

AUTONOMOUS UNIVERSITY OF BARCELONA

MASTER THESIS

Solving a Complex Optimization Problem for Product Delivery with Reinforcement Learning and Artificial Neural Networks

Autor:

Daniel Salgado Rojo

Director:

Toni Lozano Bagén¹

Master's degree in Modelling for Science and Engineering
Mathematics Department¹

Thursday 3rd May, 2018



“Artificial Intelligence is the new electricity.”

Andrew Ng

Data Scientist and co-founder of Coursera

Abstract

To do.

[1]

Acknowledgements

To do.

Nomenclature

(DEL TFG DE MATES)

To indicate that some magnitude X is a dimensional quantity we use the notation X^* , while for dimensionless quantities or the ones we do not care about their dimensions, simply by X .

The units we specify here are the ones that should be used in the model equations, and for convenience they will usually be the ones from the International System of Units.

Symbol	Units	Description
t^*	s	Time,
C	arbitrary	Mean field monomer concentration,
c^*	mol m^{-3}	Monomer concentration,
V_T	m^3	Total volume of solution,
M_p	kg mol^{-1}	Molar mass of a NP,
ρ_p	kg m^{-3}	Mass density of a NP,

Contents

Abstract	iii
Acknowledgements	v
Nomenclature	vii
1 Introduction	1
1.1 Goal optimization problem	2
1.2 Overview	2
2 Markov Decision Processes and Reinforcement Learning	3
2.1 Mathematical formalism of Markov Decision Processes	3
2.1.1 Markov Decision Processes	4
2.1.2 Policies	6
2.1.3 Optimality Criteria and Discounting	6
2.1.4 Value Functions	7
2.2 Reinforcement Learning	8
3 Problem approach	9
3.1 Problem description and constraints	9
3.2 Mathematical model (WORKING)	10
3.2.1 States: simple approach	12
3.2.2 States: complex approach	12
3.2.3 Actions	12
3.2.4 Rewards function	13
3.3 A simplified (toy) model	15
4 Classical Reinforcement Learning	17
4.1 The Q-learning algorithm	17

4.2	Simulations	19
4.3	Results	19
5	Neural Networks and Deep Learning	21
5.1	Introduction to Artificial Neural Networks	21
5.2	Training of Deep Neural Networks	21
5.3	Learning a simple policy with a Deep Neural Network	21
6	Deep Reinforcement Learning	23
6.1	Policy Gradient algorithms	23
6.2	Simulations	23
6.3	Results	23
7	(? En caso que hiciésemos algo más como lo de separar en k softmaxes y aprendiese, o bien rutas, etc.)	25
8	Conclusions and Future Work	27
A	A Product Delivery gym environment	31
B	Derivation of $J_{\text{levels}}^{(i)}$	33

Chapter 1

Introduction

Introducción/ co

- Motivation artificial intelligence? AlphaGO,...

1.1 Goal optimization problem

Imagine we are a petrol company which owns some number of petrol stations and we are paying a transport company to bring our product to the stations with their trucks in order to refuel them (see Figure 1.1).

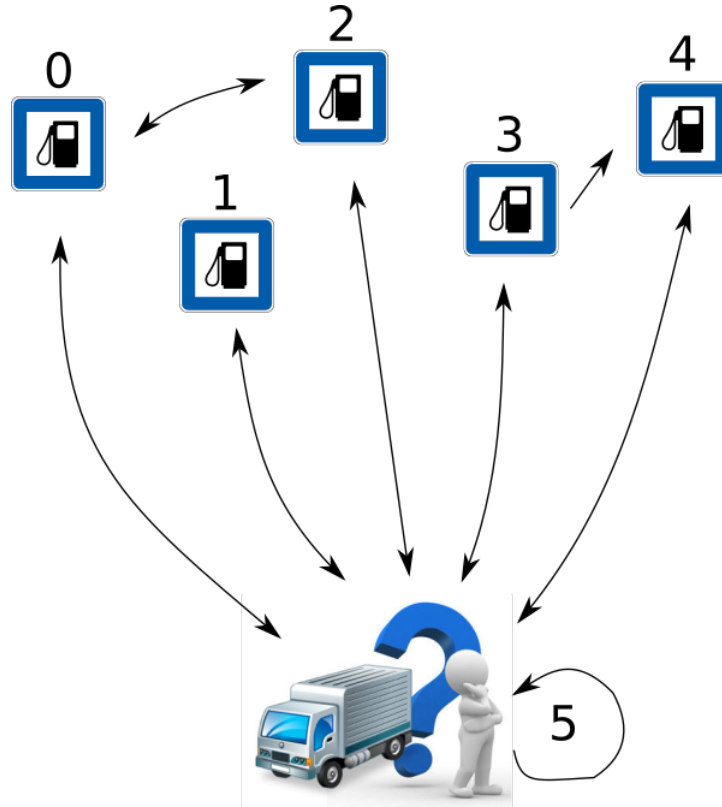


Figure 1.1: Product-delivery problem representation. (Truck image from: <https://www.pinterest.es/pin/521925044288845297/>)

The goal is to minimize the total amount of money (lets say per month or per year) that the petrol company has to pay to the transport company, ensuring that petrol is always available for costumers (among other constraints). Note that this problem can be generalized to the situation in which we have a company that is selling some product, and we have to pay to a transport company to distribute it from the loading dock so that it is available in all of our shops at any time.

solo hay que min-
imos de cosas
nó también qual-
ie no mueran las

1.2 Overview

Overview de las secciones para ubicar al lector.

Chapter 2

Markov Decision Processes and Reinforcement Learning

Markov Decision Processes (MDP) [2] are the fundamental mathematical formalism for *decision-theoretic planning* (DTP) [3], reinforcement learning (RL) [4, 5, 6] and other learning problems in stochastic domains [7].

In this chapter we introduce the basic concepts about Markov Decision Processes and Reinforcement Learning, focusing on the approaches that are adaptable to solve our goal optimization problem of product delivery. We mainly follow the first chapter in [7].

2.1 Mathematical formalism of Markov Decision Processes

In MDP models an *environment* is modelled as a set of *states* and *actions* that can be performed to control the system's state. The goal is to control the system by means of some *policy*, in such a way that some performance criteria is maximized (or minimized). MDP are commonly used to model problems such as stochastic planning problems, learning robot control and game playing.

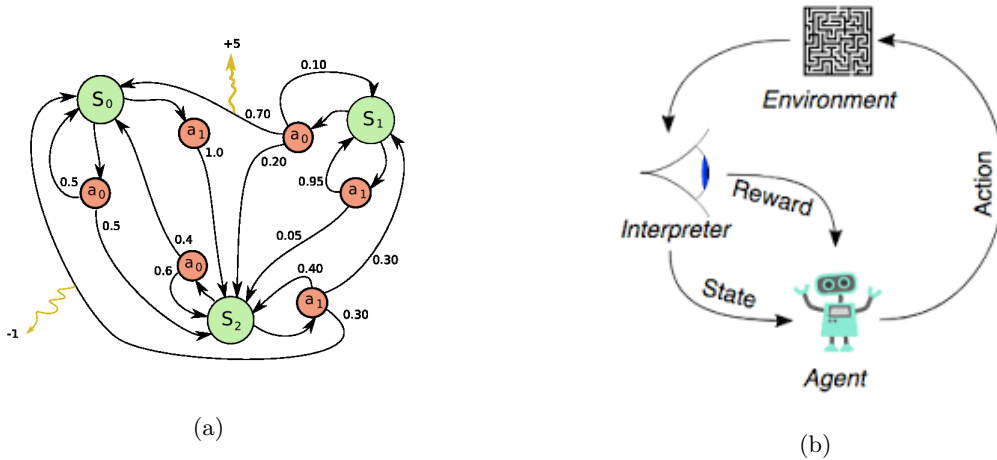


Figure 2.1: (a) “Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows)” [8]. (b) “The typical framing of a Reinforcement Learning (RL) scenario: an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent” [9].

RL is a general class of algorithms whose goal is to make an *agent* learn how to behave in an environment, where the only feedback consists of a scalar *reward* signal [7]. The goal of the agent is to perform actions that maximize rewards in the long run.

In Figure 2.1a we see the directed graph representation of a simple MDP with three states (green nodes) and some actions (red nodes). In a given state, only some actions can be performed, and each can lead to one or more states according to different probabilities. Transitions between states and the intermediate actions are represented by directed edges. In some state transitions, a positive or negative reward can be obtained (orange arrows) if we think of a MDP in the context of Reinforcement Learning (see Figure 2.1b).

2.1.1 Markov Decision Processes

The definition of MDPs we present at the end of this section consist of states, actions, transitions between states and a reward function. Although general MDPs may have infinite state and action spaces, we focus on discrete MDP problems with finite state and action spaces.

States

We denote $\mathcal{S} = \{s^1, \dots, s^N\}$ the set of possible states that can define the environment in a particular instant, and $|\mathcal{S}| = N$. Each $s \in \mathcal{S}$ is going to be considered as a tuple of *features* or properties that uniquely determine the environment state in a certain situation. For instance, in the *tic-tac-toe* game (see Figure 2.2 on the right), a state could be a tuple of 9 positions corresponding to each cell of the game's board and it may contain either 0, 1 or 2 (or any other three symbols) depending on if the cell is empty or occupied by one of the players.

Actions

We denote $\mathcal{A} = \{a^1, \dots, a^\Lambda\}$ the set of actions that can be applied to control the environment by changing its current state; $|\mathcal{A}| = \Lambda$. Usually not all actions are going to be applicable when the system is in a particular state $s \in \mathcal{S}$. Thus, we define $\mathcal{A}(s) \subseteq \mathcal{A}$ the set of actions that are applicable in state s .

The transition operator

If an action $a \in \mathcal{A}$ is applied in a state $s \in \mathcal{S}$, the system makes a transition from s to a new state $s' \in \mathcal{S}$ based on a probability distribution over the set of possible transitions [7]. We define the transition function T as

$$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow [0, 1]$$

$$(s, a, s') \longmapsto P(s'|s, a)$$

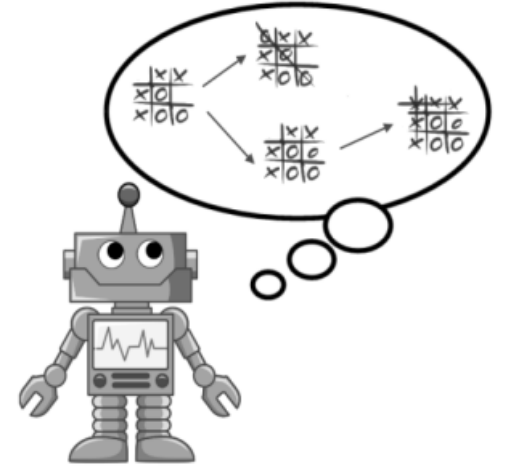


Figure 2.2: <https://mostafa-samir.github.io/Tic-Tac-Toe-AI/>

Hence, $T(s, a, s')$ is the probability of a system in a state s to make a transition to a new state s' after being applied action a .

There are some conditions to be satisfied so that T defines a proper probability distribution over possible next states:

- i) For all $s, s' \in \mathcal{S}, a \in \mathcal{A}$, $0 \leq T(s, a, s') \leq 1$,
- ii) To model the fact that some actions are not applicable when being in some states, one sets $T(s, a, s') = 0$ for all triples (s, a, s') with $s', s \in \mathcal{S}$ so that $a \notin \mathcal{A}(s)$.
- iii) For all $s \in \mathcal{S}, a \in \mathcal{A}$, $\sum_{s' \in \mathcal{S}} T(s, a, s') = 1$.

For talking about the *order* in which actions occur one defines a discrete *global clock*, $t \in \{0, 1, 2, \dots\}$, so that s_t, a_t denote the state and action at time t , respectively.

Definition 2.1. *The system being controlled is Markovian if the result of an action does not depend on the previous actions and visited states, but only depends on the current state, i.e. [7]*

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1}|s_t, a_t) = T(s_t, a_t, s_{t+1}). \quad (2.1)$$

In our context one assumes that the system being controlled is Markovian.

The rewards function

In classical optimization problems one seeks a *cost function* and aims to either maximize or minimize it under some set of conditions. In the context of RL, one talks about *rewards*, and the goal is to make an agent learn how to maximize the rewards obtained. For the type of problems we are interested to solve, we define the reward function R as

$$\begin{aligned} R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} &\longrightarrow \mathbb{R} \\ (s, a, s') &\longmapsto R(s, a, s') \end{aligned}$$

Thus, R is a scalar feedback signal which can be interpreted as a *punishment*, if negative, or a *reward*, if positive.

For the *tic-tac-toe* example, one could assign either high positive or negative rewards, or zero rewards, to actions that (when being in a given state) lead to win or lose the game, or having a draw, respectively. In this situation, the goal of the agent is to reach positive valued states, which means winning the game. Sometimes the problem is more complex and it is common to assign non-zero reward to combinations of states and actions that are good or bad under some criteria. For instance, having two aligned pieces in *tic-tac-toe* is good, so we may assign a positive reward for these situations.

In conclusion, rewards are used to give a *direction* in which way the MDP system should be controlled [7].

The Markov Decision Process

With all this stuff, we can now give the definition of a *Markov decision process*.

Definition 2.2. A *Markov decision process* (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, T, R)$ in which \mathcal{S} is a finite set of states, \mathcal{A} a finite set of actions, T a transition probability function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ and R a reward function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. One says that the pair T, R define the model of the MDP.

This definition of MDP allows us to model *episodic tasks* and *continuing tasks*:

- In episodic tasks, there is the notion of *episodes* of some length τ where the goal is to take the agent from some starting state to a *goal* state. In these cases, one can distinguish between *fixed horizon tasks* in which each episode consists of a fixed number of steps, or *indefinite horizon tasks* in which each episode can end but episodes can have arbitrary length [7]. An example for the first type would be *tic-tac-toe* and for the second *chess* board game.

In each episode the initial state of the system is initialized with some *initial state distribution* $I : \mathcal{S} \rightarrow [0, 1]$.

- In continuing or *infinite horizon* tasks the system does not end unless done it in purpose.

2.1.2 Policies

Given an MDP $(\mathcal{S}, \mathcal{A}, T, R)$, for us a (deterministic) policy is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps states to actions that are applicable in that state, i.e., $\pi(s) = a \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$.

A policy π can be used to make evolve, i.e. to control, a MDP system in the following way:

- Starting from a initial state $s_0 \in \mathcal{S}$, the next action the agent will do is taken as $a_0 = \pi(s_0)$.
- After the action is performed by the agent, according to the transition probability function T and the reward function R , a transition is made from s_0 to some state s_1 , with probability $T(s_0, a, s_1)$ and a obtained reward $r_0 = R(s_0, a_0, s_1)$.
- By iterating this process, one obtains a sequence $s_0, a_0, r_0, s_1, a_1, r_1, \dots$ of state-action-reward triples which constitute a trajectory (or path) of the MDP.

If the task is episodic, the sequence of state-action-reward triples ends in a finite number of iterations τ , the system is restarted and the process starts again with a new sampled initial state. If the task is continuing, the sequence can be extended indefinitely ($\tau = \infty$) [7].

2.1.3 Optimality Criteria and Discounting

The core problem of MDPs is to find an optimal policy π^* to control the system optimally. Therefore, we need a model for optimality with regards to policies.

Although there are several models of optimality for a MDP, we focus here on the so called *discounted average reward* criteria, which is applicable in both finite (episodic) and infinite (continuing) horizon tasks.

The discounted sum of rewards received or *return* by the agent starting from state s_t is defined as

$$R_t = \sum_{k=0}^{\tau} \gamma^k r_{t+k} \quad (2.2)$$

where $\tau < \infty$ if the task is episodic, and $\tau = \infty$ if it is continuing, and $\gamma \in [0, 1]$ is a discount factor for future rewards ($\gamma < 1$ if the task is continuing).

The factor γ determines the importance of future rewards:

- For $\gamma = 0$ (and taking $\gamma^0 = 1$) the only term that survives in equation (2.2) is the one corresponding to the present reward. In this case the agent will be *myopic* (short-sighted) by only considering current rewards.
- On the contrary, for $\gamma \approx 1$ the discount factor will make the agent strive for a long-term high reward [10]. Put the examples of gamma 0.95 and 0.99, 13, 69
- For episodic tasks and the case $\gamma = 1$, R_t is the sum of rewards at each step of an episode.

(cuando toque reescribirlo, ver [11]) The goal of the discounted average reward criteria in the context of MDP is to find a policy π^* that maximizes the expected return, $\mathbb{E}_\pi[R_t]$, over all episodes that start in state $s_t \in \mathcal{S}$. (???? policy depends on initial state?) This idea is formalized with the definition of the so-called value functions.

2.1.4 Value Functions

In order to link the criteria of optimality introduced in the precedent subsection to the policies, one considers the so-called *value functions*, which are a way of quantify “how good” it is for the agent being in a certain state or making a certain action when being in some particular state.

ver [11] para la notación de variables los valores que tom

Definition 2.3. The *value of a state s under policy π* , denoted $V^\pi(s)$ is the expected return when starting in state s and following π thereafter [7]. If we consider the discounted average reward criteria, we have the following expression:

$$V^\pi(s) = \mathbb{E}_\pi [R_t | s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\tau} \gamma^k r_{t+k} | s_t = s \right] \quad (2.3)$$

where $\tau < \infty$ if the task is episodic, and $\tau = \infty$ if it is continuing.

Definition 2.4. The *state-action value of s, a under policy π* , denoted $Q^\pi(s, a)$ is the expected return when starting in state s , taking action a and thereafter following π [7]. For the discounted average reward criteria we have the following expression:

$$Q^\pi(s, a) = \mathbb{E}_\pi [R_t | s_t = s, a_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\tau} \gamma^k r_{t+k} | s_t = s, a_t = a \right] \quad (2.4)$$

where $\tau < \infty$ if the task is episodic, and $\tau = \infty$ if it is continuing.

The value functions are known to satisfy the so-called *Bellman equations* [12] that we do not introduce in this work (see for example [7, 12, ?, ?]).

The main goal of a given MDP is to find the policy that receives the most reward. This is equivalent to find the policy that maximizes the value function for each possible state.

Definition 2.5. A policy π^* is said to be *optimal* if it is such that $V^{\pi^*}(s) \geq V^\pi(s)$ for all $s \in \mathcal{S}$ and all policies π . $V^* := V^{\pi^*}$ is called the *optimal value function*. Similarly one defines the *optimal Q-value*, Q^* .

Assuming we know the optimal Q-values for each state-action pair (i.e., $Q^*(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$), or an algorithm able to estimate them; one can greedily select an optimal action using the greedy Q-policy π_Q defined as

$$\pi_Q(s) = \arg \max_{a \in \mathcal{A}(s)} Q^*(s, a), \quad \forall s \in \mathcal{S}. \quad (2.5)$$

2.2 Reinforcement Learning

Once we have defined MDPs, policies, optimality criteria and value functions, the next step is to consider the question of how to compute optimal policies. Although there are several approaches, there are usually distinguished two types of algorithms: *model-based* and *model-free*.

Model-based algorithms assume that a model of the MDP is given, i.e. that a transition and a reward function pair (T, R) is known. This class of algorithms is also known as Dynamic Programming (DP).

On the contrary, model-free algorithms, which are under the general name of Reinforcement Learning, do not rely on the availability of a perfect model. “Instead, they rely on *interaction* with the environment, i.e. a *simulation* of the policy thereby generating *samples* of state transitions and rewards” [7]. These samples are then used to estimate the Q-values for each visited state-action pair $(s, a), s \in \mathcal{S}, a \in \mathcal{A}(s)$, by means of some iterative algorithm. Since a model of the MDP is not known a priori, the agent has to learn how to behave by experience. Thus, the agent has to *explore* the MDP to learn about its environment and how to behave in order to maximize the rewards obtained. “This naturally induces a *exploration-exploitation* trade-off which has to be balanced to obtain an optimal policy” [7].

As we are going to see during the following chapters, for the product delivery problem we introduced in section 1.1, we will have a model for the rewards function but we do not know anything about the transition function of our system modelled as an MDP. For this reason, we focus on model-free algorithms. In particular, we consider the Q-learning algorithm to estimate the optimal Q values.

Chapter 3

Problem approach

3.1 Problem description and constraints

A single product is to be delivered to several shops from a depot using several trucks (see Figure 3.1). We call *unload* the fact that a truck delivers (unloads) some quantity of product to a shop; unloads can be either *simple*, if a truck only delivers to one shop and then returns to the depot, or *shared* if a trucks delivers to more than one shop before returning to the depot. Trucks can only be loaded in the depot.

Moreover, some initial restrictions and assumptions are considered:

1. Trucks leave the depot fully loaded (lets say at 8 in the morning) and return completely empty (lets say at the end of the day, i.e., 23:59:59).
2. Trucks perform at most one *unload* (simple or shared) every day. Hence, it is possible that some day some of the trucks do not go to any shop.
3. A truck can not visit the same shop to perform a *unload* more than one time during the same day.
4. Trucks can perform a shared unload of at most $V \geq 1$ shops. In general V may depend on the truck.
5. The price that the company owning the shops has to pay to the transport company is given by some cost function J_{costs} which may depend on the distance travelled by the trucks from the depot to the corresponding shops, the quantity of product delivered and other contributions imposed by the transport company (for example, impose an additional cost if that day is a holiday, an additional cost depending on the number of shops visited for *shared* unloads, etc.).

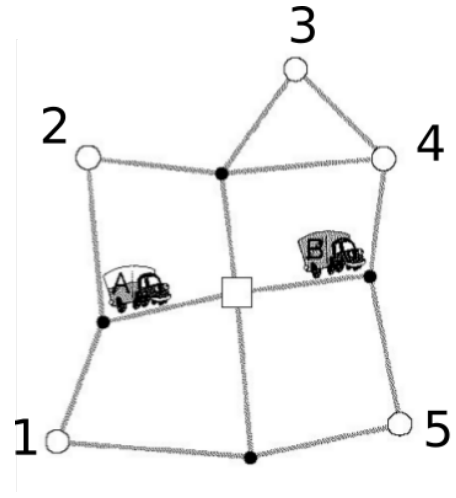


Figure 3.1: Product-delivery domain: the square in the centre is the depot (load doc) and white circles are the shops.. Adapted from [13].

Aquí faltaria alguna frase de enlace

Lets consider a particular shop and imagine that it stores the product in a cilindric container (see Figure 3.2). There are considered several levels of stock:

1. **Minimum capacity or zero level:** when there is no product in stock at all.
2. **Minimum level:** below this level it is considered as having a break of stock (high risk). Thus, we consider that below this level is like having no product in stock.
3. **Danger level:** from this level to the minimum level it is considered that we are taking a moderate risk, so that the shop should receive a *unload* of product soon. The smaller the value of stock below this level, the higher the risk of having a break of stock.
4. **Maximum level:** the desired maximum stock. This level can be overpassed in some quantity but never surpassing the maximum capacity level. The higher the value of stock above this level, the higher the risk of having a break of stock.
5. **Maximum capacity level:** when it is physically impossible to put more product in the container (i.e., however much we compress the products inside the container, we cannot fit more product).

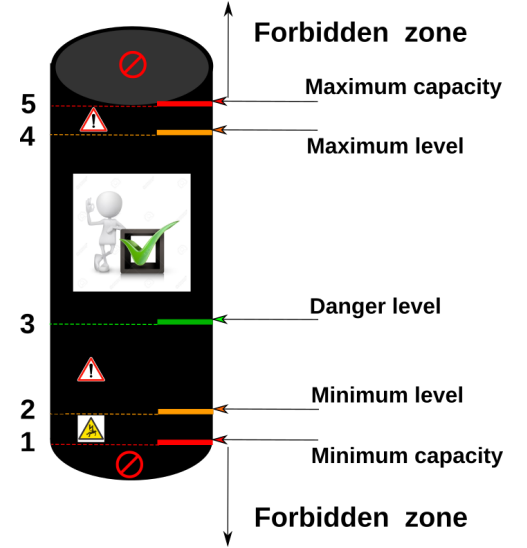


Figure 3.2: Stock levels for an abstract storage container of a given shop.

The ideal and desired state of stock is one between the danger level (3.) and the maximum level (4.).

3.2 Mathematical model (WORKING)

To formalize the model we use the Markov Decision Processes (MDP) nomenclature introduced in chapter 2.

Our system of shops and trucks can be modelled as a weighted and directed graph \mathcal{G} , a set of Trucks \mathcal{K} and a set of Tanks \mathcal{N} (the abstract storage containers of each shop, as in Figure 3.2). \mathcal{G} is determined by an adjacency matrix $A_{\mathcal{G}}$ and a matrix of weights $W_{\mathcal{G}}$, which are defined as follows:

$$(\mathcal{A}_{\mathcal{G}})_{ij} = \begin{cases} 1 & \text{if there is a path from shop } i \text{ to shop } j \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

$$(W_{\mathcal{G}})_{ij} = \begin{cases} w_{ij} \in \mathbb{R}^+ & \text{if there is a path from shop } i \text{ to shop } j \\ \infty & \text{otherwise} \end{cases} \quad (3.2)$$

Hence, weights are used to quantify the cost of going from a shop to another, and this cost is infinity when it is not possible to go from some shop to some other. Note that a component of W_G is ∞ if and only if the corresponding component of matrix A_G is zero.

As an example, the following matrix is the adjacency matrix of the system of 5 shops from Figure 1.1. The depot is considered to be the 6th node.

$$A_G = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

We consider n tanks and k trucks. Thus, $|\mathcal{K}| = k$ and $|\mathcal{N}| = n$.

If V_{\max} is the maximum number of shops that any of the k trucks of the system can visit each day. Then, we obtain a sequence of time labels that define a clock for our product delivery system

$$t \in \{t_{0,0}, \dots, t_{0,V_{\max}}, \dots, t_{\tau-1,0}, \dots, t_{\tau-1,V_{\max}}\},$$

such that the first index indicates the day and the second is aimed to divide a day in $V_{\max} + 1$ parts, and τ is some “last day” to consider. When the second index is zero, we assume that all trucks are fully loaded in the depot. We assume that it is only at the end of the day when the total amount of product in each tank is reduced by some amount according to what was consumed during all that day.

Each $N_i \in \mathcal{N}$ have the following intrinsic properties:

- An integer id, $i \in \{0, \dots, n-1\}$,
- A maximum load (maximum capacity) C_i ,
- The current load, $c_i = c_i(t)$,
- A discrete partition of the interval $[0, C_i]$ with d_i parts: $h_\beta = [\beta \cdot C_i/d_i, (\beta+1) \cdot C_i/d_i]$, $\beta = 0, \dots, d_i - 1$.

Each $K_i \in \mathcal{K}$ has the following intrinsic properties:

- An integer id, $i \in \{0, \dots, k-1\}$,
- A maximum load (maximum capacity) L_i ,
- The current load, $l_i = l_i(t)$,
- Its current position: characterized by the tank’s id where the truck is located, we call it $p_i = p_i(t)$. If $i = n$, the truck is in the depot.
- A discrete partition of the interval $[0, L_i]$ with m_i parts: $f_\alpha = [\alpha \cdot L_i/m_i, (\alpha+1) \cdot L_i/m_i]$, $\alpha = 0, \dots, m_i - 1$.

This leads to a discrete set of possible ‘deliveries’ that a truck can do when it delivers product to some tank: $\{\lambda_0^{(i)}, \dots, \lambda_{m_i-1}^{(i)}\}$, where $\lambda_j^{(i)} = (j+1) \cdot L_i/m_i$, $j = 0, \dots, m_i - 1$.

3.2.1 States: simple approach

Following the notation introduced above, we define the state of the system s_t in a given instant t by a tuple of three tuples consisting of the positions (p_0, \dots, p_{k-1}) of each truck, the current load of each truck (l_0, \dots, l_{k-1}) and the current load of each tank, (c_0, \dots, c_{n-1}) ¹. We assume that the consumption rate of each tank is deterministic and is the same every day.

In equation (3.6) we have the real (continuous) states of the system, and in equation (3.4) its discretized version.

Real state of the system

$$s = ((p_0, \dots, p_{k-1}), (l_0, \dots, l_{k-1}), (c_0, \dots, c_{n-1})) \quad (3.3)$$

Discrete state of the system

$$s = ((p_0, \dots, p_{k-1}), (f_{\alpha_0}, \dots, f_{\alpha_{k-1}}), (h_{\beta_0}, \dots, h_{\beta_{n-1}})) \quad (3.4)$$

with $0 \leq \alpha_0, \dots, \alpha_{k-1} < m_i$, $0 \leq \beta_0, \dots, \beta_{n-1} < d_i$; and f_{α_j} , h_{β_j} are such that $l_j \in f_{\alpha_j}$ and $c_j \in h_{\beta_j}$, where j varies between 0 and $k-1$ and between 0 and $n-1$ respectively.

For the discrete case, the total number of possible states is:

$$|\mathcal{S}| = (n+1)^k \times \prod_{i=1}^k m_i \times \prod_{i=1}^n d_i \quad (3.5)$$

3.2.2 States: complex approach

In this approach actions are the same as before. Now we consider the consumption rates of the tanks as state variables, so that they could model the fact that consumption rates are not the same every day and allow them to be the predicted consumptions coming from another model.

Thus, the real state of the system in this case would be like

$$s = ((p_0, \dots, p_{k-1}), (l_0, \dots, l_{k-1}), (c_0, \dots, c_{n-1}), (q_0, \dots, q_{n-1})), \quad (3.6)$$

where q_i is the consumption of product for the next time (day) and for tank i . Concretely, it is the consumption after transitioning from state s at the end of some day d (corresponding to time $t_{d, V_{\max}}$) to some other state s' (corresponding to time $t_{d+1, 0}$) through an action a .

3.2.3 Actions


In a given state, an action consists in deciding the next location, either a shop or the depot, that each truck has to visit, and in the case of being a shop, how much product has to be delivered. Therefore, if p'_i denotes the new location where the i -th truck has to go and $\lambda^{(i)}$ the quantity of product to be unload.

$$a = ((p'_0, \dots, p'_{k-1}), (\lambda^{(0)}, \dots, \lambda^{(k-1)})) \quad (3.7)$$

¹For simplicity we omit the dependencies on t .

The dimension of the actions space \mathcal{A} when the truck loads are discretized satisfies:

$$|\mathcal{A}| \leq (n+1)^k \times \prod_{i=1}^k m_i \quad (3.8)$$

The inequality comes from the fact that not all actions are valid actions in a given state. 

3.2.4 Rewards function

As we commented in section 2.1.1, the rewards function is a scalar score that allows the agent to know how good or bad it is to take an action in a given state.

In the product delivery problem we want to solve, we can split the function of rewards in three main contributions (see eq. (3.13)):

$$R(s, a, s') = \mu_1 J_{\text{costs}} + \mu_2 J_{\text{levels}} + \sum_j \mu_{3,j} J_{\text{extra},j}, \quad \mu_i \geq 0, \quad (3.9)$$

- The first term, $J_{\text{costs}}(s, a, s')$, would be the one representing the economical costs to pay to the transport company. In particular, $J_{\text{costs}} \leq 0$.
- The second term, $J_{\text{levels}}(s')$, should be a reward function that penalizes negatively in dangerous and forbidden regions, and positively in the optimal region of stock according to Figure 3.2 and the qualitative importance of the stock levels explained in section 3.1.
- And finally a third term of extra penalties $J_{\text{extra},j}(s, a, s')$ that the agent may need to learn additional rules or prohibitions (such as forbidden actions).

Transport and unloads costs contribution

If we assume that costs are proportional to the distance travelled by trucks and to the total amount of product unloaded, we can consider a function of the form

$$J_{\text{costs}}(s, a, s') = C_{\text{costs}} \sum_{i=0}^{k-1} w_{p_i, p'_i} \cdot \lambda^{(i)}, \quad (3.10)$$

for some constant C_{costs} that makes J_{costs} be dimensionless.

Levels of stock contribution

If $x^{(i)}(s')$ denotes the fraction between the current load c_i of tank i with respect to its maximum capacity C_i , then we consider the following decomposition for the levels of stock contribution for the total reward function:

$$J_{\text{levels}}(s') = \sum_{i=0}^{n-1} J_{\text{levels}}^{(i)}(x^{(i)}(s')), \quad (3.11)$$

where

$$J_{\text{levels}}^{(i)}(x) = \begin{cases} P_2 & \text{if } x < a \text{ or } x > f \\ C_{ab} \exp\left(\frac{\alpha_{ab}}{x}\right) & \text{if } a \leq x \leq b \\ m_{bc}x + n_{bc} & \text{if } b < x \leq c \\ m_{cd}x + n_{cd} & \text{if } c < x \leq d \\ m_{de}x + n_{de} & \text{if } d < x \leq e \\ C_{ef} \exp\left(\frac{\alpha_{ef}}{1-x}\right) & \text{if } e < x \leq f, \end{cases}$$

and the coefficients m_{ij} , n_{ij} and C_{ij} are determined such that the functions $J_{\text{levels}}^{(i)}(x)$ look like the one in Figure 3.3. As it is detailed in Appendix B, for the suggested function there are 6 free tunable parameters: b, c, e and M, P_1, P_2 .

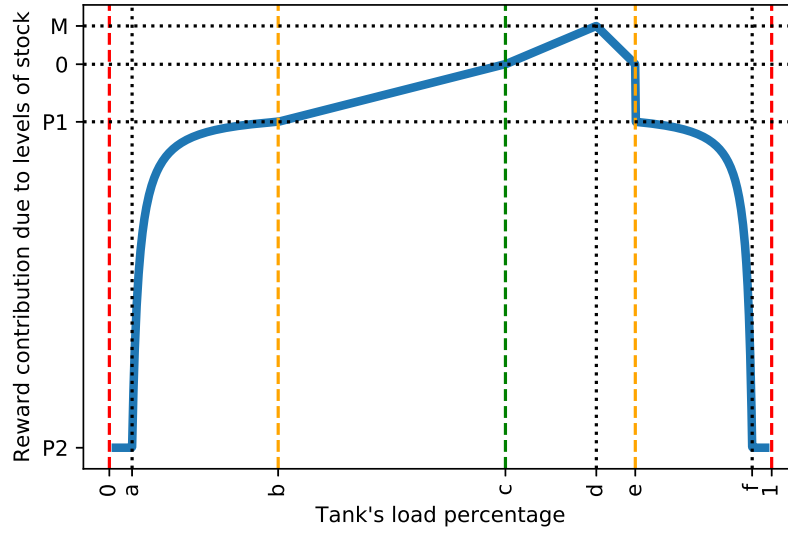


Figure 3.3: TO DO

Additional contributions

A list of examples for additional contributions on the rewards function are:

-
-

Moreover, since during the simulations the agent does not know which actions are valid or not in a given state so that it may choose non-performable actions, we need to add an extra (negative) penalty to make the agent know that this action is “bad” in some sense.

For instance, the agent may at some point try to fill up a tank **with more product than what can be fit in it**. For this reason we may add a contribution to the reward of the form:

$$J_{\text{extra, forbidden}}(s, a, s') = -C_{\text{extra, forbidden}} \sum_{i=0}^{k-1} \mathbb{1}(\text{the action performed by truck } i \text{ is forbidden}) \quad (3.12)$$

3.3 A simplified (toy) model

For the simulations and computations we will present in the next chapters, otherwise noted we are going to restrict ourselves to a simplified version of the model we have introduced so far in this chapter. We are going to focus on the first states approach described in section 3.2.1.

As it can be deduced from Figure 3.4, the first important assumption we make is that the only connections available are the ones between shops and the depot. Thus, each truck can at most visit one shop every day so that we are restricting to the case where $V_{\max} = 1$ (the maximum number of visited shops by a truck).

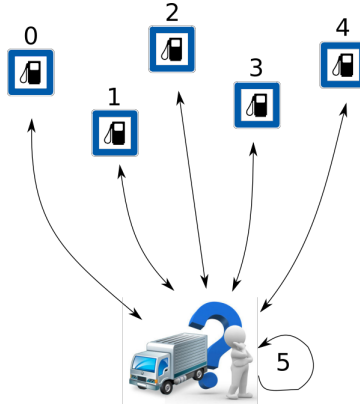


Figure 3.4: Simplified Product-delivery problem representation. Case $n = 5$ shops.

In this frame, the adjacency matrix that model our system's graph is the following:

$$A_G = \begin{pmatrix} 0_{n \times n} & 1_n \\ 1_n^T & 1 \end{pmatrix}$$

where $0_{n \times n}$ is a matrix with n rows and n columns, and 1_n is a column vector with n components equal to 1.

The first assumption we considered in section 3.1, requires that if a truck visits a shop, it must hold that all the capacity of the truck fits in that shop when it is being visited since that truck must unload all its carried product. For these reason we should consider a term on the reward function such as the one from equation (3.12) to help the agent avoid these forbidden actions.

Moreover, for now we forget about the transport and unload costs contribution to the reward function and just focus on the contribution made by the levels criteria to see if the reinforcement learning algorithms we will present are able to learn how to maintain the shops alive, or better around the optimal level of stock.

In short, the function of rewards for the simplified model version that we will use to perform most of the simulations has the following form:

$$R(s, a, s') = \mu_2 J_{\text{levels}}(s') + \mu_{3, \text{forbidden}} J_{\text{extra, forbidden}}(s, a, s') \quad (3.13)$$

where the two terms are given in equations (3.11) and (3.12), respectively.

Chapter 4

Classical Reinforcement Learning

4.1 The Q-learning algorithm

In practice, the value functions we introduced in section 2.1.4 are learned through the agent's interaction with the environment. If we focus on the Q-values, the popular algorithms that have been used more are Q-learning [14, 15] and SARSA ¹ [16] from the so-called *temporal difference approach* ², which are typically used when the model of the environment, i.e. the pair (T, R) , is unknown.

Assuming we can simulate the system in `n_episodes` episodes of length τ by means of some exploitation-exploration strategy for selecting actions in a given state, this leads to a set of simulated episodes E_k that we can write as

$$E_k = (s_0^k, a_0^k, r_0^k, s_1^k, a_1^k, r_1^k, \dots, s_{\tau-1}^k, a_{\tau-1}^k, r_{\tau-1}^k, s_{\tau}^k)$$

for $k \in \{1, \dots, \text{n_episodes}\}$.

The most exploitative action-selection criteria would be the greedy one, which would consist on using equation (2.5) for the current learned Q -values if the current state is known (and otherwise chose an action randomly). The most exploratory criteria would be the one which selects an action completely randomly in every step.

An interesting compromise between the two action-selection extremes is the ε -greedy policy, we denote π_ε , which is widely used in practice [?, ?, ?, ?], and it is defined as follows:

$$\pi_\varepsilon(s) = \begin{cases} \text{random action from } \mathcal{A}(s) & \text{if } p < \varepsilon \\ \arg \max_{a \in \mathcal{A}(s)} Q(s, a) & \text{otherwise,} \end{cases} \quad (4.1)$$

where $p \in [0, 1]$ is a uniform random number drawn at each time step (of each episode).

Policy (4.1) executes the greedy policy (2.5) with probability $1 - \varepsilon$ and the random policy with probability ε . The use of this combination of policies gives a balance between exploration and exploitation that both guarantees convergence and often good performance [20].

In Algorithm 1 we present the pseudocode for Q-learning which is hopeful self-explanatory. There is only one parameter we have not talked about yet which is the learning rate α :

¹Which stands for State-Action-Reward-State-Action)

²See for example [17, 7] for an academic point of view, or the original papers [18, 19].

The learning rate or step size determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing, while a factor of 1 makes the agent consider only the most recent information. In fully deterministic environments, a learning rate of $\alpha = 1$ is optimal. When the problem is stochastic, the algorithm converges under some technical conditions on the learning rate that required it to decrease to zero (see Theorem 4.1) [10]. In principle the learning rate can be considered a function time step, the current state and the current chosen action.

Algorithm 1 Q-learning (train) algorithm

```

1: procedure Q-LEARNING(  $\gamma, \alpha(a, s), \tau, \mathbf{n\_episodes}$ )
2:    $Q \leftarrow 0$  ▷ Initialise  $Q(s, a)$  for all  $(s, a) \in \mathcal{S} \times \mathcal{A}$ .
3:   for  $k \in \{1, \dots, \mathbf{n\_episodes}\}$  do
4:      $s_0^k \leftarrow \text{Random\_choice}(s \in \mathcal{S})$  ▷ The system is initialized to some initial state randomly
5:     for  $t \in \{0, \dots, \tau - 1\}$  do
6:       Choose  $a_t^k \in \mathcal{A}(s_t^k)$  based on the current  $Q$  and an exploration strategy (e.g.  $\pi_\epsilon$ ).
7:       Perform action  $a_t^k$ .
8:       Observe the new state  $s_{t+1}^k$  and the received reward  $r_t^k$ .
9:       Update  $Q$  with the following rule:

$$Q(s_t^k, a_t^k) \leftarrow Q(s_t^k, a_t^k) + \alpha_t^k(s_t^k, a_t^k) \left[ r_t^k + \gamma \max_{a \in \mathcal{A}(s_{t+1}^k)} Q(s_{t+1}^k, a) - Q(s_t^k, a_t^k) \right]$$

10:    end for
11:  end for
12: end procedure

```

For when the problem we are dealing with is not completely deterministic, the following theorem gives us a criteria to ensure that the Q-learning algorithm will converge as long as the learning rate satisfies some conditions.

Theorem 4.1. *Given a finite MDP $(\mathcal{S}, \mathcal{A}, T, R)$ such that \mathcal{S}, \mathcal{A} are finite, $\gamma \in (0, 1)$ and R is deterministic and bounded, the Q-learning algorithm given by the update rule*

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) \left[r_t + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (4.2)$$

converges with probability 1 to the optimal Q-value function as long as

$$\sum_{t=0}^{\infty} \alpha_t(s, a) = +\infty, \quad \sum_{t=0}^{\infty} \alpha_t(s, a)^2 < +\infty, \quad 0 \leq \alpha_t(s, a) < 1 \quad \text{for all } (s, a) \in \mathcal{S} \times \mathcal{A} \quad (4.3)$$

Condition (4.3) requires that all state-action pairs are visited infinitely often.

Proof. The proof of this theorem can be found for example in [21]. □

Example 4.1. The conditions for the learning rates that appear in Theorem 4.1; for the case of an episodic task they can be simply rewritten as

$$\sum_{k=0}^{\infty} \sum_{t=0}^{\tau-1} \alpha_t^k(s, a) = +\infty, \quad \sum_{k=0}^{\infty} \sum_{t=0}^{\tau-1} \alpha_t^k(s, a)^2 < +\infty, \quad 0 \leq \alpha_t^k(s, a) < 1 \quad \text{for all } (s, a) \in \mathcal{S} \times \mathcal{A} \quad (4.4)$$

where the index k denotes the episode.

As an example, consider a learning rate that decreases with the number of episodes but does not depend on t (when fixing an episode), neither on the state nor the action. For example, take $\alpha_t^k = \frac{\alpha_0}{1+\beta k}$, where $\alpha_0, \beta \in (0, 1)$. We call α_0 the initial learning rate, and β the learning rate decay.

Note that α_t^k satisfies the conditions in (4.4) since:

$$\sum_{k=0}^{\infty} \sum_{t=0}^{\tau-1} \frac{\alpha_0}{1+\beta k} \sim \sum_{k=0}^{\infty} \frac{1}{1+k} \sim \sum_{n=1}^{\infty} \frac{1}{n} = +\infty,$$

and

$$\sum_{k=0}^{\infty} \sum_{t=0}^{\tau-1} \left(\frac{\alpha_0}{1+\beta k} \right)^2 \sim \sum_{k=0}^{\infty} \frac{1}{(1+k)^2} \sim \sum_{n=1}^{\infty} \frac{1}{n^2} < +\infty.$$

4.2 Simulations

4.3 Results

Chapter 5

Neural Networks and Deep Learning

5.1 Introduction to Artificial Neural Networks

5.2 Training of Deep Neural Networks

5.3 Learning a simple policy with a Deep Neural Network

- Mention tensorflow library somewhere [22]
- Open AI gym environment (explicar que hemos hecho uno para los algoritmos de reinforcement learning que simulamos, todos menos los del Q-learning que hice un código a parte) [23]

Chapter 6

Deep Reinforcement Learning

6.1 Policy Gradient algorithms

6.2 Simulations

6.3 Results

Chapter 7

(? En caso que hiciésemos algo más como lo de separar en k softmaxes y aprendiese, o bien rutas, etc.)

Chapter 8

Conclusions and Future Work

Bibliography

- [1] Daniel Salgado Rojo. Rl-project. <https://github.com/dsalgador/RL-Project>, 2018.
- [2] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.
- [3] Craig Boutilier, Thomas L. Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *CoRR*, abs/1105.5460, 2011.
- [4] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997.
- [5] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS’99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.
- [6] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.
- [7] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [8] Wikipedia contributors. Markov decision process — wikipedia, the free encyclopedia, 2018. [Online; accessed 18-March-2018].
- [9] Wikipedia contributors. Reinforcement learning — wikipedia, the free encyclopedia, 2018. [Online; accessed 18-March-2018].
- [10] Wikipedia contributors. Q-learning — wikipedia, the free encyclopedia, 2018. [Online; accessed 20-March-2018].
- [11] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML’14, pages I–387–I–395. JMLR.org, 2014.
- [12] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.
- [13] Scott Proper and Prasad Tadepalli. Scaling model-based average-reward reinforcement learning for product delivery. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 735–742, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

- [14] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.
- [15] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King’s College, Cambridge, UK, May 1989.
- [16] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994.
- [17] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O’Reilly Media, Sebastopol, CA, 6 edition, 2017.
- [18] R. S. Sutton and A. G. Barto. Time-derivative models of Pavlovian reinforcement. In J. W. Moore and M. Gabriel, editors, *Learning and Computational Neuroscience*, pages 497–537. MIT Press, Cambridge, MA, 1990.
- [19] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.
- [20] Eyal Even-Dar and Yishay Mansour. Convergence of optimistic and incremental q-learning. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS’01, pages 1499–1506, Cambridge, MA, USA, 2001. MIT Press.
- [21] Francisco S. Melo. Convergence of q-learning: a simple proof. <http://users.isr.ist.utl.pt/~mtjspaam/readingGroup/ProofQlearning.pdf>.
- [22] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Appendix A

A Product Delivery gym environment

Appendix B

Derivation of $J_{\text{levels}}^{(i)}$