Effective House Energy Management Using Reinforcement Learning Technical Documentation

Dawid Czarneta, Jakub Frąckiewicz, Filip Olszewski, Michał Popiel, Julia Szulc

June 3, 2018

Abstract

This is a technical documentation of our university project developed with cooperation and guidance of Rafał Pilarczyk from SAMSUNG. Document contains a short introduction and overview of main concepts, milestones, goals and effects of this project. It is followed by a technical part, that contains a setup guide, code and tests overviews, explanation of used algorithms and concepts (including the Reinforcement Learning part) and details about the development process, providing useful insights, experiences and lessons we have learned the hard way.

1 Introduction

1.1 Goals and milestones

1.2 Results

2 Reinforcement Learning

The project is based on one of the machine learning areas called reinforcement learning. The research and development of this field has been rising as of late, because of, among others, the improvements in deep learning methods and constantly increasing computing power. Combined with concepts like convolutional neural networks, RL algorithms achieve super-human performances in a variety of tasks. The main field of success is probably the gaming world, including both the traditional games like Chess or Go, and computer games, where, for example, classic Atari games are learned and solved by RL agents taking the input directly from raw pixels.

Reinforcement learning is often explained by comparison to supervised learning. It is usually named the most successful field of Machine Learning and it is based on type of datasets, in which you are given the correct answer for each observation. For example, the dataset consists of images, and each image has it is correct label as well - a word that describes the object on the image. In Reinforcement Learning, on the other hand, there is no 'supervisor'. Instead, the environment returns a reward signal, which indicates how good or bad the current situation is. The main goal of the agent is to maximize the cummulative reward.

There is a common vocabulary that is used to describe this setup. The **agent** (RL algorithm) has to interact with the **environment**, by performing **actions**. Environment gives the agent information about the **current state**, as well as the **reward signal** for the last timeframe. Agent has to choose the best one from the given set of possible actions. This is usually happening until the environment reaches a **terminal state**. Simulation from the start to terminal state is called an **episode**. A **Transition** is defined as a sequence - state, action, reward and next state - and is used as a minimal

To put this into perspective: In our project, the smart house manager is the agent. The simulated world is the environment. Actions are used to control devices such as heating or light. The state is a collection of information about the weather, inside parameters, current devices settings and user requests - everything

that is needed for agent to perform rational decisions. The episode can be defined as - for example - one single day, where the midnight is a terminal state.

2.1 What have we used?

Our agent uses Double DQN with Prioritized Experience Replay. This is a combination of recent developments in the field, which improve the speed, quality and stability of learning, and sometimes are essential for solving particular tasks. For an explanation of these concepts and more detailed information about the agent, please see the Agent module section (4.2).

2.2 PyTorch Framework

To implement the agent and the concepts named above, we have used the PyTorch framework. It is now commonly believed, that this frameworks becomes the go-to Python framework used for Deep Learning, with some essential advantages when compared to TensorFlow. While this is still a matter of loud discusions, we have chosen the framework more out of curiosity, and because it is often called 'more Pythonic'.

We are using PyTorch for a very clear neural network model declaration, ready-to-use optimizers like SGD or Adagrad, automatic computation of the gradients of network's parameters, and to perform the Double DQN learning process. It also provides a clear way to use GPUs for the computations.

We have used the version 0.3.1 and it is important to note, that the newest version of the framework (0.4.0 on the day of this documentation's publication) requires some changes to the code.

3 Getting Started

3.1 Setup and Requirements

3.2 Learning and Simulation

3.3 Testing

4 Code overview

4.1 HouseEnergyEnvironment module

4.2 Agent module

4.3 Configuration file

5 Accuracy measures and tempo of learning

6 Development Process

6.1 Chronology

6.2 What Failed

7 Conclusions

8 Bibliography and Useful Sources