Reinforcement Learning for Elevator Control

Ayoube Ighissou - 901365

Abstract
Reinforcement Learning (RL) is learning what to do -how to map situations to actions- so as to maximize a numerical reward signal. The learner (Agent) is not told which actons to take, but instead his goal is to find out what actions will lead him to maximize the reward. In this project I applied RL fo an elevator control system, whose underlying mathematical model is based on several assumptions to simplify the model as much as possible and thus provide a system that is easy to re-implement and modify at will. In particular, I tried to compare different policies with each other to understand under which conditions the agent was able to perform better, during the training phase.

Keywords
Reinforcement Learning — Policy Evaluation — Custom Environment

[1] *Department of Physics, University of Milan - Bicocca*
*__Corresponding author__: a.ighissou@campus.unimib.it

Contents

Introduction

For most of the people in the world, elevators have become an integral part of daily lives. Indeed, since the invention of this technology back in 1880, they have revolutionised the world of vertical transport and shaping the way we move through complex structures such as high buildings, transport hubs (metropolitan, train stations, airports), shopping centers and so on. But with the rapid spread of such technology and the increasing complexity of facilities, it is necessary to flank the mechanisms that regulate it, with algorithms that can optimise them in multiple aspects, such as waiting time, energy consumption and traffic flow. One of the primary objectives in elevator control system optimization is reducing the waiting time and this for several reasons. For example, if the average waiting times is too high could lead to inconvenience but also to some psychological and physical ills. For this reasons, RL could be a powerful tool for this specific domain. By leveraging its ability to learn from interactions and make smart decisions based on rewards, it could be developed robust and adaptive control framework. From a different perspective, elevator systems can serve as useful benchmark applications for the study of RL. Elevator control is well.suited for this branch of artificial intelligence, because reward signals encoding relevant performance indices are easy to obtain, but at the same time a model of the task is difficult to derive, and an optimal control policy is unknown. In this project I apply RL to a single elevator system. In particular, the report is divided into four main sections. In the first. i will attempt to briefly set out the fundamentals necessary to understand the problem, and then describe the experiment I conducted, highlighting the assumptions I made to develop the model and the algorithms I used, with their relative performance. Finally, in the last two sections, I have tried to draw conclusions and reflections regarding the algorithms used and possible improvements that can be applied in the future.

1. Models Overview

1.1 Reinforcement Learning in-a-nutshell

RL, in order to maximize a cumulative reward signal, searches for a control policy, which is a mapping from states to probabilities of selecting each possible action. In RL, the agent interacts with the environment in discrete time. The big challenge is to design a policy of what actions to take given a state s to maximize the chance of getting a future reward. This is called policy instead of control law for several reasons: partly because the environment is not deterministic, but its probabilistic and so this policy is going to be probabilisitic.

$$\Pi(s,a) = \mathbb{P}(A = a|S = s)$$

Once I decided the policy I might want to know whihc is the value of being in a certain state s given a certain policy Π. So once I choose a policy I can start to learn what is the value of each state of the system

$$V_\pi(s) = \mathbb{E}[\sum_t \gamma^t r_t|S_0 = s]$$

The formula above represents the expectation of how much reward I'll get in the future if I start in that specific state and I act that policy. The γ inside the expectation represents a discount factor, which basically tells you how much you favor getting a reward right now versus far in the future (can be also regarded as encoding uncertainty about future events).

Having made these general remarks on the basics of reinforcement learning, there are 3 fundamental concepts in this project that need to be properly explored and they concern the policies chosen to drive the model. Below is a brief explanation of each of the 3:

1.2 Q-Learning

Q-Learning is an off policy temporal difference method that considers future rewards while updating the value function for a given stat-action pair. An advantage we get with Value-based methods is we don't need to wait until the end of the episode to get the final reward. The value of the state-action pair (s,a) under the policy π, denoted by $Q^\pi(s,a)$, represents the expected return when starting in state s, taking action a and following the policy π

$$Q^\pi(s,a) = \mathbb{E}^\pi[R_t|s_t = s, a_t = a]$$

The optimal action value function Q^* is defined as the maximum Q-function over all the policies:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

Once Q^* is know, an optimal policy can be found by a maximization over the action argument:

$$\pi^*(s) = arg\max_{a\in A} Q^\pi(s,a)$$

The q-Learning algorithm iteratively estimates Q^* from interaction with the environment by using an updating formula.

1.3 Proximal Policy Optimization

The PPO policy was developed trying to answer the following question: how can we take the biggest possible improvement step on a policy using the data we currently have, without stepping so far that we accidentally cause performance instability and collapse? PPO does so with first-order methods that use a few tricks to keep new policies close to old. PPO trains a stochastic policy in an on-policy way. This means that it explores by sampling actions according to the latest version of its stochastic policy. The amount of randomness in action selection depends on both initial conditions and the training procedure. Over the training, the policy typically becomes less random, as the update rule encourages it to exploit rewards that it has already found, thus resulting more conservative. Since it is a policy gradient method an objective function to be minimized is needed, indeed in PPO it is:

$$L = \min(r_t(\theta)\widehat{A}_t, clip[r_t(\theta), 1-\varepsilon, 1+\varepsilon]\widehat{A}_t)$$

Where $r_t(\theta)$ is :

$$r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$$

And the ε is an hyperparameter which says how far away the new policy is allowed to go from the old. For this reason PPO results being a more conservative policy, because the smaller is ε the smaller will be the changes.

2. Experiment

In this section I will offer a panaromic overview of the model used for this porject and the assumptions made up.

2.1 Model Assumptions

The elevator control systems that I thought can cover a space from the ground floor to the second floor. I imagined being in a place where the entrance and exit was only on the ground floor, while on the other two floors people could either move to a different floor or go down to the ground floor to exit. so if I am on the first floor, I can either go up to the second floor, or go down to the ground floor and exit the place. People waiting for the elevator on floor 0 are coming from outside and they want to go on the other floors, and we also supposed that people can arrive at floor zero from outside at any moment while people arriving at a positive floor at the same time. For the sake of simplicity, I consider one single elevator in the first place. This simplification does not remove much generality because it is always the case that only one elevator is available in the structure. For instance, if we thought about the metropolitan in Milan, it often happens that there's only one elevator available to move up to the ground floor. Every ten minutes people arrive at each positive floor and 7 people will join the waiting list of the elevator together. This makes sense because generally, nowadays many of the elevators are inside places that host or are subject to a huge amount of people flow, so they usually have large capabilities. For every person having arrived on a positive floor he will have the following probabilities to go to the other floors $p = [0.25, 0.75]$

Meanwhile I supposed that every 40 seconds, there is a 0.9 probability that one person will join the waiting list on the ground floor, he will have 0.5 probability to go to each of the two positive floors. Moreover the elevator need 10 seconds every time it stops and 2 seconds to travel between floors. You can see a representation in the figure 2

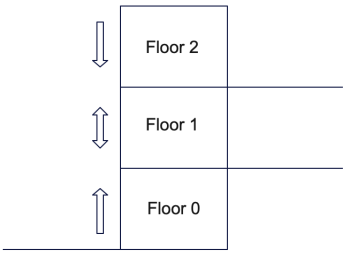

Figure 1. Elevator Model

2.2 State-Action space

The state space of the elevator system is discrete and has 5 dimensions. the state signal x is composed of the following variables:

$$x = [f_0, f_1, f_2, p, o]$$

where:

- $f_i, i = 0,1,2$: Binary values, representing call request on each floor i
- p: Discrete elevator position, taking values in 0,1,2
- o: Discrete elevator occupancy, taking values in [0,6]. the number 0 means no passengers are inside the elevator, while the number 7 means that the elevator is at capacity (reached maximum number of person)

The cardinality of the state space is then:

$$2^3 * 3 * 7 = 168$$

The elevator controller can choose among three discrete actions -1,0,1. The -1 action moves down, the 0 action stops, while 1 moves up the elevator. Furthermore two constraints are enforced on actions, corresponding to physical constraints in the elevator system. For example if we are at floor 2 we can't move up, due to the fact that over this one we cannot move.

- The controller cannot choose the action 1 when the elevator is on the top floor. Similarly, it cannot choose the action -1 when the elevator is on the floor 0
- The elevator cannot switch directions during a single sample. This means the 0 action must be taken between the 1 and -1 actions.

As in the reference paper [1], the reward function is:

$$r(s) = -\sum_{i=0}^{2} c_i - o$$

the reward at state s_t is the negative of the number of all the passengers waiting for the elevator on the floors, and passengers inside the elevator waiting to exit on the ground floor. The optimal policy will attempt to drive the system into this situation in minimum time, thereby transporting passengers to their destination in minimum time.

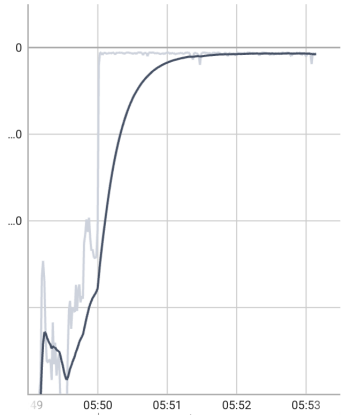

Figure 2. Entropy Loss - PPO

2.3 Performance Measures

In order to measure the performance of each chosen algorithm I used Tensorboard integrated with the library Stable Baselines, the main library used for this project. For each algorithm, unfortunately there's not the same plot that could let us compare properly the algorithms against each other, but we can extract some useful informations and lead to better insights about each of them.

2.3.1 PPO - Performance

The plots that are worthy of analysis in this section are:

- Entropy Loss: this measure tells basically how random the decisions of the model are. Usually should slowly decrease during a successful training process. As we can see from the plot, the model has after some episodes decrease too quickly, despite the hyperparameter calibration made during the developing of the code, thus indicating that a more detailed investigation should be made. Despite this details the plot shows that the action made by the agent over the episodes become less random, this highlighting the

2.3.2 DQN - Performance

For this algorithm a remarkable measure to interpret the performance is the plot of time/fps. Basically in this type of plots we have of the x-axis the time (which could be measured in various units such as episodes, training steps and so on) and it shows the progression of the training process; while in the y-axis represents the n umber of frames processed per unit of time. But in our domain we can translate this into interaction between the agent and the environment, where the agent observes a state, takes an action and receives a reward. In order to interpret this kind of plot, one must first be able to identify different phases within the graph:

- Initial phase: At the beginning of training the value on the y-axis might be low or unstable. This is because the DQN network is still learning to approximate the Q-values accurately, and the exploration-exploitation

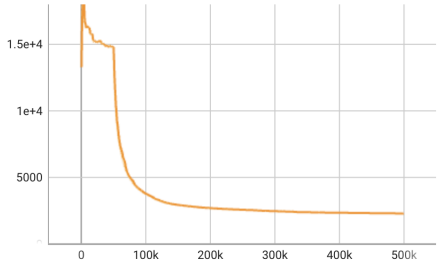

Figure 3. Time/FPS - DQN

trade-off is in effect. The agent explores the environment and the Q-network is updated, leading to slower processing

- Learning phase: As the training progresses, the values on the y-axis should generally increase. this indicates that the agent becomes more efficient in processing interactions and making decisions. During this phase the network learns to predict Q-values more accurately, which leads to faster decision-making and exploration

- Saturation Phase: At some point, the FPS might plateau or reach a steady.state. This suggests that the agent has learned a good policy and has become more stable in its decision-making

All these considerations can be found in the figure 3

3. Discussion

In this study, I applied two popular reinforcement learning algorithms, DQN and PPO, to develop an elevator control system. The aim to train an agent capable of efficiently controlling elevator movements to optimize passenger waiting time. In this section, I will summarize the work done, highlight the positive aspects of each algorithms performance and identify areas for improvement.

- The application of DQN to the elevator control system yielded promising results. Despite the results are not as expected before starting the project, in general it showed a good balance in the trade-off between exploration and exploitation. Furthermore, despite it is well known that one of the drawback of DQN is the instability problem during the training phase, here we can asses that after an initial phase, it has shown a reasonable performance in terms of decision making during the training. The initial phase shows that additional exploration strategies or modifications to the reward function structure should be made to encourage more efficient learning during the early stages.

- Applying Proximal Policy Optimization in this domain allowed me to reach a more stable results. Indeed this could be possible because PPO leverages the advantages of policy gradient methods and demonstrated slightly better performance than DQN algorithm. The train-entropy loss plot revealed a significant reduction in entropy loss over the training iterations, indicating the agent's improved policy determinism as well as that the agent transitioned from exploration to exploitation as training progressed.

References

[1] Robert Babuska Xu Yuan, Lucian Busoniu. Reinforcement learning for elevator control. 2008.

[2] https://www.gymlibrary.dev/.

[3] Domagoj Tolic´ Jens Kober Ivana Palunko Lucian Busoniu, Tim de Bruin. Reinforcement learning for control: Performance, stability, and deep approximators. 2018.

[2] [3]