Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Controlling a heating system using deep reinforcement learning

Tomáš Paleček

Supervisor: Mgr. Viliam Lisý, Ph.D.
Field of study: Open Informatics
Subfield: Open Informatics
May 2018

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Paleček Tomáš**	Personal ID number:	**457070**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Controlling a Heating System Using Deep Reinforcement Learning

Bachelor’s thesis title in Czech:

Řízení vytápění budovy pomocí posilovaného učení s neuronovými sítěmi

Guidelines:

The goal of this thesis is to evaluate how well can generic deep reinforcement learning (RL) methods perform in the task of controlling the heating system in a building.
The student will:
1) review the recent progress in reinforcement learning with function approximation;
2) design suitable representation of the features for the studied task;
3) integrate an existing model of heating in a building as an environment to the OpenAI reinforcement learning framework;
4) design and optimize a suitable neural network architecture to be used by the RL agent;
5) experimentally evaluate the effectiveness of deep reinforcement learning for the task.

Bibliography / sources:

[1] Sutton, R.S. and Barto, A.G., 2018. Reinforcement learning: An introduction. (Second Edition) Cambridge: MIT press.
[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), pp.529-533.
[3] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, ICLR 2016
[4] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. and Silver, D., 2017. Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv preprint arXiv:1710.02298.

Name and workplace of bachelor’s thesis supervisor:

Mgr. Viliam Lisý, MSc., Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **12.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Mgr. Viliam Lisý, MSc., Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I thank my supervisor for the topic he had chosen for me and for the consultations, during which he had helped me with problems I had encounter during my work on the project.
Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....
signature

Abstract

The goal of this thesis was to evaluate how well can deep reinforcement learning (deep RL) methods perform in the task of controlling the heating system in a building. We had focused on the deep Q-learning algorithm. The algorithm was tested in a simulation of a building equipped with boiler heating, written using MATLAB/Simulink. We had compared the efficiency of our agents against the rule based thermostat solutions. Our results suggest that the deep RL methods are able to control the heating system in the building more efficiently than the traditionally used rule based thermostat solutions.

Keywords: Deep reinforcement learning, Deep Q-Network, Optimal control, Building thermoregulation

Supervisor: Mgr. Viliam Lisý, Ph.D.

Abstrakt

Cílem této práce bylo zjistit účinnost řízení vytápění budovy pomocí posilovaného učení s neuronovými sítěmi. Naše algoritmy byly testovány na simulaci budovy vytápěné pomocí ústředního topení, která běžela v programu MATLAB/Simulink. Z algoritmů na posilované učení s neuronovými sítěmi jsme se zaměřily zejména na deep Q-network algoritmus. Účinnost deep Q-Network algoritmu jsme porovnali vůči účinosti termostatů s pevně definovanými teplotami pro zapnutí a vypnutí vytápění. Naše výsledky ukázaly, námi implemntovaný algoritmus je schopen lépe řídit vytápění budovy, než termostaty s pevně definovanými teplotami pro zapnutí a vypnutí vytápění.

Klíčová slova: posilovaného učení s neuronovými sítěmi, Deep Q-Network, Optimální řízení, termoregulace budovy

Překlad názvu: Řízení vytápění budovy pomocí posilovaného učení s neuronovými sítěmi

Contents

1 Introduction **1**

2 Theoretical background for our work **3**

2.1 Reinforcement learning 3

2.2 Function approximation 4

2.3 Deep learning 5

2.3.1 General overview 5

2.3.2 Neuron 5

2.3.3 Activation functions 6

2.3.4 Fully connected neural network 7

2.3.5 Gradient descent 7

2.3.6 Neural network computation . 8

2.4 Deep Q-Network 10

2.4.1 Algorithm description 10

2.4.2 Algorithm properties 11

2.5 Asynchronous advantage actor-critic 12

2.6 Comparison of algorithms considered for our problem 12

3 Problem definition **13**

3.1 Problem analysis 14

4 Model of the building **17**

4.1 Model Information 17

4.2 Model implementation details . . 19

4.3 Python Simulink interface 19

4.3.1 MATLAB/Python API 20

4.3.2 Message passing 20

4.3.3 Shared memory 21

5 Frameworks used **23**

5.1 Open AI gym 23

5.2 Deep reinforcement learning frameworks 23

5.2.1 Open AI Baselines 24

5.2.2 Open AI Baselines alternatives 24

5.3 Metacentrum 24

6 Our Environment 27

6.1 Feature selection 27

6.2 Overview of environments used for learning 28

6.2.1 Building version 0 information 29

6.2.2 Building version 1 information 30

6.3 Weather analysis 30

6.4 Thermostat overview 31

6.5 Reward function 32

7 Experimental results 33

7.1 Training and Testing 33

7.2 Hyper-parameter overview 33

7.3 Hyper-parameter tuning 35

7.4 Hyper-parameters used 36

7.5 Training results - building version 0 37

7.5.1 Comparison with the rule based solutions 38

7.6 Training results - building version 1 42

8 Conclusion 45

A Bibliography 47

B User manual 51

B.1 Requirements 51

B.2 DVD structure 52

B.3 Script usage 53

Figures

2.1 Interaction of the agent with the environment, taken from [SB18, p. 38] 4

2.2 Neuron unit with three inputs x_1, x_2 and x_3 and a bias unit b 6

2.3 The two most commonly used activation functions. The ReLu function is shown on the left. The Sigmoid activation function is shown on the right 7

2.5 An example of a simple 3 layered fully connected feed forward neural network 7

2.6 An example of gradient descent performed on a function of one variable, taken from [Sur18] 8

3.1 Visualized behavior of the rule based thermostat solution during the simulation, we can observe that the thermostat is able to control the heating in the building effectively, however not optimaly. The thermostat is unable to make an optimal transition when the value of the predefined temperature changes. Another deficiency is that the rule for turning the heating on and off can't be decided in the case of the rule based thermostat only from the difference between the predefined temperature and the current temperature. Lastly we can observe a domain specific problem between the time steps 200 and 300 when the thermostat is trying to increase the temperature, however there are small drops in the temperature, this is caused by the safety measure implemented inside of the boiler which prevents the boiler from overheating by automatically turning the boiler off when the temperature of the supply water reaches 70 °C. 14

4.1 Model of the building used in the Simulation 18

6.1 A histogram of the average temperatures for each day, on the left side of the picture the temperatures from the entire dataset are used, on the right side of the picture the temperatures from the training dataset are used. The average temperatures from the training dataset are equally distributed to stabilize the performance of our algorithm. 31

6.2 A visualization of the weather during the day, on the left side the weather for the entire data set is visualized, in the middle is the data set used for training and on the left is the data set used for validation. . . 31

7.1 The performance of the neural networks during the training, the term ftr refers to the reward obtained on the testing data, the term sttr refers to the reward obtained on the cross validation data, we can see that during learning the neural networks are stable 38

7.2 Figure shows how the agent 1 performs when the average outside temperature is between -16 and -12 °C, the agent 1 is able to better control the thermoregulation of the building better than the agent 2, when the outside temperature is low. The main diffrence is in the ability to adapt to the change in the predefined temperature occuring in time step 480. The act of agent 2 is shown in Figure 7.4 40

7.4 Figure shows how the agent 2 performs when the average outside temperature is between -16 and -12 °C, we can see that the reward -597 obtained by the agent 2 is even lesser than the reward of -583 obtained by the thermostat 40

7.6 Figure shows how the agent 1 performs when the average outside temperature is between 0 and 4 °C, we can see that the agent 1 is able to outperform the thermostat, however the agent 2 is starting to get better rewards than the agent 1 because the temperature is getting higher 41

7.8 Figure shows how the agent 2 performs when the average outside temperature is between 0 and 4 °C, we can see that the agent 2 is able to outperform the agent 1 by better regulating the temperature when the predefined temperature is set to 22 °C 41

7.10 Figure shows how the agent 1 performs when the average outside temperature is between 12 and 16 °C, we can see that the agent 1 is able to outperform the thermostat by better adapting to the change in the predefined temperature. 41

7.12 Figure shows how the agent 2 performs when the average outside temperature is between 12 and 16 °C, we can see that the reward obtained by the agent 2 is better than the reward obtained by the agent 1. This is caused by the difference in the behavior when the predefined temperature is set to 22 °C. The agent 2 is able to better keep the constant temperature than the agent 1. 42

Tables

6.1 Building version 0 overview of the features used for training 29

6.2 Building version 0 overview of the available actions 29

6.3 Building version 0 overview of the simulation settings 29

6.4 Building version 1 overview of the features used for training 30

7.1 An overview of the hyper-parameters used for the training of the neural network 36

7.2 The ten best performing hyper-parameter configurations for Building version 0, sorted by mean 37

7.3 Comparison of the rewards obtained by the two best performing agents with the rewards obtained using the continous thermostat. The term AVG refers to the average reward obtained during the testing. The W0 label represents the performance of the agent when tested on the data where the average outside temperaure was between -16 and -12 °C. Every subsequent column W1, W2, ..., W7 shifts the bound of the average temperature by 4, this means W1 represents the agents performance on the test data with the average outside temperature between -12 and -8. W7 represents the ouside temperature between 12 and 16. . . 39

7.4 The ten best performing hyper-parameter configurations for Building version 1, sorted by mean 42

Chapter 1

Introduction

The goal of this work was to efficiently control the heating system of a building using deep reinforcement learning. Deep reinforcement learning is a combination of reinforcement learning which is a computation approach which learns an optimal policy for given problem by interacting with the environment [SB18, p. 10–11] and deep learning which allows to train an artificial neural network (ANN) that can automatically learn task-relevant features [SB18, pg.361-362]. For obtaining the optimal policy for our agent we will be using the deep Q-Network algorithm [MKS$^+$15]. This algorithm allows the agent to make judgments about the value of each action, that can be executed in given state, and tries to optimize the long term reward of the agent. The values for actions are approximated using artificial neural network. The overview of deep reinforcement learning and the reasons behind using deep Q-Network are documented in Chapter 2.

Our agents were tested on a simulation of building equipped with a boiler heating. The simulation of the building had been provided for our work by Honeywell. The simulation accurately reflects a physical example of a heating system in the building. During the simulation the agent periodically receives data regarding the current state of the building. The data includes for example the current room temperature, the outside temperature and the temperature of water in the heating system. Based on the data the agent makes decision about turning the heating system on or off. The exact specifications of our problem are provided in Chapter 3. The agent chooses the actions in order to minimize the difference between the room temperature and the predefined temperature for the room. In this work we have compared the performance of a deep Q-Network algorithm against the rule based thermostat solutions. Our hypothesis was that the Q-Network system could better react to ambient temperature changes and thus provide better control of the room temperature. The reasons why are specified in the section 3.1 and then experimentally confirmed in Chapter 7. The simulation of the building was written in MATLAB and Simulink. The Chapter 4 provides a broad overview of the capabilities of the building simulation.

For training of the deep Q-Network algorithm the Open AI Baselines framework[DHK+17] was used. The framework is written in Python and is using TensorFlow library for computations. To enable the use of Open AI Baselines framework it was required to solve the problem of interprocess communication between Python and MATLAB. In our work the TCP/IP protocol for interprocess communication was used. The reasoning behind this decision is documented in Section 4.2. A Python environment which is capable of controlling the simulation process in Simulink had been created. The environment is compatible with the Open AI Gym [BCP+16] Toolkit. The exact specifications of the environment are shown in Chapter 6. This chapter also contains a discussion about the features which were used for training the algorithm.

The evaluation methodology used in the experimental part of our task, along with an overview of the impact of the hyper-parameter setting on the performance of the neural network are provided in Chapter 7. In our work we had experimentally evaluated the effectiveness of using deep Q-Network for heating regulation of the building. Our results have shown that the deep Q-Network is a suitable solution for this problem. We had also statistically confirmed that our solution is able to beat the rule based solutions. The preceding statements are shown in Chapter 7.

Chapter 2

Theoretical background for our work

In order to explain the role of using function approximation in reinforcement learning a brief introduction to the reinforcement learning is given.

2.1 Reinforcement learning

Reinforcement learning is a method used for solving problems which are defined as finite Markov decision processes(MDPs). Markov decision processes are used to formalize the problem of sequential decision making, where the future states of the process depend only on the current state. This property of MDPs is known as the Markov property. Because the future states of the Markov decision process are affected by the current state, a delayed reward needs to be taken into consideration in order to solve MDPs optimally. MDPs are defined as a 5-tuple (S,A,P,R,γ), where S is a finite set of states, A is a finite set of actions, P is the probability of taking action a in state s_t and ending in state s_{t+1}. R is the reward obtained after transitioning from state s to state s'. γ is the discount factor used for defining the trade-off between immediate and delayed rewards.

The purpose of the reinforcement learning is to calculate an unknown Markov decision process of an environment by interacting with it and making an optimal policy for the agent to follow based on the calculated Markov decision process. The goal of the agent's optimal policy is to maximize the

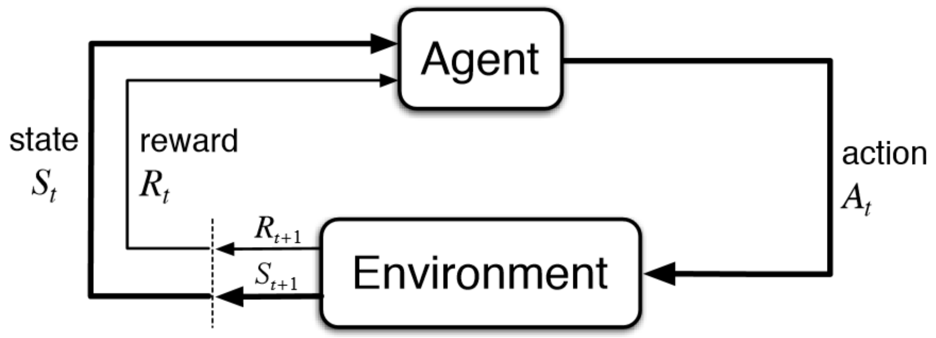


Figure 2.1: Interaction of the agent with the environment, taken from [SB18, p. 38]

long term reward of the agent. The term agent refers to the part of the algorithm interacting with the environment. Agent receives information from the environment about the state $S_t \in S$ it is currently in and based on the information received selects an appropriate action $A_t \in A(s)$ to perform. After performing the action, the agent receives a numerical reward, $R_{t+1} \in R$ and information regarding the state S_{t+1} it is in after performing the action. The agent chooses which action to perform based on his current policy π. Agent's policy defines what action will the agent perform in each state. The optimal policy tries to get the agent into states with the highest expected, accumulative, discounted reward. The state value function $v_\pi(s)$, represents the value of the accumulative sum of rewards the agent will obtain if it is in the state s and follows the policy π thereafter. We will also define the Q function $q_\pi(a, s)$, which represents the value of the accumulative sum of rewards of taking action a in state s and thereafter following policy π.

2.2 Function approximation

Function approximation is used in reinforcement learning when the environment state space is continuous, or too large to be computed effectively. In these cases we are unable to calculate the exact value for each state separately. Instead of trying to calculate the value for each state separately we will use an approximation function which predicts the value of the state based on the input provided. By using approximation function we are able to effectively reduce the state space of the environment. Another advantage of using the approximation function is that our algorithm will be able to evaluate the value of a state which it had never encountered before based on the experience obtained from other states it had visited before. For the approximation function various algorithms can be used. In our work we have focused on deep reinforcement learning which is using deep neural network for function approximation. Deep reinforcement learning is a rapidly evolving area and it is beyond the scope of this thesis to provide a full overview of all of the existing algorithms. In our work we will focus on the deep Q-Network algorithm [MKS$^+$15] and the Asynchronous advantage actor-critic [MBM$^+$16] algorithm. These two algorithms were one of the first algorithms used in deep reinforcement learning and both of them are still in the area of active research. For full overview of the deep reinforcement learning field we recommend the following paper [Li17].

2.3 Deep learning

2.3.1 General overview

The neural networks are algorithms that are capable of learning to perform complex tasks such as playing games at human level performance or driving cars. They are able to do so by automatically recognizing patters in the data provided to them. The neural network is made up from a set of neurons which are interconnected with each other. A single neuron is capable of doing a simple logical reasoning and by combining multiple neurons together the neural network is able to solve complex problems. The logical reasoning is achieved by assigning weight to each neural network connection. During each step of the training data is fed into the network and the network outputs it's solution for the task. The solution provided by the neural network is then compared to the expected result of the neural network. The difference between expected output and provided output is measured using loss function. Then the impact of each neural connection on the final reward is measured and it's weight is updated using gradient descend to minimize the loss. The following paragraphs contain the technical details of the deep neural network learning process described in this paragraph.

2.3.2 Neuron

Neuron is the basic building unit of a neural network, shown in Figure 2.2. Neuron takes in n inputs $x_1, x_2, ...x_n$ and multiply them by their respective weights $w_1, w_2, ...w_n$. The inputs multiplied by their respective weights are then summed together and a bias constant b is added to the sum. The sum is denoted as z, and is used as an input for the neuron activation function δ, which output is the output of the whole neuron, denoted y. The preceding statements are displayed in Equation 2.1.

$$y = \delta(b + \sum_{i=1}^{n} w_i x_i) \tag{2.1}$$

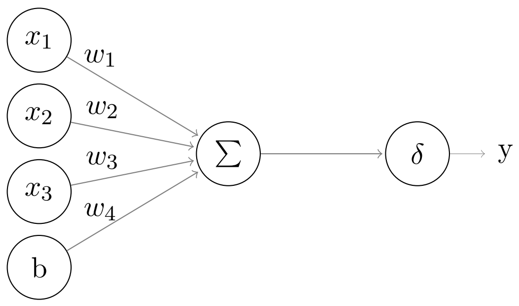


Figure 2.2: Neuron unit with three inputs x_1, x_2 and x_3 and a bias unit b

2.3.3 Activation functions

There exists a wide variety of different activation functions which can be used in neural networks. The two most commonly used activation functions are ReLu [GBB11], defined in Equation 2.2 and sigmoid, defined in Equation 2.3. The activation functions are displayed in Figure 2.3. The ReLu function is currently considered the default activation function to use in neural network. The main advantage of using ReLu over sigmoid is that ReLu doesn't suffer from the vanishing gradient problem. However the problem of dead neurons can occur, for further information on the differences between activation functions please read the paper where ReLu function was introduced in [GBB11]. By using activation function we are adding non-linearity to our neural network, this allows us to connect multiple neurons together to increase the complexity of the problems that we are able to solve using neural networks. If the non-linearity hadn't been added to the neural network, we wouldn't be able to approximate non-linear function, because combining multiple linear function together results only in an another linear function.

$$f(x) = max(0, x) \tag{2.2}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

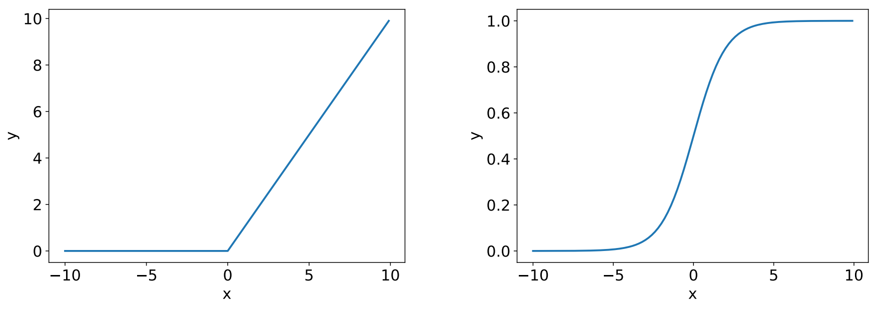


Figure 2.3: The two most commonly used activation functions. The ReLu function is shown on the left. The Sigmoid activation function is shown on the right

2.3.4 Fully connected neural network

An example of a fully connected neural network is depicted in Figure 2.5. The neural network is called fully connected because every neuron is connected with each neuron in the previous layer. The columns of neurons are referred to as the layers of the neural network. The term input layer is used for the first layer of the neural network. The last layer is referred to as the output layer. The layers in between the input layer and output layer are called the hidden layers.

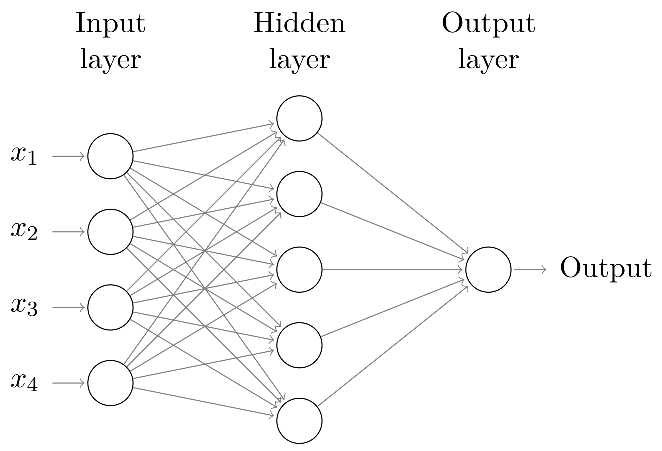


Figure 2.5: An example of a simple 3 layered fully connected feed forward neural network

2.3.5 Gradient descent

In neural networks the optimal values for the weights are usually found using the Gradient descent algorithm. Gradient descent is an optimization technique used to find the minimum of a function. For finding the minimum of a function the gradient, which represents the direction of the greatest increase of the value of the function, needs to be calculated. The gradient corresponds to the first-order derivation of the function. After finding the gradient we can minimize the value of function by taking a step into the opposite direction of the gradient. A step is taken by multiplying the gradient by a constant α, referred to as learning rate and subtracting the result of the multiplication from the function value. An example of using gradient descent on a function of one variable is shown in Figure 2.6. If the learning rate is too high divergence occur. If the learning rate is set too low the learning will be too slow.

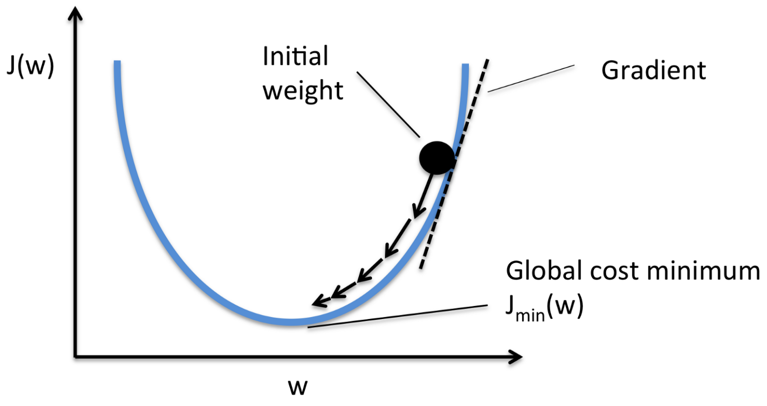


Figure 2.6: An example of gradient descent performed on a function of one variable, taken from [Sur18]

2.3.6 Neural network computation

The neural networks starts with a randomly initialized weights, we recommend using Xavier initialization [GB10], this initialization provides a initialization of the weights with zero mean and unit variance. The choice of neural network weight initialization influences the stability and performance of the algorithm. After the weights are initialized, the train data are fed into the neural network and the output of the neural network is computed using forward propagation. Forward propagation speeds up the computation by placing the parameters w and b of neurons which share the same layer, into a matrix. Then by using matrix multiplication the parameters are multiplied with their corresponding inputs. The activation function is then applied to every element of the final matrix. This allows us to calculate the output of all neurons in one layer using two operations. Based on the output of the final layer a loss of the neural network is calculated using Equation 2.4. The choice of loss function depends on the problem we are trying to solve using the neural network. This concrete function is used in the task of classification, which is used as an introductory topic to neural networks, where y represents the label of the image generated by neural network and $\hat{y}$ represents the label of the image in the test data.

$$E(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 \tag{2.4}$$

Based on the loss of the neural network the weights are updated in order to minimize the loss. This is done by using the gradient descent algorithm introduced in Section 2.3.5. The computation of gradients used for gradient descent are computed using backpropagation. Backpropagation is using the chain rule to calculate the gradients. Chain rule is used in mathematics to compute the derivative of the composition of two or more functions. Neural network is a composition of functions where each neuron is a function. The backpropagation algorithm computes the gradients starting from the output layer and ending in the input layer. By starting the computation of gradients from the back of the neural network we are able to use the values of gradients calculated in previous layer to speed up the computation. After computation of gradients we can apply gradient descend. The last step is to update the weights of the neural network. After updating the weights if the result of the neural network isn't sufficient, another cycle of neural network training is executed, this continues until convergence or the predefined number of training steps is exceeded. The end of neural network training varies from implementation to implementation. The pseudocode for algorithm used to train neural network is shown in Algorithm 1. For more details on deep learning we recommend reading the book [GBC16].

```
input  : Training data
output : Updated neural network
initialize the weights in neural network
for episode = 1 to episode = training length do
    Perform forward propagation to compute the output of the neural
      network
    Compute the error from the output layer
    Compute the gradient using backpropagation
    Update network weights by substracting the gradient multiplied by
      the learning rate
end
```

Algorithm 1: A pseudocode for the algorithm which is used for the training of the neural network

2.4 Deep Q-Network

2.4.1 Algorithm description

The first approach is using the deep Q-Network (DQN)[MKS+15]. The deep Q-Network learns to approximate the value of the Q-function. Q-function represent the accumulative, discounted reward of performing an action a in state s. In each step the algorithm performs the action with the highest value of Q-function. During training an ϵ-greedy exploration strategy is used. The ϵ-greedy exploration strategy performs a random action in each state with probability ϵ, if the random action is not performed the action with the maximal Q-value is performed. An alternative for ϵ-greedy exploration policy is for example the exploration strategy described in paper [PHD+17], which explores the state space by adding noise to the neural network of the agent. After action is performed the transition is stored inside the experience buffer. The transitions stored in the experience buffer are then used for training of the neural network. During each training a minibatch of transitions is randomly selected from the experience buffer. An alternative to selecting the transitions randomly is to use prioritized replay which selects the transitions based on the Temporal Difference error. For more information on prioritized replay please refer to the paper [SQAS15]. After choosing the transitions for the algorithm to train on we estimate the Q-value of the state s_j and action a_j recorded in the transition using the neural network. We will then approximate the value of the state s_{j+1} by taking the maximum Q-values from all the actions that can be performed in the state s_{j+1}, the reward r_j from the transition is then added to the value of the state s_{j+1}, we will denote this variable y_j. After obtaining the value of action a_j in state s_j and the value of variable y_j we can subtract these two values to obtain error in estimation of our network on Q-value of action a_j in state s_j. We then update the parameters of the neural network by performing the gradient descent. The preceding algorithm description is written using pseudocode in Algorithm 2.

```
input  : the pixels and the game score
output : Q action value function (from which we obtain policy and select
         action
Initialize replay memory D
Initialize action-value function Q with random weight $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode = 1 to M do
    Initialize sequence $s_1$={ $x_1$} and preprocessed sequence $\phi_1 = \phi(s_1)$
    for t = 1 do T do
        Following $\epsilon$-greedy policy, select
        $a_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ argmax_a Q(\phi(s_t), a, \theta) & \text{otherwise} \end{cases}$
        Execute action $a_i$ in emulator and observe reward $r_t$ and image
          $x_t + 1$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        \\experience replay
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        set $y_j =$
          $\begin{cases} r_j, & \text{if episode terminates at step j + 1} \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a', \theta^-), & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
        \\periodic update of target network
        Every C steps reset $\hat{Q}$ = Q, i.e, set $\theta^- = \theta$
    end
end
```

Algorithm 2: A pseudocode of the deep Q-Network algorithm, taken from [MKS+15], where the deep Q-Network had been introduced, in our work the data about the current state of the building will be used as input, the neural network used in our work will be feed forward fully connected neural network. In our work preprocessing of the states is unnecessary, because we are not receiving data from raw input

2.4.2 Algorithm properties

Deep Q-Network is an off-policy method which means that the network is able to train even on data which weren't produced by it's current policy. This makes it possible to train the neural network on data generated by another algorithm, or even on data generated by examining the human solution for the given problem. Recent progress in the area of deep reinforcement learning has shown that by combining extensions of DQN the algorithm is able to outperform the A3C algorithm[HMvH+17] mentioned below.

2.5 Asynchronous advantage actor-critic

The second algorithm considered was asynchronous advantage actor-critic (A3C)[MBM+16].The algorithm learns both a policy and a state-value function, where the value function is used to speed up the learning. In the paper where the A3C algorithm was introduced the authors proved that for Atari games A3C algorithm is able to learn faster and perform better than or comparably to vanilla deep Q-Network[MKS+15]. A3C algorithm is an on-policy method which means that it is able to learn only from the data produced by it's own policy. Another property of A3C is that the stability of the algorithm is influenced by the number of actors employed in parallel. A detailed description of A3C algorithm is not provided because the algorithm wasn't used in the project. For a full description of the algorithm please visit the paper that introduced it.

2.6 Comparison of algorithms considered for our problem

We found that the deep Q-Network algorithm is better suited for our problem than the A3C algorithm. The main deciding factor when considering which algorithm to use was the fact that the deep Q-Network is an off-policy method. This means that in further work we could combine the training data obtained in the simulation with the data obtained in real life situations to make the results of our algorithm more realistic. When considering the real life application of our algorithm, the stability of A3C training is dependent on the number of the agents run in parallel. Because of this property the algorithm would need to be applied to multiple buildings at once to achieve optimal results. In our case it also spares us from the need to implement parallelization for our simulation of the building.

Chapter 3

Problem definition

The goal of this thesis was to evaluate how well can deep Q-learning perform in the task of controlling the heating system in a building. The performance of our algorithm was compared against the basic rule based thermostat solutions. The agent was trained on a simulation of a building equipped with a boiler heating, over which the agent had full control. The goal of the agent was to keep the temperature inside of the room as close to the user defined temperature as possible. The inside temperature in our simulation is influenced by the weather outside of the building. The temperature data set for weather outside of the building is based on the weather measured in Basel, Switzerland. The building isn't equipped with any cooling system. Because of this, we had limited the examples of the outside temperatures only to days when the use of the heating system is required.

In our simulation the predefined desired room temperature was set to 16 degrees C during hours when the building is unoccupied and to 22 degrees C when it is occupied. The testing period of the simulation was set to 1 day. Our agent periodically receives data regarding the current state of the system during the simulation. The default frequency set for data acquisition had been set to 2,5 minutes, which should provide the agent with enough information in order to control the heating system of the building efficiently. After every action the agent receives a negative reward calculated from the quadratic difference between the predefined and the current temperature. An example of the agent controlling the heating system of the building is displayed in Figure 3.1.

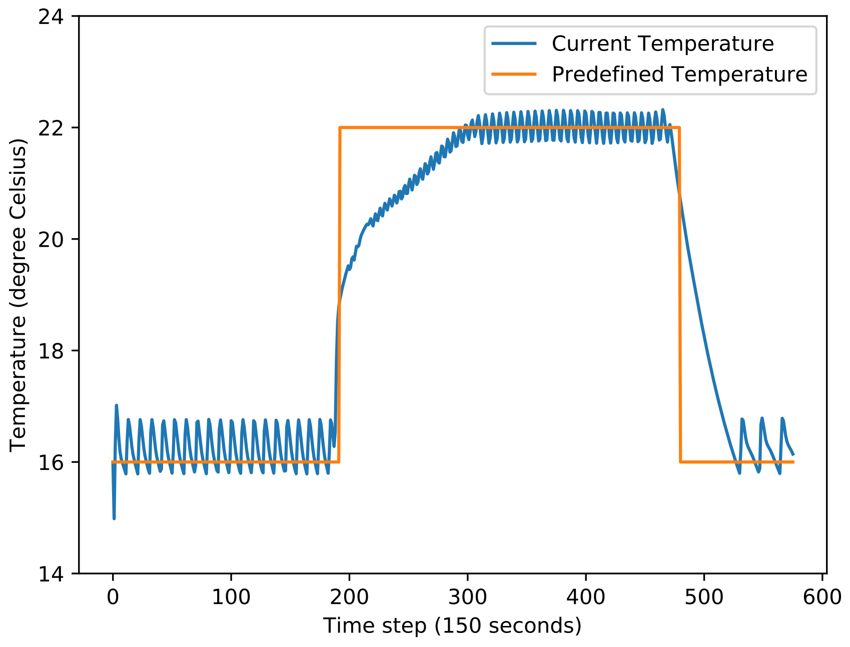


Figure 3.1: Visualized behavior of the rule based thermostat solution during the simulation, we can observe that the thermostat is able to control the heating in the building effectively, however not optimaly. The thermostat is unable to make an optimal transition when the value of the predefined temperature changes. Another deficiency is that the rule for turning the heating on and off can't be decided in the case of the rule based thermostat only from the difference between the predefined temperature and the current temperature. Lastly we can observe a domain specific problem between the time steps 200 and 300 when the thermostat is trying to increase the temperature, however there are small drops in the temperature, this is caused by the safety measure implemented inside of the boiler which prevents the boiler from overheating by automatically turning the boiler off when the temperature of the supply water reaches 70 °C.

3.1 Problem analysis

We have started by examining the capabilities and limitations of the rule based thermostat solutions. As we can see the thermostat is able to control the temperature in the building quite effectively shown in Figure 3.1. However the policy could still be improved. We can see that the thermostat is unable to make optimal transition when the value of the predefined temperature changes. Another deficiency is that the rule for turning the heating on and off can't be decided in the case of the rule based thermostat only based on the difference between the predefined and current temperatures. This basic system does not take into account the fact that the heat required for keeping the optimal room temperature is greatly influenced by the outside temperature and by the temperature of the water in the heating system. We believe that by using deep reinforcement learning we will be able to train the agent which will be able to take into consideration all these dependent variables mentioned above, when deciding the optimal action for the agent to make. By using deep neural networks the agent should be able to act based on non-linear features of the dependent variables.

Chapter 4

Model of the building

4.1 Model Information

For purpose of testing deep reinforcement learning algorithms a physically accurate simulation of a building, equipped with boiler heating had been provided by Honeywell. The following paragraphs provides an overview of the capabilities of the Simulation and the underlying construction details of the simulated building.

The building consists of two rooms. For each room information about the room temperature, furniture temperature, perimeter temperatures, heat losses and sum of heat flows can be obtained. Each room has 4 walls. In our model we have 3 different types of walls: External wall, Neighbor wall and Internal wall. Monitoring information from walls will allow our algorithm to use external weather conditions to its benefit. It also allows model to get better information about the heat flows thorough the building.

Rooms are equipped with heating radiators. Room A is heated by one radiator and Room B is heated by two radiators. We will monitor water heat content utilization and zone heat flow is monitored. The water is distributed into heating system from the Pump. From the pump we get information about its flow(kg/h), pressure(kPa) and the pump speed. Water from the pump is first sent to gas heating boiler. Gas boiler warms up the water to the required temperature and then the water continues in to the system. During the simulation the gas boiler can be turned on/off during simulation. On

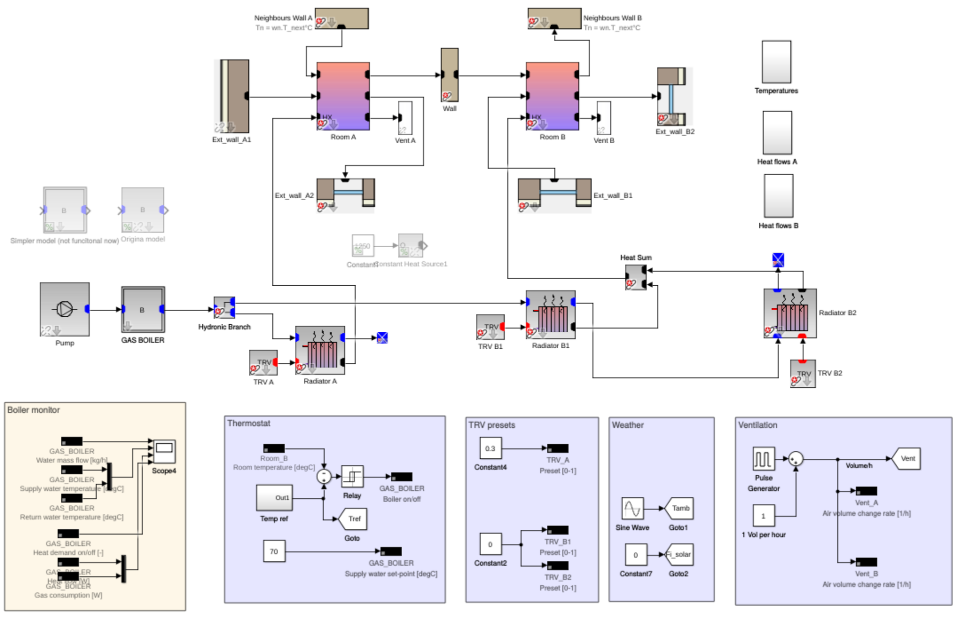


Figure 4.1: Model of the building used in the Simulation

gas boiler we can monitor the gas consumption(W), heat flow(kW), supply water temperature(°C) and return water temperature(°C). The gas boiler has active limiter on the supply water temperature. When the supply water temperature exceeds 70 degrees, the gas boiler is turned off.

A basic implementation of a functional thermostat was included with the model. It tries to keep the temperatures in rooms close to the predefined ones by activating and deactivating when the difference between the temperatures exceeds 0.2 degrees. We will use this thermostat as a baseline for our algorithm to beat.

The model also allows us to specify different types of buildings. However in our work we will use only one type building with fixed construction details. We have the option to choose the height of ceiling, size of the building, construction(wall thickness and heat capacity), insulation, and a type of radiators.

Another factor we can decide is how much the gas boiler corresponds to the reality. We can use the physically accurate gas boiler model or the physically inaccurate model. The main disadvantage of using the realistic model is that it has 10 times longer execution time. This makes the realistic model difficult to use for us, because the deep reinforcement learning requires a large number of samples in order to train the agent. Because of the limitations of deep reinforcement learning we will be using only the physically inaccurate model of the gas boiler.

4.2 Model implementation details

The model is written using Simulink and MATLAB. Simulink is a simulation and model-based design environment for dynamic and embedded systems, integrated with MATLAB. For the usage of the model first a MATLAB function (in our project init_fce.m) needs to be executed. This function initializes all the variables needed by the model. In the function we can also specify which type of building we want to use in our simulation. The body of the function contains command sim. This command runs the simulation in Simulink. We use the FastRestart option in our simulation. This leads to a significant speedup of the program, because the Simulink model is compiled only once instead of compiling it every time command sim is used.

My code for data manipulation is located within Enabled and Triggered Subsystem block. This block is activated each 150 time steps and thus saves computation power. 1 time step in simulation corresponds to 1 second in real life. For getting the data from an another subsystems Goto and From blocks were used. These blocks are used for the communication between the subsystems in Simulink. During the implementation of the Simulink code we have followed the programming practices used in the model of the building. The reinforcement learning agent which optimizes the central heating system of the building access the data from the sensors 2,5 minutes, and based on the data it decides what action to take. The RL agent is implemented using Python whilst the Simulation is written in Simulink/MATLAB, thus we need to develop a method for the inter-process communication between the programming languages.

4.3 Python Simulink interface

Because the model of the building is written in Simulink/MATLAB and our algorithm controlling the heating system is written in Python we needed to solve the problem of communication between these two programming languages. By using Python as the programming language for the implementation of the algorithm which controls the heating system a variety of frameworks for deep reinforcement learning became available to us, the DRL frameworks are discussed in Chapter 5. In the following paragraph the different methods for the problem of inter-process communication between programming languages are discussed.

4.3.1 MATLAB/Python API

The first choice for Python/Simulink communication was the official MATLAB API for Python. The MATLAB API for Python provides a package for Python to use MATLAB as a computational engine. After implementation was done we had found a critical error which made this solution unusable in our project. The MATLAB API for Python was unable to exchange information regarding the Simulink simulation while the simulation was running, the information was exchanged only after the simulation had terminated. In our project the agent needs to obtain the data during the simulation in order to be able to control the heating system of the building, therefore this solution is unusable for our project.

4.3.2 Message passing

This method allows two processes to communicate with each other, by establishing a communication link between them. The Transmission Control Protocol(TCP) will be used for communication between Python and MATLAB. The TCP protocol offers a reliable, ordered, and error-checked delivery of the data. All properties of TCP are needed in our project. In the implementation of message passing used in the project the server is written in Python whilst the client is written in MATLAB. After message passing was implemented the time to complete the simulation of one day increased from 1.05 second to 2.45 seconds. The increase in computational time was expected because the connection needs to verify that all the data sent using TCP have been received correctly. The speed of the message passing method for data sharing turned out to be sufficient solution for our problem and will be used in our project.

4.3.3 Shared memory

Shared memory allows two processes to communicate by writing data into the random access memory (RAM). This method for inter-process communication, should achieve the fastest transfer speed out of the methods mentioned above. If in the future we would want to increase the speed of the execution of the model, implementing shared memory is a viable method to do so. The reason behind not using shared memory is that the shared memory implementation is system dependent. This could turn out to be a problem when executing the computations on a computational server, such as MetaCentrum.

Chapter 5

Frameworks used

5.1 Open AI gym

Open AI Gym is a toolkit for developing and comparing reinforcement learning algorithms [BCP$^+$16].The Open AI Gym Toolkit offers a wide variety of environments with a common interface with which the agent can interact. Most Python frameworks with implementations of deep reinforcement learning algorithms are using Open AI Gym interface for the communication between the agent and the environment. The framework also allows to add user created environments if they are compatible with it's interface. In our work we had implemented an Open AI Gym environment which controls the state of the MATLAB/Simulink simulation of the building.

5.2 Deep reinforcement learning frameworks

The implementation of deep reinforcement learning algorithms can be very challenging because even the slightest mistake in the implementation of the algorithm can dramatically affect the performance it's performance. A thorough testing needs to be done in order to prove that the implementation of the algorithm is correct. In order to speed up the development time of our solution we had chosen to use a framework with implementations of deep reinforcement learning algorithms and modify it to suit our problem.

5.2.1 Open AI Baselines

In our project we have decided to use the Open AI Baselines framework [DHK$^+$17]. This framework uses TensorFlow library for the neural network related computations. The interaction of the agent with the environment is written in Python. The available deep reinforcement learning algorithms at the time of writting were DQN, A2C, ACER, ACKTR, DDPG, GAIL, PPO1, PPO2 and TRPO. As mentioned in Chapter 2, we will be using DQN algorithm in our work. The Open AI Baselines includes along with the vanilla version of the algorithm also several extensions. The extensions are Double DQN [vHGS15], DQN with prioritized experience reward [SQAS15] and DQN with exploration using parameter noise [PHD$^+$17]. In our work we had used the Double DQN extension.

5.2.2 Open AI Baselines alternatives

Open AI Baselines contains the most implementations and extensions out of the algorithms used for deep reinforcement learning. The authors actively add new deep reinforcement algorithms when they are introduced. When considering a framework to use for deep reinforcement learning we recommend using the Open AI Baselines framework. One of the shortcomings of the framework is that the computational logic is spread out between Python and TensorFlow this choice impairs the readability of the code. The framework TensorForce [SKF17] addresses this limitation by moving all of the computational logic inside the TensorFlow. Another alternative to Open AI Baselines is the deep reinforcement learning extension of Chainer framework [TOHC15]. This framework uses Chainer for the neural network computations. Chainer framework is written purely in python which could be preferred by some programmers to the option of writing the code in TensorFlow.

5.3 Metacentrum

In order to search the hyper-parameter space of the neural networks adequately and to statistically confirm our results our computational resources turned out to be insufficient. In our work we have used for computations the computational resources provided by the National Grid Infrastructure MetaCentrum. We had developed bash scripts which allow the user to schedule multiple jobs into the MetaCentrum Grid Infrastructure for computation. The scripts for computation using the National Grid Infrastructure MetaCentrum are included on the DVD.

Chapter 6

Our Environment

6.1 Feature selection

In this section the choice of features used by our agent is discussed. The ranges of the features are documented in Section 6.2

- Current temperature - based on the value of the current temperature and predefined temperature the agent should be able to tell whether to turn the heating on or off.
- Predefined temperature - the agent should strive to keep the current temperature as close as possible to the predefined temperature.
- Supply water - gives the agent information about the current temperature of water being sent into the radiators. The information about supply water allows our agent to predict how the temperature will change in the next state. For example if the current temperature is 21.9 and the predefined temperature is 22 and the temperature of supply water is close to 70, the agent can turn off the heating because in next time step the current temperature will raise because of the supply water and keeping the boiler on for another time step would result in overheating which can be seen in rule based thermostats. Also as mentioned in Chapter 4 the boiler is equipped with a safety control, which turns the heating automatically off if the supply water temperature reaches 70 degrees.
- Return water - provides information about how much heat is sinked during heating. Also the water needs to be heated again after returning

from the system. Based on the temperature of the return water our agent can calculate the amount of heat it can provide to the system.

- Gas consumption - in future work the gas consumption will be incorporated into the system. Neural network should be able to evaluate during the early stages of the training that this feature does not affect the system and set it's weight to 0.
- Future predefined temperature - this feature allows the agent to react to the expected temperature in future and prepare in advance to make a smooth transition between the two different predefined temperatures. Future predefined temperature can be given to the algorithm in two different forms. The first form is giving the current time of the simulation. The second choice is giving the algorithm value of the predefined temperature x time steps in advance. The main advantage of the first form is that it contains more information about the system, another advantage is that the information extends the input space only by one. The main disadvantage is that it is harder for the algorithm to process the feature properly. We will prove this in Section 7.6. The second choice is easier for the algorithm to interpret because it acquires only two values during the training. Where the neural network can increase the Q-value of turning the heating on when the current temperature is lower than the predefined temperature x steps ahead, and increase the Q-value of turning off when the current temperature is higher than predefined temperature x steps ahead. The disadvantage is that the agent needs to know the values for predefined temperature for 5,10,15 .... steps ahead to be able to choose the optimal time for changing the temperate, during every weather profile. In the experimental part both of the methods were tested.
- Outside temperature - the outside temperature affects the inside temperature. When the outside temperature is lower than the inside temperature, the temperature in the room decreases, also the heating system needs to produce more heat to keep the room temperature at the user predefined value. On the other hand when the outside temperature is high the thermostat could overestimate the heating required and increase the temperature outside of the user comfort zone. The outside temperature should help the agent to learn a strategy which fits the outside weather temperature the most.

6.2 Overview of environments used for learning

Our algorithms were tested in two different environments. The environments differ in the features available to the agent. The respective features and their ranges are documented in the tables below. The Table 6.1 provides an overview of the features available to the agent when the building version 0 environment is used for the training, Table 6.2 shows a summary of the actions available to the agent during training, Table 6.3 displays the settings of the simulation. Table 6.4 shows the list of features that are available to the agent when the building version 1 environment is used for the training of the agent.

6.2.1 Building version 0 information

Number	Observation	Min value	Max value
0	Current temperature	-20	40
1	Current predefined temperature	16	22
2	Predefined temperature 12.5 minutes in advance	16	22
3	Predefined temperature 25 minutes in advance	16	22
4	Predefined temperature 37.5 minutes in advance	16	22
5	Predefined temperature 50 minutes in advance	16	22
6	Predefined temperature 62.5 minutes in advance	16	22
7	Predefined temperature 75 minutes in advance	16	22
8	Predefined temperature 87.5 minutes in advance	16	22
9	Predefined temperature 100 minutes in advance	16	22
10	Predefined temperature 112.5 minutes in advance	16	22
11	Predefined temperature 125 minutes in advance	16	22
12	Water supply temperature	0	70
13	Water return temperature	0	70
14	Gas consumption	0	30
15	Outside temperature	-20	40

Table 6.1: Building version 0 overview of the features used for training

Number	Action
0	Turn boiler off
1	Turn boiler on

Table 6.2: Building version 0 overview of the available actions

Number of Days per episode	Number of time steps in episode	Time step in minutes
1	576	2,5

Table 6.3: Building version 0 overview of the simulation settings

6.2.2 Building version 1 information

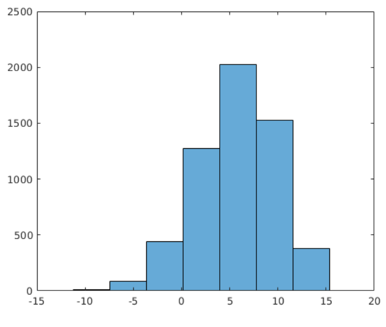
Number	Observation	Min value	Max value
0	Current temperature	-20	40
1	Current predefined temperature	16	22
2	Current time step	0	576
3	Water supply temperature	0	70
4	Water return temperature	0	70
5	Gas consumption	0	30
6	Outside temperature	-20	40

Table 6.4: Building version 1 overview of the features used for training

6.3 Weather analysis

The outside temperature is based on the temperatures measured in Basel, Switzerland during years 1985 to 2017, the data were acquired from the website [Met18]. The data contains hourly measurements of the temperature in the provided area. The building simulation requires the data input to be continuous. In order to make the measurements continuous we linearly interpolated between them. The data from the year 2018 weren't used to prevent an unequal distribution of the weather, caused by adding only the winter weather profiles from the year 2018. All of the weather profiles during which the average day temperature exceeded 16 °C were excluded from the training data set. This decision was made because we are testing the performance of our agent only on days when the heating is required.

In order to get the best performance of our agent it is important to train the agent in every possible weather profile it can encounter during a real life situation. This will allow us to make general conclusions about the performance of the agent. Another important aspect influencing the performance of our agent is the distribution of the weather profiles during the training. If we would train our agent on weather profiles taken from each season equally we would overfit the policy for certain weather profiles. This could lead to better averaged rewards. However the agent's strategy on the weather data which are under represented could lead to an unpredictable behavior. To prevent this we took the data samples of ambient temperature for each day and analyzed the frequency of their average temperature and grouped the data in 8 bins of equal range, which can be seen in Figure 6.1a. In the next step we have created experimentation data set for training of our algorithm where the frequency of the temperature taken from each bin is


(a) : Entire dataset

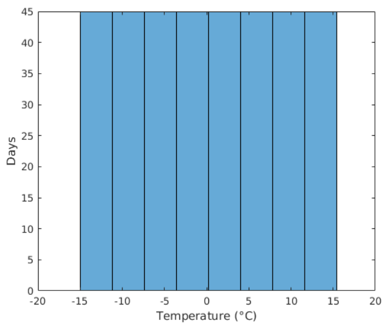
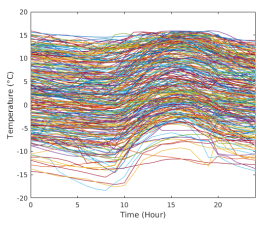
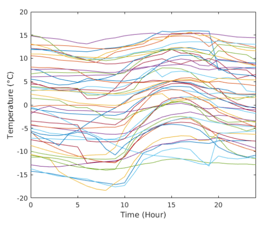

(b) : Train dataset

Figure 6.1: A histogram of the average temperatures for each day, on the left side of the picture the temperatures from the entire dataset are used, on the right side of the picture the temperatures from the training dataset are used. The average temperatures from the training dataset are equally distributed to stabilize the performance of our algorithm.


(a) : Entire data set


(b) : Test data set

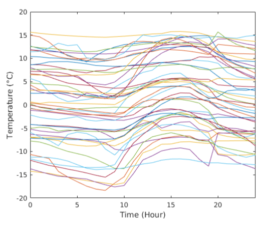

(c) : Validation data set

Figure 6.2: A visualization of the weather during the day, on the left side the weather for the entire data set is visualized, in the middle is the data set used for training and on the left is the data set used for validation.

equally represented, this can be seen in Figure 6.1b . This data set helped us to stabilize the performance of our algorithm on weather conditions which are under represented in the whole data set. The weather profiles during day for the entire dataset can be seen in Figure 6.2a, the weather profiles used for validation are plotted in Figure 6.2b and the weather profiles used for testing are depicted in Figure 6.2c. By using our algorithm we had managed to generate the training and the validation data sets which represent accurately the whole weather dataset.

6.4 Thermostat overview

In order to measure the performance of our agents we have implemented a rule based thermostat algorithm in Simulink. The algorithm receives information continuously during the simulation and reacts based on the difference between the current temperature and the predefined temperature 5 minutes ahead of the current time. If the difference between the temperatures exceeds 0.2 °C the algorithm turns the heating On/Off. This solution was able to achieve the reward of -678 in the simulation. We had also implemented a rule based thermostat in Python which receives information about the environment with the same frequency as our agent. The agent was able to receive the reward of -710. For testing the implementation of the thermostat written in Simulink will be used.

6.5 Reward function

For the computation of the reward obtained by the agent after each step we are using the following Equation 6.1.

$$reward = -(current\_temperature - predefined\_temperature)^2 \tag{6.1}$$

By making the temperature difference quadratic, we are penalizing our algorithm more if there are large differences in the temperatures. This should make the algorithm prefer making more small mistakes to making one large mistake. This behavior is preferred in real life, because large fluctations in the temperatures are more noticeable by the humans.

Chapter 7

Experimental results

7.1 Training and Testing

The duration of the training was set to 360 episodes. Our agents were tested in two different environments which were specified in Section 6.2. After every 60 training episodes the agent was tested on 48 weather profiles. During the testing the agent was acting only based on his policy and there was no randomness in his actions. The testing episodes weren't counted into the duration of the training. The average reward over all 48 test weather profiles had been used as the metric for comparing the algorithms.

7.2 Hyper-parameter overview

In the following section the effects of deep Q-Network hyper-parameters are described. The description should provide the reader with a general idea about how the hyper-parameter influences the performance of the algorithm.

- Learning rate - affects how large steps the neural network will make using the gradient descend algorithm introduced in Section 2.3.5. If the learning rate is set too high the learning will diverge. If the learning rate is set too low the neural network won't be able to converge in a reasonable amount of time. Learning rate has usually the largest effect on the performance of the neural network when sufficient neural network architecture is chosen for the given problem. It is recommended to tune the learning rate first [Ben12].

- Hidden layers - the number of hidden layers and the size of hidden layers decides how complex functions the neural network is able to approximate. By increasing the size of hidden layers we also enable the neural network to better generalize. On the other hand increasing the complexity of hidden layers increases the computational cost.

- Training length - increasing the training length leads to more states explored by the agent. This results in the increase of the neural network's performance on the training data. When the training length is set too high it could lead to overfitting of the neural network. Overfitting occurs when the performance on the train set is substantially higher than the performance of the algorithm on the testing set. Cross validation is used in order to prevent the algorithm from overfitting. Cross validation divides the training data into training, testing, and validation data groups. The algorithm is then trained using the training data, and tested during training on the validation data. The final result of the algorithm is measured using the test data, which the algorithm hadn't encountered during the training.

- Buffer size - after every transition made by the agent the transition is recorded in the experience buffer. When the experience buffer size exceeds the value defined by this hyper-parameter the oldest recorded transition is replaced by the new transition.

- Exploration steps - the exploration rate starts at the value of 1 and linearly decreases to the value defined by the exploration final value hyper-parameter, during the first x steps defined by Exploration steps hyper-parameter.

- Exploration final episode - sets the value of exploration rate after the number of steps exceeds the value of exploration steps hyper-parameter.

- Train frequency - decides how often the training of the neural network is executed.

- Batch size - the number of transitions which will be used for training during each step. The samples are taken randomly from the replay buffer.

- Learning starts - defines the number of steps at the beginning of the simulation during which the agent isn't learning and is only recording the transitions he had made.

- Gamma - influences how much into the future the algorithm looks. By setting gamma to 0 the agent would react only based on the value of the current state. It is recommended to set the value of gamma to 0.9 or to 0.95.
- Gradient norm clipping - the gradient norm clipping is used in order to stabilize the learning of the algorithm [MKS+15]. If the L2-norm value of the gradient exceeds the value specified by the gradient norm clipping, the gradient is clipped to be equal the gradient norm clipping value in the L2-norm. Which prevents the problem of exploding gradients. For understanding the full effects of gradient norm clipping we recommend to study the following paper [PMB12].
- Target network update freq - during the training of the deep Q-Network algorithm the target network is updated after every x-th step defined by the value of this hyper-parameter. This leads to better stability of the algorithm because the weights of the target Q-Network aren't changed after every step. However this solution comes in with the possible cost of decreasing the training speed of the algorithm if the hyper-parameter value is set too high. The decrease in the learning speed is caused by the fact that the weights of the target Q-Network aren't updated frequently enough.

7.3 Hyper-parameter tuning

The combinatorial space of the hyper-parameters is too large for us to search properly. In our work we will optimize the values for the learning rate and we will also try to find a suitable network architecture. We will be searching the hyper-parameter space by defining a subset of values for each hyper-parameter and then searching through the space by using every possible combination of the elements from the subsets. This technique is known as grid search. Grid search suffers from the curse of dimensionality. For high dimensional spaces it is recommended to use random search. However in low dimensional space the grid search proved to be a sufficient solution [BB12] for systematic hyper-parameter search. We have searched through the hyper-parameter space for optimal values for the learning rate and the number an size of hidden layers of neural network. The following learning rates had been searched through 0.005, 0.0005, 0.00005 and 0.000005. The following sizes and numbers of hidden layers had been searched through 64, 128, 256, 512, 1024, 2048, 128-128.

The less important hyper-parameters were set constant during the trainings.

The initial values for the hyper-parameters had been taken from the Open AI baselines implementation of DQN which were optimized for classic control problems. The hyper-parameters which are dependent on the environment the algorithm is trained in had been changed to fit our better environment. The batch size has been increased from 32 to 64. The target network update frequency has been decreased from 500 steps to 50. The start of learning has been changed from 1024 to 576. Which means the training starts after finishing one episode of simulation. During the first 60 episodes the exploration rate linearly decreases from 1 to to 0.01.

We used the Adam optimizer introduced in [KB14]. The values for the ADAM optimizer have been set to the values defined in the paper. The learning rate remains constant during the training. The Q-function is approximated using a fully connected neural network. The rectified linear unit (ReLu) [GBB11] is used as the activation function.

7.4 Hyper-parameters used

Hyper-parameter	Value
Optimizer	Adam
Activation function	ReLu
Layer type	fully connected
Learning rate	[0.005, 0.0005, 0.00005, 0.000005]
Learning rate decay	constant
Gradient norm clipping	10
Hidden layers	[64, 128, 256, 512, 1024, 2048, 128-128]
Buffer size	50000
Batch size	64
Gamma	0.9
Train freq	1
Target network update freq	50
Learning starts	576
Training length	360 episodes
Exploration steps	60 episodes
Exploration final episode	0.01

Table 7.1: An overview of the hyper-parameters used for the training of the neural network

7.5 Training results - building version 0

hidden layers, learning rate	mean	std	max	min	CI Up	CI Low	samples
[1024], 5e-05, 360	-512.7	25.1	-476.5	-545.7	-498.3	-527.1	10
[2048], 5e-05, 360	-578.5	73.5	-502.8	-710.0	-536.4	-620.6	10
[128, 128], 5e-05, 360	-634.6	135.1	-540.2	-898.9	-557.3	-711.9	10
[256], 0.0005, 360	-637.2	41.9	-573.1	-717.8	-613.2	-661.2	10
[2048], 0.0005, 360	-644.8	46.5	-575.6	-741.7	-618.2	-671.4	10
[512], 0.0005, 360	-650.7	56.2	-565.9	-753.0	-618.5	-682.9	10
[1024], 0.005, 360	-661.0	91.7	-524.0	-821.8	-608.5	-713.5	10
[1024], 0.0005, 360	-667.5	49.3	-611.8	-777.5	-639.3	-695.7	10
[128], 0.0005, 360	-681.5	76.9	-582.1	-852.2	-637.5	-725.5	10
[256], 0.005, 360	-686.2	134.5	-553.2	-953.9	-609.2	-763.2	10

Table 7.2: The ten best performing hyper-parameter configurations for Building version 0, sorted by mean

The ten best results of our algorithms are documented in the Table 7.2. From the analysis of the results we can conclude that the neural network with one hidden layer with 1024 neurons trained using learning rate 0.00005 had been able to outperform the other neural network architectures. We have supported this conclusion by running each of the best performing neural network architectures 10 times and then we have calculated the values for the upper bound and lower bound values of the mean with 95 percent confidence interval. The confidence intervals can be seen in the Table 7.2 in columns CI Up and CI Low. After observing the training data it is apparent that further increasing the number of neurons inside of the hidden layers won't result in better performance of the neural network. The stability of learning is shown in Figure 7.1, the figure contains the rewards of the ten best performing neural network architectures during their training. From the results it can be seen that the algorithms are steadily converging towards their optimal values. The term ftr in the legend of the Figure 7.1, denotes the reward obtained by the agent on the training data and the term sttr, denotes the maximal reward obtained on the cross validation data during the training. As we can see the algorithm is able to generalize properly and overfitting to training data hadn't occurred. We can also see that our optimal neural network architecture had been able to obtain the 3 best testing scores.

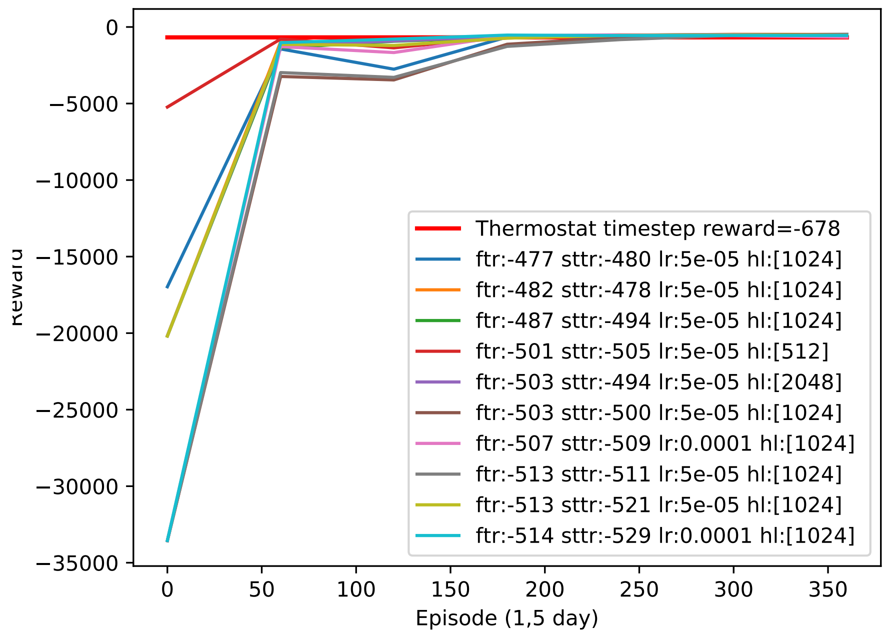


Figure 7.1: The performance of the neural networks during the training, the term ftr refers to the reward obtained on the testing data, the term sttr refers to the reward obtained on the cross validation data, we can see that during learning the neural networks are stable

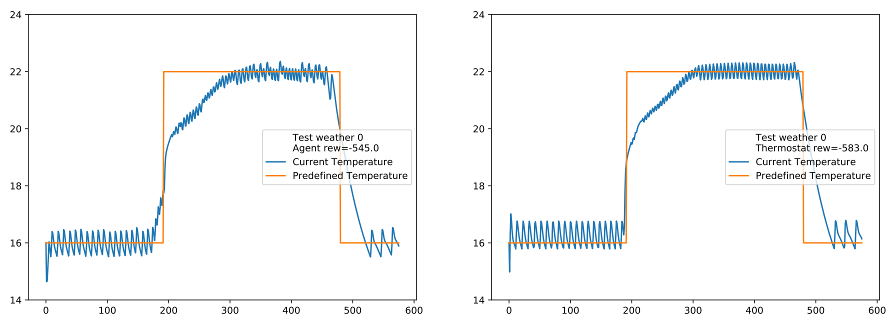
7.5.1 Comparison with the rule based solutions

We have compared the results of our agents against the solutions obtained using the rule based thermostats. We have compared the performance of both our solutions on the score obtained on the testing data and we have also examined the results obtained on different weather profiles which can occur during the year. The first weather profile, denoted W0 has the average temperature between -16 °C and -12 °C, each subsequent weather profile has average day temperature approximately by 4 °C higher than the previous temperature. The results are summarized in Table 7.3. The two best performing agents were used for this comparison. We can see that our agents were able to outperform the rule based solutions. With 95 percent confidence interval the mean of the test rewards of our best performing agent is -527 as can be seen in Table 7.2. Performance of the best rule based thermostat is -678 as can be seen in Table 7.3. We had compared the visualized strategies of our agent against the strategies of the rule based thermostat. We can see that our agent is able to better distribute the temperatures during the time when the user predefined temperature is constant, plotted in Figure 7.2. If we examine the behavior of the thermostat we can see that when predefined temperature is set to 22

°C, the thermostat is able to keep the temperature within the defined range. However when the temperature is set to 16 °C the thermostat actions leads to overheating. Our agent is able to keep the room temperatures within the defined range for both predefined temperatures. Our agent manages to do so with an update frequency of 2,5 minutes, while the thermostat has continuous control over the system. We expect that increasing the update frequency of our agent would lead to a better performance of the agent, at the cost of additional computation resources. Our agent is able to better react to the change in the predefined temperature by heating in advance after receiving information about the upcoming change in the predefined temperature. The ability to react to the change of the predefined temperature had shown to be the area where our agent is able to outperform the policies obtained by the rule based thermostat solutions by the highest margin. If we compare the results obtained by our two best performing agents we can see that the first agent was able to perform better in days when the outside temperature was lower. While the second agent performed better when the outside temperature was higher. After close inspection we have found out that the agent 1 had problems with keeping the temperature constant during the time when the predefined temperature was set to 22 °C, but the agent 1 have managed to better adapt to the changes in the temperature than the agent 2. This insight shows that the performance of the deep Q-Network can still be increased.

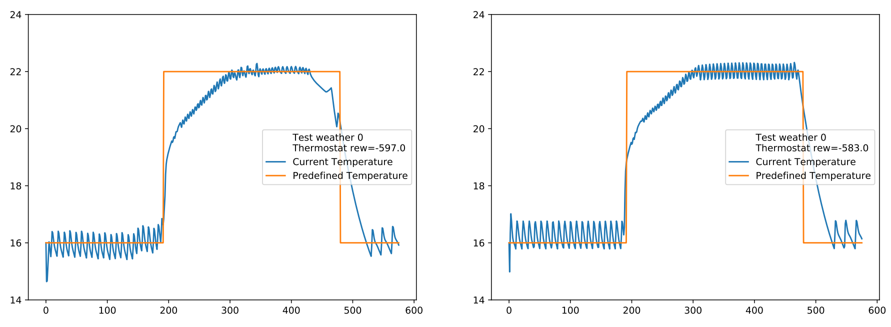
	AVG	W0	W1	W2	W3	W4	W5	W6	W7
Agent 1	-476.5	-545	-473	-355	-299	-370	-453	-696	-575
Agent 2	-482.5	-597.0	-503.0	-408.0	-326.0	-347.0	-417.0	-694.0	-537.0
Termostat	-678.0	-583	-493	-422	-455	-540	-748	-988	-783

Table 7.3: Comparison of the rewards obtained by the two best performing agents with the rewards obtained using the continous thermostat. The term AVG refers to the average reward obtained during the testing. The W0 label represents the performance of the agent when tested on the data where the average outside temperaure was between -16 and -12 °C. Every subsequent column W1, W2, ..., W7 shifts the bound of the average temperature by 4, this means W1 represents the agents performance on the test data with the average outside temperature between -12 and -8. W7 represents the ouside temperature between 12 and 16.



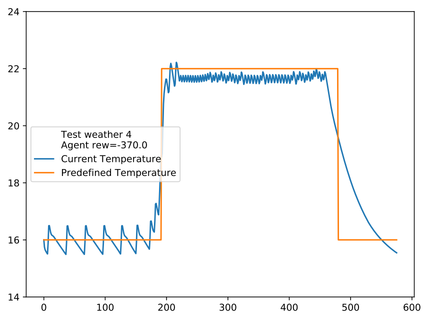
(a) : Agent1 **(b) :** Thermostat

Figure 7.2: Figure shows how the agent 1 performs when the average outside temperature is between -16 and -12 °C, the agent 1 is able to better control the thermoregulation of the building better than the agent 2, when the outside temperature is low. The main diffrence is in the ability to adapt to the change in the predefined temperature occuring in time step 480. The act of agent 2 is shown in Figure 7.4

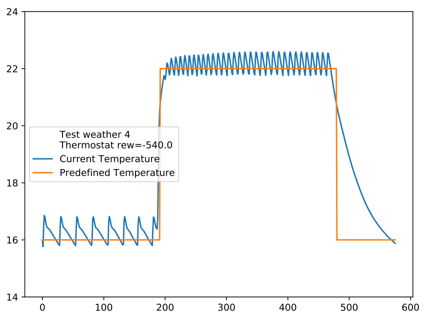


(a) : Agent2 **(b) :** Thermostat

Figure 7.4: Figure shows how the agent 2 performs when the average outside temperature is between -16 and -12 °C, we can see that the reward -597 obtained by the agent 2 is even lesser than the reward of -583 obtained by the thermostat

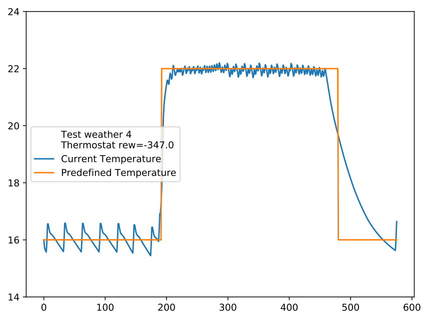


(a) : Agent1

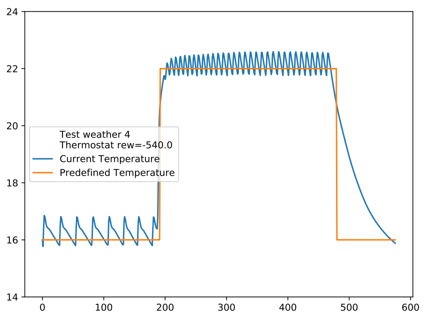


(b) : Thermostat

Figure 7.6: Figure shows how the agent 1 performs when the average outside temperature is between 0 and 4 °C, we can see that the agent 1 is able to outperform the thermostat, however the agent 2 is starting to get better rewards than the agent 1 because the temperature is getting higher

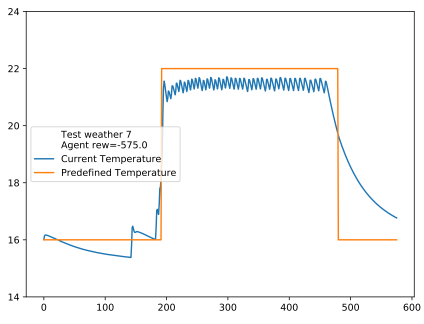


(a) : Agent2

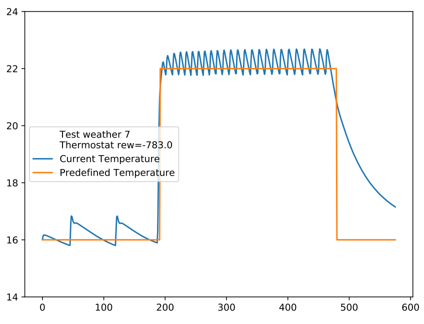


(b) : Thermostat

Figure 7.8: Figure shows how the agent 2 performs when the average outside temperature is between 0 and 4 °C, we can see that the agent 2 is able to outperform the agent 1 by better regulating the temperature when the predefined temperature is set to 22 °C

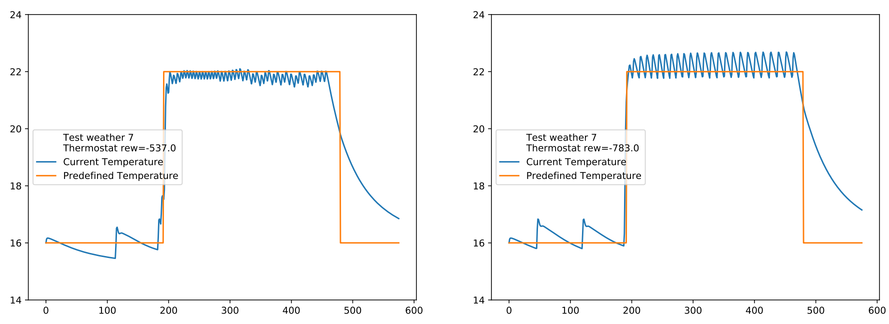


(a) : Agent1



(b) : Thermostat

Figure 7.10: Figure shows how the agent 1 performs when the average outside temperature is between 12 and 16 °C, we can see that the agent 1 is able to outperform the thermostat by better adapting to the change in the predefined temperature.



(a) : Agent2 **(b) :** Thermostat

Figure 7.12: Figure shows how the agent 2 performs when the average outside temperature is between 12 and 16 °C, we can see that the reward obtained by the agent 2 is better than the reward obtained by the agent 1. This is caused by the difference in the behavior when the predefined temperature is set to 22 °C. The agent 2 is able to better keep the constant temperature than the agent 1.

7.6 Training results - building version 1

hidden layers, learning rate	mean	std	max	min	CI Up	CI Low	samples
[1024], 0.0005	-822.1	85.8	-723.7	-898.9	-740.4	-903.8	5
[128, 128], 0.0005	-840.6	47.9	-768.5	-899.5	-795.0	-886.2	5
[128, 128], 0.005	-870.4	37.5	-811.8	-911.6	-834.7	-906.1	5
[64], 0.005	-879.9	48.1	-823.6	-946.2	-834.1	-925.7	5
[512], 0.005	-899.6	34.3	-866.2	-956.5	-866.9	-932.3	5
[2048], 0.0005	-900.3	40.8	-848.5	-939.1	-861.4	-939.2	5
[128], 0.005	-922.7	161.2	-703.4	-1063.9	-769.1	-1076.3	5
[2048], 0.005	-925.9	45.6	-859.7	-988.1	-882.5	-969.3	5
[1024], 0.005	-935.1	52.8	-872.0	-988.0	-884.8	-985.4	5
[512], 0.0005	-941.3	92.9	-779.6	-999.7	-852.8	-1029.8	5

Table 7.4: The ten best performing hyper-parameter configurations for Building version 1, sorted by mean

In Section 6.1 we had discussed the option of using the current time of the simulation as the feature based on which the algorithm would obtain the information about the change in the predefined temperature in the future. In our analysis we have came to the conclusion that our agent would face difficulties when learning an optimal policy for transition between the temperatures based solely on the current time of the simulation. We proved this conclusion by training multiple agents using this feature, the results of our trainings can be seen in Table 7.4. After the inspection of the results, we can see that the highest average reward we were able to obtain was -785. If we compare this result to the result obtained by training the DQN algorithm in the building environment version 0 we can conclude that for the training length of 360 episodes the building environment version 0 proved to be better environment. We statistically confirmed this statement using the 95 % confidence interval. The 95 % confidence interval upper bound of the agent's average reward in the building version 1 environment was -740, while the 95 % confidence lower bound of agent's average reward in the building version 0 was -527.

Chapter 8

Conclusion

In this thesis we have evaluated the effectiveness of using deep reinforcement learning for controlling the heating system of a building. The algorithms were evaluated in a simulation of a building developed in MATLAB. We have compared the performance of our solution to the performance achieved using the traditional rule based thermostats solutions.

At the beginning of the thesis the issue of using an approximation function with reinforcement learning was introduced to the reader. After that an overview and comparison of deep reinforcement learning algorithms which were considered to use for solving the problem of optimal control for the heating system were given. After establishing the theoretical background behind the deep reinforcement learning we have specified the exact specifications of our tasks and defined the parts of the task in which our algorithm could outperform the current solutions. After that we have documented the software development part of our project where we solved the problem of inter-process communication between Python and MATLAB/Simulink. After solving the problem of inter-process communication we have created an environment in Python which was able to control the simulation of the building. We have integrated an Open AI gym interface into our environment. This allowed us to use a wide variety of frameworks with implementations of deep reinforcement learning algorithms. Afterwards we have designed a suitable representation of the features used in our task.

After systematically searching through the hyper-parameter space of neural networks used in our work, we have obtained a neural network configuration which was able to solve the problem of optimal heating control in the building effectively. Our solution proved to be capable of achieving better performance than the rule based solutions traditionally used in this field. Previous conclusion was statistically proven. The average reward of our solution was -512 while the rule based thermostat was able to achieve the reward of -678. Our results showed that deep reinforcement learning is a viable option when considering an algorithm for optimal heat control in the building. We believe that the results we have achieved can be further improved and we have mentioned few areas to focus on in order to improve the performance of deep reinforcement learning.

Appendix A

Bibliography

[BB12] James Bergstra and Yoshua Bengio, *Random search for hyper-parameter optimization*, J. Mach. Learn. Res. **13** (2012), 281–305.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, *Openai gym*, 2016.

[Ben12] Yoshua Bengio, *Practical recommendations for gradient-based training of deep architectures*, CoRR **abs/1206.5533** (2012).

[DHK+17] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu, *Openai baselines*, `https://github.com/openai/baselines`, 2017.

[GB10] Xavier Glorot and Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Chia Laguna Resort, Sardinia, Italy) (Yee Whye Teh and Mike Titterington, eds.), Proceedings of Machine Learning Research, vol. 9, PMLR, 13–15 May 2010, pp. 249–256.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, *Deep sparse rectifier neural networks*, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (Fort Lauderdale, FL, USA) (Geoffrey Gordon, David Dunson, and Miroslav Dudík, eds.), Proceedings of Machine Learning Research, vol. 15, PMLR, 11–13 Apr 2011, pp. 315–323.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[HMvH$^+$17] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver, *Rainbow: Combining improvements in deep reinforcement learning*, CoRR **abs/1710.02298** (2017).

[KB14] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, CoRR **abs/1412.6980** (2014).

[Li17] Yuxi Li, *Deep reinforcement learning: An overview*, CoRR **abs/1701.07274** (2017).

[MBM$^+$16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, CoRR **abs/1602.01783** (2016).

[Met18] Meteoblue, *Weather data*, 2018, data retrieved from , `https://www.meteoblue.com/en/weather/archive/export/basel_switzerland_2661604`.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis, *Human-level control through deep reinforcement learning*, Nature **518** (2015), 529 EP –.

[PHD$^+$17] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz, *Parameter space noise for exploration*, CoRR **abs/1706.01905** (2017).

[PMB12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, *Understanding the exploding gradient problem*, CoRR **abs/1211.5063** (2012).

[SB18] R.S. Sutton and Barto, *Reinforcement learning: An introduction. (second edition)*, Cambridge: MIT press, 2018.

[SKF17] Michael Schaarschmidt, Alexander Kuhnle, and Kai Fricke, *Tensorforce: A tensorflow library for applied reinforcement learning*, Web page, 2017.

[SQAS15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, *Prioritized experience replay*, CoRR **abs/1511.05952** (2015).

[Sur18] Suryansh S., *Image of gradient descent*, image retrieved from, `https://hackernoon.com/gradient-descent-aynk-7cbe95a778da`, 2018, [Online; accessed 18-May-2018].

[TOHC15] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton, *Chainer: a next-generation open source framework for deep learning*, Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS), 2015.

[vHGS15] Hado van Hasselt, Arthur Guez, and David Silver, *Deep reinforcement learning with double q-learning*, CoRR **abs/1509.06461** (2015).

Appendix B

User manual

B.1 Requirements

The following programs and their respective modules are needed in order to execute the programs included with our work.

- MATLAB version 2017a and newer
  - Simulink
  - Hydronic Heating Toolbox(Request from Honeywell)
- Python version 3.4
  - Open AI Baselines module for Python
  - Open AI Gym module for Python
  - Matplotlib

Without the proper MATLAB version the simulation of the building will not run. The Python needs to be exactly version 3.4 in order to work with the saved neural network data provided alongside our work. A newer version than Python 3.4 can be used for training, but the saved data will still have the dependency on the Python version used for the training execution. We used Python 3.4 because it was already installed on the National Grid Infrastructure MetaCentrum. The list of all python modules required for the project are exported in the file requirements.txt.

The codes for the simulation will not be provided with this work. The reason behind this is that the source codes for the simulation of the building are not publicly available. After showing us the permission from Honeywell to use the codes for the simulation of the building we will give you access to our modified versions of them.

B.2 DVD structure

```
root
├── Python
│   └── Gym_env
├── Thesis
├── Saved_trainings
│   ├── bv0
│   ├── bv1
│   └── thermostat
└── data_preprocess
```

The structure of the project is shown above. The deep reinforcement learning algorithms alongside with the tools used for visualization are placed inside the Python Folder. The Python environment for controlling the simulation of the building is located inside the Gym_env folder. By placing the environment inside a separate folder we allow it to be easily transfered into another projects. The thesis in pdf format is located within the Thesis folder. The Saved_trainings folder contains saved data from our trainings. The bv0 folder contains saved data obtained from training the agent in the building version 0 environment. The bv1 folder contains saved data obtained from training the agent in the building version 1 environment. The thermostat folder contains data regarding the performance of the rule based thermostat solutions.

The data_preprocess folder contains the code for generating the data for training and testing with equal weather distributions. The bash scripts for running the training on MetaCentrum are located inside the root folder. The root folder also contains the bash scripts for training and testing the learning algorithms. The manual and readme are contained inside the root folder.

B.3 Script usage

The description of the scripts used in our project, along with the examples of the commands used for their execution is provided in the file manual.pdf contained inside the root folder of the project.