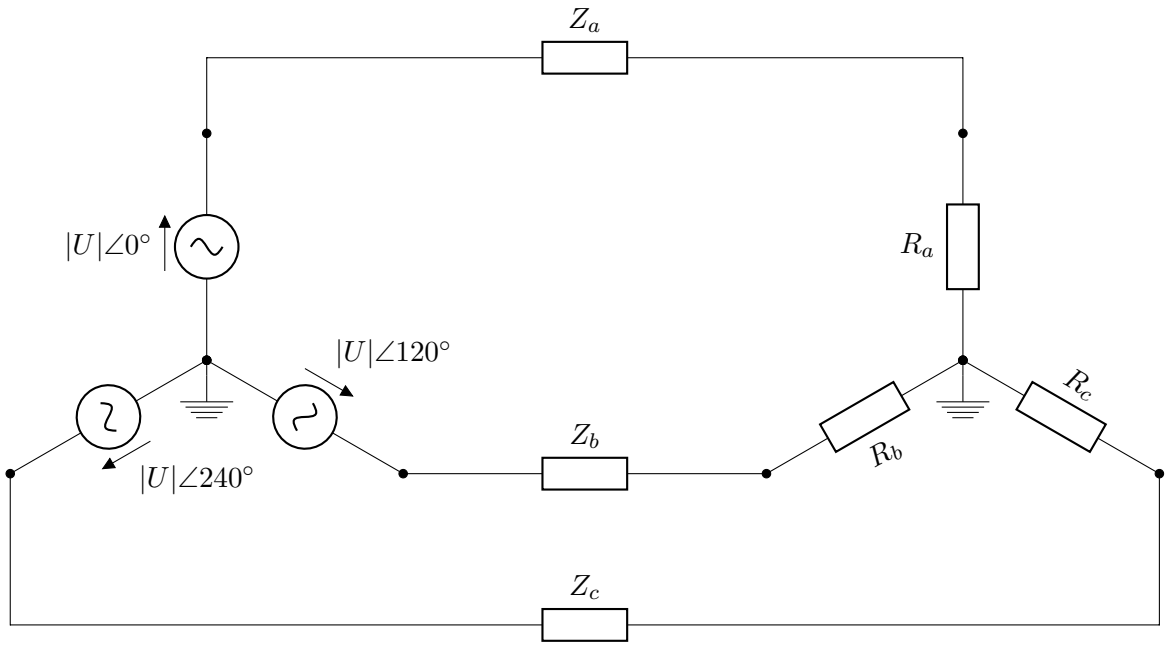


Figure 2.9: Electric model of a three-phase transmission system. Three conductors transfer power to the loads R_a, R_b and R_c

The apparent power $|S|$ transferred in a three-phase system is given by

$$|S| = \sqrt{3}|U_{LL}||I| \tag{2.24}$$

where $|U_{LL}|$ is the line voltage magnitude and $|I|$ is the current magnitude in each conductor. The active power P and reactive power Q is determined by

$$\begin{aligned} P &= |S|\cos\varphi \\ Q &= |S|\sin\varphi \end{aligned} \tag{2.25}$$

where φ is the angle between the phase voltage U_{ph} and the current in the same phase. A symmetric system is assumed, so it is arbitrary which phase is used.

2.6 Per-unit system

An electric power system generally consist of lines with different voltage magnitudes that can range from a few kV to many hundreds of kV. As a result, it is difficult to see if the power flow in a line is high or low, because it must always be compared to the voltage level. To simplify this, quantities are generally measured relative to base values. This is called the per-unit system. Specifically,

$$\begin{aligned} U &= |U_b|U_{pu} \\ I &= |I_b|I_{pu} \\ S &= |S_b|S_{pu} \\ Z &= |Z_b|Z_{pu} \end{aligned} \tag{2.26}$$

The subscripts b and pu denote base and per-unit respectively. The per-unit quantities are still complex numbers, but are dimensionless. A line is given by a nominal value for

apparent power $|S_b|$ and voltage magnitude $|U_b|$. The base current $|I_b|$ is defined as

$$|I_b| = \frac{|S_b|}{\sqrt{3}|U_b|} \tag{2.27}$$

By the definition in equation (2.27), the apparent power in per-unit S_{pu} is given as

$$S_{pu} = \frac{S}{|S_b|} = \frac{\sqrt{3}UI}{\sqrt{3}|U_b||I_b|} = U_{pu}I_{pu} \tag{2.28}$$

In other words, the apparent power takes the form of a one-phase system, although it in reality is a three-phase system. A similar motivation gives the definition of the per-unit impedance base $|Z_b|$

$$|Z_b| = \frac{|U_b|}{\sqrt{3}|I_b|} \tag{2.29}$$

By the definition in equation (2.29), the per-unit voltage U_{pu} is given as

$$U_{pu} = \frac{U}{|U_b|} = \frac{\sqrt{3}ZI}{\sqrt{3}|Z_b||I_b|} = Z_{pu}I_{pu} \tag{2.30}$$

The per-unit notation thus result in the same relation between current, voltage and impedance as Ohm's law in a one-phase system.

2.7 Components in the power system

An electrical power system consists of a set of nodes $\mathbf{N}$, commonly referred to as buses, and a set of branches $\mathbf{L}$ that connects some or all of the buses. The branches between buses can be power lines, cables, transformers or other power electronics equipment. The buses and branches define the topology of the electrical power system. Figure 2.10 is an illustration of an electric power system consisting of 5 buses and 7 branches. Note that the branches are one-line representations of a three-phase system. Formally, this network is described as

$$\begin{aligned} \mathbf{N} &= \{1, 2, 3, 4, 5\} \\ \mathbf{L} &= \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 5), (4, 5)\} \end{aligned} \tag{2.31}$$

A bus is electrically modelled as a point where electrical power is injected. A positive injected power corresponds to generation of power at that bus. This is the case for a bus that is connected to a power plant. A negative power injection corresponds to a consumption of power, as would be the case for a bus connected to a factory producing aluminium. The sum of power production and consumption determines the net power injection to that bus. Bus k in an electric power system is physically described by four quantities: The voltage magnitude $|U_k|$, the voltage angle δ_k, the active power injection P_k and the reactive power injection Q_k.

2.8 Two-bus system

Figure 2.11 displays power flow between to buses connected by a transmission line. From the left we have active power P_1 and reactive power Q_1 flowing. The power flows through the line and continues out from bus 2 as P_2 and Q_2. U_1 is the complex representation of

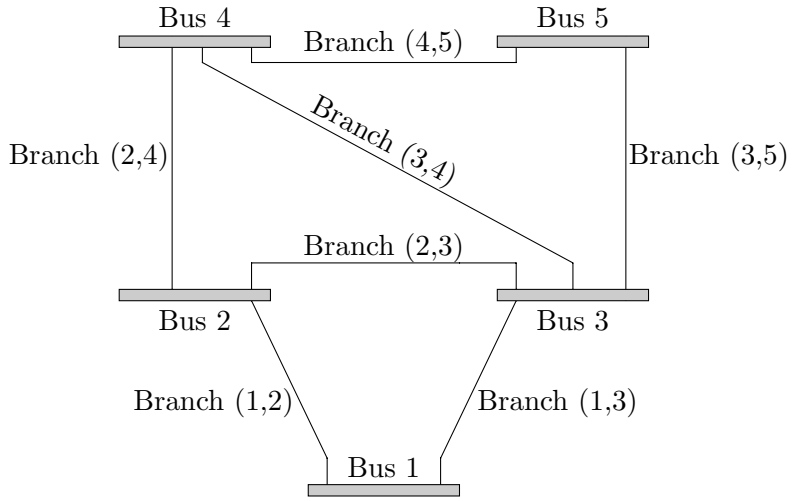


Figure 2.10: Example of a network with 5 buses and 7 branches connecting them

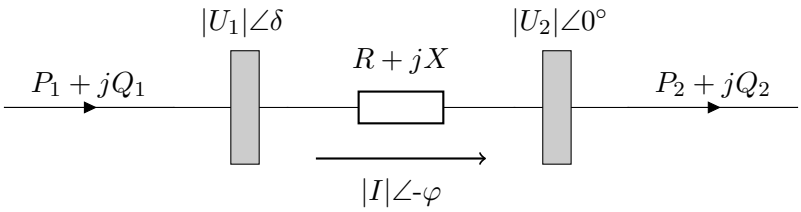


Figure 2.11: Simple two-bus system connected by a line. P and Q are the active and reactive power flow, R and X are the resistance and reactance of the line, U is the voltage and I is the current flowing

the voltage at bus 1, $U_1 = |U_1|e^{j\delta}$. Similarly, $U_2 = |U_2|e^{j0} = |U_2|$. The relation between voltage and current for the system is given by Ohm's law.

$$U_1 - U_2 = (R + jX)I \tag{2.32}$$

R is the resistance and X is the reactance of the line, U_1 and U_2 are the voltages at bus 1 and 2, and I is the current flowing in the line. The impedance Z of the line can be expressed as $Z = |Z|e^{j\epsilon}$ where $\tan(\epsilon) = X/R$. The current I is commonly defined to be lagging, so that $I = |I|e^{-j\varphi}$. By this definition, φ is a positive real number when the current is lagging the voltage. In figure 2.12, the current, voltages and impedance are drawn as phasors in the complex plane for a line with zero resistance (R=0).

Using Ohm's law, the current I in the line can be expressed as

$$I = \frac{U_1 - U_2}{Z} = \frac{|U_1|e^{j\delta} - |U_2|}{|Z|e^{j\epsilon}} = \frac{|U_1|e^{j(\delta-\epsilon)} - |U_2|e^{-j\epsilon}}{|Z|} \tag{2.33}$$

where $|U_1|$ and $|U_2|$ are the voltage magnitudes at bus 1 and 2, I is the current flowing in the line and $|Z|$ is the impedance magnitude for the line. Using the definition of power flow in (2.3) on bus 1 gives the active power P_1

$$P_1 = \frac{|U_1|^2}{|Z|}cos(\epsilon) - \frac{|U_1||U_2|}{|Z|}cos(\epsilon - \delta) \tag{2.34}$$

To make it easier to investigate a situation with zero resistive losses in the line (R = 0), it is convenient to introduce a loss angle $\alpha = \tan(R/X)$. By the sum of angles in a triangle,

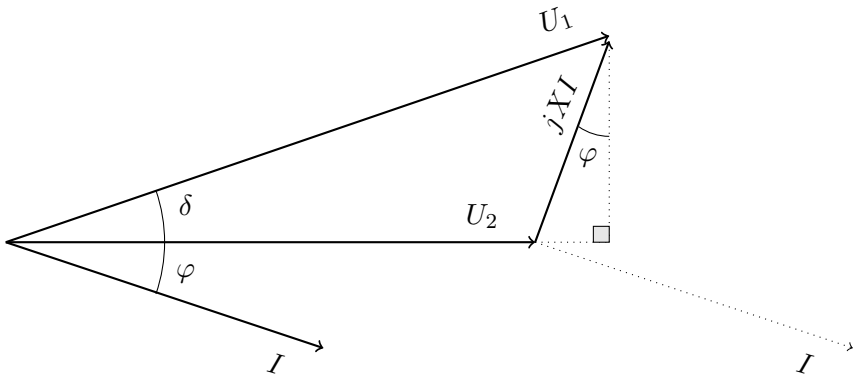


Figure 2.12: Phasors of current and voltages in a two-bus network connected by line with 0 resistance (R=0)

we have the relation $\alpha + \epsilon = \pi/2$. By using the loss angle α, the active power flow at bus 1 P_1 can be expressed as

$$P_1 = \frac{|U_1|^2}{|Z|} sin(\alpha) + \frac{|U_1||U_2|}{|Z|} sin(\delta - \alpha) \tag{2.35}$$

A line without resistive losses can now be examined simply by setting $\alpha = 0$. The resulting active power flow P_1 at bus 1 reduces to

$$P_1 = \frac{|U_1||U_2|}{|Z|} sin(\delta) \tag{2.36}$$

where $|U_1|$ and $|U_2|$ are the voltage magnitudes at bus 1 and 2, $|Z|$ is the impedance magnitude of the line and δ is the phase angle at bus 1 with respect to bus 2. The takeaway from this analysis is that the direction of the active power flow is determined by the phase angle δ of the voltages between the buses. In a two-bus system, the bus with the leading voltage is supplying active power, while the lagging voltage is receiving. Another thing to note is that for an AC power system the voltage magnitude between the buses may differ in size, and active power can still flow in both directions. This would not be possible in a DC power system. Figure 2.13 shows the relation between a loss free power line and a sample line in `pandapower`, a Python toolkit for power flow analysis. We see that the active power P for a line with resistive losses is also mainly controlled by the voltage angle δ. This is especially true around 0 voltage angle δ, which is the region it generally is located.

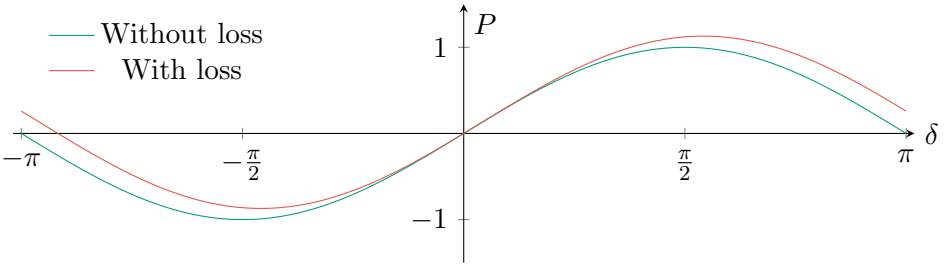


Figure 2.13: Active power flow P between two buses as a function of voltage angle with no losses ($\alpha = 0$) and a loss angle α of 0.13. The bus with a leading voltage angle δ supplies active power, while the lagging bus voltage is receiving.

A similar argument will give that the reactive power flow Q_1 at bus 1 for a line without resistive losses is given by

$$Q_1 = \frac{|U_1|}{|Z|}(|U_1| - |U_2|) \tag{2.37}$$

where $|U_1|$ and $|U_2|$ are the voltage magnitudes at bus 1 and 2 respectively and $|Z|$ is the impedance magnitude of the line. The voltage angle δ is assumed to be small so that $cos(\delta) \approx 1$. Equation (2.37) shows that the direction of the reactive power is determined by the difference of the voltage magnitudes. In other words, the bus with the highest voltage magnitude is supplying reactive power in a two-bus system.

2.9 The power flow equations

Figure 2.14 is an electrical model of bus k in a power system consisting of n buses. I_k is the net injected current to the grid from bus k [5]. The current flowing from bus k to bus i is $I_i = (U_k - U_i)Y_{ki}$. The current flowing out of bus k must equal the injected current I_k. This gives

$$I_k = U_k Y_{k0} + \sum_{i=1, i\neq k}^{n} (U_k - U_i)Y_{ki} \tag{2.38}$$

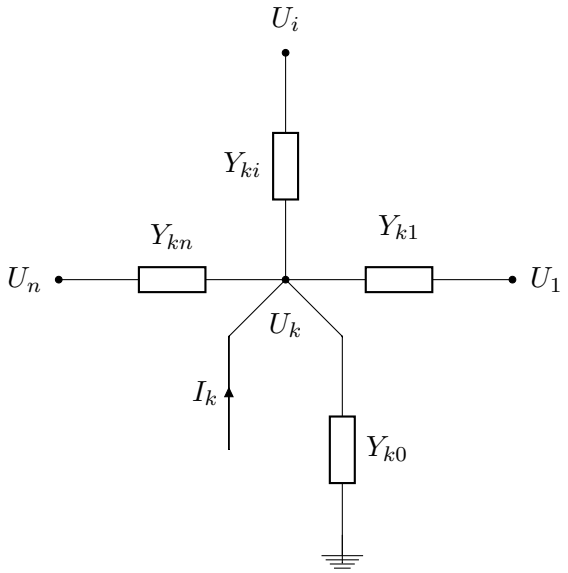


Figure 2.14: Electrical model of a bus in a grid consisting of n buses

Let the admittance components y_{ki} be defined as

$$y_{ki} = \begin{cases} -Y_{ki} & \text{if } k \neq i \\ Y_{k0} + \sum_{j=1, j\neq k}^{n} Y_{kj} & \text{if } k = i \end{cases} \tag{2.39}$$

It is now possible to express the injected current I_k as a linear combination of all the bus voltages in the system

$$I_k = \sum_{i=1}^{n} y_{ki} U_i \tag{2.40}$$

Equation (2.40) is valid for any bus k. For the case when bus k does not inject any current, $I_k = 0$. The voltages and injected currents at all buses can therefore be expressed compactly as a matrix equation with the bus voltages ordered in a vector $U_{\text{bus}} \in \mathbb{C}^n$

$$I_{\text{bus}} = Y_{\text{bus}} U_{\text{bus}} \tag{2.41}$$

where $I_{bus} \in \mathbb{C}^n$ is a vector whose k-th component is the injected current at bus k. $Y_{bus} \in \mathbb{C}^{nxn}$ is called the admittance matrix of the network and its elements are given by equation (2.39). It is more convenient to work with power injection instead of current injection at a bus. The complex conjugate injected apparent power S_k^* at bus k can be found by using equation (2.40)

$$S_k^* = U_k^* I_k = U_k^* \sum_{i=1}^{n} y_{ki} U_i = \sum_{i=1}^{n} y_{ki} U_i U_k^* \tag{2.42}$$

Let

$$\begin{aligned} U_i &= |U_i| e^{j\delta_i}, \;\; i = 1, ..., n \\ y_{ki} &= |y_{ki}| e^{j\lambda_{ki}}, \;\; i = 1, ..., n, \;\; k = 1, ..., n \\ \delta_{ki} &= \delta_k - \delta_i, \;\; i = 1, ..., n, \;\; k = 1, ..., n \end{aligned} \tag{2.43}$$

Substitution into equation (2.42) gives

$$S_k^* = \sum_{i=1}^{n} |y_{ki}||U_i||U_k| e^{-j(\delta_{ki} - \lambda_{ki})} \tag{2.44}$$

Applying Euler's formula on equation (2.44) gives the injections of real power P_k and reactive power Q_k at bus k.

$$\begin{aligned} P_k &= \sum_{i=1}^{n} |y_{ki}||U_i||U_k| \cos(\delta_{ki} - \lambda_{ki}) \\ Q_k &= \sum_{i=1}^{n} |y_{ki}||U_i||U_k| \sin(\delta_{ki} - \lambda_{ki}) \end{aligned} \tag{2.45}$$

The equations in (2.45) are known as the power flow equations. The admittance y_{ki} of a line is known and is treated as a constant. By inspection of (2.45), it is evident that there are 4n variables ($|U_k|, \delta_k, P_k$ and Q_k at each bus) and 2n equations. Therefore, 2 variables must be specified at each bus to have a unique solution of the power flow equations.

Types of buses

There are three types of buses in an electric power system [5].

- Slack bus / reference bus

The voltage angle δ at the slack bus is defined to be 0 and the angles at other buses are measured relative to it. There is only one slack bus in an electrical power system.

- Load bus

A load bus is the most common bus in an electrical grid. The load buses have a fixed injection of active power P and reactive power Q, as this is determined by the demand in the market. As a result, it is also called a P-Q bus.

- Voltage controlled bus or generation bus

The voltage magnitude $|U|$ and active power $|P|$ are fixed for the voltage controlled bus, while the voltage angle δ and reactive power Q are unknown. It is also called a P-V bus. A bus connected to a hydro power plant is an example of a P-V bus.

Table 2.1 summarises known and unknown quantities at different bus types.

Table 2.1: Known and unknown quantities at different bus types in a system with n buses in total.

Bus type	Known quantities	Unknown quantities	Number of buses
Voltage control	P, $\|U\|$	Q, δ	m
Load bus	P, Q	$\|U\|$, δ	$n-m-1$
Reference bus	$\|U\|$, δ	P, Q	1

2.10 Electric model of a power line

A transmission line can be electrically modelled by using the π-equivalent circuit, as shown in figure 2.15

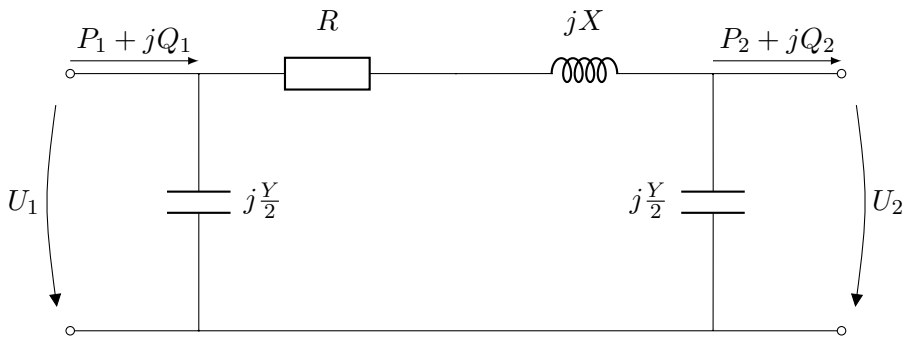


Figure 2.15: π-equivalant model for a transmission line.

The transmission line is modelled by a resistance and inductance in series. In addition, there are two shunt capacitors connected in parallel at each end of the line. The capacitors are there because the flow of charges will give an associated capacitance that is proportional with the line length. The π-model splits the admittance Y of the capacitor into two and puts one part on each end of the line.

Chapter 3

Reinforcement learning

Reinforcement learning is an algorithm that learns through trial and error. The system consists of an agent that observes a numerical state representing an environment and responds to that by taking an action. Simply put, the agent will get a positive reward when it takes good actions and negative rewards for bad actions. When the agent takes a bad action, it will be less likely to choose that action again later. Similarly, when it gets a positive reward it will more likely choose a similar action given the same observed state. By letting the agent see many states and explore different actions, it can eventually learn a behaviour that yields a lot of positive rewards.

Reinforcement algorithms are similar to how humans and animals learn. Imagine a dog seeing its owner holding a bag of treats. Obviously, the dog is keen on getting the treats, but is not sure what to do. The dog sees that the owner is putting his hand in front of its nose and yelling some command, but does not quit understand what to do. So, it simply tries doing something. First, it might try to lean forward and smell the hand. Sadly, this does not result in any treat. Therefore, it continuous to try different actions, until it eventually happens to lift its front paw in the hand of the owner. At last, it receives a tasty treat from the owner. It has learned what action to take to get a treat. Next, the owner might rotate its arm in front of the dog. The dog might try to lift its front paw again, since that worked last time. Sadly, it does not get a reward this time. Therefore, it starts to explore new actions until it after some time tries to spin around. Again, it receives a treat. It has now learned that simply raising its front paw does not always result in a treat. It has to evaluate its observation before taking an action.

The dog can also learn to be farsighted. For instance, when the dog hears that the doorbell rings, it wants to run to the door and bark in excitement. If the owner is smart, it can teach the dog to sit still when visitors come, and give the dog a treat when it behaves well. Over time, the dog will learn to suppress its excitement when the door rings, because it has learned that the long-term reward is greater if it behaves well. The dog is capable of reducing the short-term reward (barking in excitement) for a greater long-term reward (treat).

The dog training is similar to the mechanisms in a reinforcement learning algorithm. The dog is the agent that tries to figure out what actions to do, while the owner is the reward system. The advantage of a reinforcement algorithm is that it does not need a physical reward, but is happy with a numerical reward. In addition, a computer can experiment much more quickly and efficiently than a dog can.

3.1 Reinforcement learning and machine learning

Algorithms in machine learning and artificial intelligence are often divided into either supervised or unsupervised learning. Supervised learning is an algorithm using input data and labelled output data (target). The algorithm tries to map the input to the target in a manner that generalises well to unseen input data. Examples of supervised learning are regression and classification algorithms. Algorithms using unsupervised learning attempt to find structure in unlabelled data. In other words, the goal is not to find a mapping between input and output as there is no output data. Examples of unsupervised learning are clustering and anomaly detection. The terms *supervised* and *unsupervised* do not describe well the mechanisms of reinforcement learning algorithms. A reinforcement learning agent learns from interaction with an environment and receiving rewards based the action it takes. The agent's goal is not to use labelled data in some sense or explicitly finding general structures in the data. As a result, reinforcement learning is considered a category of its own [6]. However, it should be noted that there are reinforcement algorithms that use supervised learning in the learning process. The relation between supervised, unsupervised and reinforcement learning is shown in figure 3.1

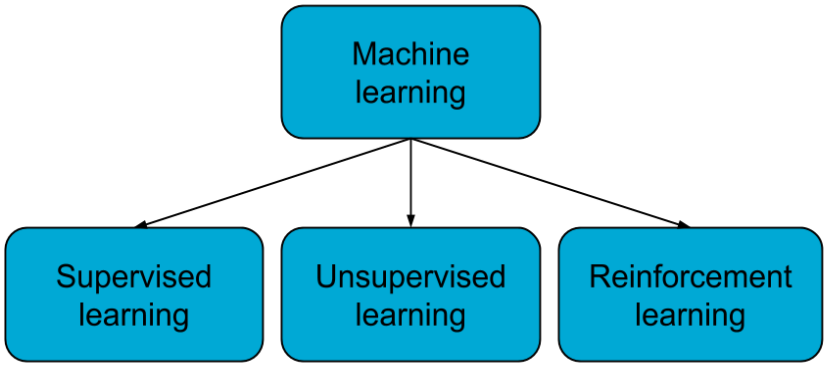


Figure 3.1: The three main categories of machine learning: supervised, unsupervised and reinforcement learning

3.2 Elements in a reinforcement algorithm

The agent and the environment with which it is interacting is fundamental to any reinforcement learning algorithm. As mentioned in the introduction, an agent is the decision maker that observes a state s and decides what action a to take. An example can be a self-driving vehicle that receives an observation s in the form of input from cameras and sensors placed on the vehicle. The observation s represents the state of the system, and based on that the self-driving car must choose an action a. The action might be to turn the wheel to the right, which in turn leads to a new observation from the cameras and sensors on the vehicle.

The set of possible actions and states are respectively called the action space $\mathcal{A}$ and state space $\mathcal{S}$. For some reinforcement learning tasks, such as chess, the action space depends on the state s. For instance, it is not allowed to castle if an opponent's piece is attacking your king. In such cases, the action space $\mathcal{A}(s)$ is given by the state s. The

action space in an electric power system is dependent on the state, for instance if a power plant or transmission line is out of operation. A central element in the reinforcement algorithm is the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that evaluates how "good" action a is in state s. For instance, the reward given to the agent in car race can be the speed of the car so that the agent is encouraged to drive fast. It is not always possible to give a reward after an action is taken, because it is not possible to say if the action was good or bad. For instance, it is hard to evaluate if a move is bad or good. In such cases, the reward can be the same as the outcome of the game: +1 for victory, 0 for draw and -1 for loss. This setup is termed Monte Carlo learning. A setup where rewards can be given after every action is called time-difference learning.

The goal at each time step t is to maximise the rewards in the future. How to formally define the reward maximising criterion depends on the nature of the task. Some tasks, such as playing a video game, are called episodic and have well-defined boundaries for initial and terminal state. On the other hand, the electric power system is a continuous task that never should end if the agent does its job. For continuous tasks, let the discounted return r_t^γ at time t be defined as

$$r_t^\gamma = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=t}^{\infty} \gamma^{k-t} r_{k+1} \tag{3.1}$$

where r_{t+1} is the reward received after taking action a_t in state s_t, and $\gamma \in [0, 1]$ is the discount factor. The goal of the agent at every time step t can be defined to maximise the discounted return r_t^γ. The gamma term is a hyper-parameter that can be tuned, and it determines how relevant future rewards are. If $\gamma = 0$, then the agent only considers the immediate reward as relevant. If $\gamma = 1$ then all future rewards count equally to the total future reward r_t^γ. For values between 0 and 1, the importance of a reward decreases exponentially with every time step. For instance, if $\gamma = 0.5$ the rewards for the next steps are weighted $0.5, 0.25, 0.125, ...$ Having γ smaller than 1 is also a mathematical convenience that ensures that the discounted return is finite in a continuous task, as long as the rewards are bounded.

3.3 Markov decision process

A Markov decision process is a mathematical framework describing sequential decision making and interaction with an environment, where the outcome can be stochastic [6]. The environment starts at $t = 0$ and is described by an initial state $s_0 \in \mathcal{S}$. The agent performs some action $a_0 \in \mathcal{A}$ and receives a reward $r_1 \in \mathcal{R} \subseteq \mathbb{R}$ based on how good that action is. The action a_0 interacts with the environment and gives a new state s_1. This starts the sequence of states, actions and rewards.

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, ... \tag{3.2}$$

The interaction between the agent and environment is visualised in figure 3.2 as a feedback loop. The loop continues until the environment reaches a terminal state, for instance when the self-driving car reaches its destination or if it crashes. The transitions from start state s_0 to terminal state s_T constitutes an episode in the reinforcement algorithm.

Formally, a finite Markov decision process $\mathcal{M}$ is a tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle \tag{3.3}$$

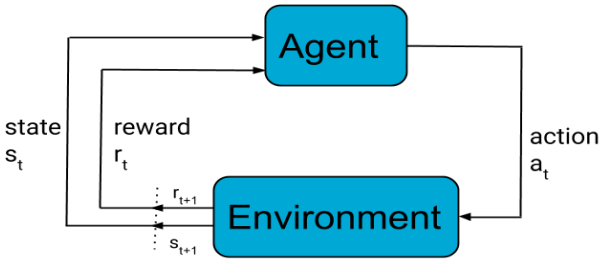


Figure 3.2: Interaction in a Markov decision process

where $\mathcal{S}$ and $\mathcal{A}$ respectively are finite sets of states and actions, $\mathcal{P}$ is the matrix with state transition probabilities, $\mathcal{R}$ is a reward function and γ is a discount factor [7]. The probability of transitioning to the next state s_{t+1} and receiving r_{t+1} only depends on the previous state s_t and action a_t in a Markov decision process [6]. Formally, a state s_t is Markov if and only if

$$\mathbb{P}[s_{t+1}|a_t,\ s_t] = \mathbb{P}[s_{t+1}|a_t, s_t,\ a_{t-1}, s_{t-1}, ..., a_0, s_0] \tag{3.4}$$

where $\mathbb{P}$ is the symbol for probability. This is called the Markov property of the state [7]. In other words, the history of states and actions leading up to the current state is not relevant for the probability of transitioning to state s_{t+1}. Let the transition function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ be the probability of transitioning from state s to s' and receiving reward r given the action a

$$p(s', r, s, a) = \mathbb{P}[s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a] \tag{3.5}$$

If the transition function p in (3.5) is known, it can be used for planning actions in a reinforcement algorithm.

3.4 Value and policy functions

The reinforcement agent selects an action in a given state through its policy π. The policy of the agent decides what action to take in a given state by mapping the state space to the action space, $\pi : \mathcal{S} \to \mathcal{A}$. The policy can both be deterministic and stochastic. A deterministic policy maps a given state to the same action every time, while a stochastic policy maps the state to a probability distribution over the action space. For the deterministic case, the policy function π is given by

$$\pi(s) = a \tag{3.6}$$

where a is the action chosen by the policy. For the stochastic case, it gives the probability of choosing action a in state s.

$$\pi(a|s) = \mathbb{P}(a|s) \tag{3.7}$$

A central tool in many reinforcement algorithms is to evaluate a certain state before taking an action. The state-value function V^π is defined as the expected discounted future

return in state s under the policy π.

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[r_t^{\gamma}|s = s_t] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s = s_t] \end{aligned} \tag{3.8}$$

The reason for including "under the policy π" is that a state is valued differently depending on the policy used. For instance, the start position in chess will be evaluated better under the policy of a chess grand master than under the policy of an amateur, because the grand master has a much higher expected future reward.

The action-value function Q^{π}, also called the Q-function, quantifies the expected discounted return given the action a_t in state s_t and that the policy π is followed thereafter. In other words, it can evaluate a specific action in a given state, in contrast to the value function V that only evaluates the state.

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= \mathbb{E}_{\pi}[r_t^{\gamma}|a = a_t, s = s_t] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|a = a_t, s = s_t] \end{aligned} \tag{3.9}$$

There is an important recursive relation between the action-value function in two successive states s_t and s_{t+1}, known as the Bellman equation. Assuming a deterministic policy π, we have

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|\ s_t, a_t] \\ &= \mathbb{E}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))|\ s_{t+1}] \end{aligned} \tag{3.10}$$

In other words, the action-value for state s_t and action a_t is the expected sum of the immediate reward r_t and the action-value in the next state. The Bellman equation is used in several reinforcement algorithms to guide the estimates of the Q-values closer to the true values. For instance, consider a case where the estimated Q-value Q_t is 4, with a discount factor of 0.8 . The agent takes and action, receives the reward +2 and evaluates the next state and action to be worth 5. According to Bellman's equation, we have that

$$2 + 0.8 \cdot 5 = 6 \neq 4 \tag{3.11}$$

The right-hand and left-hand side of the equation are not equal. The true Q-values satisfies the Bellman equation, so the estimate Q_t should be higher, and the difference of the right and left hand side can be used to update the estimate Q_t. In this case, the new estimate can be $4 + 0.1 \cdot (6 - 4) = 4.2$, where 0.1 is the learning rate.

The action-value function Q^{π} and state-value function V^{π} are similar to each other and can together be used to measure the advantage of an action a. The advantage $A^{\pi}(a, s)$ of action a in state s under policy π is defined as

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \tag{3.12}$$

If the advantage is positive, it means that it is better to take action a than following the action chosen by the policy in state s. Similarly, a negative advantage means that action a is worse than following the action chosen by the policy.

3.5 The exploration - exploitation dilemma

A problem that arises when constructing a reinforcement learning algorithm is how to both be able to exploit a good policy and at the same time explore new policies. If an agent always follows its policy and picks the action it believes is the best, it will never

explore new and perhaps better approaches to solve a problem. At the same time, the agent cannot simply explore new behaviour all the time, since its goal is to maximise future rewards. Therefore, it must also exploit the behaviour that works. This is called the exploration-exploitation dilemma in reinforcement learning [6]. There are several different approaches to this dilemma, one of which is to have two different policies. One policy is called the target policy that is to be the optimal solution, while the other is called the behaviour policy and is for exploration of new behaviours [6]. A reinforcement learning algorithm using a behavioural and target policy is said to be learning off-policy, because it can learn from the experiences made from another agent [6]. On the other hand, a reinforcement learning algorithm that only learns from its own experiences is said to be learning on-policy.

3.6 Artificial neural networks

The use of artificial neural networks (ANN) in machine learning algorithms is one of the import factors for the recent progress in reinforcement learning [6]. The key role of neural networks in reinforcement learning is that they are used as function approximators for the policy function π and action-value function Q. Formally, a neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that maps a n-dimensional input space to a m-dimensional output space. The input space can for instance be the pixel values of a picture, or any other numerical representation of the state of an environment. The output space of a neural network approximating the policy function is the action space. For instance in an car driving environment, the input space could be numerical information about the speed, lane position, distance to closest car etc., and the output space would have one component each for the acceleration, break, and angle of the steering wheel.

An ANN is organised in different layers, where each layer consists of several nodes or neurons, as visualised in figure 3.3. The network has a 2-hidden layer architecture, with 4 neurons each. The input features are sent to the input layer of the network which consists of a neuron for each of the input values. All the input neurons are connected with the neurons in the next layer by a scalar weight, represented by arrows in figure 3.3. The first step for determining the value of a neuron in the first hidden layers is by computing a linear combination of all the input features that are connected to it, and then adding a bias factor. It should be emphasised that the weights and bias are randomly initialised, and the whole point of training a neural network is to find appropriate values for them. Once the linear combination z is computed, it is sent through a non-linear activation function $\Phi : \mathbb{R} \to \mathbb{R}$, whose output value will determine the value of the neuron. There are several activation functions that are used, such as the hyperbolic tangent (tanh), the sigmoid function and rectified linear unit (ReLU). Once all the neurons in a layer are found, the next layer can be calculated with the same process. When an ANN is used as an approximator of the action-value function Q, the training process is all about updating the weights and biases such that the output layer gives the true Q-value for different states and actions. An advantage of using an ANN to approximate the Q-value is that the algorithm can evaluate an action in a new and unseen state. This is useful when the state representation is very large, such as for pictures.

3.7 Actor-critic reinforcement learning

There are two main categories of reinforcement learning algorithms: model-based and model-free reinforcement learning. A model-based algorithm uses the dynamics of the

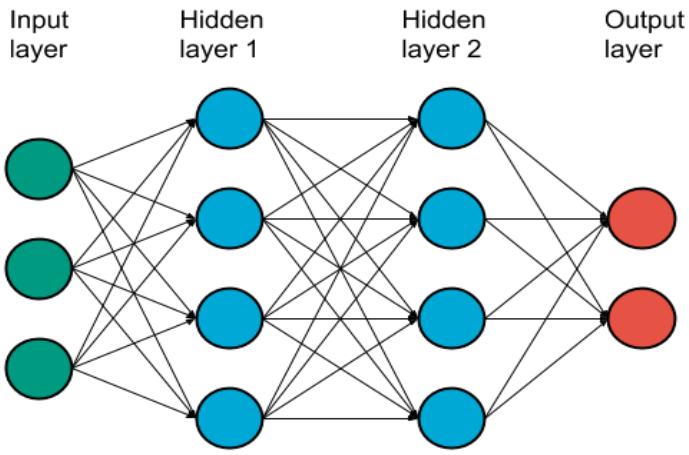


Figure 3.3: Illustration of a neural network with two hidden layers

system to plan actions. For instance, the transition function p in equation (3.5) gives a probability distribution over the next state and reward, which can be used for planning in dynamic programming [7]. The reinforcement algorithm is explicitly using a model of the environment to choose actions. However, for many situations the transition function is unknown, and even if it is known it can be computationally very expensive to use in a reinforcement algorithm.

The second category is called model-free reinforcement learning. As the name suggest, it requires no model or information about the dynamics in the environment. This is very useful in situations where there is no transition function that describes the dynamics in the environment, but experiences can be sampled. Model-free algorithms can be divided into two subcategories: Value-based and policy-based. The categories are visualised in figure 3.4.

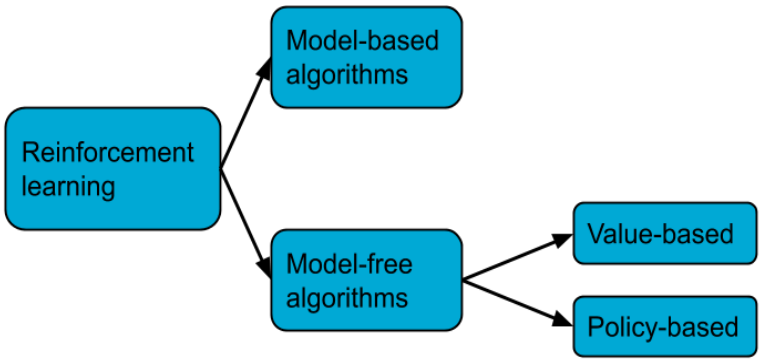


Figure 3.4: Classes of reinforcement algorithms

The first subcategory of model-free algorithm is called value-based methods, where the approach is to approximate the action-value function Q, and use that to take an action. Examples of value-based algorithms are Q-learning, State Action Reward State Action (SARSA) and Deep Q-Network (DQN) [6]. An advantage of value-based methods

is that they can learn off-policy, for instance by learning from the behaviour of experts [8]. Value-based method are therefore very sample efficient as they do not need to find optimal moves themselves, but can learn from a behaviour that is known to be good. The disadvantage is that value-based methods are not well suited for function approximation, such as neural networks, as they tend to be unstable [6].

Policy-based methods (also called policy gradient) directly parametrise the policy function π without involving the action-value function Q in the decision-making [6]. In contrast to value-based methods, they are stable when using function approximation, but very sample inefficient [8]. In other words, the weakness of value-based methods is the advantage of the policy-based methods, and vice versa. A natural idea is then to combine the two methods into a more robust method. This is called an actor-critic model, and is a mix of policy-based and value-based reinforcement learning, as illustrated in figure 3.5. The policy π is called the actor, because it chooses the action to take. The action-value function Q is named the critic because it evaluates the action picked by the actor.

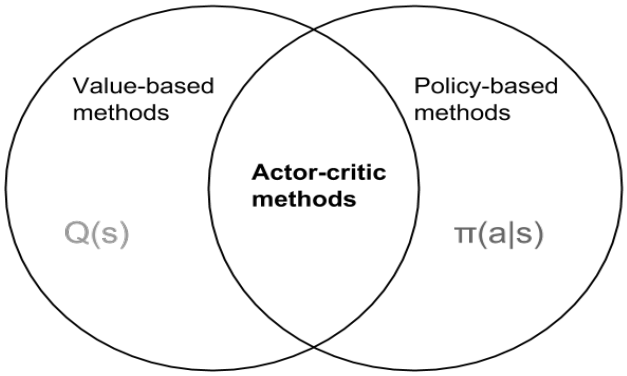


Figure 3.5: Actor-critic in relation to value-based and policy-based methods

3.8 Deep deterministic policy gradient

Silver et. al have developed an off-policy actor-critic reinforcement algorithm called deep deterministic policy gradient (DDPG) [9]. As the name suggest, the algorithm uses a deterministic approximation of the policy function instead of a stochastic version. The advantage with a deterministic policy is that it does not have to integrate over action space in the search for the best policy. This section will outline the differences between a stochastic and deterministic policy function. Both cases share similarities with each other. DDPG models the problem as a Markov decision problem as described in section 3.3.

The initial state distribution is described by $p_1(s_1)$. There is a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and an action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that evaluates an action a in state s under the policy π. The policy interacts with the environment by taking actions and generates a trajectory of states, action and rewards

$$h_{1:T} = s_1, a_1, r_1, ..., s_T, a_T, r_T \tag{3.13}$$

The objective function $J(\pi)$ that the agent will maximise is the expected discounted

return under the policy π from the initial state s_0

$$J(\pi) = \mathbb{E}_\pi[r_0^\gamma] = \mathbb{E}_\pi[r_1 + \gamma r_2 + \gamma^2 r_3 + ...] \tag{3.14}$$

Stochastic policy approximation

Consider a parametrised stochastic policy $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ with parameters θ^π that maps the state space to a probability distribution over the action space. Let $p(s \to s', t, \pi)$ be the probability of transitioning from state s to s' in t time steps. The discounted state distribution $\rho^\pi(s')$ is defined as

$$\rho^\pi(s') = \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \pi) ds \tag{3.15}$$

The performance objective can then be expressed as an expectation

$$\begin{aligned} J(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(a|s) r(a, s) da ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[r(s, a)] \end{aligned} \tag{3.16}$$

The way the agent learns is by computing the gradient of the objective with respect to the policy parameters θ^π and update the weights in that direction. The gradient of the stochastic policy is given by the policy gradient theorem (Sutton [6]) in equation (3.17).

$$\begin{aligned} \nabla_{\theta^\pi} J(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_{\theta^\pi} \pi_\theta(a|s) Q^\pi(a, s) da ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_{\theta^\pi} \log \pi_\theta(a|s) Q^\pi(a, s)] \end{aligned} \tag{3.17}$$

Equation (3.17) shows that the parameter update for the stochastic policy needs to integrates over the action space, which can be computationally costly.

Deterministic policy approximation

Consider a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$ with parameters θ^μ. Using the same definitions of discounted state distribution ρ^μ as in equation (3.15), the objective function is given as

$$J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) r(s, \mu_\theta(s)) ds = \mathbb{E}_{s\sim\rho^\mu}[r(s, \mu_\theta(s))] \tag{3.18}$$

Silver et al. [9] proved that the deterministic policy gradient is given as

$$\begin{aligned} \nabla_{\theta^\mu} J(\mu_\theta) &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_{\theta^\mu} \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s\sim\rho^\mu}[\nabla_{\theta^\mu} \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}] \end{aligned} \tag{3.19}$$

In other words, it is the expectation of the matrix-vector product of the policy Jacobian matrix and the action-value gradient with respect to the actions under policy μ_θ. By convention, the columns of $\nabla_\theta \mu_\theta$ are the gradient of each action dimension. Silver et al. argue that there are a class of function approximators that follow the gradient of the action-value function. Let θ^Q be the parameters of the action-value approximator $Q(s, a|\theta^Q)$ such that

$$Q(s, a|\theta^Q) \approx Q^\mu(s, a) \tag{3.20}$$

and replace the action-value gradient $\nabla_a Q^\mu(s,a)$ in equation (3.19) by the gradient of the approximation $\nabla_a Q(s,a|\theta^Q)$

Deterministic off-policy actor-critic

Deep deterministic policy gradient is an off-policy learning algorithm. An off-policy algorithm has two different policies: a target policy $\mu_\theta(s)$ that is to be the final policy, and a stochastic behavioural policy $\beta(s)$ that is used for exploration to find different strategies. In other words, the target policy learns from the experiences of the behavioural policy. The off-policy objective performance in equation (3.18) is altered by replacing the reward r with the value function V^μ under the target policy μ and following the discounted state distribution of the behavioural policy ρ^β.

$$\begin{aligned} J_\beta(\mu_\theta) &= \int_{\mathcal{S}} \rho^\beta(s) V^\mu(s) ds \\ &= \int_{\mathcal{S}} \rho^\beta(s) Q^\mu(s, \mu_\theta(s)) ds \end{aligned} \tag{3.21}$$

The gradient of the modified objective is

$$\begin{aligned} \nabla_\theta J_\beta(\mu_\theta) &\approx \int_{\mathcal{S}} \rho^\beta(s) \nabla_\theta \mu_\theta(s|a) Q^\mu(s,a) ds \\ &= \mathbb{E}_{s\sim\rho^\beta}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}] \end{aligned} \tag{3.22}$$

Critic loss function

The bellman equation in (3.10) can be used in the design of the loss function for the action-value function approximator $Q(s,a|\theta^Q)$. Let the action-value target y_t be defined as

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) \tag{3.23}$$

The action-value loss can be defined as

$$L(\theta^Q) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim\beta, r_t\sim\mathcal{E}}[Q(s_t, a_t|\theta^Q) - y_t)^2] \tag{3.24}$$

Notice that the state and action follows the distribution of the behavioural policy β. The loss is zero if the action value approximator satisfies the Bellman equation in (3.10). Therefore, the action-value approximator is encouraged to satisfy the Bellman equation.

There are however some practical problems with the loss definition in equation (3.23). The critic target y_t is calculated using $Q(s, \mu(s)|\theta^Q)$, which after each parameter update of θ^Q may change quickly. Therefore, the critic target can fluctuate rapidly during learning. We are changing our approximator closer to the target, but the target itself is moving. To avoid this, Lillicrap et al. suggests a soft update of the target weights [10]. This is done by creating a copy of the actor μ' and critic Q' at the beginning of learning that will serve as the target approximators. After each parameter update, the original actor and critic approximator μ, Q are updated as before, while the target weights are soft updated by $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, where τ is a small positive number ($\tau \ll 1$). The target approximator are more locked in place, and will slowly change during training compared to the original approximators. This greatly increase the stability during learning.

Experience replay

It has been proven difficult to use a naive implementation of the described DDPG algorithm with neural network as function approximators [10]. Learning challenging problems is generally unstable. A problem with the naive approach is that the states and actions explored by the agent during learning are sequential, and by no means uncorrelated. It is best that the samples in a batch used for updating the parameters of a neural network are independent and uncorrelated. A way to get around this problem is by introducing a replay buffer $\mathcal{R}$ where experiences during learning are stored. Specifically, each tuple (s_t, a_t, r_t, s_{t+1}) is put into the replay buffer. When the weights of the functions approximators are updated, a batch of N random transition tuples is drawn from the replay buffer. This avoids the problem with correlated samples in the parameter update and increases the stability of DDPG. In addition, it ensures that the agent does not forget experiences from early in the training.

Algorithm for DDPG

Algorithm 1: Deep deterministic policy gradient, as described by Lillicrap et al.[10]

Randomly initialise critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialise target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialise replay buffer $\mathcal{R}$
for *episode 1:M* **do**
  Initialise random process $\mathcal{N}$
  Receive initial state s_1
  for *t 1:T* **do**
    Select $a_t = \mu(s_t|\theta^Q) + \mathcal{N}_t$
    Execute a_t and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{R}$
    Sample random minibatch of N transitions from the replay buffer
    Set critic target $y_i = r(s_i, a_i) + \gamma Q'(s_{i+1}, \mu'(s_{i+1})|\theta^{Q'})$
    Update critic by

$$L = \frac{1}{N}\sum_{i=1}^{N} Q(s_t, a_t|\theta^Q) - y_t)^2 \tag{3.25}$$

    Update actor policy by

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_i} \tag{3.26}$$

    Soft update the target weights

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{aligned} \tag{3.27}$$

  end
end

3.9 Reward engineering

Designing the reward for a reinforcement agent is often a challenging and time-consuming task. For an algorithm attempting to play a game it is usually not a problem, since the reward could simply be the score in the game, or a positive reward every time the agent wins an episode. However, for other tasks there will often be a need for a lot of handcrafted rewards that require domain-specific knowledge. This section will present an example of this challenge in an electric power system.

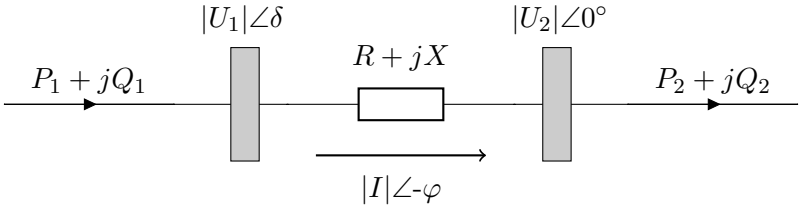


Figure 3.6: A two bus system connected by a line. P and Q are the active and reactive power flow, R and X is the resistance and reactance of the line, U is the voltage and I is the line current

Consider the two-bus system shown in figure 3.6. The bus at the left represent a power plant that produces electric power $S_1 = P_1 + jQ_1$. The power flows through the transmission line and with some losses and arrives at bus 2 with apparent power $S_2 = P_2 + jQ_2$. An agent using the deep deterministic policy gradient (DDPG) algorithm was given a simple load balancing task on the power network. The agent controls the active power production P_1 at bus 1, and has to tune it so that bus 2 receives the desired active load P_2. For simplicity, the demanded load at bus 2 is constant at 1.3 MW. The voltages magnitudes at each bus is locked to 1 p.u, while the reactive power injections and voltage angle can vary. The reward r_t is defined as the negative of the absolute deviation between the received and desired load at bus 2. The idea is that the agent should be encouraged to move closer to the desired load. For simplicity, the initial state, represented as the voltage magnitude, voltage angle the apparent power at each bus, was always the same. The task should therefore be very manageable for the agent because it only has to learn one simple behaviour. After training it converged and received rewards close to 0, indicating that bus 2 got the desired load. The problem was that the agent produced over 10 times as much active power as bus 2 wanted. Where did the rest go? It was lost in the line. In other words, the system had under 10 % efficiency. Figure 3.7 shows what was happening. A line cannot transfer an arbitrary amount of active power due to the power flow equations. As a result, there will be two different production points for P_1 that give the load the desired amount of active power. However, the production point around 12 MW is obviously not desired, as the system efficiency is below 10 %. However, one cannot blame the agent for finding this point, and sticking with it. It has no incentive to change its behaviour when the reward is defined as it is. A natural solution would be to include a term in the reward that punishes loss in the line. This can be done, and the agent would probably find the right power production level. This illustrates the challenges with manual reward shaping.

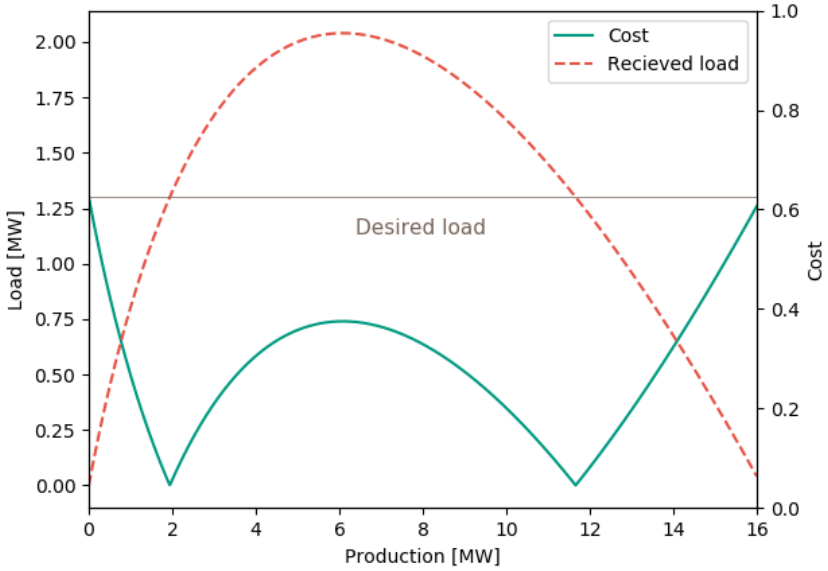


Figure 3.7: Received load as a function of production of the two-bus system in figure 3.6, and cost given to the agent

Chapter 4

Problem Description

Solar power production occurs during day-time and is, not surprisingly, controlled by the sun. As photovoltaic modules become more prevalent in residential areas, they will grow to a size where they produce a significant amount of power that must be transported out on the grid around noon. Later in the afternoon, power must be imported from the external grid because the solar power production is low when residential areas consume power. A normal day will therefore have an import and export period that can challenge the grid in terms of power capacity and voltage quality. A consequence of this is that safety bounds for both voltage and line capacity are in danger of being violated. A strategy for avoiding such problems is called demand response. Albadi and El-Saadany give the following definition of demand response:

> "Demand response can be defined as the changes in electricity usage by end-use customers from their normal consumption patterns in response to changes in the price of electricity over time. Further, DR can be also defined as the incentive payments designed to induce lower electricity use at times of high wholesale market prices or when system reliability is jeopardised. DR includes all intentional electricity consumption pattern modifications by end-use customers that are intended to alter the timing, level of instantaneous demand, or total electricity consumption"[11].

The idea is to modify the demand of electric power such that the size of the peak power in network is reduced. An electric transmission system must be designed to withstand the peak powers, although the network only operates in this period a small proportion of the time. As an example, the municipality Hvaler in southern Norway multiplies its population during vacations, which means that the capacity of the transmission lines to the islands of Hvaler must be built to withstand a very rare peak power situation. To avoid congestion in the grid, the network operator in Hvaler has introduced a power term in the electricity bill, to give an incentive to the consumers to change their consumption pattern [12]. Avoiding congestion in the transmission network is not the only positive consequences of demand response. Vázquez-Canteli and Nagy list the following advantages[13]:

- Improved grid stability due to increased demand flexibility.
- Shift of peak demand towards periods of peak renewable energy generation.
- Lower thermal costs and electricity prices. Since the peak to average ratio of the demand decreases, less peaking plants need to be operated.

- Reduction of the investments in generation, transmission, and distribution assets, which are sized to meet peak demand.
- Lower capacity reserves requirements.
- Reduced energy bills for consumers.

Demand response can be categorised into two main programs: Price-Based Programs (PBP) and Incentive-Based Programs (IBP) [11]. PBP and IBS are also respectively called system-led and market-led programs, which reflect the main mechanism for achieving demand response. In classical IBP for demand control, costumers are given some sort of participation payment, such as a discount rate [11]. There is also a market based IBP, which compensates the costumers based on how much and when they participate. This thesis will not go into the details of the different demand response programs, but instead focus on the implementation of a reinforcement algorithm under an ideal assumption where the agent can change power consumption without a cost. This is done so that it is possible to evaluate the reinforcement agent solely based on its safety behaviour. In other words, the goal is to investigate if the agent safely can control the grid using an ideal demand response program, where flexibility is a free resource. A cost of changing the demand is presented as a term in the reward function in section 4.3, but is not used in the trained reinforcement algorithm.

The scope of this thesis is to continually control the absolute demand at different buses in a power grid, given some interval of flexibility at that bus. It might seem strange to use continuous control since power consuming units generally cannot be set to an arbitrary power level. The nature of a power component is binary: it either consumes power or it does not consume power. Consequently, a valid question is whether a continuous control setup even is realisable in a real power system. Although individual power consuming units are binary, a large collection of units connected to a bus can be approximated as continuous. Imagine 1000 households connected to a bus and that each household has several power consuming components that can be controlled, such as electric vehicles, water heater and heat pumps. Assuming these units have some flexibility in terms of operation time, it is possible to coordinate all these devices in such a way that the power consumption appear continuous in some interval at the bus. A coordinator that does this in a power system is called a flexibility aggregator [14].

This thesis will assume the aggregator exists, so the centralised agent can modify the power consumption continuously in some interval of flexibility. For instance, when the agent wants to decrease the consumption by 2 MW at a bus, there exists a system that turns off electric equipment in households connected to that bus such that the total power change is 2 MW. The advantage of continuous control is that the action space of the algorithm is dramatically reduced, as there is only one action per bus in the grid. Controlling individual components in a power grid quickly gives a very large and impractical action space since the number of possible actions doubles for every additional control device. In addition, Vázquez-Canteli and Nagy recommended in their literature review of demand response and reinforcement learning that there should be more case studies using modern algorithms such as deep deterministic policy gradient (DDPG), which have a continuous action space [13].

The specific setup in this thesis is based on several assumptions for a hypothetical future power grid. The electrical power grid used is constructed by the International Council on Large Electric Systems (CIGRE) as a benchmark network that can be used to analyse the integration of distributed energy resources [15]. It is a 20 kV distribution

network that is predefined in `pandapower`, the Python toolkit used for power flow calculations, and can be defined with different renewable energy sources connected at different buses. The grid used is visualised in figure 4.1.

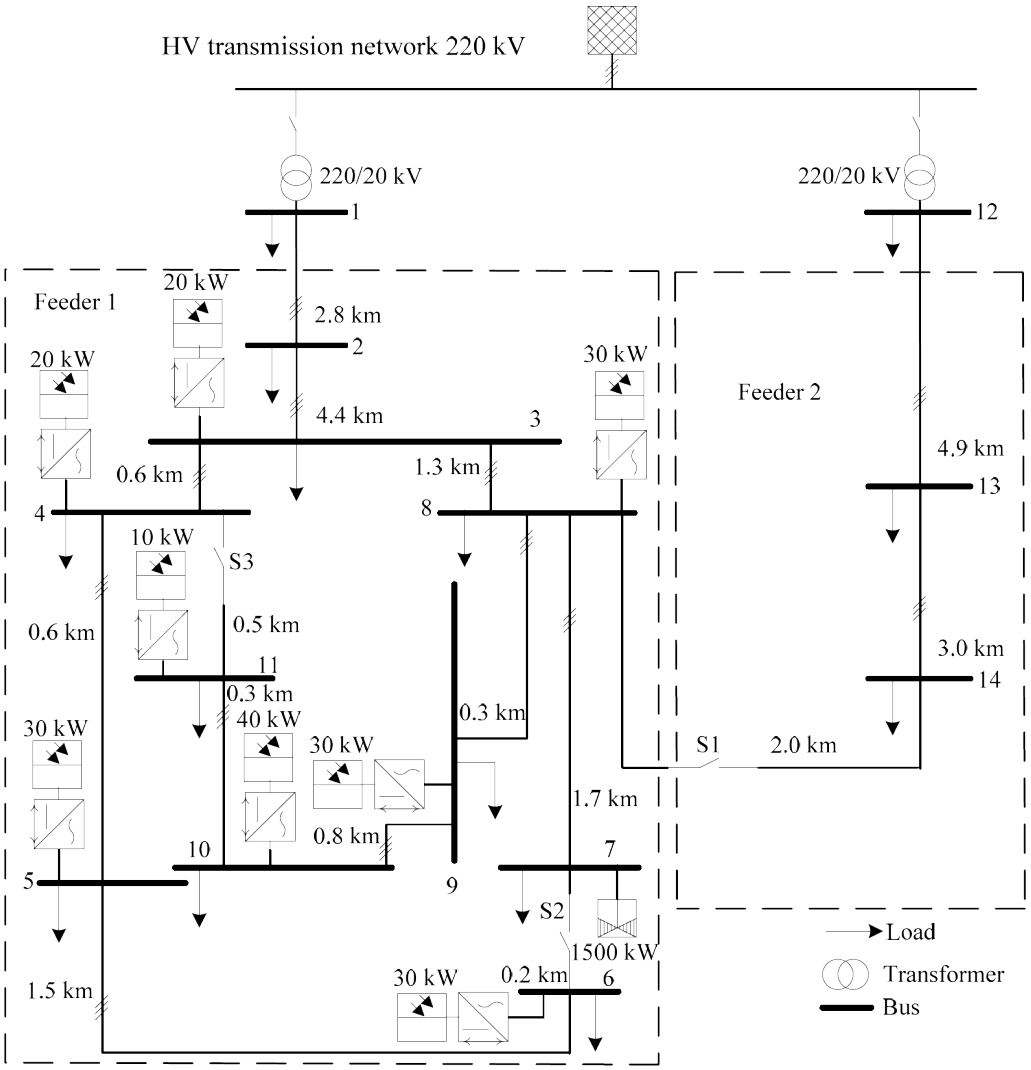


Figure 4.1: CIGRE network with solar and wind power that is used in the reinforcement learning algorithm [15]

The left-most feeder has several solar producing units connected and wind power connected to bus 7. From a power flow perspective, there is no difference between a wind farm (without voltage regulation) and solar production since they are both modelled as static generators (PQ-node). For simplicity, the wind farm connected at bus 7 is assumed to be solar in this thesis, i.e. its power production follows the sun profile through the day. The time resolution in this task is chosen to be hourly. In other words, the demand and solar production at every bus is updated every hour, and assumed constant in between hours. Naturally, the solar intensity varies continuously in an hour, and average values do not tell if or when clouds are blocking the sun. Because of this, the timescale is not short enough to help solve immediate problems that require actions within seconds or minutes, but can be used to plan on an hourly time scale.

There exist demand response programs for large power-consuming industries in several European countries, but programs that aggregate the consumption of residential

costumers are not implemented in large scale [14]. However, the emerge of the Internet of things is enabling communication and coordination between small power consuming equipment in residences, which can be used for demand control. Therefore, the consumption pattern in this thesis will follow the pattern of residential consumers. The daily demand profile is generated using `enlopy`, a Python toolkit for energy demand time series [16]. The demand signal is generated based on data of the energy consumption of private households.

A data set with satellite-derived solar irradiance in central Norway is used as input to define the production of solar power [17]. Figure 4.2 plots a time series of the solar irradiance, and a couple of example days. The solar data is scaled so the maximum value is 1.

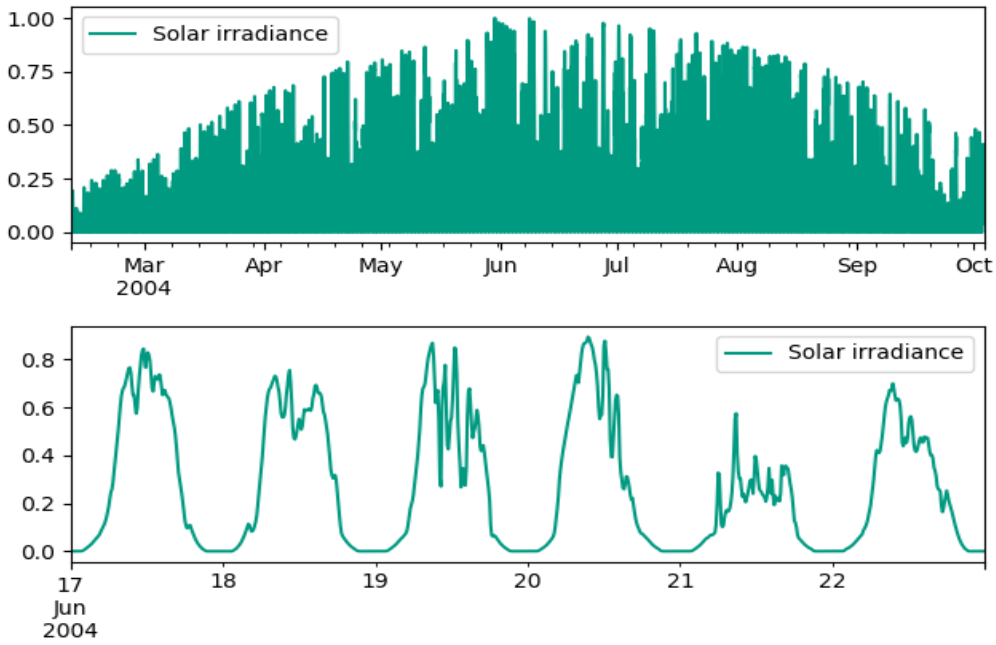


Figure 4.2: The solar irradiance data used in the reinforcement model. The period is 2004-02-11 to 2004-10-03, generated from SoDa [17]. (lon,lat) = (63.395,10.381)

The signal of solar production and power demand is assumed equal at all buses and scaled up based on the nominal values for consumption. It should be noted that the nominal values vary from bus to bus, so they do not consume the same amount of power in absolute terms. For instance, if the demand signal is 0.5, the consumption at each bus is determined by multiplying 0.5 with their respective nominal power that is predefined in `pandapower`. The same is true for the generation of solar power. This reduces the state space, as there is no need for a unique demand forecast for all the loads. Still, the reinforcement agent receives the enough information to determine the demand situation at each bus.

For simplicity, the power factor is assumed constant at every bus. In other words, if the consumption of active power increases by 1 %, then the reactive power consumption also increases by 1%. The power factor at each load is the same as the default values in the CIGRE network. The nominal values of the solar production are intentionally amplified so that the safety margins for line current and voltage magnitudes in the grid frequently are violated if no actions are taken. The solar producing units do not output

any reactive power, only active power.

It is assumed that the loads at all buses have some flexibility in terms of power consumption, say 10 % of the actual demand, that can be centrally controlled at every hour. The reinforcement agent can modify the power consumption independently between the different buses. For instance, it can increase the consumption by 5 % at one bus, and decrease it by 2% at another bus in the same hour. The absolute change in consumption at each load in the net is found by multiplying the demand change by the forecasted power consumption in that hour. A significant simplification in the simulation is that the flexibility is assumed constant in all hours, which is far from a realistic scenario. For instance, decreasing the consumption in an hour corresponds to turning off electrical equipment, and that equipment cannot be turned off again in the next hour. Activation of flexibility should decrease the future flexibility. However, by doing this the agent is given a simpler environment to work in. If a reinforcement algorithm is to be successful in a realistic modelling of flexibility, it should be successful in a simplified model as well. This thesis explores the simplified model.

The CIGRE network consists of several power components, such as transformers, lines, switches and loads. Table 4.1 summarises the different components in the network.

Table 4.1: Component in the CIGRE network

Component	Symbol	Amount
Bus	N	15
Switch	-	8
Static generator	G	9
Line	L	15
Load	F	18
Transformer	-	2

Table 4.1 shows that there are 18 loads and 15 buses in the system. The reason is that there are several load elements connected to some buses which makes the number of loads greater than the number of buses. The net power consumption at a bus is found by summing over the consumption of all the loads connected to that bus.

4.1 State space

This section presents several spaces that could be useful for the reinforcement algorithm. Not all the presented spaces are used in the trained reinforcement learning agent which is analysed in chapter 7.

Let $\mathcal{S}_{\text{sun}} \subseteq \mathbb{R}^H$ be space of the forecasted solar irradiance in the grid H hours into the future. This gives the agent information about the coming solar production and can be used for finding control strategies. For instance, consider an hour with high production of solar power, where the transmission lines are overloaded. If the agent sees that the forecast predicts a dip in solar production in the next hour due to clouds, it can increase the local power consumption at solar producing buses, to relieve the overloaded power lines. It can then safely decrease the consumption the next hour, since the sun is blocked by the clouds and thereby restore the total energy consumption over the two hours. This is an example of a desired behaviour that the agent ideally should find. The buses are assumed to be geographically close, so the solar irradiance is the same everywhere. If a cloud blocks the sun at bus 1, then it also blocks the sun at all the other buses in the

system. This significantly reduces the state space compared to a unique forecast for every bus.

Let $\mathcal{S}_{\text{demand}} \subseteq \mathbb{R}^{F \cdot H}$ be the space of the forecasted power demand H hours into the future in a grid with F flexible loads. It is also possible to let the total demand in the market represent the demand state. A problem with this approach is that it only partially describes the demand situations in the market since the agent does not receive information about demand at individual loads. An advantage, however, is that the size of the state vector from $\mathcal{S}_{\text{demand}}$ is much smaller than a corresponding vector for all loads. Specifically, the CIGRE network that is used has 18 loads, which would make the state vector 18 times greater.

Let $\mathcal{S}_{\text{bus}} \subseteq \mathbb{R}^{4N}$ be the space representing the state of all the buses in the net. Specifically

$$\mathcal{S}_{\text{bus}} = \{|U_i|, \delta_i, P_i, Q_i| \; i = 1, ..., N\} \tag{4.1}$$

where $|U_i|, \delta_i, P_i$ and Q_i respectively are the voltage magnitude, voltage phase angle, active power injection and reactive power injection at bus i in a N-bus system. This could give information to the agent about how stressed the system is. For instance, large values for active power P_i and voltage magnitude $|U_i|$ indicate a stressed situation at bus i, possibly due to high solar production.

The agent should also get information that ensures that the consumption at a load merely is shifted and not altered in absolute magnitude. In other words, a state that contains information about the energy imbalance in the grid. A positive energy imbalance means that the agent has forced the loads to use more energy than the real demand. If the agent changes a load by -1 MW for an hour, the agent should ideally increase the load by 1 MW some time not far into the future. Gemine et al. did this by making a commitment when the demand is changed at a load. When a load is changed, it follows a predefined modulation curve for T_d time steps (for instance 4 steps) [18]. The modulation signal is constructed so that it sums to zero over the time period T_d, which guarantees that the total energy consumption of the bus is constant. Note that Gemine et al. did not implement a reinforcement learning algorithm. The state vector could indicate in what modulation step a load is at. For instance, if the modulation started at t_0 for a load and the current time step is $t_0 + 3$, then the load component of this vector is 3. The agent is then always fed with a signal that tells it the commitment stage of that load. The desired action of the agent at a time step in the commitment period will simply be ignored so that the load follows the modulation signal. This has the drawback that the agent's actions frequently are overridden. Given a modulation period of k steps, the agent desired action is only performed $1/k$ times. Hopefully the agent will pick up on this through the commitment state vector. Formally, the commitment space $\mathcal{S}_{\text{commitment}} \subseteq \mathbb{R}^F$ can be defined as

$$\mathcal{S}_{\text{commitment}} = \{c_i| \; i = 1, ..., F\} \tag{4.2}$$

where $c_i \in \{0, 1, .., k\}$ is the commitment stage of flexible load i in a k-period commitment period in a grid with F flexible loads.

The agent does not have to follow a predefined commitment signal to ensure that the power consumption is shifted, but can instead be penalised for changing the total consumption during a day. The agent must then be fed some information about the energy imbalance at the flexible loads, i.e. if the loads have consumed more or less energy

than the original demand over a period. This could be done by introducing the energy imbalance $B_{i,t}$ of flexible load i at time t.

$$B_{i,t} = \sum_{k=1}^{24} \Delta P_{i,t-k} \tag{4.3}$$

$\Delta P_{i,t-k}$ is the change in power consumption k time steps before t at flexible load i. Simply put, the imbalance $B_{i,t}$ expresses if a flexible load is in balance in terms of energy consumption over the last 24 hours. A positive imbalance means the flexible loads have consumed more power than they originally planned. The goal is to have the imbalance close to zero, which means that power consumption has been shifted and not altered in absolute magnitude. Let $\mathcal{S}_{\text{imbalance}} \subseteq \mathbb{R}^F$ be the state space representing the power imbalance, defined by

$$\mathcal{S}_{\text{imbalance}} = \{B_i |\ i = 1, ..., F\} \tag{4.4}$$

where B_i is the energy imbalance at flexible load i defined by (4.3). Table 4.2 summarises the different state spaces that are presented in this section. The reinforcement learning algorithm will be tested with a subset of the spaces, and the final state variable is constructed by concatenating the individual state vectors.

Table 4.2: State spaces that can be used in the reinforcement algorithm. H is the forecast horizon, i.e. the number of hours into the future the agent receives a forecast. N and F are respectively the number of buses and flexible loads in the net. r_i and d_i are, respectively, the forecasted solar irradiance and power demand i hour in the future. $|U_i|, \delta_i, P_i$ and Q_i are respectively the voltage magnitude, voltage phase angle, active power injection and reactive power injection at bus i. c_i and B_i are respectively the commitment stage and energy imbalance of flexible load i

State space	Symbol	Size	Definition
Solar forecast	$\mathcal{S}_{\text{sun}}$	H	$\{r_j \mid j = 1, ..., H\}$
Demand forecast	$\mathcal{S}_{\text{demand}}$	$H \cdot F$	$\{d_{j,i} \mid j = 1, ..., H, i = 1, ..., F\}$
Bus state	$\mathcal{S}_{\text{bus}}$	$4N$	$\{\delta_i, P_i, Q_i, \|U_i\| \mid i = 1, ..., N\}$
Commitment state	$\mathcal{S}_{\text{commitment}}$	F	$\{c_i \mid i = 1, ..., F\}$
Imbalance state	$\mathcal{S}_{\text{imbalance}}$	F	$\{B_i \mid i = 1, ..., F\}$

4.2 Action space

The reinforcement agent can manipulate the power demand at flexible loads in the power grid. The amount of flexibility at a load is defined to be certain percent of the forecasted power demand. It is assumed that the flexibility is symmetric so the demand can both be tuned up and down. The action space in a power grid with F flexible load $\mathcal{A} \subseteq \mathbb{R}^F$ is defined as

$$\mathcal{A} = \{a_i \mid i = 1, ..., F\} \tag{4.5}$$

where $a_i \in [-1, 1]$ is the activation at flexible load i in the power grid. $a_i = 1$ means that load i increases its active power consumption as much as possible. The change in demand ΔP_i is then scaled up according to the action signal a_i by

$$\begin{aligned} \Delta P_i &= f_i a_i P_{i,\text{forecast}} \\ P_i &\leftarrow P_i + \Delta P_i \end{aligned} \tag{4.6}$$

where f_i is the flexibility, P_i is the active power demand and $P_{i,\text{forecast}}$ is the forecasted power demand at load i. Note that every flexible load has a unique demand change that

is independent of the demand change at the other loads. The resulting reactive power demand is found by multiplying the active power with a load constant, because the loads are assumed to have a constant power factor.

It is possible to include more control variables that the agent can control. For instance, the reinforcement agent could control the tap position of the transformer and the switches in the system. There is also a CIGRE benchmark network in pandapower that has storage units in the power net. A possible extension is to let the agent control the charging and discharging of storage units.

4.3 Reward function

The reward function is a central element in any reinforcement algorithm. The terms *cost* and *reward* are used interchangeably in this thesis. In all cases the reward is the negative of the cost. By this definition, maximising the reward is the same as minimising the cost.

The reward function should give a signal that is used to reinforce "good" behaviour. The goal of the agent is to avoid violations of safety margins for voltage and current in the power grid. Gemine et al. formulate a reward function aimed to safely operate a power grid at a low cost, where they punish the agent proportionally to the violation of safety margins [18]. There are multiple cost terms that can be included, and this section will present several of these.

There are safety margins in terms of voltage magnitudes in an electric system. In the Norwegian power system, there are regulations that state that the voltage can vary by +/- 10 % of nominal levels for voltage levels below 1 kV [19]. This thesis will assume a stricter safety region around nominal voltage values, motivated by the higher voltage level in the CIGRE distribution network (22 kV). The default safety margin for voltage is set to +/- 5 % deviation from nominal voltage. Let $C_{\text{voltage},i}$ be the cost for violating voltage margins at bus i

$$C_{\text{voltage},i} = \max(0, |U_i| - U_{\text{upper}}) + \max(0, U_{\text{lower}} - |U_i|) \tag{4.7}$$

where U_{upper} and U_{lower} are the upper and lower per-unit voltage limit respectively. The voltage cost for a bus in the grid with a safety margin of +/- 5 % is visualised in figure 4.3.

Let $C_{\text{current},i}$ be the cost of violating current margins in line i

$$C_{\text{current},i} = \max(0, |I_i| - I_{\text{upper}}) \tag{4.8}$$

where I_{upper} is the per unit upper current loading limit in lines. Note that the current loading is not measured in ampere, but is the percentage of used capacity in the transmission line. The current cost is therefore weighted by the capacity of a line. The current cost for a line is plotted in figure 4.4 for an upper current loading limit of 90 %.

Costumers must be given an incentive to change their consumption pattern in a realistic modelling of a demand response program. In other words, costumers should be economically compensated when their flexibility is activated. Let the activation cost $C_{\text{activation},i}$ be defined as

$$C_{\text{activation},i} = \lambda |\Delta P_i| \tag{4.9}$$

where $|\Delta P_i|$ is the absolute change in power consumption at flexible load i and λ is the flexibility price at the time of activation [1]. In other words, the cost is the same if the

[1] The currency is arbitrary, and does not affect the cost as it can be scaled by the weight of activation

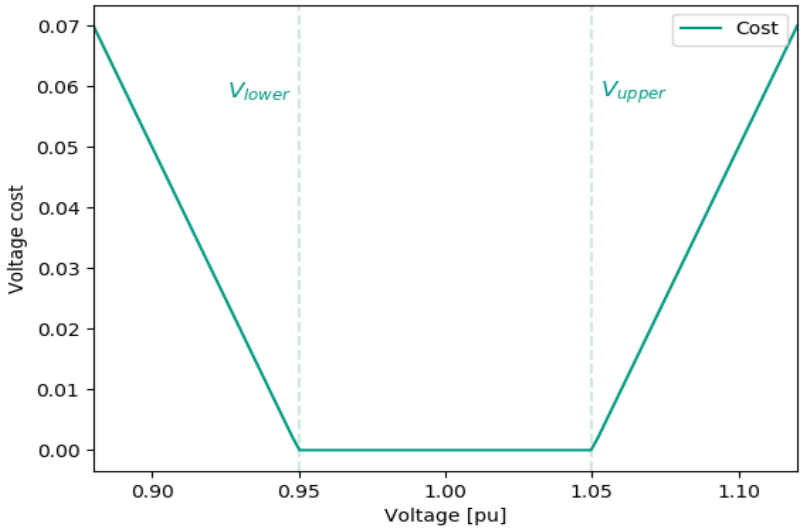


Figure 4.3: Voltage cost at a bus in the power system where the lower and upper voltage limits respectively are 0.95 pu and 1.05 pu

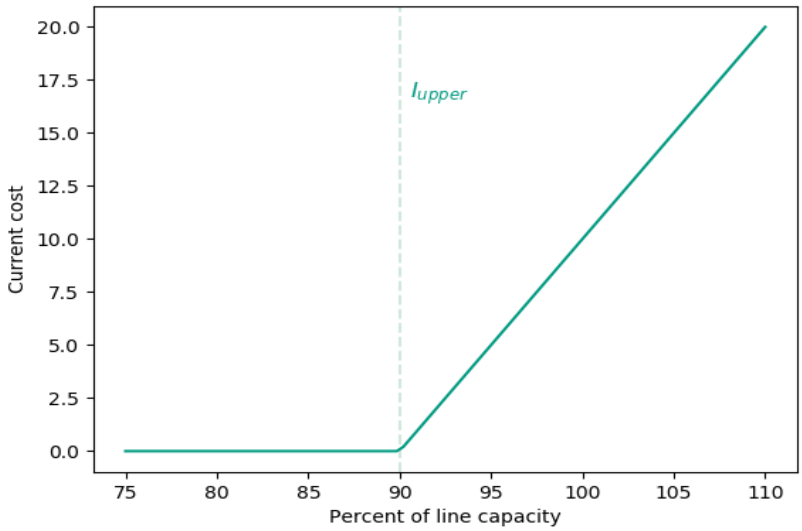


Figure 4.4: Current cost for a line in the power system where the upper current loading is 90 %

consumption is increased or decreased.

It is desired that the flexible loads consume more power during periods with heavy solar production. In addition, flexible loads should use less energy during peak demand, both to reduce violations in the grid and to ensure that the total energy consumption in a day is not altered. The agent should be punished for having a large imbalance. Consequently, it should be penalised for taking an action that increases the absolute balance quantity. Let $C_{\text{imbalance},i}$ be the energy imbalance cost at time t for flexible load

i.

$$C_{\text{imbalance},i} = |B_{i,t}| - |B_{i,t-1}| \tag{4.10}$$

where the energy imbalance $B_{i,t}$ is given by equation (4.3). The agent is penalised if the result of an action increases the energy imbalance in absolute magnitude.

The total cost of an agent at each step is defined as a linear combination over the voltage, current, activation and imbalance cost

$$C = \kappa_1 \sum_{i=1}^{F} C_{\text{activation},i} + \kappa_2 \sum_{i=1}^{L} C_{\text{current},i} + \kappa_3 \sum_{i=1}^{N} C_{\text{voltage},i} + \kappa_4 \sum_{i=1}^{F} C_{\text{imbalance},i} \tag{4.11}$$

Where L, F and N are the number of lines, flexible loads and buses respectively. The weights κ_i can be tuned, based on what the desired behaviour of the agent is. If safety margins are most important, the weights for cost of activation and imbalance should be small. Note that it is possible to discard a cost completely by setting its weight equal to zero. Lastly, the cost must be turned into a reward that a reinforcement algorithm can use. The reward R is simply defined to be the negative of the cost

$$R = -C \tag{4.12}$$

where the total cost C is defined by (4.11). Table 4.3 summarises the different costs and their definition.

Table 4.3: Cost terms that can be used in the reinforcement algorithm. The subscripts 'upper' and 'lower' means the upper and lower safety margins. λ is the unit cost for activation of a flexible load ($£/MWh$). ΔP_i is the change in consumption of flexible load i. $B_{i,t}$ is the daily energy imbalance as defined in (4.3)

Cost	Definition	Comment
Voltage	$\max(0, \lvert U_i \rvert - U_{\text{upper}}) + \max(0, U_{\text{lower}} - \lvert U_i \rvert)$	Violation of voltage margin
Current	$\max(0, \lvert I_i \rvert - I_{\text{upper}})$	Violation of current margin
Activation	$\lambda \lvert \Delta P_i \rvert$	Activating flexibility
Imbalance	$\lvert B_{i,t} \rvert - \lvert B_{i,t-1} \rvert$	Changing daily energy consumption

4.4 Playing an episode

This section will describe how the state of the electric grid is updated in an episode. The reinforcement agent is each hour given a state that represents the system, which includes a forecast for power demand and solar irradiance. Naturally, no forecasts are perfect, so the actual power demand and solar irradiance should deviate from the predicted values. This is done by adding a noise term to the forecasted values. The noise terms for the demand and solar irradiance are assumed to follow a Gaussian distribution with mean 0 and a standard deviation that is proportional to the forecast in that hour. The hyperparameters σ_{solar} and σ_{demand} determine the uncertainty in the forecasts, and the default values are set to 3 % in both cases.

The reinforcement agent evaluates a state s and picks an action a to perform in each hour. The action determines the change in power consumption at the flexible loads in the

power grid. The power flow equations are ready to be solved when the power demand resulting from the action is computed. There are no voltage regulating generators in the CIGRE network. Consequently, all the buses in the network are modelled as PQ-buses, except for the external grid (slack bus) where active power P and voltage angle δ are known. After the power flow calculations have been performed, the reward is calculated and used to evaluate the action of the agent. Finally, the forecasts for demand and solar irradiance are updated and the state s_{t+1} for the next hour is found. This concludes the processes involved in one time step of the reinforcement model. The learning process is summarised in pseudo code in algorithm 2

Algorithm 2: Process for updating power demand and solar production in the electrical power grid

Select state space $\mathcal{S}$, flexibility $f \in [0, 1]$, reward function r
for *episode 1:M* **do**
  Randomly select start day and start hour
  Receive initial state s_1
  for *t 1:T* **do**
    Let the reinforcement agent select an action a_t based on s_t
    Set global solar irradiance r based on forecasted solar irradiance r_{forecast}

$$r = r_{\text{forecast}}(1 + \epsilon), \quad \epsilon \sim \mathcal{N}(0, \sigma_{\text{solar}}) \tag{4.13}$$

    for *Static generator 1:G* **do**
      Update solar production P_{sun} based on nominal value S_N

$$P_{\text{sun}} = rS_N \tag{4.14}$$

    end
    for *Load 1:F* **do**
      Set power demand P based on forecasted power demand P_{forecast}

$$P = P_{\text{forecast}}(1 + \epsilon), \quad \epsilon \sim \mathcal{N}(0, \sigma_{\text{demand}}) \tag{4.15}$$

    end
    for *Load 1:F* **do**
      Update power demand P based on the agent's action a and flexibility f

$$P \leftarrow P + afP_{\text{forecast}} \tag{4.16}$$

    end
    Solve the power flow equations
    Calculate reward with reward function r
    Find next state s_{t+1}
  end
end

Chapter 5

State of the art

5.1 Reinforcement learning

Reinforcement learning is a discipline in rapid development. Reinforcement algorithms have gained from the recent success of methods in supervised learning, such as convolutional neural networks (CNN) and recurrent neural networks (RNN). Dan Cireşan et al. had a breakthrough in 2010 when they were able to train a dense neural network using backpropagation with a graphic processor unit (GPU) instead of a conventional central processor unit (CPU) [20]. The neural network was trained on the MNIST dataset of handwritten digits using image augmentation such as rotations, horizontally shearing and scaling. Their results showed that the time of the backpropagation routine was cut by a factor of 40 with the use of the GPU and they achieved a record breaking low error rate of 0.35 % in the classification task. In 2011 the team of Dan Cireşan et al. continued the development and presented an implementation of backpropagation for CNN using GPU, cutting the training time from months to days [21].

In 2013, Mnih et al. at DeepMind Technologies implemented a Q-learning algorithm using CNN as function approximators, inspired by the recent success of supervised learning [2]. They called their method Deep Q-Network (DQN) and applied it on seven of the Atari 2600 games. DQN was able beat all previous solutions in all games expect for space invaders, and achieved super-human performance in three games. DQN only learns from raw pixel input, without using low dimensional feature engineering of the input values. In addition, the same network architecture and hyperparameters were used for all games, proving the generality of DQN. Before DQN, most reinforcement learning used linear function approximators for the action-values function because nonlinear approximated had problems with divergence [2]. In addition, Tsitsiklis and Van Roy had presented a proof of convergence, in addition to a bound on the approximation error for linear approximators [22].

A problem with using neural networks as a function approximator in online learning is that samples in reinforcement learning are highly correlated. States visited in chronological order are naturally dependent on each other. Therefore, it is difficult to update the network online, since the gradient estimate will suffer from large variance during training, heavily depending on the recent visited states. Mnih et al. avoided this problem by using an experience replay buffer that stores the N last tuples $\langle s_t, a_t, r_t, s_{t+1} \rangle$ of experiences made by the agent. The neural network is updated using stochastic gradient descent by drawing random samples from the experience replay buffer. In addition to avoiding correlated samples, it is a more sample efficient approach as a single experience can be used in different parameter updates [2].

In each time step, Mnih et al. converted the last 4 frames from RGB to grayscale images, reduced them to 84x84 pixels, and stacked into an 84x84x4 image. This was the state representation that was sent as input to the CNN. The output layer of the neural network was the action-value for all the actions in that game. Note that the action space in these games all are discrete actions, i.e. move right, move left, shoot etc. Until DQN, similar algorithms all relied on some sort of domain knowledge that was used for feature engineering. DQN, on the other hand, learns to extract the relevant features on its own, from a stack of grayscale images, through its neural network function approximator.

In 2015, Mnih et al. tested DQN on more atari games, and outperformed 43 of the 47 games where reinforcement learning research previously had been conducted [23]. The algorithm had a slight modification to that presented in the 2013 paper. The Bellman equation in (3.10) is the foundation for calculating the target values used for updating the parameters in the function approximator. However, the parameters of the neural network are used to calculate the target values which in turn are used to calculate the parameter update. As a result, the learning situation can become unstable because the target values are changing as the network parameters are updated. To solve this, they only periodically updated the parameters used for calculating the target, thereby making it more stable during training.

Although DQN was a great improvement compared to existing methods, it is not able to tackle tasks with a continuous action space. A self-driving car must not only decide to turn left or right, but it must determine the exact turn angle of the wheel. A possible solution is to discretise the action space, for instance by dividing the possible wheel angles into 5 possible categories: $[-90^o, -45^o, 0^o, 45^o, 90^o]$. This is realisable for tasks consisting of few independent action variables, albeit the steering would be very abrupt, but the action space grows exponentially with the number of variables. For instance, imagine a reinforcement task with 10 independent action variables that all are discretised into 5 categories. The resulting action space is finite, but has a size of $5^{10} \approx 10,000,000$. The consequence is that the final layer in the neural network in the DQN algorithm must consists of 10 million neurons, which all would have a parameter for all the neurons in the previous layer, in addition to a bias. The number of parameters in the neural network would easily surpass 1 billion. Therefore, a better approach is to use policy gradient methods that are well suited for continuous actions.

In 2016, Lillicrap et al. presented the deep deterministic policy gradient (DDPG) as an continuous extension of the DQN algorithm [10]. DDPG is an actor-critic model that builds on the deterministic policy gradient methods presented by Silver at. el in 2014, where it was shown that deterministic policy methods were better than stochastic methods, especially in tasks with a large action space [9]. The extension made by Lillicrap et al. was to use a neural network, as in DQN, as function approximators. Specifically, they parametrise both the policy (actor) and action-value function (critic) by neural networks. DDPG also uses the replay buffer to ensure uncorrelated samples during parameter updates. They use a separate target network for both the actor and the critic, in a similar fashion to DQN where target values parameters only are periodically updated. DDPG was tested on 33 physics tasks with continuous actions, such as the cartpole and pendulum environment. They trained both on low-dimensional input data, such as angle velocities, acceleration etc., and also using raw pixels from rendering of the environment. DDPG performed well on most of the tasks, and was able to beat a model-based planning algorithm (iLQG) on several occasions, not only using low-dimensional features, but also using raw pixels as input [10].

The success of neural networks as function approximators continued in 2016 with

AlphaGo (Silver et al.), a reinforcement algorithm trained to play the Chinese board game Go [24]. Go is a complex game with a large state and action space that computer programs have been struggling to solve for decades. AlphaGo was the first program to beat the current world champion of Go, winning 4 of 5 games against Lee Sedol [6]. AlphaGo was not entirely learning from self-play, but was pretrained using expert-moves. However, an extension of the method, called AlphaGo Zero, learned entirely from self-play through the use of Monte Carlo tree search and CNN as function approximators. Finally, Silver et al. introduced AlphaZero, and showed that the same algorithm and neural network architecture could be generalised to also learn chess and the Japanese board game Shogi through self-play [3]. AlphaZero was able to beat the current world-record computer programs in all three games, without any domain-specific knowledge or handcrafted features.

On April 13 2019, OpenAI Five became the first reinforcement algorithm to beat a world champion team in an esport event [25]. OpenAi Five played against the world champions of Dota 2, a multiplayer online battle arena video game consisting of two teams of 5 playing against each other. The game is highly strategic, where it is important to plan and collect resources to be able to destroy the opponents main building [26]. Once again, the successful reinforcement algorithm uses neural network as function approximators, although the exact algorithm (PPO) is different from DDPG [27]. Still, it shows that neural networks have not only been revolutionary in supervised learning, but also in reinforcement learning. OpenAI Five demonstrates one of the main strengths of neural networks: it continually learns from new input data/experiences. OpenAI Five had a solid 10-month wall-clock time of training, corresponding to 45 000 year of self-play in Dota-2 [25].

5.2 Demand response

Demand response is a subject that is well researched. Vázquez-Canteli and Nagy have conducted a literature search where they make a review of published research concerning the use of reinforcement learning algorithms and demand response [13]. The different search terms are summarised in table 5.1 and they used the following combination for a search in Web of Science: *A and {B or (C and D)}*

Table 5.1: Search terms used by Vázquez-Canteli and Nagy to find literature about demand response and reinforcement. learning [13]. * means that singular and plural form were used

A	B	C	D
"Reinforcement learning"	"Demand-side management"	Heating	Building*
Q-learning	"Demand response"	Cooling	House*
	"Electric vehicle*"	"Electricity price*"	Residential
	HVAC	Comfort	Home*
		Energy	Household*
		Photovoltaic	
		PV	
		Solar	

The review includes 105 articles in total, mainly from the literature search in Web of Science. The studies have objectives such as minimising energy cost, peak energy and user discomfort. A way of minimising energy cost and peak energy is by shifting the

demand to hours of peak solar production, which in turn helps the power system by avoiding voltage and current safety violations. However, the goal of the studies included in Vázquez-Canteli and Nagy's review do not explicitly use the voltages and line current in the reinforcement algorithm in a similar way as this thesis. In contrast, the studies often focuses on the energy system of individual houses or collection of houses. For instance, Yang et al. focus on controlling a heat pumps, geothermal boreholes and PV modules to minimise a building's energy usage [28]. Dusparic et al. use reinforcement learning to control the charging of electrical vehicles such that the current limit for a transformer is not violated [29]. The problems are similar, but do not involve the power flow equations as part of the algorithm. Vázquez-Canteli and Nagy give recommendation for future work on reinforcement learning and demand response. They propose that more studies should implement state-of-the-art reinforcement learning algorithms, such as DDPG [13].

Gemine et al. have formulated a similar problem as the research question in this thesis, allowing the agent to not only change the consumption in but also to curtail generation of power [18]. They use tests grids of 5, 33 and 77 buses and the goal is to avoid violations operational limits at a low cost. They include the power flow equations explicitly in the problem, but do not solve the task using a reinforcement learning algorithm, although they formulate the problem as a Markov decision process. The safety margins for current and voltage are taken out of the objective function, and put as constraints into different mathematical programs. In addition, they do not consider continuous actions.

Zarrabian et al. use Q-learning to avoid line congestion and cascade failure in an electric transmission system [30]. The agent is punished for increasing the current in an overloaded line and the agent's actions space is to increase or decrease the power produced at generators in the system. Again, the action space is discrete and not continuous as in this thesis. They test the agent for several scenarios, such as a N-1 and N-1-1 contingency. A scenario with a loss of a single transmission component is termed a N-1 contingency, and a scenario with two consequent losses of transmission components is termed N-1-1 contingency. The Q-learning approach is successful, and the agent is able to satisfy all power system constraints and at the same time avoiding cascading failures [30].

The work of Gemine et al. and Zarrabian et al. are the most relevant studies found for the research question in this thesis.

Chapter 6

Implementation

This chapter introduces the Python libraries `gym`, `stable-baselines` and `pandapower`, which are used for setting up the reinforcement learning algorithm and solving the power flow equations. Also, the class `ActiveEnv` developed for this thesis is presented.

6.1 Pandapower

The central goal in this thesis is to keep values for current and voltage in an electrical power system within safety margins. It is therefore necessary to have a way to solve the power flow equations (2.45) and use the results to see if safety margins are violated. The tool for solving the power flow equation in this thesis is the Python package `pandapower` [31]. They describe themselves in the following way:

> "pandapower builds on the data analysis library pandas and the power system analysis toolbox PYPOWER to create an easy to use network calculation program aimed at automation of analysis and optimization in power systems. What started as a convenience wrapper around PYPOWER has evolved into a stand-alone power systems analysis toolbox with extensive power system model library, an improved power flow solver and many other power systems analysis functions." [32]

Pandapower is a time independent simulation tool that finds steady state solutions to a power flow problem. The power consumption and production at nodes in the power system can easily be updated as desired, and the power flow calculation will give the resulting voltage and current magnitudes.

This section will describe the different electrical elements available in pandapower and how they are physically modelled. In addition, it is shown how to take actions in the power system using the API of pandapower. It should be noted that this thesis uses version 1.6.1 of pandapower, which has some differences compared to the 2.0.1 version available at the current time of writing. For instance, the base power unit in the 1.6.1 is kilowatts, while it for the 2.0.1 is megawatts.

Lines

The method `pandapower.create_line` is used to create a line element that can be connected between two buses. Pandapower offers many standard types lines for both underground cables and overhead transmission lines. Alternatively, a line with custom parameters for impedance, line length, line diameter etc. can be specified with the

method `pandapower.create_line_from_parameters`. The lines are modelled using the π-equivalent model, described in section 2.10.

Generators

Pandapower has two generator types. The first type is simply called generator and is modelled as a PV-bus. In other words, the active power production P and voltage magnitude $|U|$ are known when solving the power flow equations. Generators can be created using the method `pandapower.create_gen`, where nominal values for apparent power and voltage can be specified. The second type is the static generator, where the active power P and reactive power Q is specified (PQ-bus). Static generators are created using `pandapower.create_sgen`. Pandapower models power from the consumer perspective, so negative values for active power P and reactive power Q correspond to generation of power.

Loads

The load in pandapower is modelled as a PQ-bus, where active and reactive power are known. In other words, they are modelled like static generators, but are separated in two different data structures. Loads are created using the method `pandapower.create_load`. The loads can also be modelled with constant impedance Z, current I and P. In other words, replacing reactive power with current and impedance.

Transformers

Pandapower offers both two-winding and three-winding transformers that can be created from standard types using the `pandapower.create_transformer` method, or from parameters using the method `pandapower.create_transformer_from_parameters`. The transformers can be modelled as a π-transformer or a t-transformer. The transformers have settings for tap-position and phase-shifting of the voltage, which are possible control variables for a reinforcement agent.

Storage

The method `pandapower.create_storage` creates a storage element in a power net. The storage element is modelled as a PQ-node. Because simulations in pandapower is time-independent and only find steady state solutions, it does not update the capacity of a storage during power flow calculations. Available energy in the storage element must therefore be updated manually according to some predefined timescale when several power flow calculations are performed.

6.1.1 Data structures in pandapower

Pandapower stores information about elements in an electric transmission system in pandas DataFrames, which make it easy to inspect the numeric output of a power flow calculation. Figure 6.1 shows a 4 bus case net and the components included in it.

```
In [2]: import pandapower as pp
        import pandapower.networks as pn
        net = pn.case4gs()
        net

Out[2]: This pandapower network includes the following parameter tables:
           - ext_grid (1 element)
           - bus (4 elements)
           - bus_geodata (4 elements)
           - load (4 elements)
           - line (4 elements)
           - gen (1 element)
         and the following results tables:
           - res_ext_grid (1 element)
           - res_gen (1 element)
           - res_line (4 elements)
           - res_load (4 elements)
           - res_bus (4 elements)
```

Figure 6.1: Loading an example net in pandapower

Each of the elements listed has a corresponding pandas DataFrame. The elements without "res" at the beginning are parameter tables and have information about nominal and max/min values for the components. Figure 6.2 shows the parameter table for the lines and buses in the system.

```
In [3]: net.bus
```

Out[3]:

	in_service	max_vm_pu	min_vm_pu	name	type	vn_kv	zone
0	True	1.1	0.9	0	b	230.0	1
1	True	1.1	0.9	1	b	230.0	1
2	True	1.1	0.9	2	b	230.0	1
3	True	1.1	0.9	3	b	230.0	1

```
In [6]: net.line
```

Out[6]:

	c_nf_per_km	df	from_bus	g_us_per_km	in_service	length_km	max_i_ka
0	513.969177	1.0	0	0.0	True	1.0	0.627555
1	388.610841	1.0	0	0.0	True	1.0	0.627555
2	388.610841	1.0	1	0.0	True	1.0	0.627555
3	639.327512	1.0	2	0.0	True	1.0	0.627555

Figure 6.2: The parameter table for buses and lines in pandapower. There are more columns in `net.line`

All of the components will have a result table after the power flow calculation is performed, using the method `pandapower.runpp`. Figure 6.3 shows the result tables for the buses and lines in the net.

6.1.2 Plotting results

It is important to be able to inspect the resulting state after a reinforcement agent performs an action on the system. This can be complicated for large network with many buses and lines that each have a voltage magnitude, voltage angle and so on. Fortunately,

```
In [8]:  pp.runpp(net)
         net.res_bus
```

Out[8]:

	vm_pu	va_degree	p_kw	q_kvar
0	1.000000	0.000000	-136809.077861	-83510.840631
1	0.982421	-0.976122	170000.000000	105350.000000
2	0.969005	-1.872177	200000.000000	123940.000000
3	1.020000	1.523055	-238000.000000	-131849.643142

```
In [11]:  net.res_line
```

Out[11]:

	p_from_kw	q_from_kvar	p_to_kw	q_to_kvar	pl_kw	ql_kvar
0	38691.532265	22298.455796	-38464.824941	-31236.318554	226.707324	-8937.862758
1	98117.545596	61212.384835	-97086.107117	-63568.702389	1031.438479	-2356.317554
2	-131535.175059	-74113.681444	133250.652398	74919.557635	1715.477339	805.876191
3	-102913.892876	-60371.297572	104749.347607	56930.085508	1835.454732	-3441.212064

Figure 6.3: The result table for lines and buses in pandapower. There are more columns in `net.res_line` than those shown in the figure

pandapower has support for plotting both grid architecture and results from the power flow. It is possible to get static plots using matplotlib and interactive plots using plotly [33]. Figure 6.4 shows the interactive results from a power flow calculation performed on the Oberrhein power grid using plotly. The lines are coloured based on the line loading (100% means maximum current), while the buses are coloured based on the voltage magnitude. Such plots help to get an overview of the grid and quickly identify critical areas in the transmission system. It is possible to zoom into the net since the plot is interactive, and by clicking on a line or bus you can see the values for voltage, current and power.

6.1.3 Controlling a pandapower net

Reinforcement learning is all about taking actions and getting rewards based on how good that action was. It must therefore be possible to control certain elements in a pandapower net. This section will show how to update the demand in the net, and how to control transformers. The demand update is the only control that is implemented into the reinforcement algorithm in this thesis.

Transformers in a transmission system can have controllable taps that change the winding ratio between the low and high voltage side of the transformer. By doing this it is possible to control the voltage magnitude at buses connected to a transformer. There also exists phase-shifting transformers that can manipulate the voltage angle between the low and high voltage side. Transformers in pandapower allow control of both voltage magnitude $|U|$ and voltage angle δ. The code in figure 6.5 demonstrates how to control a transformer in pandapower. First, a two-bus system with a transformer is created. It is a standard type transformer 25 MVA, 110/20 kV and the external grid is connected to the high voltage side of the transformer. The taps are placed on the low voltage side of the transformer with the command `net.trafo['tp_side'] = 'lv'`. Performing a power flow calculation gives the result tables without changing the tap position in the transformer. The voltage angle is manipulated to 20 degrees by specifying `net.trafo['shift_degree']`. The voltage magnitude is changed by first specifying

```
from pandapower.networks import mv_oberrhein
import pandapower
from pandapower.plotting.plotly import pf_res_plotly
net = mv_oberrhein()
pandapower.runpp(net)
pf_res_plotly(net)
```

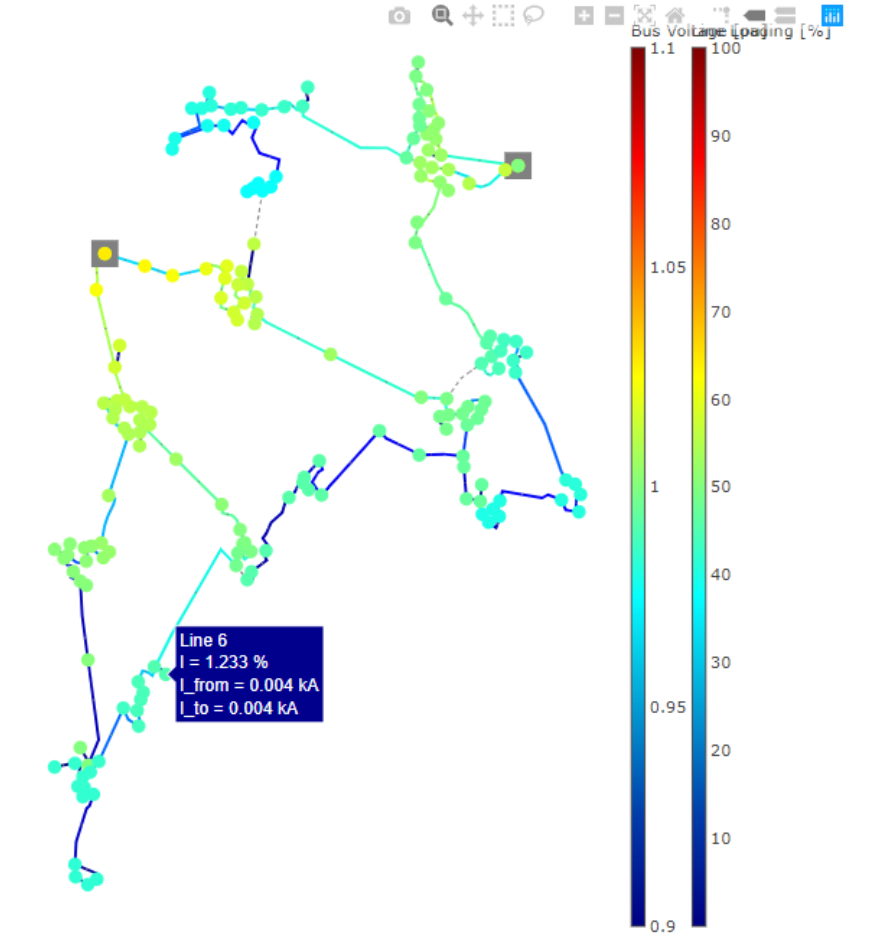


Figure 6.4: Interactive plot of the results from a power flow calculation on the Oberrhein case power grid using plotly. The grid consists of 179 lines and 180 buses

`net.trafo['tp_st_percent']` which is the percentage change in voltage magnitude per tap position. This is set to 10 % and the tap position is set using `net.trafo['tp_pos']` to -1. By running another power flow calculation and inspecting the table `net.res_bus`, it is evident that the voltage angle is shifted 20 degrees and that the voltage magnitude is reduced by 10 % with respect to the first power flow calculation.

`net.load` gives a DataFrame where all each load in the system is represented as a row. There can be several loads connected to a bus in pandapower. In such cases, the net power injection at that bus is the sum over all its loads. Figure 6.6 shows how to double the active and reactive power consumed at the loads. The general pattern for controlling an element in pandapower is to change the element table, run the power flow calculation which in turn updates the result table. The solar production at a bus can be set in the same manner, by updating the values in the `net.sgen` table.

In [1]:
```
import pandapower as pp
net = pp.create_empty_network()
b1 = pp.create_bus(net, vn_kv=110)
b2 = pp.create_bus(net, vn_kv=20)
t = pp.create_transformer(net, hv_bus=b1, lv_bus =b2,
                          std_type='25 MVA 110/20 kV')
net.trafo['tp_side'] = 'lv'
load = pp.create_load(net, bus=b2, p_kw=10000)
pp.create_ext_grid(net,bus=b1)
pp.runpp(net)

net.res_bus
```

Out[1]:

	vm_pu	va_degree	p_kw	q_kvar
0	1.00000	0.000000	-10030.468579	-493.595172
1	0.99717	-2.759365	10000.000000	0.000000

In [2]:
```
net.trafo['shift_degree'] = 20
net.trafo['tp_st_percent'] = 10

net.trafo['tp_pos'] = -1
pp.runpp(net,calculate_voltage_angles=True)
net.res_bus
```

Out[2]:

	vm_pu	va_degree	p_kw	q_kvar
0	1.000000	0.000000	-10030.468579	-493.595172
1	0.897453	-22.759365	10000.000000	0.000000

Figure 6.5: Code showing how to control the tap position and phase angle for a transformer in pandapower

6.2 Gym, stable-baselines and ActiveEnv

The Python library `gym` is a toolkit used for managing environments in a reinforcement learning algorithm [34]. It is developed by OpenAI and includes thousands of predefined environments of classical video games and control theory tasks such as the cartpole, swinging pendulum etc. In addition, it is possible to construct own environments that easily can be used in reinforcement algorithms. OpenAI also has a toolkit called `baselines` with implementations of many reinforcements algorithm that can interact with `gym` environments [35]. However, the `baselines` library is currently lacking a unified code structure between the algorithms and generally have poor documentation. As a result, a fork called `stable-baseline` has been created by the AI community, that offers a major cleanup of the code, with a scikit-learn like interface [36]. Along with this, it supports tensorboard that can be used to monitor the rewards and objective losses during training.

The implementation of the reinforcement algorithm in this thesis is done using `gym` and `stable-baselines`. Specifically, an environment class called `ActiveEnv` is implemented, that follows the standard `gym` environment structure. The environment is thoroughly

In [1]:
```
import pandapower as pp
import pandapower.networks as pn

net = pn.case4gs()
pp.runpp(net)
net.res_load
```

Out[1]:

	p_kw	q_kvar
0	50000.0	30990.0
1	170000.0	105350.0
2	200000.0	123940.0
3	80000.0	49580.0

In [2]:
```
net.load[['p_kw','q_kvar']] *= 2
pp.runpp(net)
net.res_load
```

Out[2]:

	p_kw	q_kvar
0	100000.0	61980.0
1	340000.0	210700.0
2	400000.0	247880.0
3	160000.0	99160.0

Figure 6.6: Code showing how to double the consumption at loads in pandapower

tested using pytest. The main job of `ActiveEnv` is to take in an action chosen by a reinforcement algorithm, perform that action, find the next state resulting from that action, and calculate the reward. Specifically, `ActiveEnv` receives an action vector a where each component determines the percentage change in power consumption at each flexible load. First the consumption and production of power at nodes in the net are changed according to the demand forecast and solar forecast, respectively. The action vector is then processed and updates the power consumption at each load in the `pandapower` net. The power flow equations for the network are then solved using `pandapower`. Once the power flow equations have been solved and the new voltage and current magnitudes in the net are determined, the reward can be calculated. This summarises the steps involved with one action in the reinforcement algorithm. A more detailed description can be found in chapter 4.

There are many possible ways of constructing the reward function and state space, as discussed in chapter 4. The `ActiveEnv` class has a method that can be used to specify the setup of the reinforcement learning algorithm, in addition to several parameters. Table 6.1 gives a description of all the parameters together with their default values.

Table 6.1: Description of the parameters that determine the setup of the `ActiveEnv` environment

Parameter name	Value	Description
activation_weight	0.0001	Weigh factor for activation cost
current_weight	0.01	Weight factor for current cost
demand_scale	10	Scale factor for power demand
demand_std	0.03	Standard deviation as a ratio of forecasted demand
episode_length	200	Number of steps (hours) before the environment resets
flexibility	0.1	Quantity describing max demand change at a load
forecast_horizon	4	Number of hours in the forecasts
i_upper	90	Upper current limit as percentage of line capacity
imbalance_change	False	Imbalance change used to calculate cost if True
imbalance_weight	0.0001	Weight factor for imbalance cost
reactive_power	True	Determines if reactive power is modified by the action
reward_terms	['voltage', 'current', 'imbalance','activation']	Terms to include in the reward function
solar_scale	0.8	Scale factor for solar irradiance
solar_std	0.03	Standard deviation as a ratio of forecasted solar irradiance
state_space	['sun', 'demand', 'imbalance', 'bus']	State spaces to be included in the model
total_imbalance	False	Calculates total demand imbalance in the system if True
v_lower	0.95	Lower voltage margin
v_upper	1.05	Upper voltage margin
voltage_weight	1	Weight factor for voltage cost

Chapter 7

Results

This chapter presents results from a model trained using deep deterministic policy gradient (DDPG).

7.1 Feasibility

It is important to investigate the potential effect demand response has on current loading and voltage magnitudes before a model is evaluated. It is not given that it is physically feasible to avoid safety violations in critical periods by modifying the power consumption in grid. Certainly, the voltage magnitude at buses in the net will decrease when the consumption is increased, but how much? This section shows that changing the consumption indeed can affect the voltage and current enough to reduce the number of violations in CIGRE network.

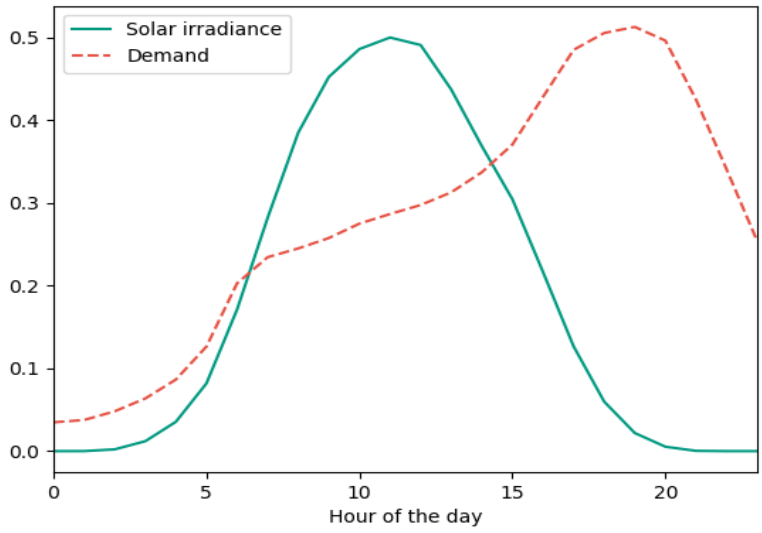


Figure 7.1: Mean hourly power demand curve and solar irradiance

As mentioned in chapter 4, the main challenges for the network is that large amounts of power must imported from and exported out to the external grid in a normal day. Figure 7.1 plots the daily mean solar irradiance signal and power demand used for training

the reinforcement agent. The first period that is critical for the grid is when the solar irradiance is at its maximum.

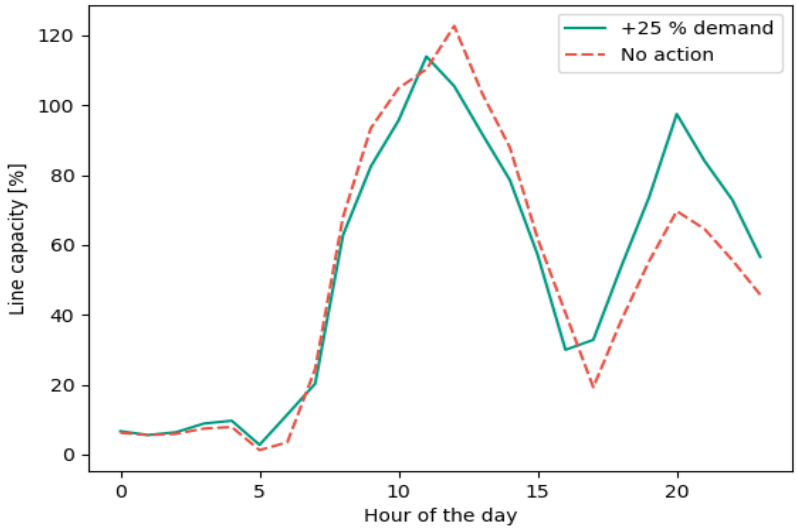


Figure 7.2: Current capacity in a critical line when increasing the power consumption with 25 % in all hours and regular situation. The line connecting bus 1 and 2 in figure 4.1 is showed

Figure 7.2 illustrates the effect of activating flexibility (increasing consumption) has on the current in a critical line. The consumption of active power is increased by 25 % for all the loads in every hour. The first and second peak in plot correspond, respectively, to the solar peak and the power demand peak. It is clear from the first peak that increasing the consumption decreases the line current. This is because the buses in this period generate so much solar power that they act as producers and not consumers. Increasing the consumption means that less power needs to be transported out to the external grid, which decreases the line current. As a result, the desired behaviour in periods with high solar irradiation is to increase the consumption.

The effect of increasing consumption in the second peak is the opposite of that in the first peak. This is because the loads now act as consumers since there is no solar production. The active power must therefore be imported from the external grid. Increasing the consumption in this period simply corresponds to drawing even more power from the external grid, which in turn increases the line current loading. The desired behaviour in this period is therefore to lower the power consumption, since this would decrease the current in the line.

Figure 7.3 illustrates the effect activation of flexibility has on voltage magnitude at a critical bus bar. In this case, the consumption of active power is decreased by 25 % at all loads in the network throughout a day. There are two critical periods during a normal day for this case as well. The effect of decreasing the consumption in periods with high solar is that the voltage magnitude increases. This can be seen between around hour 13 in figure 7.3. The buses are in this period acting as producers because of the excess solar power that is transported out to the external grid. Generally, higher production at a bus bar means higher voltage magnitude. Reducing the consumption means that even more power from solar production must be sent out to the external grid, raising the voltage magnitude ever further. Consequently, decreasing the consumption in periods with high

solar irradiance is not a desired behaviour.

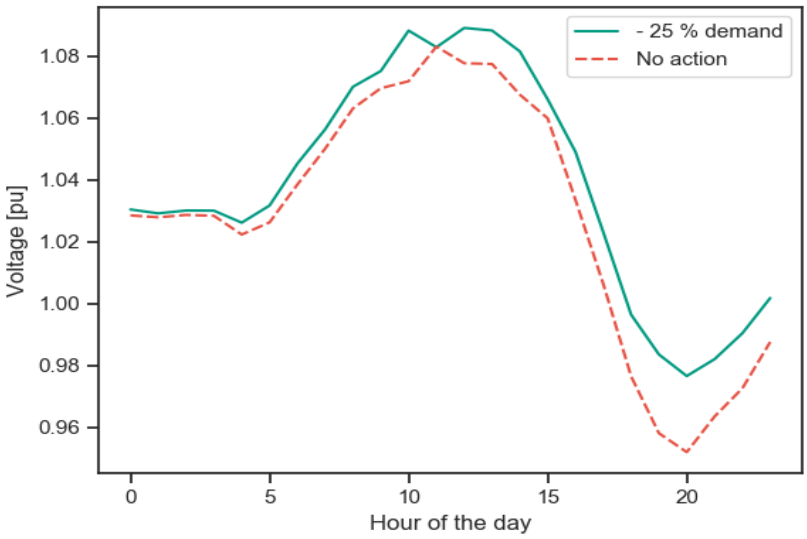


Figure 7.3: The voltage magnitude at a critical bus when the power consumption is reduced with 25 % in all hours and regular situation. Bus bar 9 in 4.1 is used

The second critical period in a normal day can be seen around hour 20 in figure 7.3. At this point, there is no solar power production, and power must be imported from the external grid to meet the demand. The voltage magnitude is now lower than its nominal value because the loads are consumers. Generally, higher consumption means lower voltage magnitude. Therefore, it is better to reduce the consumption of power in this period because the voltage magnitudes will be closer to nominal values.

To summarise, it is physically feasible to impact the current and voltage magnitudes in the CIGRE net such that safety violations can be avoided or reduced by the means of demand response. The desired behaviour in terms of both current and voltage safety is to increase consumption in periods with high solar irradiance and decrease it during peak demand.

7.2 Simulation - Free activation

In this simulation, a reinforcement agent with a constant flexibility of 10 % is trained with a reward function that does not include the cost of activation. This is not a realistic case of demand response, since households that offer flexibility should be compensated for altering their energy profile and the flexibility is not a constant quantity. However, it shows how an agent would activate flexibility if there was no direct cost associated with altering the power consumption. The agent is penalised for changing the total daily energy consumption in the power net. Note that it is not penalised for changing the daily consumption at individual loads as long as the total consumption in the network is preserved. The specific reward terms with weights are shown in table 7.1. The cost function C in a time step with energy imbalance B is

$$C = 10^{-2} \sum_{i=1}^{L} C_{\text{current},i} + 1 \sum_{i=1}^{N} C_{\text{voltage},i} + 10^{-4} B \tag{7.1}$$

where L and N respectively are the number lines and buses in the system.

Table 7.1: Reward terms and weights for formulation 1

Cost	Weight	Comment
Voltage	1	Per-unit values
Current	10^{-2}	Percentage of max current
Activation	0	No activation cost
Imbalance	10^{-4}	Units of energy imbalance is kWh

Table 7.2: State used in formulation 1

State space	Size	Comment
Solar forecast	4	4-hour solar forecast
Demand forecast	4	4-hour demand forecast
Imbalance state	1	Total energy imbalance in the net

The state space is constructed to be as small as possible. The state is represented by a 4-hour forecast for both solar irradiance and active power demand. The power demand as a percentage of nominal consumption is assumed equal at all the flexible load, so there is not an individual power demand for each load. In addition to the forecasts, the total power imbalance for the whole power network is included. Table 7.2 summarises the state space

The DDPG agent was trained for 100 000 time steps, with no uncertainty in the forecasts. In other words, the agent receives perfect information of the solar and demand situation in the next hours. This can be seen as a model training on historical data for solar irradiance and demand. Once training is done, the agent is tested in an environment with uncertainty in both the solar and demand forecasts. The forecast error follows a Gaussian distribution with standard deviation equal to 3 % of the forecasted value. A complete summary with all hyperparameters used can be found in appendix A.1.

Figure 7.4 visualises the actions of the trained agent throughout an arbitrary day (24 hours) together with the solar irradiance. Simply put, it is desired that the demand change follows the curve of the solar irradiance around noon, and goes down in the afternoon.

By inspecting figure 7.4 it is evident that the agent activates flexibility in hours with high solar production for several of the loads. The plot `load = 12` clearly follows this pattern. On the other hand, the plot for `load = 8` shows some peculiar behaviour. The demand change does not follow the solar profile that day, but is negative most of the day. This behaviour could be a result of how the reward function is defined in this experiment. The agent is penalised according to the total energy imbalance in the system, and not at individual loads. As a result, if the energy imbalance is +1 MWh at one load and -1 MWh at another load, they perfectly cancel each other, and the agent is not penalised. From the agent's perspective, the system is in energy balance, although individual loads may have a large absolute energy imbalance. This illustrates the problem with constructing state variable that accounts for the system as a whole, and not individual loads. The agent uses the same strategy consistently at this load. It can appear as if this load functions as an energy balance, whose main job is to ensure that the total power imbalance in the grid is kept as small as possible. However, the behaviour of the agent gives a negative energy imbalance in the long term, as seen in figure 7.5. The agent controls a 200-hour long episode, and the energy imbalance quickly decreases and reaches an equilibrium around

-13 MWh. The agent prefers to decrease the total consumption in the system. It indicates that the reward function needs further parameter tuning.

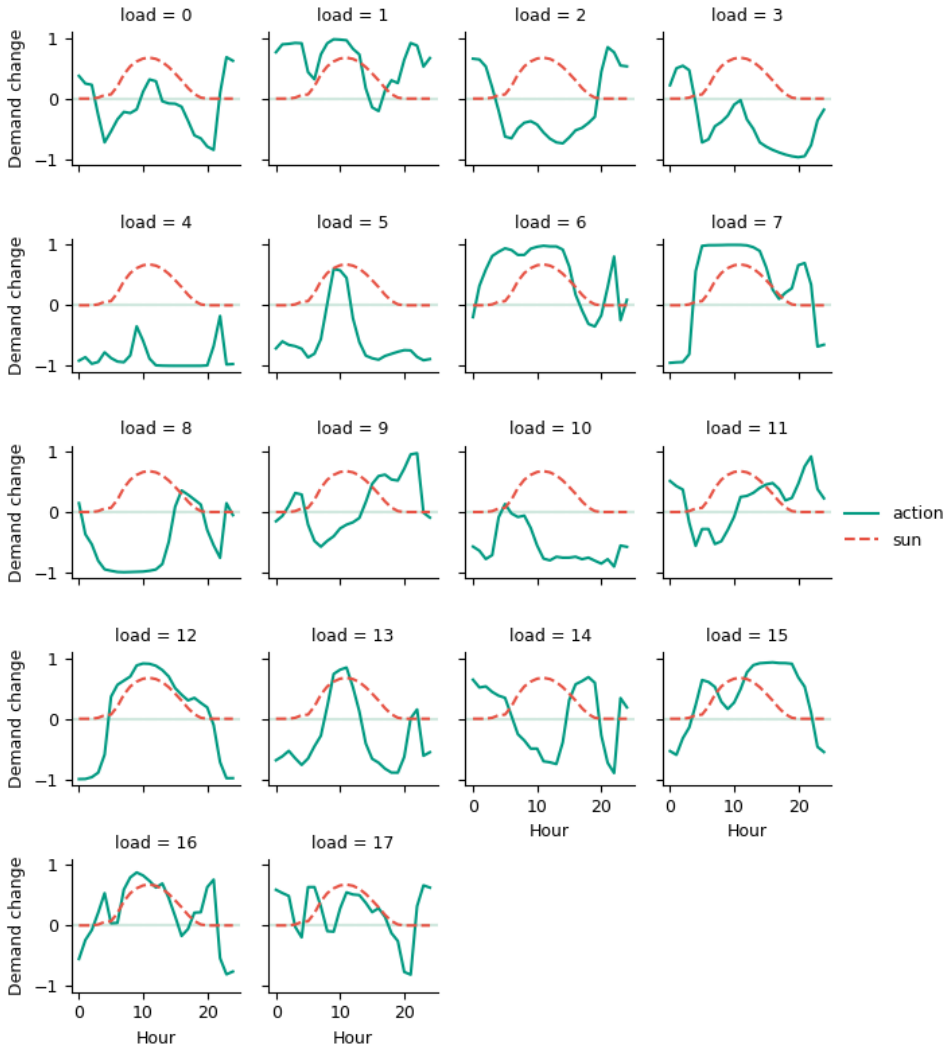


Figure 7.4: Activation of flexibility at the different flexible loads of the trained agent throughout a day. The red line is the solar irradiance during the day, which is the same for all loads. The actions in green is the activation of flexibility. Demand change = 1 means that the flexible load increases its power consumption with 10 %, while -1 means that it decreases its power consumption with 10 %. The solar irradiance is unitless, and plotted to show the relation between the action and solar profile.

7.2.1 Voltage violations

The reward function has an energy imbalance term during training that is meant to incentives the agent to shift the energy consumption. However, the main goal of the agent is to keep the values for current and voltage within safety margins. Therefore, it is interesting to see the rewards of the trained model when only the voltage and current terms are considered. In other words, the agent is only penalised for violating safety margins for current and voltage in the grid. The trained model was tested for 500 episodes, each consisting of 200 steps, and evaluated in terms of voltage penalty and current penalty

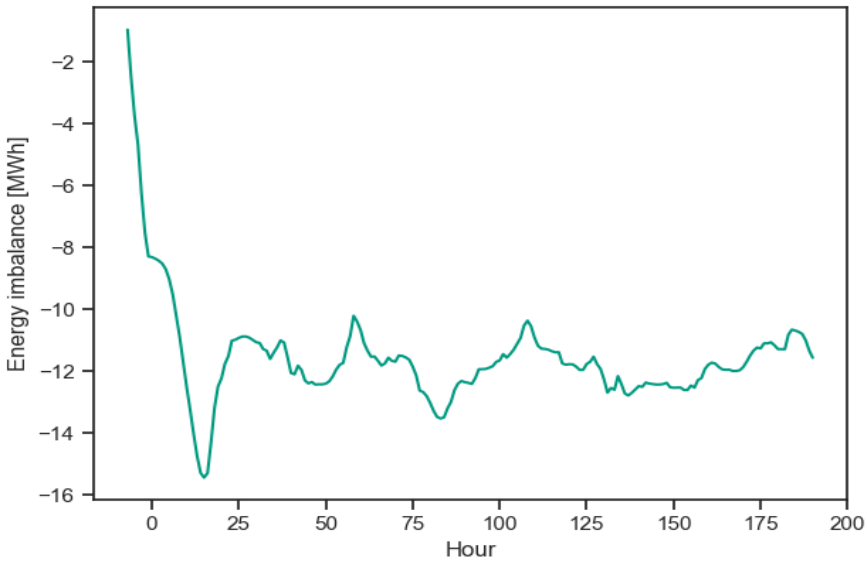


Figure 7.5: Total energy imbalance in the system during a 200-hour episode controlled by the trained agent

independently. The statistics for the hours with a non-zero reward using only voltage reward are presented in table 7.3. Note that every non-zero reward corresponds to a violation of voltage margin because only the voltage term is used to calculate the reward. The trained agent reduces the number of voltage violations by 18 % from 8804 to 7253. Note that there is no guarantee that it is physically feasible to satisfy voltage safety margins given a demand flexibility of 10 %. In addition to reducing the number of violations, the trained agent reduces the mean voltage penalty by 8%.

Table 7.3: Statistics of the voltage penalty given to the trained agent and no agent scenario over 500 episodes, each with 200 hours

	count	mean	std	25%	50%	75%	max
Agent	7253	0.035	0.037	0.006	0.021	0.052	0.237
No agent	8804	0.037	0.039	0.007	0.023	0.057	0.227

Figure 7.6 shows the voltage penalty distribution for the trained agent and no agent scenario for hours with violations of voltage safety margins. These hours are termed critical since there would be voltage violations if no actions were taken. The trained agent is better in critical situations, as more of the penalties are situated closer to zero.

The statistics presented in table 7.4 show that the mean voltage penalty for the trained agent is reduced by 24 % in critical hours.

Table 7.4: Statistics of the voltage penalty given in critical hours to the trained agent and no agent scenario over a 500-episode simulation with 200 hours each

	count	mean	std	25%	50%	75%	max
Agent	8804	0.028	0.036	0.000	0.014	0.044	0.237
No agent	8804	0.037	0.039	0.007	0.023	0.057	0.227

All voltage penalties in the no agent scenario for critical hours are sorted and plotted in

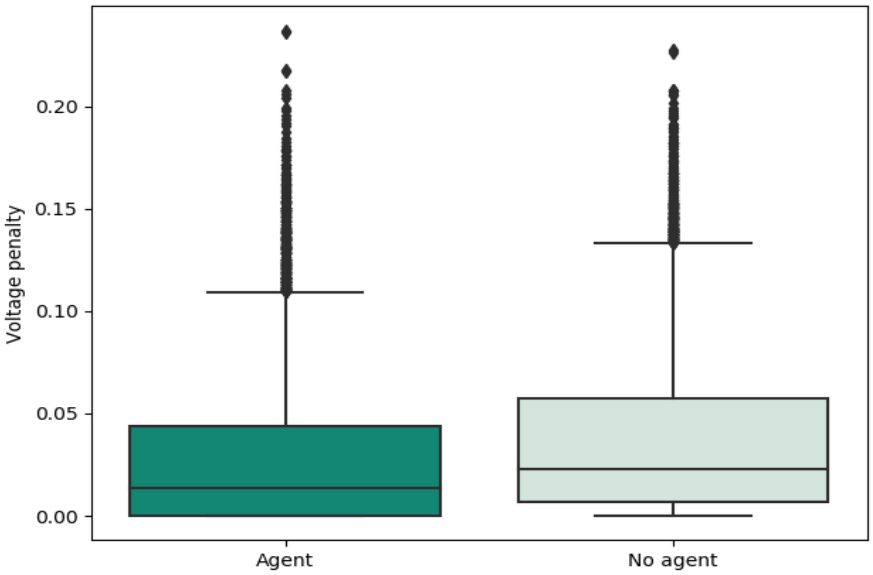


Figure 7.6: Box plot of the voltage penalty distribution of the trained agent and no agent scenario for critical hours where safety margins are violated.

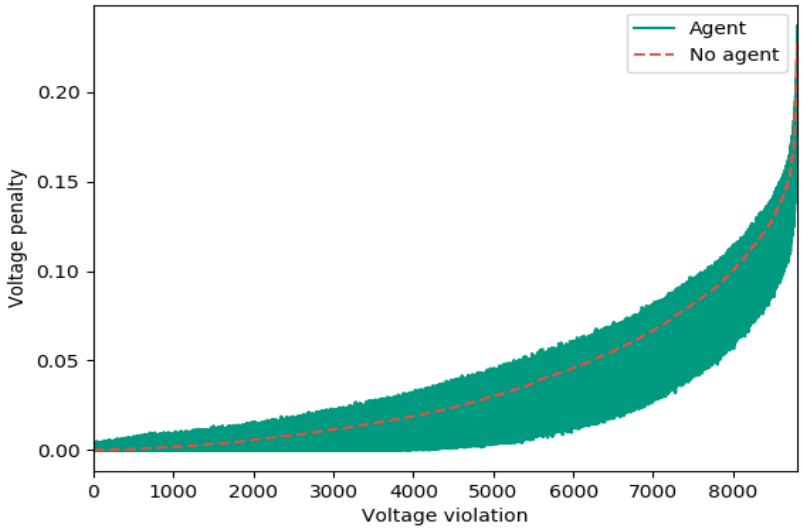


Figure 7.7: Voltage penalties given to the trained agent and no agent scenario in critical hours. The penalties are sorted for the no agent scenario from least to most severe

figure 7.7. The corresponding voltage penalties given to the trained agent are also plotted. This is done to see if the trained agent is better in very critical periods with severe voltage penalties, or if it performs well overall. The plot shows that the trained agent for the most part receives lower voltage penalties regardless of the size of the violation. Still, there are occasions where the trained agent is above the no agent scenario in figure 7.7 which corresponds to periods where the trained agent aggravates the voltage magnitudes. The trained agent makes matters worse in 40% of the critical hours, and increases the mean penalty with 29 % in those hours. However, the trained agent improves the voltage

margins in 60 % of the critical hours, where it decreases the mean voltage penalty by 57%.

Figure 7.8 shows for each hour of a day a box plot of the difference between the voltage reward given to the trained agent and no agent scenario in critical hours. This plot illustrates the improvements of the trained agent, measured in terms of the increase in voltage reward. Evidently, the trained agent is only improving the voltage situations in the afternoon, in periods of peak demand. This explains the times the agent makes matters worse in figure 7.7. 40 % of the voltage violations occur before peak demand hours (18 pm), which is the same as the proportion of the critical hours where the trained agent makes matters worse.

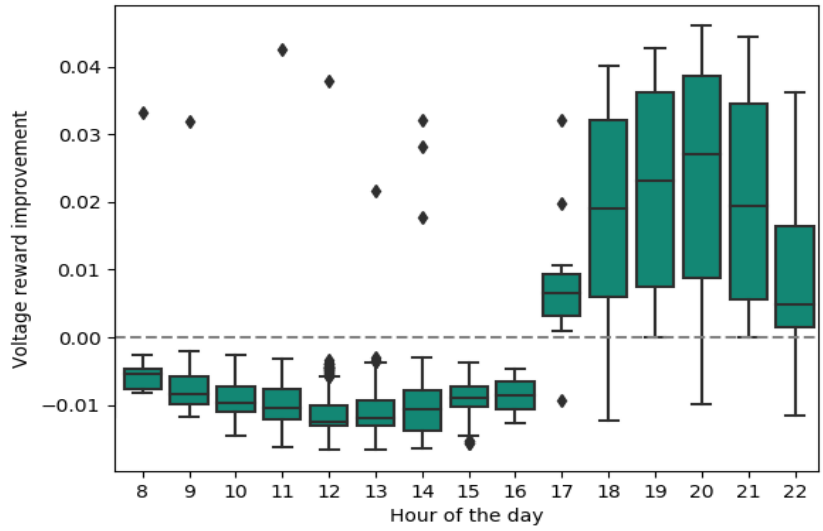


Figure 7.8: Hourly box plots of the difference in voltage rewards between the trained agent and no agent scenario in critical hours. Positive values mean that the trained agent is better

Until now, the trained agent has been evaluated in critical hours, where there would be a safety violation in the no agent scenario. However, the electric grid is within safety margins 91 % of the time in the 500-episode simulation. It is very important that the trained agent finds appropriate behaviour in both non-critical and critical hours. Investigating the voltage penalty given to the agent in non-critical hours reveals that the agent pushes the voltages out of the safety margins 0.6 % of the time. The worst of these violations is -0.004 in magnitude, which is negligible compared the violations in critical hours. In other words, the agent behaves well in non-critical hours in terms of voltage.

7.2.2 Current violations

The trained agent is tested in the same 500-episode simulation, each consisting of 200 hours, and evaluated in terms of violations of current safety margins in the power lines. Specifically, the current cost C_{current} for a line carrying a current I with capacity I_{upper} is $C_{\text{current}} = \max(0, I - I_{\text{max}})$ where I_{upper} is set to 90 % of the line capacity. The total current cost is found by summing over all lines. Note that the terms *penalty* and *reward* are used interchangeably, depending on what is most natural. In all cases the reward is the negative of the penalty. The statistics for the current violation of the trained agent and no agent scenario are presented in table 7.5

Table 7.5: Statistics of the current penalty given to the trained agent and no agent scenario over a 500-episode simulation, each with a duration of 200 hours

	count	mean	std	25%	50%	75%	max
Agent	1556	0.174	0.146	0.053	0.138	0.268	0.806
No agent	1487	0.161	0.142	0.045	0.121	0.250	0.787

The trained agent increases the number of current violations by 5 % from 1487 to 1556. In addition, the mean magnitude of current penalty increases by 9 %. The agent has not learned an appropriate behaviour to reduce current overloads in the lines. The hours in the simulation that have a current violation if no action is taken are termed critical hours. Figure 7.9 shows the distribution of the current penalties for the trained agent and no agent scenario in critical hours. The trained agent is worse because its current penalty distribution is higher than for the no agent scenario.

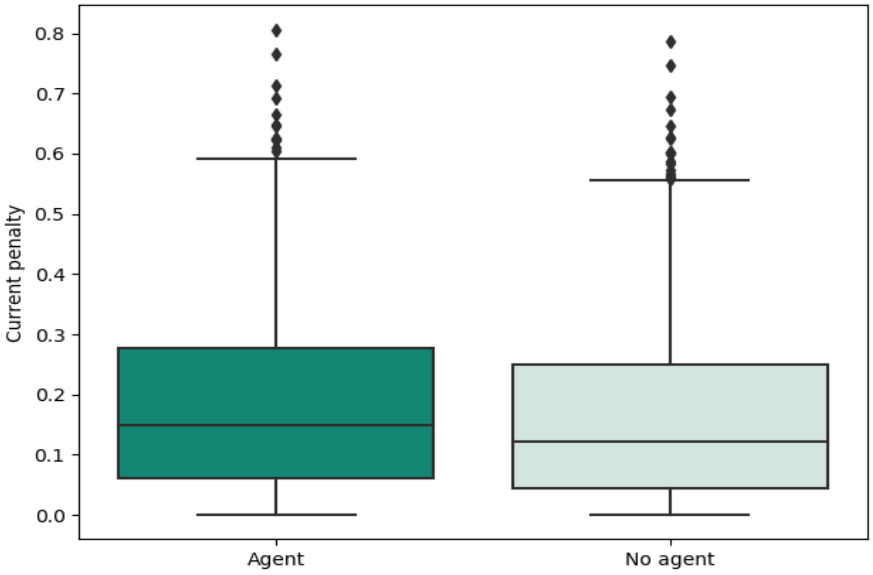


Figure 7.9: Box plot of the current penalty for the trained agent and no agent scenario in critical hours

Figure 7.10 plots the current violations for critical hours in the 500-episode simulation sorted from least to most severe. The hour with the lowest current penalty is the least critical hour. It is clear that the trained agent for the most part aggravates the situation. In fact, it only improves the situation in 9% of the critical hours, which for the most part are less critical in terms of the size of the current penalty.

Figure 7.11 shows hourly box plots for difference in current reward between the trained agent and the no agent scenario. This figure illustrates which hours of the day the trained agent performs well. For current penalty, the trained agent aggravates the situation in all hours before peak demand. The agent improves the situation in terms of current magnitudes after 18 pm. However, 91 % of the current violations happen during peak solar production, so the net effect of the improvements is small. 91 % is also the proportion of times the trained agent makes matters worse in critical hours in terms of the magnitude of the current violations, as shown in figure 7.10. Figure 7.12 illustrates a period where current violations occur during peak solar production.

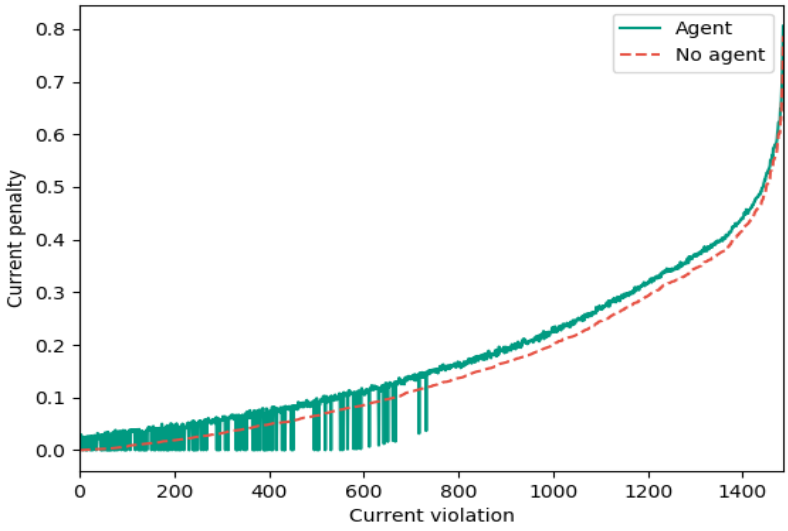


Figure 7.10: Current penalties in critical hours sorted from least to most severe for the trained agent and no agent scenario

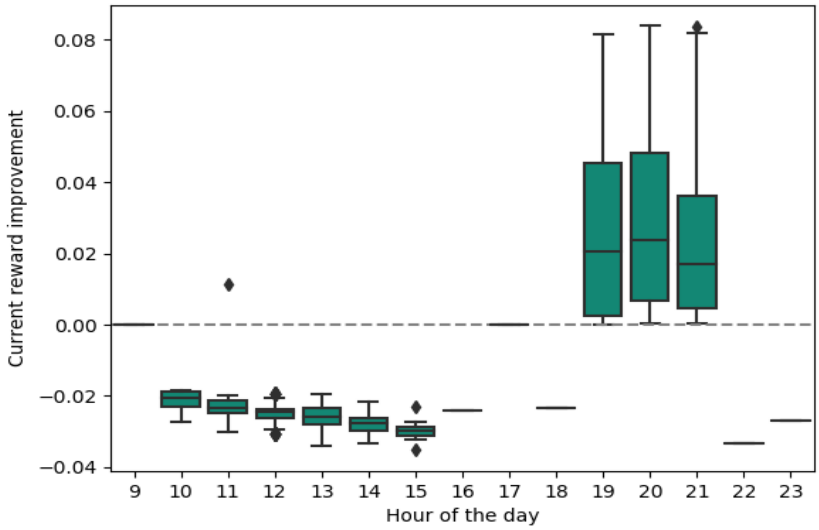


Figure 7.11: Hourly box plots of the difference in current rewards between the trained agent and no agent scenario in critical hours. Positive values mean that the trained agent is better

Critical hours in terms of current violations account for 1.6 % of the timesteps in the 500-episode simulation. The trained agent creates current violations in non-critical hours 0.2% of the times, and they are small compared to current violations in critical hours. In other words, the agent behaves well in non-critical hours in terms of current.

7.2.3 Summary

The trained agent shows a desired behaviour in terms of voltage magnitudes in periods of peak power demand. The power consumption is decreased in those periods, which keeps

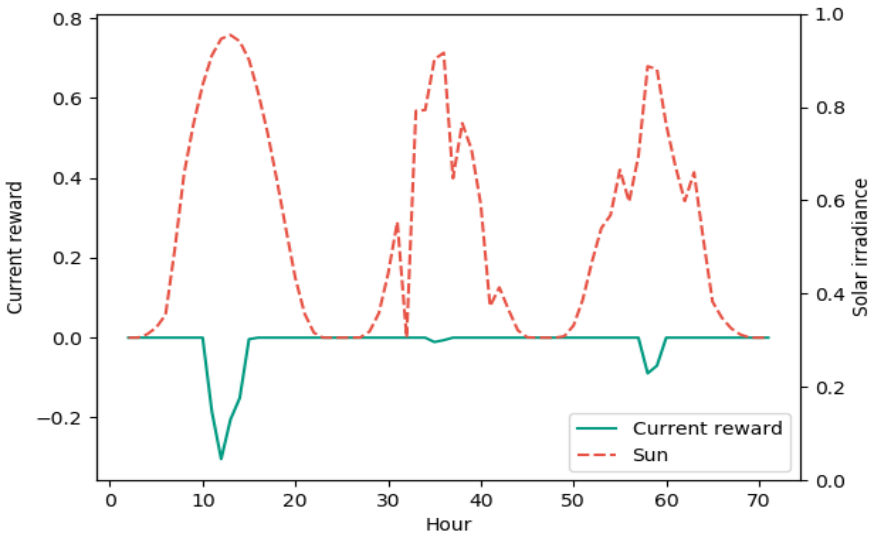


Figure 7.12: Current reward and solar irradiance profile in a three-day period. The agent is not able to prevent current overloads in hours of peak solar irradiance.

the voltage magnitudes above the lower safety margins. However, the trained agent has not found a strategy that keeps voltage magnitudes below upper margins in periods with peak solar production.

The agent performs poorly in terms of avoiding current overloads in the transmission lines. This is mainly a problem in periods of peak solar production, because there were relatively few current overloads in the lines in hours of peak demand. Still, because it is improving the voltage situation during peak demand, it also reduces the current loading in the transmission lines. Therefore, the agent performs well in terms of current in periods of peak demand. The behaviour of the agent in hours of peak demand and peak solar production is summarised in table 7.6.

Table 7.6: Behaviour of the agent in critical periods in terms of current and voltage safety margins

	Current behaviour	Voltage behaviour
Peak demand	Good	Good
Peak solar production	Poor	Poor

A summary of the performance in terms of voltage and current is presented in table 7.7. Overall, the trained agent reduces the number of voltage and current violations by 14%. However, it increases the mean penalty by 7 % due to its poor behaviour in terms of avoiding current violations. Still, the trained agent is better in terms of the total penalty received in the 500-episode simulation because voltage violations are more frequent. The total penalty in terms of both current and voltage is reduced by 8 %.

Table 7.7: Summary of number of violations and mean penalty in terms of current and voltage for the trained agent and the no agent scenario

Type		Agent	No agent	Change
Current	Violations	1556	1487	5%
	Mean penalty	0,17	0,16	9 %
	Penalty	271	239	14 %
Voltage	Violations	7253	8804	-18%
	Mean penalty	0,035	0,037	-8%
	Penalty	251.3	330.1	-24 %
Total	Violations	8809	10291	-14%
	Mean penalty	0.059	0.055	7 %
	Penalty	**522.3**	**569.1**	**-8%**

Chapter 8

Discussion

8.1 Voltage and current impact

The action of the reinforcement agent determines the percentage change in demand at each load in the interval $[-f, f]$, where f is the flexibility of demand. Naturally, the load is varying a lot throughout a day, but the nominal demand also varies a lot from bus to bus. Figure 8.1 shows the values for nominal apparent power at each bus bar in the power system, as predefined in `pandapower`.

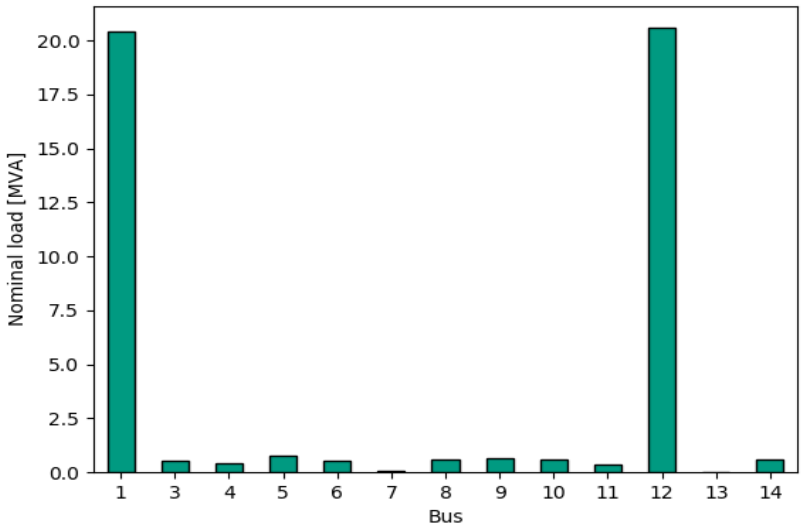


Figure 8.1: Bar plot of the nominal apparent power at each bus in the CIGRE network

Bus 1 and 12 clearly stand out, and account for nearly 90 % of the total system demand. For simplicity, they will be referred to as the dominant buses due their large demand. Note that there is no solar power connected to them. Because each action variable is scaled up by the nominal load, it is clear that action +1 for the dominant buses has a much larger impact in terms of absolute power change than it has on the rest of the buses. As shown in figure 8.2, The dominant buses are placed in the top of each feeder 1 and feeder 2, connected to each their 220/22 kV transformer.

It might seem troubling that the dominant buses have much larger nominal values. However, because they are placed next to the grid, they do not affect the line current in

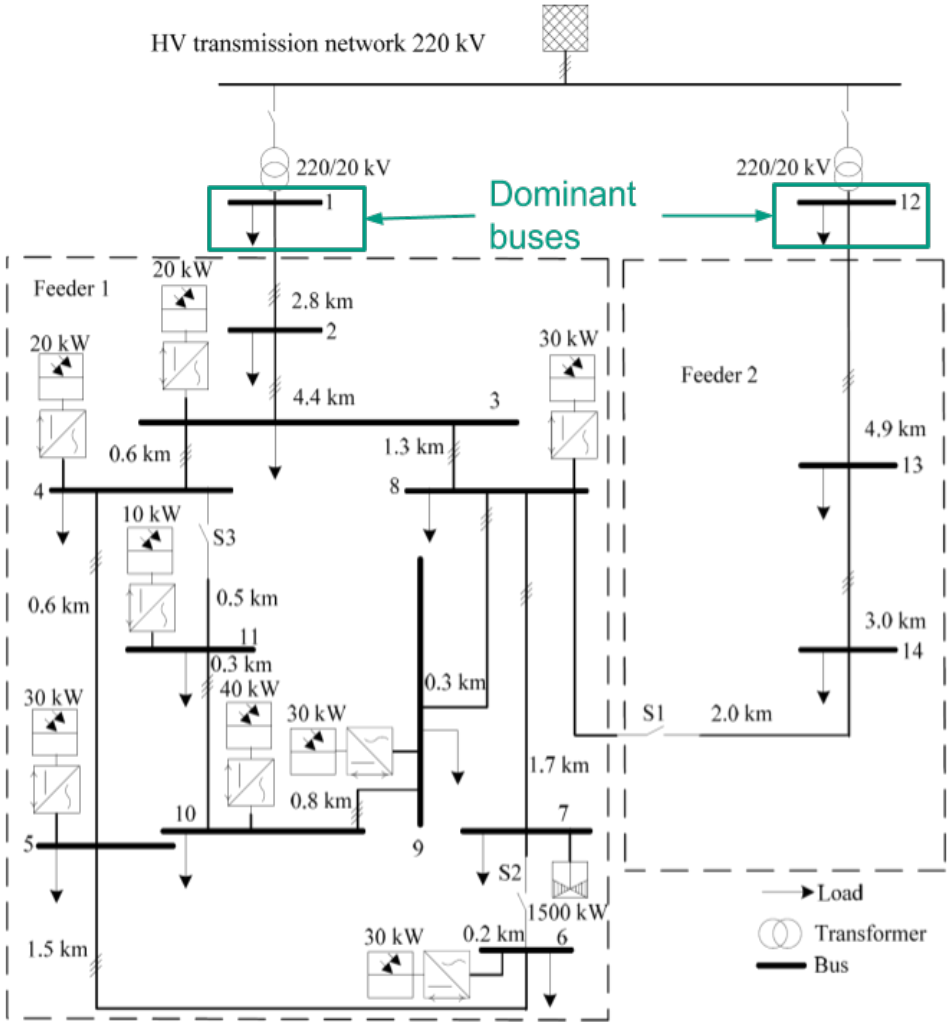


Figure 8.2: CIGRE network with solar and wind power that is used in the reinforcement learning algorithm [15]. The dominant buses account for approximately 90 % of the power consumption in the grid

the rest of the system much. For instance, if the consumption at bus 1 is doubled, the current through the transformer approximately doubles to supply the demand. Naturally, this could be very critical for the transformer, but it has a small effect on the line currents in the power system, because the rest of the loads still draw the same amount of power from the grid. The line current effect of changing the demand at bus 1 and 12 are not that decisive as one first might think, due to their position close to the external grid. On the other hand, they can greatly affect the voltage magnitudes in the grid. Figure 8.3 illustrates the effect increasing the demand at the dominant buses has on the rest of the buses. The predefined demand values are used for solving the power flow equations in situation A, while they are doubled for bus 1 in situation B, leaving the other buses unchanged.

The dominant buses serve as the starting point for the voltage magnitudes because they are at the beginning of the two feeders. This is true because there are only PQ-buses in the system and no voltage regulating units. The voltage is static at the external grid, and gradually falls as we move down the feeders in periods of peak demand. The voltage

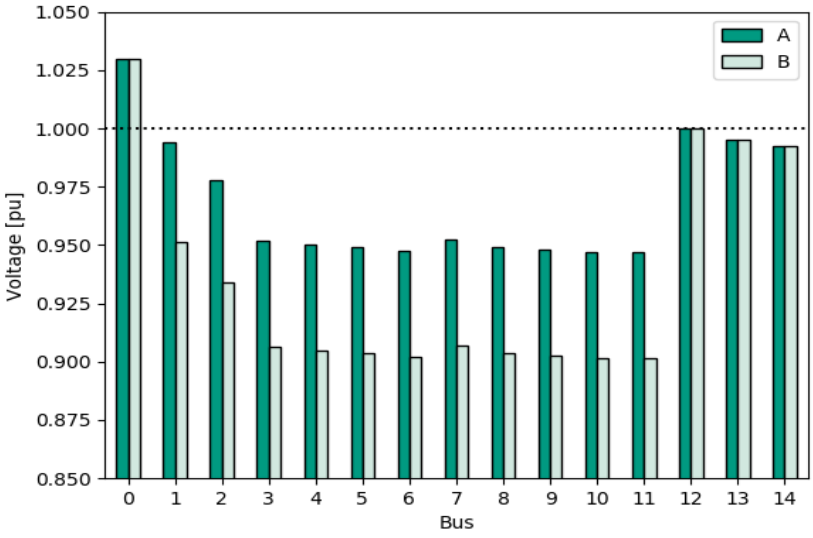


Figure 8.3: Bar plot for each of the buses showing their voltage magnitude in nominal operation (A) and after the demand at bus 1 in figure 8.2 is doubled (B). Note that the vertical axis is truncated

magnitude at bus 1 is reduced when it doubles its demand, which in turn propagates down the feeder and affects all buses in that feeder. The buses in the right feeder are unaffected by the demand change at bus 1 because switch 1 (S1) in figure 8.2 is open. The reinforcement agent is not allowed to double the demand in the reinforcement algorithm, but figure 8.3 illustrates the decisive effect of the dominant buses in terms of voltage magnitudes.

A measure of impact is needed to systematically investigate the effects of changing the demand at a bus. Let the voltage and current impact of a bus respectively be defined as the sum of the changes in voltage and current loading in the net when modifying the demand at that bus by a certain percent. Note that the rest of the buses are left unchanged when calculating the impact. For instance, the voltage impact of bus 1 in figure 8.3 is the voltage difference of scenario A and B, summed over all buses. The same can be done for the current impact, which is the difference in current loading summed over all lines. The current loading in a line is a percentage of the line capacity, not ampere. The current impact is therefore weighted against the capacity of the line. The impact of a bus will be discussed for the two critical periods of the day, namely peak solar production and peak demand. For peak solar production, the average demand and maximum solar production at 12 am during the 500-episode test simulation are used to define the values for consumption and production. For peak demand, the average solar production and maximum demand at 8 pm are used.

Figure 8.4 plots the voltage impact during peak demand for each bus where the demand change ranges from -20 to +20 %. The lower part of the box plots corresponds to increasing the demand by 20 %, since increasing the consumption in peak demand lowers the voltage magnitudes in the grid. As already discussed, bus 1 has a large voltage impact, because it is at the top of feeder 1.

Figure 8.5 plots the current loading impact in peak demand hours for the buses in the net, where the demand change ranges from -20 to +20 %. The lower part of the box

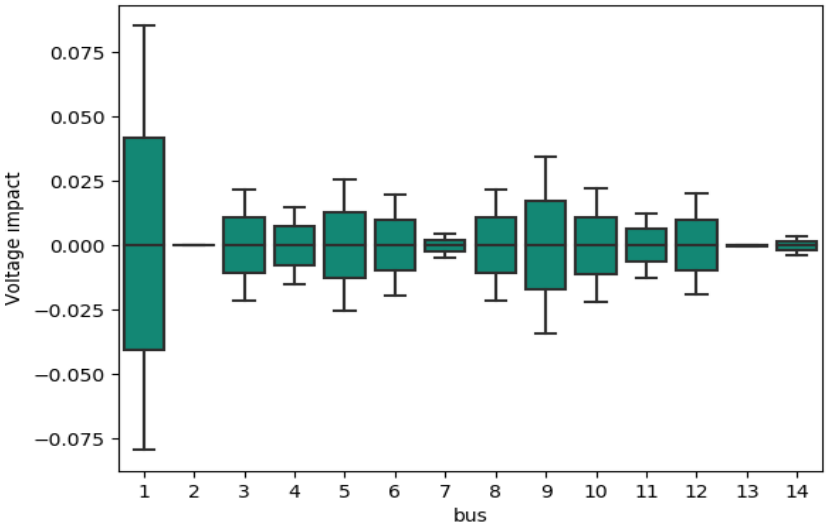


Figure 8.4: Box plot for each bus bar showing the voltage impact for a flexibility of demand ranging from -20 to + 20 % in an hour of peak demand

plots corresponds to decreasing the consumption by 20 %, since less current is needed to supply the demand. The buses with highest current impact (5, 6, 9, 10) are all located far down in feeder 1. A change in demand at them has consequences for several lines, because the current they draw must travel through the lines in the top of the feeder. As discussed, the current loading impact is low for the dominant buses (1 and 12), despite the fact that they account for nearly 90 % of the consumption in the system.

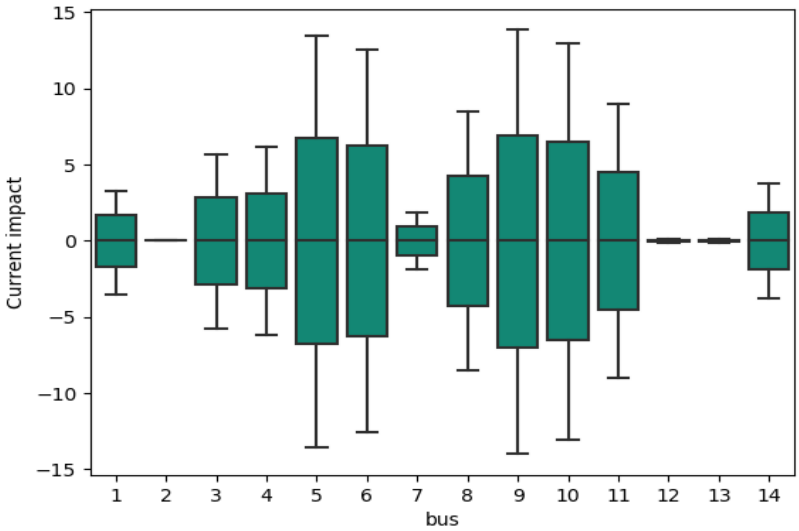


Figure 8.5: Box plot for each bus bar showing the current impact during peak demand for a demand change ranging from -20 to + 20 %

So far, the impacts of the buses have been investigated in hours of peak demand. A similar simulation in an hour of peak solar production can be done to show the impact

of the buses in the second critical period. The impact relation between the buses in peak solar hours is similar to the peak demand period. In other words, if a bus has a large impact in peak demand, it also has large impact during peak solar production compared to the other buses. However, the magnitude differs a lot for the two critical periods. Figure 8.6 plots the current loading impact factor for each bus when the demand is changed by 10% in hours of peak demand and peak solar production. The current impacts during peak solar production are lower in all cases, except for bus 1. Similarly, figure 8.7 shows the difference in voltage impact during peak demand and peak solar production. The impacts differ because the power consumption generally is lower during peak solar production. The mean demand during maximum solar production at 12 am is less than half of the max demand value at 8 pm. Consequently, changing the demand by 10 % at 12 am results in an absolute power change that is about half the change during peak demand. The agent simply has less muscles during peak solar production, because it is physically impossible to affect the grid as much as in the afternoon during peak demand. This can explain why the trained agent performs well in periods of peak demand and poorly in periods of peak solar production.

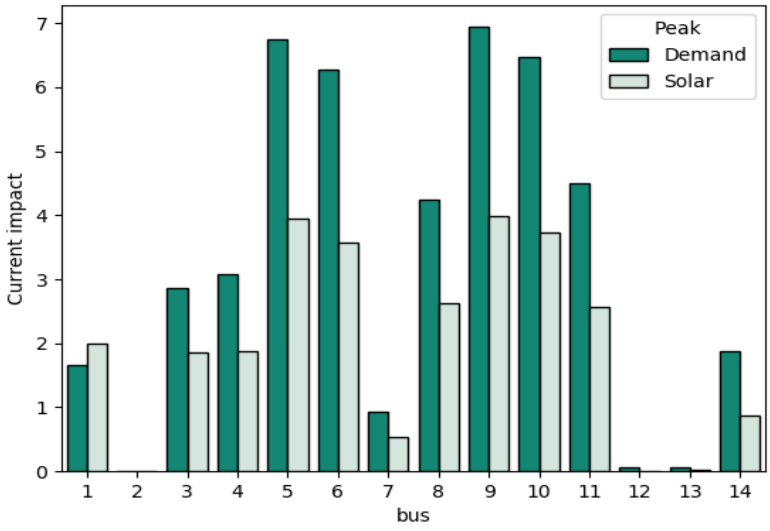


Figure 8.6: Current impact of all the buses in an hour of peak demand and peak solar production. The demand is changed by 10 %

8.2 Performance of the trained agent

The results from section 7.2 show that the trained agent is able to reduce the number of safety violation by 14%. However, the trained agent is only able to reduce violations of voltage safety margins, not current violations. In fact, it increases the number of current violations by 5 %. Still, there are large differences in terms of the quantity and magnitude of the current and voltage safety violations. There are almost 6 times more voltage violations than current violations. This is of course dependent on the voltage bounds that are used for defining the voltage cost, which in this case are 0.95 and 1.05 pu. Although there are many more voltage violations in a normal day, the average current violation is more severe. Specifically, the mean current cost is over 4 times greater than

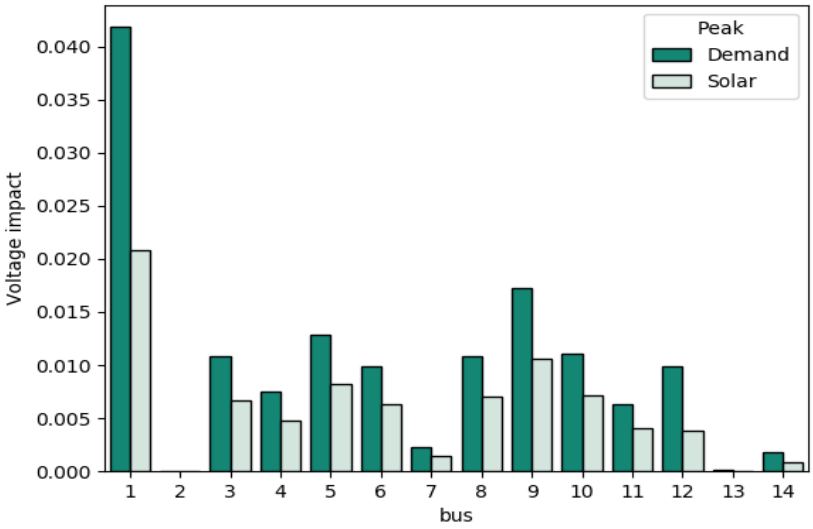


Figure 8.7: Voltage impact of all the buses in an hour of peak demand and peak solar production. The demand is changed by 10 %

the mean voltage cost. The nature of the transmission in terms of violations is therefore: the current violations are sparse and severe, while the voltage violations are numerous and faint.

Why is the trained agent better at avoiding voltage violation than current violations? A possible reason is that the agent is penalised more for voltage violations on average. The total voltage cost is 38 % greater than the total current cost in the 500-episode simulation, because they are more frequent. It is sensible that the agent learns a behaviour that reduces the most punishing term, namely the voltage cost. As stated in section 7.2.2, the trained agent only learned the appropriate behaviour in periods of peak demand. This can be explained by the time of the day the voltage violations occur. 60 % of the voltage violations occur during peak demand (after 17 pm) in the 500-episode test simulation. In other words, voltage violations are the most punishing term and they are most frequent in the afternoon. Imagine that the agent has the correct behaviour in the beginning of training, i.e. that it increases consumption during peak solar and decreases it during peak demand. The agent will experience a greater boost in reward by behaving correctly in peak demand, because then the voltage cost is highest. In addition, the agent has a greater voltage and current impact during peak demand, as discussed in section 8.1. Put differently, it is easier for the agent to learn the correct behaviour during peak demand. It is natural that the agent first focuses on the low-hanging fruits, and decreases the power demand.

The agent is worsening the situation in periods of peak solar production. The desired behaviour in such a situation is to increase the demand, so that less excess solar power needs to be exported to the grid. However, the trained agent decreases the demand in periods of peak solar production, since the number of current overload increases. It is possible that the learned behaviour simply is to decrease the demand at all times, especially when considering that the energy imbalance in the system is negative, as shown in figure 7.5.

Figure 8.8 shows the mean action taken by the agent throughout the day in the 500-

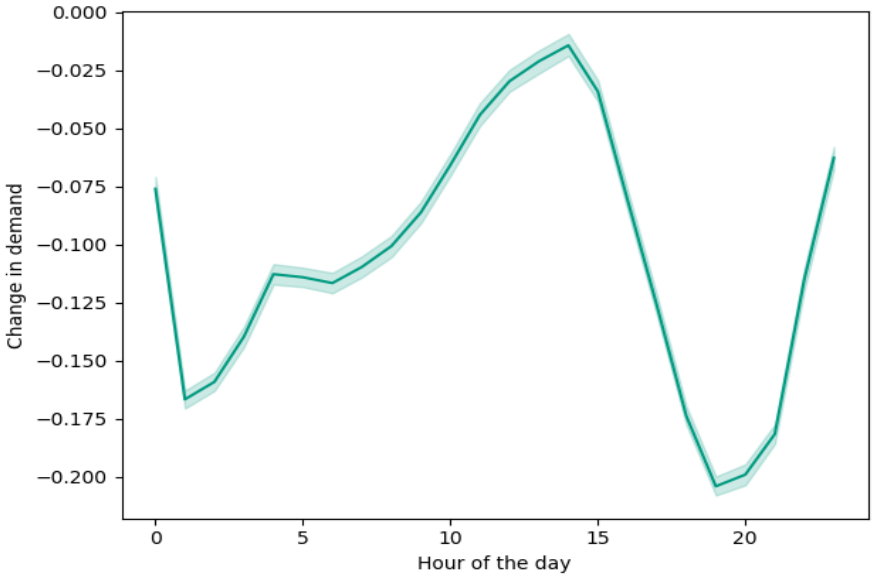


Figure 8.8: Hourly mean values for the change in demand at the buses in the net, as determined by the trained reinforcement agent. The error region is a 95 % confidence interval of the mean value

episode simulation, with a 95 % confidence interval. The pattern of the actions is as desired, because it goes up during peak solar producing and down during peak demand in the afternoon. However, the change in demand during peak demand is still negative. The action signal seems to have a negative bias. It is also worth noting that the mean action is quite low in absolute magnitude. In other words, the agent is not synchronising the buses such that the entire available flexibility in the system used.

Figure 8.9 shows mean values for demand change throughout a day for each bus. The buses 1, 6, 8 and 9 are on average negative in every hour, i.e. the agent always reduces the consumption. As discussed, the different buses have different voltage and load current impact, and it is interesting to see whether there is a relationship between actions and load current/voltage impact of the buses. First it can be noted that bus 1 (top left corner) which has the largest voltage impact in hours of peak demand mainly is negative in an average day. This can suggest that the trained agent has learned that lowering the consumption is beneficial in peak demand hours, and therefore has a strong negative bias that pulls the demand change to negative values throughout the day. The buses with strongest current impact are 5, 6, 9 and 10, and they do not show a particularly desired pattern. Only bus 10 increases the power consumption during peak solar. Bus 9, which has a high current and voltage impact, has a reasonable curve shape, but it is negatively biased which results in a decrease in power consumption in all hours. Perhaps the most sensible mean action signal is for bus 11. Still, there is no obvious pattern that suggests that the trained agent has learned a particularly well-coordinated behaviour. It might very well be that it is the agent's negative bias, i.e. its tendency to always decrease power consumption, that leads to less voltage violations in the grid.

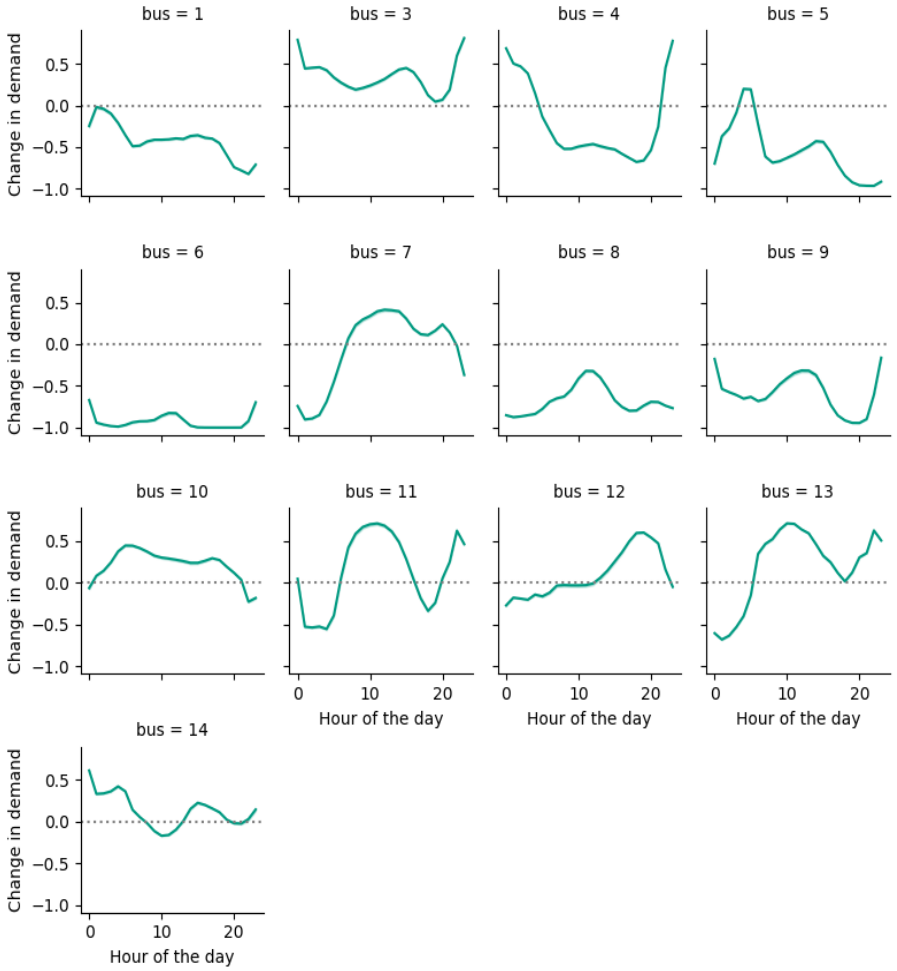


Figure 8.9: Hourly mean values for the change in demand in the net for each bus, as determined by the trained reinforcement agent. The error region is a 95% confidence interval

8.3 Solar power production

The nominal solar production level predefined in the network is scaled up by a factor of 40. This is done to challenge the grid by increasing the number of current and voltage violations in hours of peak solar production. The maximum values for solar production and consumption at the solar producing buses are presented in figure 8.10. Naturally, the demand and solar production varies throughout the day. Around maximum solar production, the relation between the mean solar production and mean consumption is similar to the relation showed in figure 8.10. The ratio between solar production and demand at bus 7 is very high, and should probably have been scaled differently than the other buses. The ratio for the rest of the buses ranges from 1.2 to 2.8.

8.4 State representation

Bus state

There are many different ways of constructing the state space in the reinforcement agent. In section 4.1 several candidates for the state space were introduced, such as the bus space.

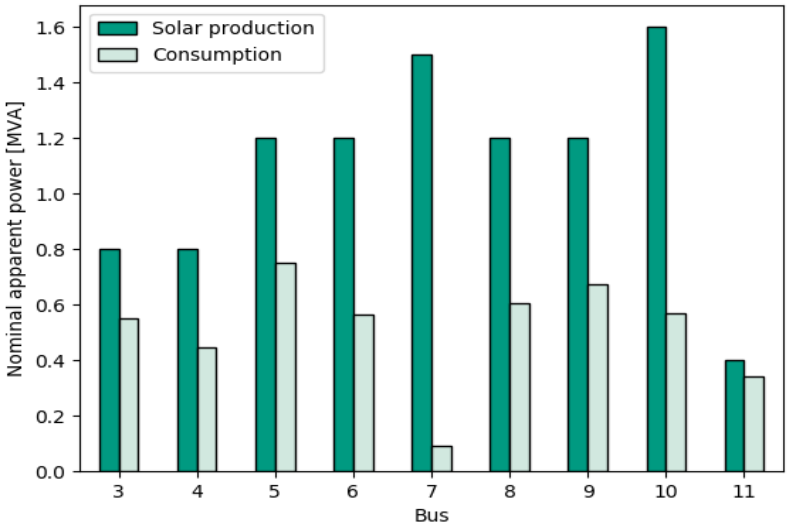


Figure 8.10: Bar plot of the nominal apparent power consumption and solar production at solar producing buses in the grid

The bus space consists of the active power P, reactive power Q, voltage magnitude $|U|$ and phase angle δ for all the buses in the system. It was not included in the state space of the trained reinforcement agent. A reason for this is that the bus state in an hour gives limited information about the future. Naturally, there is a correlation between the bus state of the current and next step. For instance, during peak demand in the afternoon, the demand between two hours are highly correlated, which means that the bus states also are correlated. However, this information can be found in the demand forecasts. In some sense, the bus state in the current time step is redundant. During peak solar production, we can have an hour with heavy production from solar units, but in the next hour the forecasts says that there will be cloudy. It is not possible to predict the clouds from the bus state. On the contrary, the agent can be fooled to use the bus state to choose actions, when it really is the forecast is should use, because the predictive information is hidden in the forecast. However, the forecasts can be used to predict the next bus state, which can be helpful information for the agent. The downside is that this will slow down the training process, because the power flows equation must be solved, possibly several times if many future bus states are going to be estimated.

State influence of actions

The actions of the agent in a reinforcement learning algorithm interact with the state of the environment. For instance, an agent playing chess can choose to move a pawn based on the state representation of the board. This move directly determines the next board position, and therefore also the next state. The pawn action is very committing since a pawn can not be moved back. The agent must therefore be very farsighted, and be able to estimate future rewards precisely. Let us for the sake of argument ignore the imbalance variable in the state space, and only consider the forecast space. The forecast space used in this thesis is entirely independent of whether the agent increases or decreases the demand power demand at a bus. There's no sophisticated long-term planning that the agent can do, because the agent does not affect the next forecast state. The optimal behaviour is

to maximise the immediate reward, because the next forecast state will be the same no matter what action is taken. Why would you not take the action that maximises the immediate reward, when your future rewards do not depend on the action? The only state variable the agent influences is the energy imbalance, which introduces the need for long term planning and not only to maximise immediate rewards.

The agent would however interact with the state variable in an a more realistic modelling of demand response, where there is a direct cost of activating flexibility, and when the future flexibility depends on the actions. If batteries also are included in the system, the agent will affect the future state more than in the experiment analysed in this thesis. A more realistic demand response model would therefore be more like a classical reinforcement learning setup, where an action heavily influences the possible actions in the future.

It should be noted that the agent still interacts with the environment, although the forecast state is not affected by the agent's actions. Altering the consumption affects the power flow equations, and therefore the number of violations in the power grid.

Bus specific information

The state space used for training the agent does not include information that is bus specific. The demand, which is expressed as a percentage of nominal power consumption levels at each bus, is assumed equal at every bus. If the demand at a bus is 60 % of nominal consumption, then the demand at every other bus is also 60 % of their nominal consumption. Naturally, this is an oversimplification since demand curves vary from bus to bus, depending on the type of load connected to the bus. For instance, a bus with power-intensive industry connected will have different daily curve than a bus with only residential buildings connected. Furthermore, the state space for a more realistic setup should include an hourly price of changing the demand.

Constant flexibility

The reinforcement learning agent is trained for a fixed flexibility. In other words, it is assumed that the consumption at all times can be increased and decreased by a fixed percentage. This is not a realistic view of flexibility. As an example, the available flexibility can change from workdays and weekends, as industry shuts down and there are also hourly variations of flexibility throughout a day. In addition, the flexibility at a bus should be influenced when the consumption in modified. For instance, an electric vehicle can start charging if the agent increases the consumption at a bus. Naturally, this would reduce the flexibility of the bus, since the electric vehicle might be fully charged the next hour, and simply cannot increase its consumption. Changing the power consumption in an hour will affect the future flexibility in the system. As a result, there must be a term in the state space that estimates the available flexibility for buses in the system in a realistic model of demand response.

8.5 Reward function

The reward function is designed to penalise violations of safety margins in the grid. Specifically, current overloads and violations of voltage safety margin are the factors that are used to calculate the reward signal. There are however two transformers in the grid, that also should be included in the reward function. Transformers are not fireproof.

`pandapower` has a result table for the transformer, that easily could be integrated in the reward function. This would make the current impact of the dominant buses greater as well.

The reward terms for current and voltage respectively sum over every line and bus bar. By this definition it is mathematically possible that the magnitude of a current violation in an individual line increases, but the overall penalty is less. For instance, consider the following scenario. Say that the current loading in three lines are 105%, 104% and 104%. With an upper current loading limit of 90 %, the current cost would be 15 + 14 + 14 = 43. Imagine that the reinforcement learning agent takes an action and that the resulting current loading in the three lines are 115 %, 97 % and 97%. The resulting current cost would then be 25 + 7 + 7 = 39, which is lower than the original one. According to the reward function, the second scenario is better, although an individual line is pushed further out of the safety margin, possibly damaging it severely. This is mathematically possible, but the question is whether this is physically realistic. The trained reinforcement agent controls the change in consumption in at the buses in the net. Referring the imagined scenario above, two of the tree lines reduce their current loading, and the way to achieve this in peak demand is by reducing the consumption at some buses. This would not affect the power drawn by the other buses or in any way cause larger current in any other line. On the contrary, decreasing the demand in peak demand would increase voltage magnitudes in the grid, which in turn would reduce the line current ($S = UI$). Consequently, it is not physically realistic that the described scenario happens in the reinforcement learning setup used in this thesis.

The same scenario is mathematically possible with voltage violations as well. Again, it is physically unrealistic that it would happen, because if the voltage magnitude at a bus increases, it will push up the voltage magnitudes in the rest of the buses.

The current and voltage cost increase linearly with the size of the violation. It is not necessary that the cost increases linearly, and it might be more appropriate to have a quadratic cost or exponential cost that would penalise large violations harder than the linear cost. Using a quadratic cost eliminates the mathematically possible reward scenario described above.

The agent is not penalised for changing the demand in the setup analysed. This is an unrealistic case of demand response, as costumers should be compensated for changing their consumption pattern. Why would someone postpone their dishwasher if they did not get some form for compensation? The reason for excluding the activation cost is that the agent then can solely focus on reducing the safety violations, without worrying about the economic side. A more complete reinforcement learning algorithm should include the activation cost, so that the agent can learn to operate the net safely at a low economic cost.

8.6 Energy imbalance

Figure 8.11 shows the distribution of imbalance rewards given to the trained agent in the 500-episode test simulation. The mean imbalance reward is -0.02, which is smaller than both the mean current and voltage cost in critical hours. However, there is a non-zero imbalance reward at every time step, in contrast to the current and voltage reward. The current and voltage reward are only non-zero when there is a violation in the grid, while an imbalance reward is given in every hour. Therefore, the imbalance cost is by far the most dominant factor in the total reward given to the agent. The total imbalance cost in the 500-episode simulation is 2039, almost four times as large as the current and

voltage costs combined. Naturally, this works against the original goal of reducing the number of safety violations in the grid. How is the agent suppose learn how to reduce the number of safety violations, when it is penalised much more for increasing the energy imbalance in the system? Moreover, why would it not simply learn to reduce the total energy imbalance? The trivial solution to this is to never change the demand in the grid, i.e. all actions are 0. Still, this is not the behaviour we observe, as it simply tends to decrease the energy imbalance in the system.

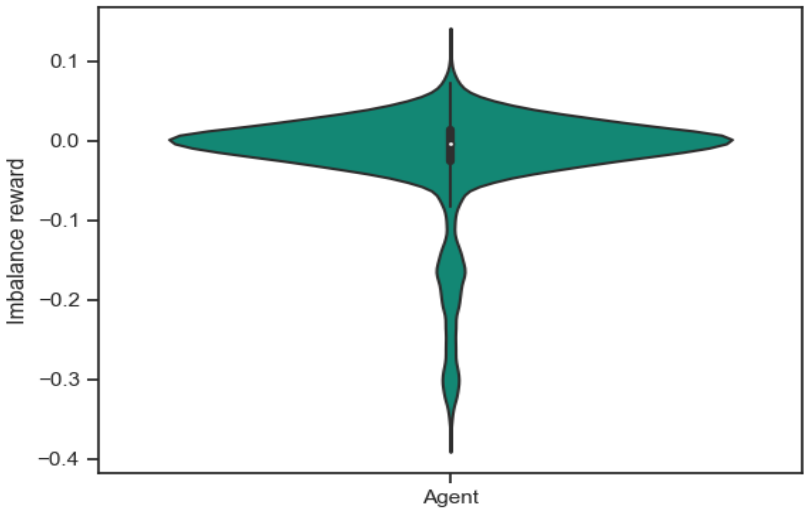


Figure 8.11: Violin plot of the imbalance reward given to the trained agent in the 500-episode simulation

The energy imbalance term gives a positive reward if the absolute energy imbalance in the system moves closer to 0. Thus, it is not penalised for the absolute magnitude of the energy imbalance in a transition from a state to the next state. This is equivalent to scenario where a freezing person is satisfied when the temperature changes from -30^oC to -28^oC, simply because it is moving closer to 0. Still, it is odd that the agent allows a decrease the energy imbalance, as this is heavily penalised. A reinforcement learning algorithm plans ahead and tries to maximise rewards in the future. The energy imbalance is proportional to the future imbalance reward with the current definition of the imbalance cost. For instance, if the energy imbalance is -10 MWh, the agent can increase the consumption by 0.1 MWh the next 100 time steps and reach 0 energy imbalance. The sum of the imbalance rewards given in the 100 steps is therefore proportional to the original energy imbalance of -10 MWh. Decreasing the energy consumption can therefore be seen as an investment that can be cashed in some time in the future by restoring the energy balance in the system. However, the trained agent does not cash in the imbalance cost. It is difficult to explain this behaviour. Nonetheless, the formulation of the energy imbalance cost can be considered a poor and misleading term in the reward function used in the reinforcement learning algorithm. A better approach is to decrease the weight for the imbalance cost, and penalise a large absolute energy imbalance.

8.7 Reinforcement learning algorithm

This thesis has not systematically tested different combinations of hyperparameters in the reinforcement learning algorithm. The default values for learning rate, neural network architecture and exploratory noise have been used. Naturally, performing a grid search to find optimal hyperparameters could boost the performance of the algorithm. Moreover, the training time consisted of 100 000 experiences, which is quite low considering that neural networks are used as functions approximators. One of the great features of neural network is that they continually can improve from new and unseen data. This was demonstrated in OpenAI's Dota Five reinforcement learning algorithm that had over 10-month wall-clock time training [25]. The relatively primitive behaviour of the trained agent could simply be a result of too little training time.

Deep deterministic policy gradient is an off-policy reinforcement learning algorithm, which means that it is capable of learning from the experiences of others. `stable_baseline` offers a method that can clone an expert behaviour using supervised learning. Considering the large action space (16 independent variables), it would probably be beneficial to pretrain the agent using the experiences of a simple baseline that increases the consumption during peak solar production, and decreases it during peak demand. The probability that the agent by chance coordinates its action such that the consumption is increased at all loads is the same as flipping a coin 16 times and getting tails every time. Pretraining would guide the agent such that it does not have to spend all the training time to find the basic behaviour, but can start exploring strategies that are closely related to the baseline.

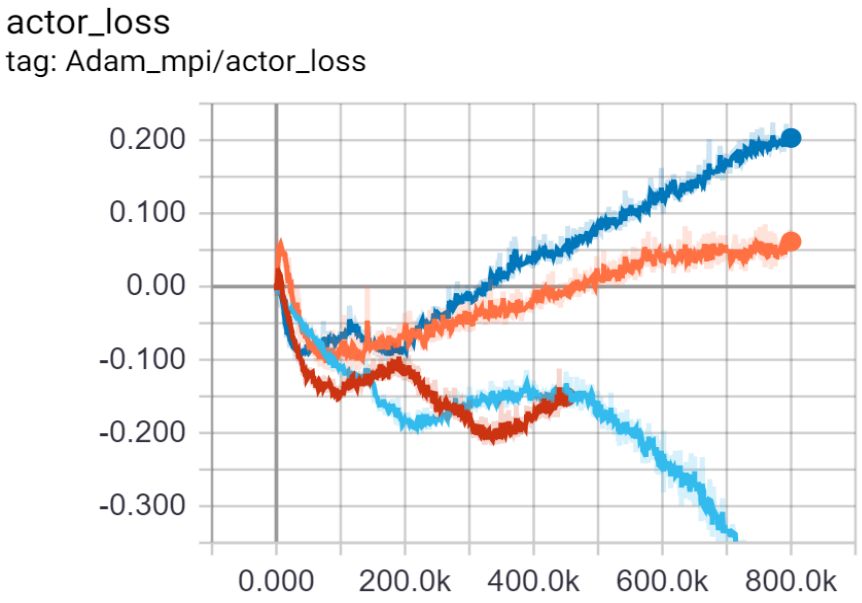


Figure 8.12: Actor loss during training of four different DDPG agents, where small hyperparameter changes have been performed.

The learning process is found to be sensitive to the hyperparameters in the reinforcement algorithm. Figure 8.12 plots the loss of the actor network during training for 4 different training sessions, with small changes in hyperparameters. The loss curves change a lot during training, and indicate that training of the environment is sensitive to hyperparameters, which is a known weakness in DDPG [37]. It might be beneficial to test other reinforcement algorithms. For instance, Soft Actor-Critic is a reinforcement algorithm available in `stable_baseline` that outperforms DDPG in several continuous

control tasks, and is shown to be stable during training for several different seeds [37]. Proximal Policy Optimization algorithms are also potential candidates [27].

There have frequently been comments about the desired behaviour of the agent in this thesis. A legitimate question is whether a reinforcement learning algorithm is needed if one can say what the desired behaviour is. Why not simply increase consumption as much as possible in hours of peak solar production, and then decrease it in hours of peak demand, as this is always referred to as the desired behaviour? This can be a reasonable baseline to which the reinforcement algorithm can be compared. The trained agent has not been tested against a baseline in this thesis, but it is unlikely that it would beat it. The trained agent only managed to decrease consumption in hours of peak demand, and made the situation worse elsewhere. The simple baseline described would probably beat the trained agent because it would reduce the violations in periods of peak solar production as well. Admittedly, a reinforcement learning algorithm is perhaps somewhat overkill considering the action space, reward function and state representation used in the simulation. However, when factors such as price of flexibility, batteries, transformer control and user discomfort are included, the task quickly gets more complicated. It is no quick way to determine the desired behaviour of the agent in such a setup, where there are many more strategies. Still, it is necessary to inspect the deep deterministic policy gradient algorithm in a simpler task, where its behaviour can be analysed and discussed, and build on those experiences when expanding the setup further.

Chapter 9

Conclusion and future work

The main goal in this thesis is to see if reinforcement learning and demand response can be used to safely operate a power grid with distributed solar energy production. The motivation is that costly upgrades of the electric transmission infrastructure can be avoided if the algorithm succeeds. The following research question was formulated in chapter 1:

> **RQ**: Is the deep deterministic policy gradient algorithm able to reduce the number of safety violations in a grid with high peak solar power production and high peak demand by the means of demand response?

The short answer to the research question is yes. The agent was able reduce the number of safety violations by 14 % by changing the power consumption in the net. However, investigating the results revealed that the reinforcement algorithm only avoids safety violations in hours of peak demand and actually produced more violations during hours of peak solar production. In total, there are more violations during peak demand than during peak solar production, so the net effect is a decrease in the number of violations. As discussed in chapter 8, the behaviour of the algorithm is found to be quite primitive, and there is room for improvement. This thesis has not performed an extensive tuning of the model, but instead used default values for hyperparameters and focused on the results that emerge from them. Therefore, it is natural to assume that improvements can be found, for instance by increasing the training time or pretraining the reinforcement algorithm.

The implementation of the power system environment is based on several simplifications, such as a constant power factor, equal demand signals and constant flexibility in the net. This is done so it is possible to analyse the behaviour of the algorithm. The implementation can be extended to be more in accordance with a real model for demand response and tested for different combinations of hyperparameters. The agent's action space can also be extended to control transformers, switches and charging of storage units. This is a way to handle the challenges that comes with increased amount of distributed solar production in the electrical power net. Reinforcement learning is a particularly attractive approach for solving this task because the environment can be simulated efficiently, providing the agent with many experiences from which it can learn. A successful reinforcement learning algorithm managing a demand response program can accelerate the incorporation of more renewable energy into the power mix, as costly and time-consuming upgrades of infrastructure can be avoided. This is of great utility, not only from a power system perspective, but also as a measure to achieve the goals of the Paris agreement.

Future work

There are several natural next steps to continue the work presented in this thesis.

Pretraining

The reinforcement agent in this thesis learns entirely from self-play and receives no input of how it should behave. However, the deep deterministic policy gradient method can be pretrained on a dataset with behaviour that is known to be good. Pretraining gives the agent a good starting point from which it can start experimenting with different behaviours and find proper strategies. This is especially attractive since the action space is very large, which means that a lot of training time is used to explore basic behaviour.

Hyperparameter search

There are many hyperparameters that can be tuned in reinforcement learning, as in most machine learning algorithms. This thesis has not performed an optimal hyperparameter search. A key parameter that should be tested more thoroughly is the training time, as neural network networks are good at continuously learning from new data, and the action space is large.

Realistic demand response model

The model of the demand response program used in this thesis is based on several simplifications. The power system is assumed to offer a demand flexibility that is constant and symmetric in all hours. Concretely, the power consumption can in every hour be changed continuously in the interval [-10%, 10%]. Realistically, changing the consumption affects the future flexibility in the system. In addition, the trained agent can change the consumption freely without any cost in this thesis. This is unrealistic, since people are not going to change their consumption patterns without an form for compensation. Future work should have a more realistic modelling of a demand response program. This would result in the need for more long-term planning because an action greatly affects the future possible actions. Reinforcement algorithms are well suited for tasks that require farsighted strategies.

More control variables

The action in the reinforcement algorithm is to change consumption in the grid. A natural extension of this is to let the agent control more elements, such as tap-changing transformers, and charging of batteries. Both transformers and batteries can be controlled using `pandapower`, the Python power flow simulation tool used in this thesis.

Finer time scale

The trained reinforcement agent performs an action every hour. Therefore, it is not able to tackle problems in real-time by activating flexibility. Instead, it is appropriate for planning the activation of flexibility in the next hours. A finer time-resolution, for instance every minute, could be implemented in a demand response program that can operate close to real-time.

Wind power

Wind power is a variable renewable energy source that is not consider in this thesis. Consequently, only a solar forecast is sent to the reinforcement agent. An extension of the setup can integrate wind power in the network as well, and test how the reinforcement agent tackles different sources of renewable energy production.

Other reinforcement algorithms

The Python code developed for activation of flexibility follows the structure of a general `gym` environment in Python, and can consequently be applied to different reinforcement algorithms without doing any modification of the code. The DDPG algorithm is known to be unstable and need a lot of hyperparameter tuning. The Python toolkit `stable_baseline` offers several possible reinforcement learning algorithms, such as Proximal Policy Optimization and Soft Actor Critic, which can improve stability without compromising performance.

Bibliography

[1] Magne Holstad and Thomas Aanensen. *Elektrisitet*. 2019. URL: https://www.ssb.no/energi-og-industri/statistikker/elektrisitet/aar (visited on 05/12/2019).

[2] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[3] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

[4] Alexandra Von Meier. *Electric power systems: a conceptual introduction*. Wiley, 2006.

[5] Stephen Frank and Steffen Rebennack. "An introduction to optimal power flow: Theory, formulation, and examples". In: *IIE Transactions* 48.12 (2016), pp. 1172–1197. DOI: 10.1080/0740817X.2016.1189626. eprint: https://doi.org/10.1080/0740817X.2016.1189626. URL: https://doi.org/10.1080/0740817X.2016.1189626.

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. ISBN: 978-0262039246. URL: http://incompleteideas.net/book/the-book-2nd.html.

[7] David Silver. *UCL course on reinforcement learning*. 2015. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html (visited on 04/18/2019).

[8] Ofir Nachum et al. "Bridging the Gap Between Value and Policy Based Reinforcement Learning". In: *CoRR* abs/1702.08892 (2017). arXiv: 1702.08892. URL: http://arxiv.org/abs/1702.08892.

[9] David Silver et al. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, June 2014, pp. 387–395. URL: http://proceedings.mlr.press/v32/silver14.html.

[10] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015). arXiv: 1509.02971. URL: http://arxiv.org/abs/1509.02971.

[11] M. H. Albadi and E. F. El-Saadany. "A summary of demand response in electricity markets". English. In: *Electric Power Systems Research* 78.11 (Nov. 2008), pp. 1989–1996. ISSN: 0378-7796. DOI: 10.1016/j.epsr.2008.04.002.

[12] Kjetil Mæland. *Betalte 500 kroner i strøm for én natt på hytta*. 2015. URL: https://www.nettavisen.no/na24/betalte-500-kroner-i-strom-for-en-natt-pa-hytta/3423150520.html (visited on 05/07/2019).

[13] José R Vázquez-Canteli and Zoltán Nagy. "Reinforcement learning for demand response: a review of algorithms and modeling techniques". In: *Applied energy* 235 (2019), pp. 1072–1089.

[14] Elta Koliou et al. "Demand response in liberalized electricity markets: Analysis of aggregated load participation in the German balancing mechanism". In: *Energy* 71 (2014), pp. 245–254.

[15] Convener Kai Strunz et. al. "Benchmark Systems for Network Integration of Renewable and Distributed Energy Resources". In: *Electra* 273 (Apr. 2014), pp. 85–89.

[16] K. Kavvadias. *Enlopy: Python toolkit for energy load time series*. URL: http://github.com/kavvkon/enlopy.

[17] SoDa. *SoDa - Solar radiation data*. 2019. URL: http://www.soda-pro.com/about-us (visited on 04/08/2019).

[18] Quentin Gemine, Damien Ernst, and Bertrand Cornélusse. "Active network management for electrical distribution systems: problem formulation and benchmark". In: *CoRR* abs/1405.2806 (2014). arXiv: 1405.2806. URL: http://arxiv.org/abs/1405.2806.

[19] Olje- og energidepartementet. *Forskrift om leveringskvalitet i kraftsystemet*. 2019. URL: https://lovdata.no/dokument/SF/forskrift/2004-11-30-1557 (visited on 05/07/2019).

[20] Dan Claudiu Ciresan et al. "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition". In: *CoRR* abs/1003.0358 (2010). arXiv: 1003.0358. URL: http://arxiv.org/abs/1003.0358.

[21] Dan C. Cireşan et al. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 1237–1242. ISBN: 978-1-57735-514-4. DOI: 10.5591/978-1-57735-516-8/IJCAI11-210. URL: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-210.

[22] John N. Tsitsiklis and Benjamin Van Roy. *An analysis of temporal-difference learning with function approximation*. Tech. rep. IEEE Transactions on Automatic Control, 1997.

[23] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[24] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[25] OpenAI. *How to Train Your OpenAI Five*. URL: https://openai.com/blog/how-to-train-your-openai-five/ (visited on 04/22/2019).

[26] Dota 2 Wiki. *Dota 2*. URL: https://dota2.gamepedia.com/Dota_2 (visited on 04/22/2019).

[27] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[28] Lei Yang et al. "Reinforcement learning for optimal control of low exergy buildings". In: *Applied Energy* 156 (2015), pp. 577–586.

[29] Ivana Dusparic et al. "Multi-agent residential demand response based on load forecasting". In: *2013 1st IEEE Conference on Technologies for Sustainability (SusTech)*. IEEE. 2013, pp. 90–96.

[30] Sina Zarrabian, Rabie Belkacemi, and Adeniyi A Babalola. "Reinforcement learning approach for congestion management and cascading failure prevention with experimental application". In: *Electric Power Systems Research* 141 (2016), pp. 179–190.

[31] L. Thurner et al. "pandapower — An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems". In: *IEEE Transactions on Power Systems* 33.6 (Nov. 2018), pp. 6510–6521. ISSN: 0885-8950. DOI: 10.1109/TPWRS.2018.2829021.

[32] *About pandapower*. URL: https://www.pandapower.org/about (visited on 04/17/2019).

[33] Plotly Technologies Inc. *Collaborative data science*. 2019. URL: https://plot.ly.

[34] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[35] Prafulla Dhariwal et al. *OpenAI Baselines*. https://github.com/openai/baselines. 2017.

[36] Ashley Hill et al. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018.

[37] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

Appendix A

Model details

A.1 Simulation 1

Table A.1 shows the parameters used in the `ActiveEnv` class in the simulation in this thesis.

Table A.1: Parameters in the `ActiveEnv` class used in the simulation in this thesis

Parameter name	Value
activation_weight	0
current_weight	0.01
demand_scale	10
demand_std	0
episode_length	200
flexibility	0.1
forecast_horizon	4
i_upper	90
imbalance_change	True
imbalance_weight	0.0001
reactive_power	True
reward_terms	['voltage', 'current', 'imbalance']
solar_scale	0.8
solar_std	0
state_space	['sun', 'demand', 'imbalance']
total_imbalance	True
v_lower	0.95
v_upper	1.05
voltage_weight	1

The used implementation of DDPG is `stable_baselines.ddpg.ddpg.DDPG`. The agent was trained for 100 000 steps, which took around 1 hour with a 2 core Intel(R) Core(TM) i3-7100U CPU @ 2.40 GHz. The memory limit (size of the experience replay buffer) was also set to 100 000. The rest of the arguments are the same as the default values.

`stable_baselines.ddpg.policies.LnMlpPolicy` is the policy that is used for training the model. It is a dense neural network with an architecture of 2 hidden layer, each with 64 neurons and layer normalisation. The exploratory noise added to the deterministic policy is `stable_baselines.ddpg.noise.OrnsteinUhlenbeckActionNoise` with standard hyperparameters: mean=0, sigma=0.2, theta=0.15.

Appendix B

Python code

The complete Python code written for this thesis can be found on GitHub: `https://github.com/vegraux/master_thesis`

B.1 ActiveEnv

```
# -*- coding: utf-8 -*-

"""
Gym environment implementing demand response in a pandapower net.
"""
import os
import dotenv
dotenv.load_dotenv()
import gym
import matplotlib.pyplot as plt
from gym import spaces
from gym.utils import seeding
import numpy as np
from gym_power.sample_net import cigre_network
from pandapower.networks import create_cigre_network_mv
import copy
import pandapower as pp
from pandapower import ppException
import pandas as pd

__author__ = 'Vegard Solberg'
__email__ = 'vegard.ulriksen.solberg@nmbu.no'
DATA_PATH = os.getenv('DATA_PATH')


class ActiveEnv(gym.Env):
    params = {'episode_length': 200,
              'reward_terms': ['voltage', 'current', 'imbalance',
                               'activation'],
              'voltage_weight': 1,
              'current_weight': 0.01,
              'imbalance_weight': 1e-8,
              'activation_weight': 1e-4,
              'forecast_horizon': 4,
              'flexibility': 0.1,
              'solar_scale': 0.8,
              'demand_scale': 10,
              'state_space': ['sun', 'demand', 'bus', 'imbalance'],
```

```
            'v_upper': 1.05,
            'v_lower': 0.95,
            'i_upper': 90,
            'demand_std': 0.03,
            'solar_std': 0.03,
            'total_imbalance': False,
            'reactive_power': True,
            'imbalance_change': False}

    def set_parameters(self, new_parameters):
        """
        sets params for the environment
        :param new_parameters: New parameter value
        :type new_parameters: dictionary
        """
        allowed_keys = ['episode_length', 'reward_terms', 'voltage_weight',
                        'current_weight', 'imbalance_weight',
                        'forecast_horizon', 'activation_weight',
                        'flexibility', 'state_space', 'solar_scale',
                        'demand_scale', 'v_lower', 'v_upper', 'i_upper',
                        'demand_std', 'solar_std', 'total_imbalance',
                        'reactive_power', 'imbalance_change']
        non_negative = ['voltage_weight', 'current_weight',
                        'imbalance_weight', 'activation_weight']
        zero_to_one = ['flexibility']
        for key in new_parameters:
            if key not in allowed_keys:
                raise KeyError('Invalid parameter name: ' + key)
            if key in non_negative and new_parameters[key] < 0:
                raise ValueError('Invalid parameter value, negative values '
                                 'not allowed: ' + key)
            if key in zero_to_one and (
                    new_parameters[key] < 0 or new_parameters[key] > 1):
                raise ValueError('Invalid parameter value, value must be'
                                 ' between 0 and 1: ' + key)

        self.params = {**self.params, **new_parameters}

        if ('state_space' in new_parameters) or \
                ('total_imbalance' in new_parameters):
            self.observation_space = spaces.Box(low=-np.inf, high=np.inf,
                                                shape=self._get_obs().shape,
                                                dtype=np.float32)

        _ = self.reset(reset_time=False)

    def __init__(self, do_action=True, force_commitments=False,
                 seed=None):
        self.np_random = None
        self._seed = self.seed(seed)
        # time attributes
        self._current_step = 0
        self._episode_start_hour = self.select_start_hour()
        self._episode_start_day = self.select_start_day()

        self.do_action = do_action

        # power grid
        self.base_powergrid = cigre_network()
        pp.runpp(self.base_powergrid)
        self.powergrid = copy.deepcopy(self.base_powergrid)
```

```
100         self.load_idx = np.arange(len(self.powergrid.load))
101         self.last_action = np.zeros_like(self.load_idx)
102         self.pq_ratio = self.calc_pq_ratio()
103 
104         # state variables, forecast + commitment
105         self.solar_data = self.load_solar_data()
106         self.demand_data = self.load_demand_data()
107         self.solar_forecasts = self.get_episode_solar_forecast()
108         self.demand_forecasts = self.get_episode_demand_forecast()
109         self.set_demand_and_solar()
110         self.force_commitments = force_commitments
111         self._commitments = np.zeros(len(self.powergrid.load)) != 0
112         self.resulting_demand = np.zeros(self.params['episode_length'])
113         self._imbalance = self.empty_imbalance()
114 
115         self.observation_space = spaces.Box(low=-np.inf, high=np.inf,
116                                             shape=self._get_obs().shape,
117                                             dtype=np.float32)
118         self.action_space = spaces.Box(-1., 1.,
119                                        shape=self.last_action.shape,
120                                        dtype=np.float32)
121 
122         self.load_dict = self.get_load_dict()
123 
124     def get_load_dict(self):
125         """
126         :return: dictionary mapping columns name to index
127         """
128         loads = self.powergrid.load
129         return {col: index for (index, col) in enumerate(loads)}
130 
131     def calc_pq_ratio(self):
132         """
133         Power factor for loads are assumed constant.
134         This method finds the PQ-ratio for all loads
135         (same as for default cigre network)
136         """
137         net = create_cigre_network_mv(with_der="pv_wind")
138         pq_ratio = net.load['q_kvar'] / net.load['p_kw']
139         return pq_ratio
140 
141     def get_demand_forecast(self):
142         """
143         Finds the forecasted hourly demand for the next T hours
144         """
145         forecasts = []
146         t = self._current_step
147         horizon = self.params['forecast_horizon']
148         for load in self.demand_forecasts:
149             day_forecast = load[t:t + horizon]
150             forecasts.append(day_forecast)
151 
152         return forecasts
153 
154     def get_solar_forecast(self):
155         """
156         Returns solar forecast for the next look_ahead hours.
157         """
158         t = self._current_step
159         horizon = self.params['forecast_horizon']
160         return self.solar_forecasts[t:t + horizon]
```

```
161
162     def get_scaled_solar_forecast(self):
163         """
164         scales each solar panel production with nominal values.
165         """
166         solar_pu = self.solar_forecasts
167         nominal_solar = self.powergrid.sgen['sn_kva']
168         scaled_solar = []
169         for sol in solar_pu:
170             scaled_solar.append((sol*nominal_solar).sum())
171         return np.array(scaled_solar)
172
173     def get_scaled_demand_forecast(self):
174         demand_pu = self.demand_forecasts[0]
175         scaled_demand = []
176         loads = self.powergrid.load
177
178         for demand in demand_pu:
179             scaled_demand.append((demand * loads['sn_kva']).sum())
180         return np.array(scaled_demand)
181
182     def select_start_hour(self):
183         """
184         Selects start hour for the episode
185         """
186         return self.np_random.choice(24)
187
188     def select_start_day(self):
189         """
190         Selects start day (date) for the data in the episode
191         """
192         try:
193             demand_data = self.demand_data
194         except AttributeError:
195             demand_data = self.load_demand_data()
196
197         demand_days = (demand_data.index[-1] - demand_data.index[0])
198         demand_days = demand_days.days
199         episode_days = (self.params['episode_length'] // 24) + 1  # margin
200         return self.np_random.choice(demand_days - episode_days)
201
202     def seed(self, seed=None):
203         self.np_random, seed = seeding.np_random(seed)
204         return [seed]
205
206     def reset(self, reset_time=True):
207         self._current_step = 0
208
209         self.powergrid = copy.deepcopy(self.base_powergrid)
210         if reset_time:
211             self._episode_start_hour = self.select_start_hour()
212             self._episode_start_day = self.select_start_day()
213
214         self.solar_forecasts = self.get_episode_solar_forecast()
215         self.demand_forecasts = self.get_episode_demand_forecast()
216         self.set_demand_and_solar()
217         self._imbalance = self.empty_imbalance()
218         self.resulting_demand = np.zeros(self.params['episode_length'])
219
220         return self._get_obs()
221
```

```
222     def empty_imbalance(self):
223         """
224         Creates imbalance array, all 0.
225         """
226         nr_loads = len(self.load_idx)
227         episode_length = self.params['episode_length']
228         return np.zeros((nr_loads, episode_length))
229 
230     def get_bus_state(self):
231         """
232         Return the voltage, active and reactive power at every bus
233         """
234 
235         return self.powergrid.res_bus.values.flatten()
236 
237     def get_episode_solar_forecast(self):
238         """
239         Method that returns the solar_forecast for the entire episode
240         """
241         episode_length = self.params['episode_length']
242         horizon = self.params['forecast_horizon']
243         start = self._episode_start_hour + self._episode_start_day * 24
244         nr_hours = episode_length + horizon + 1  # margin of 1
245         solar_forecast = self.solar_data[start:start + nr_hours].values
246         return solar_forecast.ravel() * self.params['solar_scale']
247 
248     def get_episode_demand_forecast(self):
249         """
250         gets the forecasts for all loads in the episode
251         """
252         episode_length = self.params['episode_length']
253         horizon = self.params['forecast_horizon']
254         demand_scale = self.params['demand_scale']
255 
256         start = self._episode_start_hour + self._episode_start_day * 24
257         nr_hours = episode_length + horizon + 1  # margin of 1
258         demand_forecast = self.demand_data[start:start + nr_hours].values
259         return [demand_forecast.ravel() * demand_scale]
260 
261     def get_commitment_state(self):
262         """
263         Transforms _commitments array from booleans to 0,1
264         """
265         commitments = np.zeros(self._commitments.shape)
266         commitments[self._commitments] = 1
267         return commitments
268 
269     def load_solar_data(self):
270         solar_path = os.path.join(DATA_PATH, 'hourly_solar_data.csv')
271         solar = pd.read_csv(solar_path)
272         solar.index = pd.to_datetime(solar.iloc[:, 0])
273         solar.index.name = 'time'
274         solar = solar.iloc[:, [1]]
275 
276         return solar
277 
278     def load_demand_data(self):
279         demand_path = os.path.join(DATA_PATH, 'hourly_demand_data.csv')
280         demand = pd.read_csv(demand_path)
281         demand.index = pd.to_datetime(demand.iloc[:, 0])
282         demand.index.name = 'time'
```

```
283         demand = demand.iloc[:, [1]]
284
285         return demand
286
287     def _check_commitment(self, action):
288         """
289         Checks if a load has a commitment in terms of production due
290         to its action from last step, and modifies the current action to the
291         opposite of the last action, so the consumption is not altered.
292         """
293         if self.force_commitments:
294             action[self._commitments] = - self.last_action[self._commitments
    ]
295
296             new_commitments = (action != 0)
297             new_commitments[self._commitments] = False
298             self._commitments = new_commitments
299             self.last_action = action
300             return action
301         else:
302             return action
303
304     def _get_obs(self):
305         """
306         returns the state for the power system
307         """
308         state = []
309         if 'demand' in self.params['state_space']:
310             demand_forecasts = self.get_demand_forecast()
311             for demand in demand_forecasts:
312                 state += list(demand)
313
314         if 'sun' in self.params['state_space']:
315             solar_forecasts = self.get_solar_forecast()
316             state += list(solar_forecasts)
317
318         if 'bus' in self.params['state_space']:
319             bus_state = self.get_bus_state()
320             state += list(bus_state)
321
322         if self.force_commitments:
323             commitment_state = self.get_commitment_state()
324             state += list(commitment_state)
325
326         if 'imbalance' in self.params['state_space']:
327             balance = self.calc_imbalance() / 30000
328             if self.params['total_imbalance']:
329                 state += [balance.sum()]
330             else:
331                 state += list(balance)
332
333         return np.array(state)
334
335     def calc_imbalance(self):
336         """
337         Calculates how much power the agent ows to the system,
338         i.e the amount of extra energy the loads have received
339         the last 24 hours. Reward function penalises a large imbalance.
340         """
341         t = self._current_step
342         if t > 24:
```

```
343             modifications = self._imbalance[:, t - 24:t]
344
345         elif t > 0:
346             modifications = self._imbalance[:, :t]
347
348         else:
349             modifications = np.zeros((len(self.load_idx), 1))
350
351         return modifications.sum(axis=1)
352
353     def log_resulting_demand(self):
354         """
355         Logs the resulting demand in an episode after the agent has taken
356         its actions
357         """
358         loads = self.powergrid.load['p_kw']
359         self.resulting_demand[self._current_step] = loads.sum()
360
361     def _take_action(self, action):
362         """
363         Takes the action vector, scales it and modifies the flexible loads
364         """
365         nominal_load = self.powergrid.load['sn_kva']
366         forecasted_load = nominal_load * self.get_demand_forecast()[0][0]
367
368         action *= self.params['flexibility'] * forecasted_load
369         if self.force_commitments:
370             action = self._check_commitment(action)
371
372         self._imbalance[:, self._current_step] = action
373
374         p_index = self.load_dict['p_kw']
375         q_index = self.load_dict['q_kvar']
376
377         self.powergrid.load.iloc[self.load_idx, p_index] += action
378         if self.params['reactive_power']:
379             self.powergrid.load.iloc[
380                 self.load_idx, q_index] += action * self.pq_ratio
381
382         try:
383             pp.runpp(self.powergrid)
384             self.log_resulting_demand()
385             return False
386
387         except ppException:
388             return True
389
390     def set_demand_and_solar(self):
391         """
392         Updates the demand and solar production according to the forecasts,
393         with some noise.
394         """
395         net = self.powergrid
396         sol_std = self.params['solar_std']
397         solar_pu = self.get_solar_forecast()[0]
398         solar_pu += solar_pu * sol_std * self.np_random.randn()
399
400         demand_std = self.params['demand_std']
401         demand_pu = self.get_demand_forecast()[0][0]
402         demand_pu += demand_pu * demand_std * self.np_random.randn()
403
```

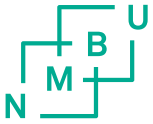
```
404         net.sgen['p_kw'] = - solar_pu * net.sgen['sn_kva']
405         net.load['p_kw'] = demand_pu * net.load['sn_kva']
406         net.load['q_kvar'] = net.load['p_kw'] * self.pq_ratio
407 
408     def calc_reward(self, old_imbalance, action, include_loss=False):
409         """
410         Calculates the reward in a time step
411         """
412 
413         state_loss = 0
414         if 'voltage' in self.params['reward_terms']:
415             weight = self.params['voltage_weight']
416             v = self.powergrid.res_bus['vm_pu']
417             v_min = self.params['v_lower']
418             v_max = self.params['v_upper']
419             v_lower = sum(v_min - v[v < v_min]) * weight
420             v_over = sum(v[v > v_max] - v_max) * weight
421             state_loss += (v_lower + v_over)
422 
423         if 'current' in self.params['reward_terms']:
424             weight = self.params['current_weight']
425             i = self.powergrid.res_line['loading_percent']
426             i_max = self.params['i_upper']
427             i_over = sum(i[i > i_max] - i_max) * weight
428             state_loss += i_over
429 
430         if 'imbalance' in self.params['reward_terms']:
431             balance = self.calc_imbalance()
432             weight = self.params['imbalance_weight']
433             if self.params['imbalance_change']:
434                 balance_change = np.abs(balance) - np.abs(old_imbalance)
435                 state_loss += balance_change.sum() * weight
436             else:
437                 state_loss += np.abs(balance).sum() * weight
438 
439         if 'activation' in self.params['reward_terms']:
440             flex = self.params['flexibility']
441             weight = self.params['activation_weight']
442             action *= flex * self.powergrid.load['p_kw']
443             act_loss = np.abs(action).sum() * weight
444             state_loss += act_loss
445 
446         if include_loss:
447             i_loss = sum(self.powergrid.res_line['pl_kw'])
448             state_loss += i_loss
449 
450         return -state_loss
451 
452     def plot_demand_and_solar(self, hours=100):
453         """
454         Visualise the total solar production and
455         demand for buses in the system
456         """
457         load = self.get_scaled_demand_forecast()
458         sol = self.get_scaled_solar_forecast()
459         resulting_demand = self.resulting_demand
460         fig, ax = plt.subplots()
461         plt.plot(sol[:hours], axes=ax)
462         plt.plot(load[:hours], axes=ax)
463         plt.plot(resulting_demand[:hours], axes=ax)
464         plt.legend(['solar', 'demand', 'modified'])
```

```
465         plt.show()
466
467     def step(self, action):
468         ep_length = self.params['episode_length']
469         self.set_demand_and_solar()
470         old_balance = self.calc_imbalance()
471         if self.do_action:
472             episode_over = self._take_action(action)
473         else:
474             pp.runpp(self.powergrid)
475             episode_over = False
476
477         self._current_step += 1
478         reward = self.calc_reward(old_balance, action)
479
480         if (self._current_step >= ep_length) or episode_over:
481             ob = self.reset()
482         else:
483             ob = self._get_obs()
484
485         return ob, reward, episode_over, {}
486
487     def render(self, mode='human', close=False):
488         pass
```

B.2 Training

```
1  # -*- coding: utf-8 -*-
2
3  """
4  Training the reinforcement agent using stable baselines
5  """
6  __author__ = 'Vegard Solberg'
7  __email__ = 'vegard.ulriksen.solberg@nmbu.no'
8
9  from gym_power.envs.active_network_env import ActiveEnv
10 from stable_baselines.common.vec_env.dummy_vec_env import DummyVecEnv
11 from stable_baselines.ddpg.policies import LnMlpPolicy
12 from stable_baselines.ddpg.noise import OrnsteinUhlenbeckActionNoise
13 from stable_baselines import DDPG
14 import numpy as np
15
16 powerenv = ActiveEnv()
17 powerenv.set_parameters({'state_space': ['sun','demand','imbalance'],
18                          'reward_terms':['voltage','current','imbalance']})
19
20 powerenv = DummyVecEnv([lambda: powerenv])
21 action_mean = np.zeros(powerenv.action_space.shape)
22 action_sigma = 0.3 * np.ones(powerenv.action_space.shape)
23 action_noise = OrnsteinUhlenbeckActionNoise(mean=action_mean,
24                                             sigma=action_sigma)
25
26 t_steps = 100000
27 logdir = 'C:\\Users\\vegar\\Dropbox\\Master\\logs'
28 powermodel = DDPG(LnMlpPolicy, powerenv,
29                   verbose=2,
30                   action_noise=action_noise,
31                   gamma=0.99,
32                   tensorboard_log=logdir,
33                   memory_limit=int(100000),
```

```
34                      nb_train_steps=50,
35                      nb_rollout_steps=100,
36                      critic_lr=0.001,
37                      actor_lr=0.0001,
38                      normalize_observations=False)
39 powermodel.learn(t_steps)
```



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway