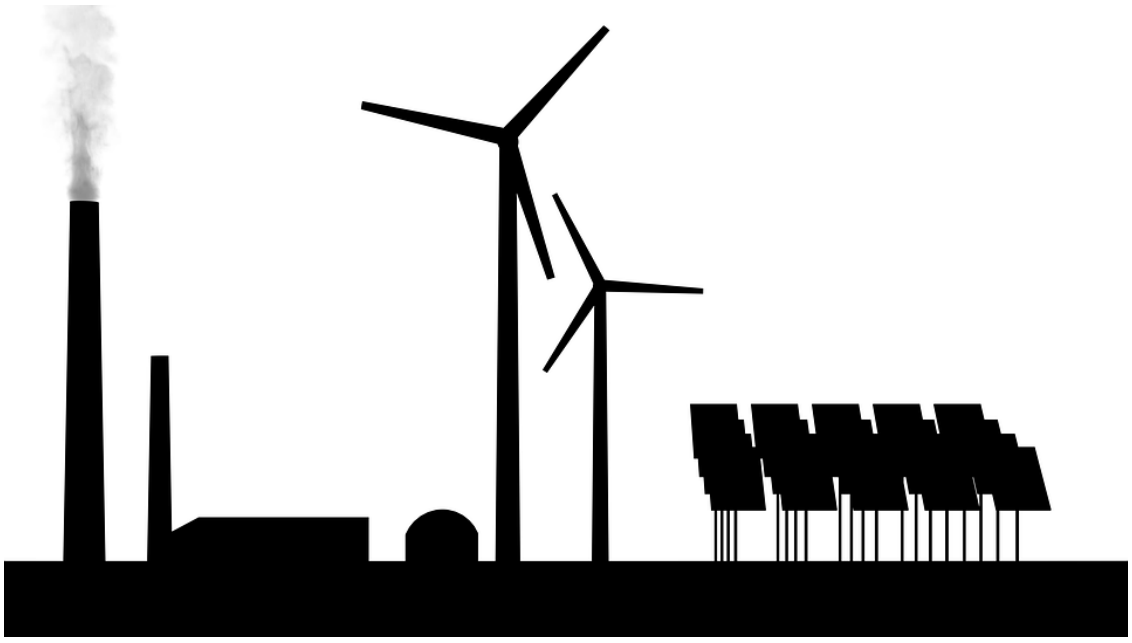
CHALMERS
UNIVERSITY OF TECHNOLOGY



Control temperature of room with reinforcement learning

Bachelor's thesis in Computer Science and Engineering

André Höjmark
Rickard Gyllensten

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg University
Gothenburg, Sweden 2019

Bachelor's thesis 2019

Department of Computer Science and Engineering

André Höjmark, Rickard Gyllensten
Chalmers University of Technology
Gothenburg University
Gothenburg, Sweden 2019

Control temperature of room with reinforcement learning

Supervisor: Sandro Stucki, Chalmers
Examiner: Peter Lundin, Chalmers

Bachelor's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg University
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2019

Abstract

The human connection to the increase of average temperature on earth is a known issue because of the energy need that is partly full-filled with fossil fuels. Currently 40% of the world's energy comes from buildings and by making heating/cooling systems more efficient there could be a big reduction of the energy need.

The purpose of this research is to explore the possibilities of implementing machine learning to regulate temperature in a room.

Using Python, Tensorflow and the Stable-baselines framework a simple model for reinforcement learning was created and trained on to explore if it was possible to use reinforcement learning to regulate the temperature in a room.

The trained model managed to control the inside-temperature in a stable manner, this with a highly fluctuating outside-temperature and with a room-size never seen before. The paper will also discuss the steps taken to create a model, a working algorithm and further work.

Keywords: Reinforcement learning, Tensorflow, Python, Stable-Baselines, Thermal control system, Temperature control, Room

Sammanfattning

Den mänskliga kopplingen till ökningen av medeltemperaturen på jorden är ett känt problem och påverkas dels utav energibehovet som täcks av fossila bränslen. I nuläget används 40% av världens energi till byggnader och genom att effektivisera uppvärmning och kylningssystemen finns det en chans att energibehovet minskar.

Syftet med denna rapport är att undersöka möjligheterna att använda maskininlärning för att reglera temperaturen i ett rum.

Genom att använda Python, Tensorflow och Stable-baselines-ramverket skapades och tränades en enkel modell för reinforcement learning. Modellen användes för att undersöka om det var möjligt att använda reinforcement learning för att reglera temperaturen i ett rum.

Den tränade modellen lyckades styra inomhustemperaturen på ett stabilt sätt på en modell som hade varierande utomhustemperatur och rumsstorlek som den aldrig sett förut. Rapporten diskuterar även utförligt stegen som gjorts för att skapa modellen och algoritmen samt förslag till framtida arbeten.

Acknowledgements

We would like to show our gratitude to the project owner Acobia FLUX for letting us do our project with them, and a special thank you to **Jakob Thorsell** for being our mentor at Acobia FLUX.

Secondly, we would like to send a special word out to our supervisor at Chalmers **Sandro Stucki**, for helping and guiding us through this project.

Thirdly we would like to thank our temporal supervisor and examiner **Peter Lundin**.

André Höjmark, Gothenburg 06 2019
Rickad Gyllensten, Gothenburg 06 2019

Contents

List of Figures

1
Introduction

The energy consumption of buildings stands for 40% of the world's energy consumption and has high potential to be reduced with a change of building layouts or with new techniques used to regulate the temperature [4]. One of these new techniques is the use of reinforcement learning which is a subsection of machine learning.

Reinforcement learning is based on having a model learn what optimal actions to take in different states to maximize a reward. In a heating system the states for example could be room temperature, water temperature & water flow and the actions water temperature & water flow. The training of the model is then done by testing different action values to see how it affects the room temperature. The changes that resulted into a room temperature closer to the target temperature is given a higher reward that is then used to identify good action changes [19].

There has already been previous work done in regard to regulating temperature with reinforcement learning. The company Google has for example recently implemented a more efficient cooling system using deep mind's machine learning algorithms that reduced energy cost by 40% at their data centre. What machine learning improved was [6].

- Taking the complex, non-linear interactions of the hardware such as pumps, chillers and cooling towers into consideration.
- Quickly being able to adapt to internal and external changes such as weather and work load of the servers where the old systems had problems to make a rule for every possible scenario.
- Creating a more general environment-model that work for large buildings with different layouts [6].

Stable Baselines is a library of reinforcement learning algorithms that is primarily used to simplify the use and tweaks of the algorithms. The library has example code of it solving reinforcement learning games from openAI that require the use of different techniques to score good. These techniques can then be transferred to work with regulating the temperature in a room [8].

1.1 Purpose and Problem definitions

The goal of this paper is to explore reinforcement learning by researching and creating software to regulate the temperature in an office room.

This report will try to answer the following questions...

- What are the benefits of using reinforcement learning to regulate temperature?
- What algorithms can be used to implement reinforcement learning to regulate temperature?
- What were the challenges in implementing reinforcement learning?

1.2 Limitations

The project will use a simplified general model that only takes heating power and outside temperature into consideration and some properties of the building.

1.3 Contributions

# We propose a simple model in Python that reinforcement learning is used on to regulate temperature.

# We briefly discuss the benefits of using reinforcement learning to regulate temperature.

# We describe the steps taken to get a working model and algorithm.

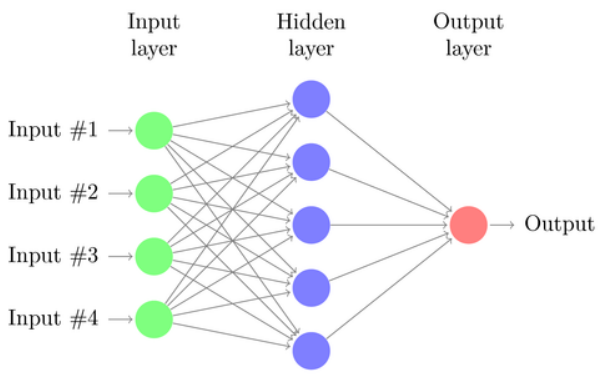
# The files are shared on GitHub.
`https://github.com/andrehoejmark/Tempearture-control`

2
Theory

This chapter will first talk about the algorithms used to do optimal control with reinforcement learning that often is a combination of neural networks and reinforcement learning. Next the different frameworks used to apply the different algorithms will be brought up and lastly an overview of our model that is meant to represent the physical environment that is needed to use the reinforcement learning algorithms.

2.1 Neural network

Neural networks consists of layers filled with neurons that are adjusted to map a certain input to an output such that all input images with, for example a dog give true otherwise false. These networks must be trained by adjusting the variables of every neuron's weight and bias by for example feeding tons of images with dogs and cats with a given result, and slowly learn from each fed image [15]. When there are more than one hidden layer which are the layers in between the input and the output layer the network gets named a deep neural network instead of neural network. [14]



The graphical representation of a feed forward neural network shows the typical structure of a neural network with an input layer, hidden layers and an output layer, all filled with neurons.

Figure 2.1: Feed forward neural network
Source: [1]

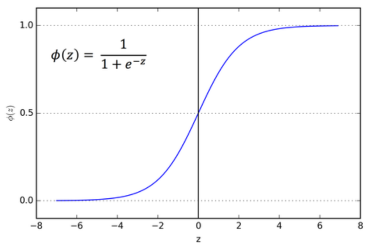
2.1.1 Neuron

The neuron has inputs x_i and weights w_i from the previous layer's neurons to the one neuron specified in the next layer. The neuron computes its output by taking the sum of the x_i * w_i from the previous layer's neurons - a bias that then gets squashed with an activation function $g()$ to get an output used as input for the next layer's neurons [15].

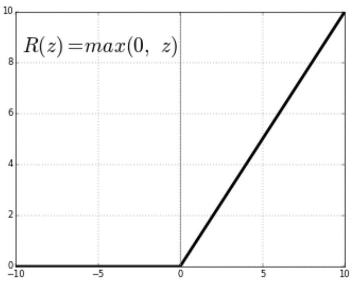
$$output = g(\sum_{i=1}^{N}(w_i * x_i) + b) \tag{2.1}$$

2.1.2 Activation function

The activation function $g()$ from equation 2.1 maps the total net input to the neuron - a bias to a value in range of a lower and upper limit usually between 0 to 1 or -1 to 1. Why we need activation functions is to introduce non-linearity into the network that makes it able to solve more complex problems [7].



(a) Softmax [3]



(b) Relu [18]

Figure 2.2: These are two commonly used activation functions: a) is a Softmax function and is used when a probability of the result is wanted, such as the probability that the object in the picture is a dog. The b) is a relu function that is commonly used because it's simple and works most of the times.[24].

2.2 Reinforcement Learning

Reinforcement learning is when the computer teaches itself from doing right and wrong on a certain task. This is done by trial and error and is measured by a potential reward for each action that the learner; also called agent, takes. For each time step t the agent performs an action based on the environment finite number of states S [4]. The environment states can in a simplified example be the balls position in the game PONG, the player's position and the opponent's position. The action, in this case, will be the probability to move the player up or down [12]. The state changes depending on the action and the environment gives a potential reward or penalty based if the S_{t+1} was desired or not. This is calculated by a reward function that differentiates depending on the environment.

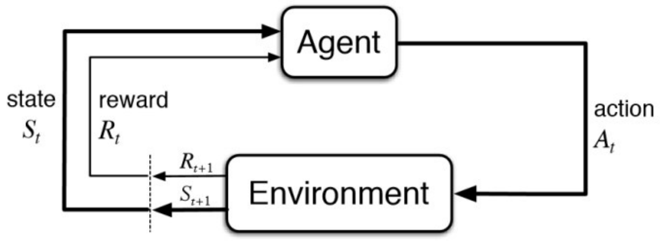


Figure 2.3: Feed forward neural network
Source: [2]

Using this reward function the agent can calculate which action yields the highest possible reward for that single time-step. Although that action might not be the most rewarding one in the long run. In excess of the reward function there is also a ***value function*** with the purpose of maximizing the reward during a longer time period. By combining the reward and value function the agent can determine the best possible step at each state [17].

For more complex tasks such as if the observation space is continuous then a combination of reinforcement learning and neural network can be implemented to better solve the task. The neural network will then primarily be used to calculate the probability of the actions the agent can use [20].

2.2.1 Q-learning

Q-learning has q-values $Q(s, a)$ for every state and action pair and is used to approximate maximum accumulated discounted reward when performing action a in state s, and continuing optimally from there. These q-values are then used to know what action to take to maximize reward in a specific state, and is updated by the following update rule [5].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma * argmax(Q(s_{t+1}, a)) - Q(s_t, a_t)]$$

In the update rule $argmax(Q(s_{t+1}, a))$ is the maximum total future reward from all possible actions in the next state, r_{t+1} is the actual reward after taking action a in state s, α is the learning rate and γ is the discount rate for future rewards [21].

2.2.2 Tabular Q-learning

Storing the state and action pair's q-values in tables works very well when the task to solve only has discrete values such as a 3x3 board game where you can move up, down, left, right. The table would then have a size of $(3x3) * 4 = 36$ which is every square on the board times the number of actions available in that state [25].

2.2.3 Deep Q-Network (DQN)

Deep Q-Networks are a combination of Q-learning and deep neural networks to be able to handle continuous state spaces or just when there are many states. The traditional Tabular Q-Learning doesn't work here since the memory would become too big to store every state and it would take a much longer time to train [25]. Worth noting that Deep Q-Learning doesn't work with large or continuous action spaces [22].

2.2.4 Value function

The value function is often denoted $V(s)$ and represents the total expected reward starting from state s and following the current used policy π to make the next actions. The difference between the value and Q-Function is that the Q-Function find the total expected reward after taking an action, and then follows the policy for next actions [26].

2.2.5 Policy Gradient Method

The policy gradient method works for continuous action and state spaces by having policy gradients that directly choose the actions instead of having a value or q-value for every state action pair. Also compared to Q-Learning there is no need for a explicit exploration and it often converges faster but needs more time-steps to train on [22].

2.2.6 Actor critic (A2C)

The actor critic method is based on having an actor that performs different actions with policy gradient and a critic that evaluates the different actions taken with a value function. This makes it possible to still work on continuous action/state spaces with policy gradient and still bring the steadiness of sampling trials from a value-based learning similar to Q-Learning [11].

2.2.7 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an reinforcement learning algorithm that works on continuous action/state spaces and is said to perform better than A2C and be easier to tune [16].

2.3 Frameworks

There are several frameworks that are useful when wanting to apply reinforcement learning and test different architectures such as Baselines, Stable Baselines, TensorForce, Qlearning4k, ChainerRL and many more [13]. Stable Baselines was used for this project.

2.3.1 OpenAI's Gym

OpenAI's gym is a platform for testing your reinforcement algorithms on already made models that are quick and often supports graphical rendering. There are different models depending on if you want to test discrete, continuous actions, algorithms, games or a custom made [8].

2.3.2 Stable Baselines

Stable Baselines is an improved version of the reinforcement learning framework Baselines with better documentation, an active community fixing bugs and making improvements. The framework is built with Python, Tensorflow and offers the functionality to work with custom models if they follow the OpenAI gym's interface. The documentation has examples of all the different algorithms and how to make custom models. What algorithms they offer are the following A2C, ACER, ACKTR, DDPG, DQN, GAIL, HER, PPO1, PPO2, SAC TRPO. For more information regarding Stable Baselines visit their GitHub page [8, 10]. In this project we used A2C and PPO2 from Stable Baselines.

2.3.3 Stable Baselines Interface

When creating custom models for Stable Baselines it must import a specific interface to be able to run it with openAI's OpenGym [8].

Stable Baseline's custom environment interface [9].

```python
import gym
from gym import spaces

class CustomEnv(gym.Env):
  """Custom Environment that follows gym interface"""
  metadata = {'render.modes': ['human']}

  def __init__(self, arg1, arg2, ...):
    super(CustomEnv, self).__init__()
    # Define action and observation space
    # They must be gym.spaces objects
    # Example when using discrete actions:
    self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)
    # Example for using image as input:
    self.observation_space = spaces.Box(low=0, high=255,
                                        shape=(HEIGHT, WIDTH,
  N_CHANNELS), dtype=np.uint8)

  def step(self, action):
    ...
  def reset(self):
    ...
  def render(self, mode='human', close=False):
    ...
```

2.4 Environment

The model of the environment works as a virtual simulation of the physical environment and is a necessity when training with reinforcement learning. This is because during training the reinforcement learning model will test many many different action values and then note the results. These results don't take effect instantaneously in the real world but varies in time. Therefore instead of waiting 10min between every test of action, the virtual simulation would give the result of 10min into the future instantly.

There are environments that are made more general and not specific to a particular room. This is not ideal since if the environment that we do the training on differs too much from the physical world it would compromise the performance and potentially not even make it applicable. Examples of what could change are the thickness of the walls, climate, furniture, ventilation and what underlying system is used for heating.

When building an environment there must be states that is what the reinforcement learning model will be able to see, actions that are what the reinforcement learning model will be able to change & adjust and a reward function that the reinforcement learning model will use to know what tested action values that are good or bad.

2.4.1 Our Environment

The model of the environment we have created is built to simulate a general room layout and not specifically tailored to a specific room. It has the interface that was required from Stable Baselines and its observation space is [power, reward, outside temp, current temp, room-size] and action space [power, cooling].

How it works is by taking the current_temperature, outside_temperature, power, cooling, room-size & target_temperature and after each step it spits out the newly updated room_temperature X minutes into the future. A scenario of this is where the model has $current\_temp = 17, outside\_temp = 20, target\_temp = 22$ and we call the function step(power=500) it would update the model's current_temp=17.5 & reward=0.25 and if power isn't big enough it would instead decrease the temperature instead of increase, depending on the outside temp.

2.4.2 Reward function

The reward function is meant to return a bigger value when the agent does something good. In our case it is good if the tested action value made the current_temperature move closer to the target temperature. In our case the reward function equal the equation below.

$$Reward = -(current\_temp - target\_temp)^2$$

This formula takes the distance between current_temp and target_temp and then makes it negative to get a more positive reward when closer to each other. The reason why the formula is quadratic is because it would make the model want to prioritize small changes rather than big ones [5]. This is because when the agent randomly tries action values it sometimes tries a big value and returns a very big negative number as a reward. The agent takes the big negative reward as a very bad thing and therefore learns to avoid it by making smaller changes to the action values to minimize the chances to get this big negative reward

3 Methods

At the start of the project, we were given a data set that included data points for water temperature, water flow, ventilation and some more. According to people who had done lots of reinforcement learning said that it could be too advanced for a student newly introduced to the field to create a model out of the data set. Therefore we instead created a simple model to see if it was possible to regulate temperature with reinforcement learning.

3.1 Environment creation

The original plan was to create the simplified environment-model within the program Simulink and have it retrieve inputs, process the data and then send the output back to our Python program for further training. After consulting with a teacher from Chalmers we decided that it would be simpler to create the model within Python itself and skip sending data between the two programs.

The model was created by finding inspiration from relevant simulations on GitHub. One model we found written in Python could set a target temperature, starting temperature and then automatically adjust heating power to move closer to target temperature. This was very similar to what we were trying to do but without machine learning. Therefore we looked more into the project [23] and used it as inspiration to create our model. We created the function $f(power)$ and for each time it was called the model moved 10 minutes into the future with that power setting and returned how the actual temperature changed. Then we applied that to the OpenAI interface that had to be followed to create a custom model for Stable Baselines. Also to make the model not overfit when training we had to make the model change the target temperature each time the model reset.

We tried to make the model as dynamic as possible right from the start. To make it easy to add or remove states and actions. But also with the ability to change the power and reward function.

3.2 Reinforcement algorithm creation

First we tried to write DQN from scratch in Python but then came to the conclusion that it didn't work with continuous action spaces but only with continuous state spaces. Therefore we decided to use Stable Baselines since it seemed too advanced to write those more advanced reinforcement learning algorithms from scratch. This enabled us to also test different algorithms with some ease.

3.3 Testing creation

During the progression of the project over 300 test has been performed to train the agents to behave in a desired way. With the environment being more simple at the beginning with fewer variables the focus was set on finding the optimal learning-rate * time-steps combination. This was time-consuming since some of the tests took hours to complete. Stable-Baselines has tensorflow integration, which enables GPU support. Performing the training on a GPU would decrease the time per test ten folds [5]. We couldn't get this to work on the computers the company had given us since their graphics cards were too old. Therefore most of the tests were done using the CPU. How we did the tests was by running 2x for-loops looping over two lists filled with time steps and learning rates then storing the models after each run, to then plot them when all was done.

Further into the project, we started to add more parameters that the agent used for learning, examples like outside temperature, room size and a cooling system.

4
Results

The result is a custom environment model written in Python with OpenAI's model interface that can take in power, room-size, cooling and outside temperature and give the corresponding change in temperature X minutes into the future. To this model, a reinforcement learning model has been created that can adjust the heating power sent to the model to have the actual temperature move closer to the target temperature. This works with a target temperature, room-size and outside temperature that varies.

4.1 One parameter

At the beginning of the project, the environment was only fed the input parameter of the *current temperature*. Using this the agent managed to learn what action to take depending on the *current temperature*. Since the *target temperature* was static during training the agent learned what action to take at each *current temperature* to yield the highest reward. Since the *target temperature* was not part of the input parameters resulted in the agent only being able to reach the static temperature that was set during training. The results can be seen in figure 4.1.

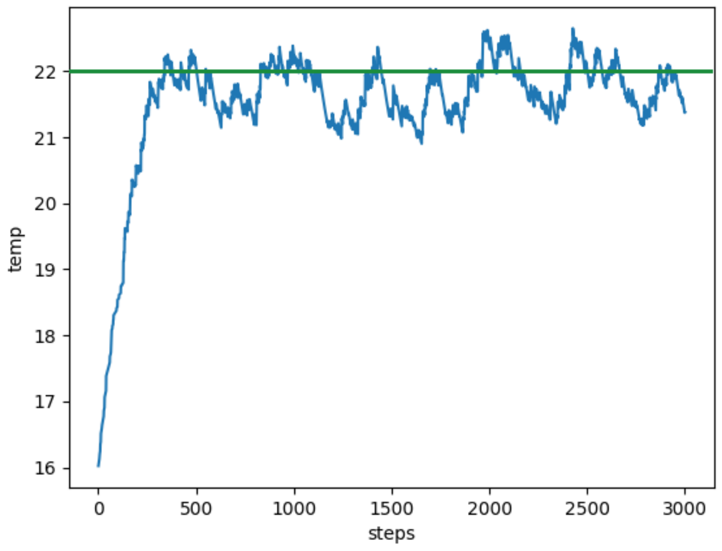


Figure 4.1: Target temperature 22°C. Environment inputs: Current temperature

4.2 Two parameters

Adding *target temperature* to the environment inputs the result changed for the better. The agent did not have to train in the blind anymore but could learn what action to take depending on the *current temperature* in regard of the *target temperature*. In figure 4.2 you can see that the agent managed to regulate the temperature in a more stable way than before. But more importantly reach any target temperature, even if it is not seen by the network before.

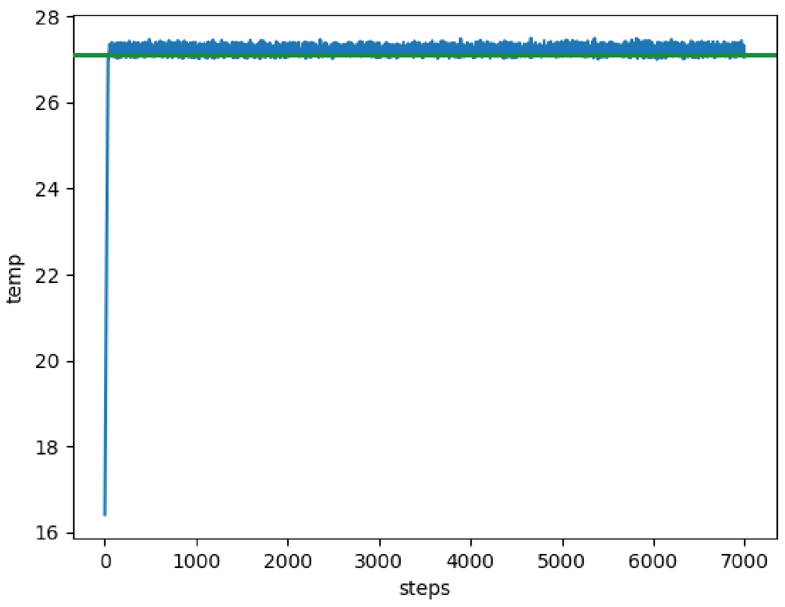


Figure 4.2: Target temperature 27°C. Environment inputs: Current temperature, Target temperature

4.3 Outside temperature

Adding a third input parameter; *outside temperature* brings the environment closer to the real situation that we are trying to simulate. Having a changing *outside temperature* during training forces the agent to learn how to compensate for this. One problem that occurs here is when the outside temperature rise above the target temperature since the agent can't lower the temperature than turning the heating-system off.

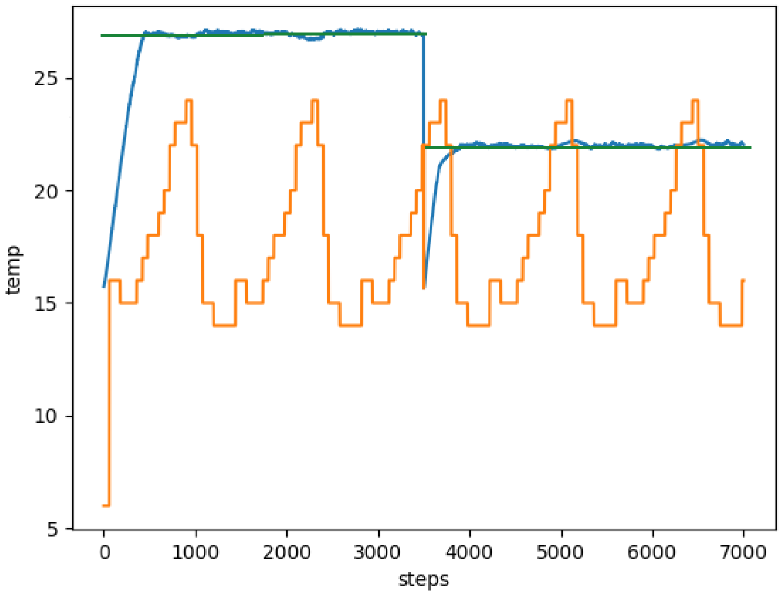
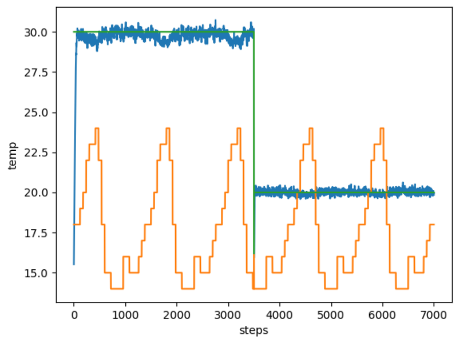


Figure 4.3: Target temperature 22°C and 27°C. Blue: Current temperature, Yellow: Outside temperature. Environment inputs: Current temperature, Target temperature, Outside temperature

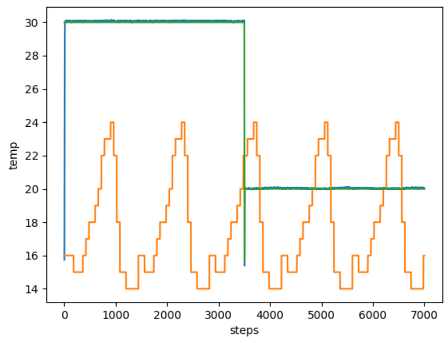
4.4 With Cooling unit and dynamic room size

This are the latest results achieved in the project. Before the agent only had the ability to control the power in a way that actively raised the temperature. This results in an unwanted temperature spike if *Outside temperature* > *Target temperate*. To solve this we added a way for the agent to lower the temperature; a cooling unit. The cooling unit was implemented in two different ways.
We gave the agent its own action space to control the cooling unit, parallel to the heating power. The result can be seen in figure (a) below. The agent solves the problem with the heat spikes but has a hard time to stabilize the temperature around the *target temperature*. The cause of this problem is our current *reward function*. Since we never tell the agent that it is recommended not to run the heating and cooling unit at the same time. Adding some kind of energy-cost to the *reward function* might solve this stability problem.

The second way for the agent to reduce the temperature was added in a simpler way. Here we changed the existing action space that earlier only was able to heat the room. By changing the action space and let it use a *negative* power we achieved the results we were looking for. It can be seen in the figure (b) below.
One thing to note here is that we also changed the size of the room during training. The tests (a) and (b) are performed on a room size never seen before but are still able to perform.



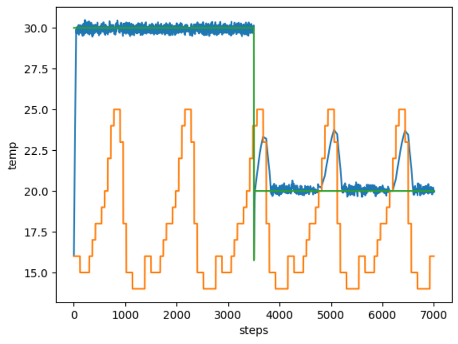
(a) 2 Action-space



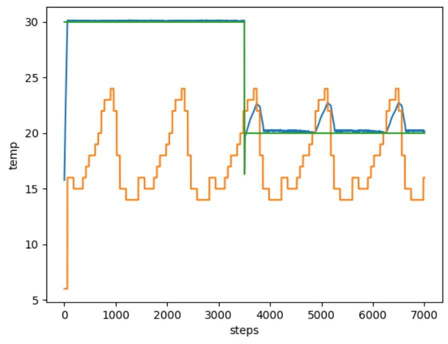
(b) 1 Action-space

4.5 A2C versus PPO2

Training the agent was done using two different reinforcement learning algorithms: A2C and PPO2. Both of theses algorithms support continuous actions and states which was a requirement in our case. During testing we noticed that the results were better when using PPO2 compared to A2C. These tests were done early in the project and we continued to use PP02 for the remaining parts. The tests were performed under the same circumstances with the same environmental properties. Notice that we did not use the cooling system here and that the heat spikes are clearly visible in figure (a) and (b). The big difference in performance can also be seen by comparing figure (a) and (b) where PP02 comes out on the top.



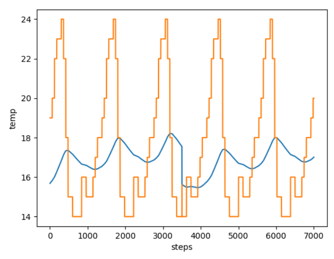
(a) A2C



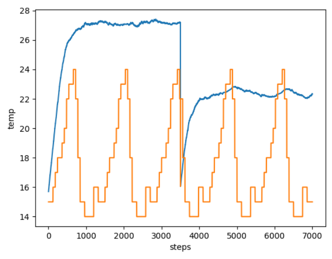
(b) PPO2

4.6 Learningrate versus Timesteps

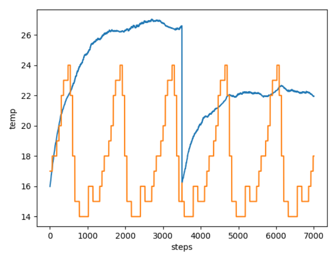
The combination of several different learning rates and time-steps was tested during the project. Changing the learning rate or the time that the agent trains makes a huge impact on the result. With a higher learning rate, the agent requires a shorter learning time but with the possibility of performing in unwanted behavior. In the figure below you can see the difference a change in learning rate can have on the results. Figure (a) is a clear example when the network fails to converge and is not close to reaching the *target temperature* of 27°C & 22°C. Comparing (b) and (c) shows that (c) learning rate is to low for the network to learn in the time given. Meanwhile (b) manages to complete the task is a desired way.



(a) Learning rate: 0.0001

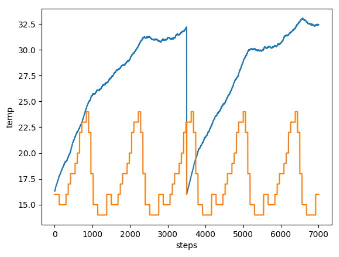


(b) Learning rate: 0.00005

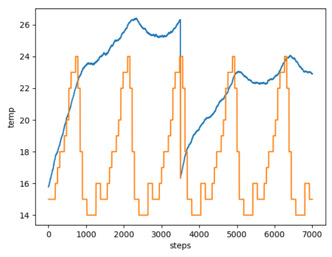


(c) Learning rate: 0.00001

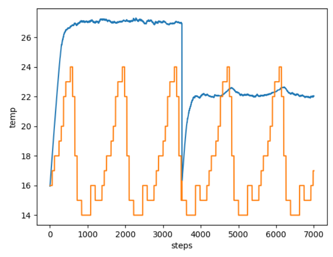
During the 300 test completed in this project, several different learning rates and time-steps have been combined to get the best results. We noticed during the testing that there is no correct answer to this but depends on the training data and the number of inputs in the environment. In the figure below can you see the difference time-step can have on the results. All trained with the same learning rate.



(a) Time-steps: 10^6



(b) Time-steps: $4*10^6$



(c) Time-steps: 10^7

5
Discussion

5.1 The benefits of using reinforcement learning to regulate temperature?

It can be very beneficial to use reinforcement learning to regulate the temperature if the system is very complex with many communicating parts such as Google's server that they managed to reduce energy consumption by 40%. But when the systems aren't that complex it can be questionable whether there will be a big noticeable increase in energy efficiency such as in a hotel rooms that uses good temperature regulators without machine learning. We didn't find any sources that had done any experiments similar to office/hotel rooms and since we couldn't test and compare our model with anything in the real world we couldn't really tell.

5.2 What algorithms are used to regulate temperature with reinforcement learning?

What algorithm to use when implementing reinforcement learning to regulate temperature depends on the system has discrete/continuous action and state spaces. For example a system that only has a boiler turned (on/off) to heat the water that is then fed to the pipes has a discrete action space (on/off) and a continuous state space (the actual temperature in the room). A system that has valves to increase/decrease water flow by opening/closing the valves with a percentage would have a continuous action space instead. The state space in a system that regulates temperature has to be continuous since it's most likely that the actual temperature in the room has to be a float value that has to be able to vary to get better results, and the action space can be discrete or continuous depending on what system.

What algorithm to use depends therefore on if the action space is continuous or discrete and for continuous action spaces we have used PPO2 and A2C which worked really well for our model and most likely for other models as well, and for discrete action spaces DQN is favorable because it is easy to implement and has loads of good and simple tutorials.

5.3 What were the challenges in implementing reinforcement learning?

5.3.1 Environment creation

The problem with our current environment-model is that it's not based on an actual physical system that we could test and compare our results on to see if it was better than their old systems. This was because creating a model from their data set was said to us from experts to be too hard and is a field of it's own, and since we hadn't done any courses in modeling we decided to only make a generic simple model to only test reinforcement learning algorithms.

If one had access to a physical room the agent could learn by performing actions in real-time and take a step every 10 minutes for example. Using this method would take longer time than a simulation but the model would reflect to the physical world and be useful when training is completed. The same agent could later be reused in similar rooms and would be a good solution for hotels or other buildings with equal sized rooms and heating system.

The middle-ground here would be as we mentioned before to create the model from existing data. A model that represent the physical world as close as possible. The difficulty here increases with the number of parameters that effects the heat in the room. In our case there were two air ventilation systems and one water heating system. Both with continuous valves that regulate water temperature, water flow, pressure in the air-drain,air temperature and airflow.

5.3.2 Algorithm creation from scratch

The creation of the algorithm was meant to be done from scratch but since there was too much to be learned to be able to implement it efficiently and get it to work we decided to go with Stable Baselines framework.

5.3.3 Environmental aspects

There are several potential benefits of applying machine learning to regulate temperature when 40% of the world's energy consumption comes from buildings. The company Google reduced their energy consumption of their server-base by 40% by regulating their cooling with reinforcement learning, but when regulating less complex buildings with for example office rooms we couldn't find any information regarding if there would be any big reduction in energy cost. Besides reducing the payment cost of heating/cooling, there could also be a reduction in the overall energy that the world has to produce from power plants and this would then in turn have a positive impact in decreasing the steadily rising temperature.

5.4 Further Development

The model is adjusting the temperature by changing the *power* variable to the system. This is not a good representation of the real world since there are often more variables in place. For example a room heated by both ventilation and water driven radiator has more than one parameter that will affect the temperature. Water temperature and water flow has its own interaction with the temperature, likewise with airflow and air temperature. Therefore one new feature to work on could be to build a better model that represents a real physical environment. This would preferably be built from a data set with gathered data from the previously implemented heating system.

The energy-consumption was an important factor in why this project is so interesting. It would be very interesting to give the reward function a way to prioritise actions that minimizes the energy cost. Maybe it would be more cost efficient to have lower water flow to reduce energy etc. This is something it would figure out itself by experimenting.

6
Conclusion

The end product is a reinforcement learning model with Stable-Baselines that is trained on a custom model that represents a general heating system built with the Stable-Baselines interface.

It is hard to tell whether it is more beneficial to use machine learning just to regulate office rooms, but when applying reinforcement learning to manage heating systems to for example servers that are more complex, there is huge potential for improvements.

There are several different algorithms to use such as A2C, PPO, Deep Q-Learning and many more depending on if the environment has continuous or discrete action/state spaces.

The hardest challenges was trying to apply the reinforcement learning algorithms from scratch and creating a model to train on. Why creating a environment-model was hard was because the given data set over the room was very bad and we were told it would take quite some time to create a custom model without any experience. The same goes for writing more advanced reinforcement learning algorithms from scratch such as A2C that is needed when the action/state space is continuous.

A brief summary of the further work is that we would try to improve the model of the environment to more closely represent the real room that we're trying to model by using data sets of a real physical room. Another thing would be to write the algorithms from scratch and understand them to make better adjustments. Last thing would be to test the actual model in a real room and potentially apply our trained model to a minicomputer such as a raspberry PI.

Bibliography

[1] Nerual-network. http://www.texample.net/media/tikz/examples/PNG/neural-network.png. Last Accessed 2019-05-29.

[2] Reinforcement png. https://www.kdnuggets.com/images/reinforcement-learning-fig1-700.jpg. Last Accessed 2019-05-29.

[3] Softmax. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Last Accessed 2019-05-29.

[4] Success stories in building energy efficiency. https://ccap.org/assets/Success-Stories-in-Building-Energy-Efficiency_CCAP.pdf. Last accessed 2019-05-28.

[5] ADL. An introduction to q-learning: reinforcement learning. https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/. Last accessed 2019-05-29.

[6] R. Evans J. Gao. Deepmind ai reduces google data centre cooling bill by 40%. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/. Last accessed 2019-05-28.

[7] Danijar Hafner. Why do neural networks need an activation function? https://www.quora.com/Why-do-neural-networks-need-an-activation-function. Last accessed 2019-05-23.

[8] https://github.com/hill-a/stable baselines/graphs/contributors. Stable-baselines-docs. https://stable-baselines.readthedocs.io/en/master/. Last accessed 2019-05-23.

[9] https://github.com/hill-a/stable baselines/graphs/contributors. Using custom environments. https://stable-baselines.readthedocs.io/en/master/guide/custom_env.html. Last accessed 2019-05-26.

[10] https://github.com/openai/gym/graphs/contributors. openai gym. https://gym.openai.com/. Last accessed 2019-05-23.

[11] Sergios Karagiannakos. The idea behind actor-critics and how a2c and a3c improve them. https://sergioskar.github.io/Actor_critics/, 2017-11-18.

[12] Andrej Karapathy. Deep reinforcement learning: Pong from pixels. http://karpathy.github.io/2016/05/31/rl/. Last accessed 2019-05-27.

[13] Edward Li. What is the most popular python framework to do reinforcement learning? https://www.quora.com/

`What-is-the-most-popular-Python-framework-to-do-reinforcement-learning`. Last accessed 2019-05-23.

[14] Bernard Marr. Deep learning vs neural networks - what's the difference? `https://bernardmarr.com/default.asp?contentID=1789`. Last accessed 2019-05-28.

[15] Matt Mazur. A step by step backpropagation example. `https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/`. Last accessed 2019-05-28.

[16] J. SchulmanOleg K. WolskiPrafulla D. Radford. Proximal policy optimization. `https://openai.com/blog/openai-baselines-ppo/`. Last accessed 2019-05-28.

[17] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An introduction.* The MIT Press, London, England, 1 edition, 5 2017.

[18] Kanchan Sarkar. Relu. `https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fe`. Last Accessed 2019-05-29.

[19] Thomas Simonini. An introduction to reinforcement learning. `https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419`,, 2018.

[20] Xander Steenbrugge. An introduction to reinforcement learning. `https://youtu.be/JgvyzIkgxF0?t=234`. Last Accessed 2019-05-28.

[21] Claes Strannegård. Lecture 4: Reinforcement learning, 2018. Chalmers University of Technology.

[22] Skowster the Geek. Policy gradient methods tutorial. `https://youtu.be/0c3r5EWeBvo?t=55`. Last Accessed 2019-02-22.

[23] Timtroendle. Simple simple. `https://github.com/timtroendle/simple-simple`. Last accessed 2019-06-14.

[24] Avinash Sharma V. Understanding activation functions in neural networks. `https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0`. Last accessed 2019-05-28.

[25] T. Xia and Z. Han. Path planning using reinforcement learning and objective data. `http://publications.lib.chalmers.se/records/fulltext/256451/256451.pdf`, 2017.

[26] Shaked Zychlinski. The complete reinforcement learning dictionary. `https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e`, 2019-02-23.

A
Appendix 1

```

# Environemnt that follows the gym interface


#  Parameters:
#  * heat_mass_capacity: capacity of the building's heat mass [J/K]
#  * heat_transmission: heat transmission to the outside [W/K]
#  * maximum_cooling_power: [W] (<= 0)
#  * maximum_heating_power: [W] (>= 0)
#  * initial_building_temperature
#  * time_step_size: [s]
#  * conditioned_floor_area: [m**2]


#Insprited from:
https://github.com/timtroendle/simple-simple/blob/develop/simplesimple/
    building.py
https://github.com/openai/gym/blob/master/gym/envs/classic_control/
    continuous_mountain_car.py




import gym
from gym import spaces
import numpy as np
from datetime import timedelta
import random
class Building(gym.Env):

  metadata = {'render.modes': ['human']}

  def __init__(self, target_temp):
    super(Building, self).__init__()

    self.step_counter = 0
    self.reset_counter = 0
    self.outside_counter = 0
    # Building settings1
    self.conditioned_floor_area = 100
    self.heat_mass_capacity = 165000 * self.conditioned_floor_area
    self.heat_transmission = 200
    self.time_step_size = timedelta(minutes=10) # How many minutes into
     future each step.
```

```

        # Outside Settings
        self.outside_temperature = 6
        self.outside_temp = [16,16,15,15,15,16,17,18,18,19,20,22,23,23,24,22,18,15,15]

        # Starting and target temperature.
        self.current_temperature = 16
        self.target_temperature = target_temp
        self.target_array = [16, 19, 21,25,37]

        # Action space for power.
        self.action_space = spaces.Box(low=0, high=1000, shape=(1,), dtype=np.float32)
        high = np.array([70,70,40])

        # Observatin space 1 variable between 0 and 70 that represents the temperature in the room.
        self.observation_space = spaces.Box(low=-high, high=high, dtype=np.float32)

    def rescale_power(self, power):
        p = power * 5000
        return p

    def step(self, action):


        self.step_counter += 1
        self.outside_counter +=1
        power = min(max(self.rescale_power(action[0]), 0), 10000)
        self.current_temperature = self.state[0]

        #Looping through the daily tempratures each hour
        if self.step_counter%6 == 0:
            if self.outside_counter >=23:
                self.outside_counter=0
            self.outside_temperature =  self.outside_temp[self.outside_counter]
            self.outside_counter += 1

        dt_by_cm = self.time_step_size.total_seconds() / self.heat_mass_capacity

        self.next_temperature = (self.current_temperature * (1 - dt_by_cm * self.heat_transmission) + dt_by_cm * (power + self.heat_transmission * self.outside_temperature))


        done = bool(self.step_counter > 10000)
        self.state = np.array([self.next_temperature, self.target_temperature,self.outside_temperature ])


        reward = -(self.next_temperature - self.target_temperature)**2

```

```python


    return self.state, reward, done, {}

  def reset(self):
    self.step_counter = 0
    self.state = np.array([np.random.uniform(low=15.6, high=16.4)])
    self.reset_counter += 1
    if self.reset_counter == 6:
      self.reset_counter = 0
    return self.state


  # NOT USED
  def render(self, mode='human', close=False):
    pass
```