POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master of Science in Computer Engineering (Software Career)

Master Thesis

Deep Reinforcement Learning algorithms for autonomous systems

Design and implementation of a control system for autonomous driving task of a small robot, exploiting state-of-the-art Model-Free Deep Reinforcement Learning algorithms

Supervisors
prof. Pietro Michiardi
prof. Elena Baralis

Candidate
Piero Macaluso
matricola: s252894

Academic year TODO (Macaluso P.): 2019-2020

Abstract

TODO (Macaluso P.): Abstract is the last thing to do

Acknowledgements

TODO (Macaluso P.): Acknowledgements must be prepared!

Contents

Chapter 1

Introduction

Autonomous systems and in particular self-driving for unsupervised robots and vehicles (e.g. self-driving cars) are becoming more and more integral part of human lives. This topic attracted much attention from both the research community and industry, due to its potential to radically change mobility and transport. In general, most approaches to date focus on formal logic methods, which define driving behaviour in annotated geometric maps. These methods can be challenging to scale, as they rely heavily on an external mapping infrastructure rather than using and understanding the local scene, leaving fully autonomous driving in a real urban environment an essential but elusive goal.

In order to make autonomous driving a truly ubiquitous technology, in this thesis we focus on systems which address the ability to drive and navigate in the absence of maps and explicit rules, relying – just like humans do – on a comprehensive understanding of the immediate environment while following simple high-level directions (e.g. turn-by-turn route commands). Recent work in this area has demonstrated that this is possible on rural country roads, using GPS for coarse localization and LIDAR to understand the local scene [1].

The majority of the approaches adopted to exploit the local scene to learn how to drive, concentrate on deterministic algorithms to recognize the surroundings and select the right action (e.g. lane following problem on well-marked structured highways). However, these methods, like the previous ones, are not able to generalize proficiently in a different environment because of their deterministic nature.

Recently, Reinforcement Learning (RL) – a machine learning subfield focused on solving Markov decision process (MDP), where an agent learns to select actions in an environment in an attempt to maximize some reward function – has been shown to achieve super-human results at games such as Go [2] or chess [3], to be particularly suited for simulated environments like computer games [4], and to be a promising methodology for simple tasks with robotic manipulators [5].

Furthermore, the great fervour produced by the widespread exploitation of Deep Learning opened the doors to function approximation with neural networks

and convolutional neural networks, developing what is nowadays known as Deep Reinforcement Learning.

TODO (Macaluso P.): Continue from here

In this thesis, we argue that the generality of RL makes it a useful framework to apply to autonomous driving. For this reason, we design and implement a control system for an autonomous driving task with the small robot Cozmo by Anki [6] on which to exploit state-of-the-art model-free deep reinforcement learning algorithms and discussing possible ways to make them data efficient.

1.1 Motivation

1.2 Structure of the thesis

TODO (Macaluso P.):

- Brief Description of Every Chapter

The aim of this section is to describe the main structure of the thesis.

Chapter 1 - Introduction

The current chapter contains the motivation of this work and the structure of the thesis.

Chapter 2 - Reinforcement Learning Fundamentals

The aim of this chapter is to present a description as detailed as possible about RL state-of-the-art in order to provide the reader with useful tools to enter in this research field. TODO (Macaluso P.): Da qui in poi questo capitolo è da fare

Chapter 3 - Tools and Frameworks

This chapter explains briefly what are the main tools, frameworks and languages used in the thesis. TODO (Macaluso P.): Continue this list

OpenAI Gym framework for developing and comparing Reinforcement Learning algorithms and environments using standardize interface.

Anki Cozmo it looks like a simple toy at first sight, but it hides an infinite potential under the hood, which make it a perfect candidate for the purposes of this thesis.

Chapter 4 - Design of the control system

Chapter 5 - Algorithms for Autonomous Systems

Chapter 5 - Experiments

This is the most important chapter. It shows all the results obtained during the numerous experiments with comments and speculations about them.

Chapter 6 - Conclusions

A summary of the results obtained from experiments with a specific part dedicated to future improvements.

Chapter 2

Reinforcement Learning

Reinforcement Learning (RL) is a field of Machine Learning that is experiencing a period of great fervour in the world of research, fomented by recent progress in Deep Learning (DL). This event opened the doors to function approximation with Neural Network (NN) and Convolutional Neural Network (CNN) developing what is nowadays known as Deep Reinforcement Learning.

RL represents the third paradigm of Machine Learning alongside supervised and unsupervised learning. The idea behind this research field is that the learning process to solve a decision-making problem consists in a sequence of trial and error where the *agent*, the protagonist of RL, could discover and discriminate valuable decisions from penalising ones exploiting information given by a *reward signal*. This interaction has a strong correlation with what human beings and animals do in the real world to forge their behaviour.

Recently RL has known a remarkable development and interests in video games: it managed to beat world champions at the game of Go [2] and Dota with superhuman results and to master numerous Atari video games [4] from raw pixels. Decisions, actions and consequences make video games a simulated reality on which to exploit and test the power of RL algorithms. It is essential to realise that the heart of RL is the science of decision making. This fact makes it compelling and general for many research fields ranging from Engineering, Computer Science, Mathematics, Economics, to Psychology and Neuroscience.

Before discussing the results of this thesis, it is good to clarify everything that today represents the state-of-the-art in order to understand the universe behind this new paradigm better. Indeed, the exploration of this field of research is the main aim of this chapter: the first section begins with the definition of the notation used and with the theoretical foundations behind RL, then in the second section it moves progressively towards what is Deep RL through a careful discussion of the most important algorithms paying more attention to those used during the thesis project.

The elaboration of this chapter is inspired by [7], [8], [9], [10].

2.1 Fundamentals of Reinforcement Learning

Reinforcement Learning is a computational approach to Sequential Decision Making. It provides a framework that is exploitable with decision-making problems that are unsolvable with a single action and need a sequence of actions, a broader horizon, to be solved.

This section aims to present the fundamental ideas and notions behind this research field in order to help the reader to develop a baseline useful to approach section 2.2 on page 16 about Deep Reinforcement Learning.

2.1.1 The Reinforcement Learning Problem

The primary purpose of RL algorithms is to learn how to improve and maximise a future reward by relying on interactions between two main components: the agent and the environment.

The *agent* is the entity that interacts with the environment by making decisions based on what it can observe from the state of the surrounding situation. The decisions taken by the agent consist of *actions* (a_t). The agent has no control over the environment, but actions are the only means by which it can modify and influence the environment.

Usually, the agent has a set of actions it can take, which is called *action space*. Some environments have discrete action spaces, where only a finite number of moves are available (e.g. $\mathcal{A} = [\text{North}, \text{South}, \text{East}, \text{West}]$ choosing the direction to take in a bidimensional maze). On the other side, there are continuous action spaces where actions are vectors of real values. This distinction is fundamental to choose the right algorithm to use because not all of them could be compatible with both types: according to the needs of the specific case, it may be necessary to modify the algorithm to make it compatible.

The *environment* represents all the things that are outside the agent. At every action received by the agent, it emits a reward, an essential aspect of RL, and an observation of the environment.

The *reward* r_t is a scalar feedback signal that defines the objective of the RL problem. This signal allows the agent to be able to distinguish positive actions from negative ones in order to reinforce and improve its behaviour. It is crucial to notice that the reward is local: it describes only the value of the latest action. Furthermore, actions may have long term consequences, delaying the reward. As it happens with human beings' decisions, receiving a conspicuous reward at a specific time step does not exclude the possibility to receive a small reward immediately afterwards and sometimes it may be better to sacrifice immediate reward to gain more rewards later.

In this context, many features make RL different from supervised and unsupervised learning. Firstly, there is no supervisor: when the agent has to decide what action to

take, there is no entity that can tell him what the optimal decision is in that specific moment. The agent receives only a reward signal which may delay compared to the moment in which it has to perform the next action. This fact brings out another significant difference: the importance of time. The sequentiality links all actions taken by the agent, making resulting data no more indipendent and identically distributed (i.i.d.).

Given these definitions, it is noticeable that the primary purpose of the agent is to maximise the cumulative reward called *return*.

The *return* g_t is the total discounted reward starting from timestep t defined by eq. (2.1) where γ is a *discount factor*.

$$g_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad \gamma \in [0,1) \tag{2.1}$$

Not only the fact that animal and human behaviour show a preference for immediate rewards rather than for the future ones motivates the presence of this factor, but it is also mathematically necessary: an infinite-horizon sum of rewards may not converge to a finite value. Indeed, the return function is a geometric series, so, if $\gamma \in [0,1)$, the series converges to a finite value equal to $1/(1-\gamma)$. For the same convergence sake, the case with $\gamma = 1$ makes sense only with a finite-horizon cumulative discounted reward.

The other data emitted by the environment is the *observation* (o_t) that is related to the *state* (s_t). It represents a summary of information that the agent uses to select the next action, while the *state* is a function of the *history* the sequence of observation, actions and rewards at timestep t as shown in eq. (2.2).

$$h_t = o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t, \qquad s_t = f(h_t) \tag{2.2}$$

The sequence of states and actions is named *trajectory* (τ): it is helpful to represent an episode in Reinforcement Learning (RL) framework.

The state described above is also called *agent state* s_t^a, while the private state of the environment is called *environment state* s_t^e. This distinction is useful for distinguishing fully observable environments where $o_t = s_t^e = s_t^a$, from partially observable environments where $s_t^e \neq s_t^a$. In the first case, the agent can observe the environment state directly, while in the second one, it has access to partial information about the state of the environment.

Beyond the fact that this chapter will focus on fully observable environments, the distinction between state and observation is often unclear and, conventionally, the input of the agent is composed by the reward and the state as shown in fig. 2.1 on the next page.

Furthermore, a state is called *informational state* (or *Markov state*) when it contains all data and information about its history. Formally, a state is a Markov state if and only if satisfies eq. (2.3) on the following page.

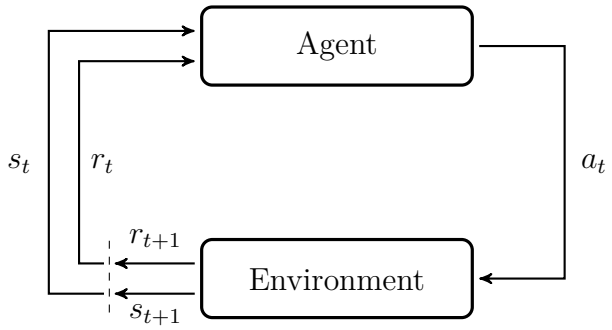

Figure 2.1. Interaction loop between Agent and Environment. The reward and the state resulting from taking an action become the input of the next iteration.

$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \ldots, s_t] \tag{2.3}$$

It means that the state contains all data and information the agent needs to know to make decisions: the whole history is not useful anymore because it is inside the state. The environment state s_t^e is a Markov state.

With all the definitions shown so far, it is possible to formalise the type of problems on which RL can unleash all its features: the Markov decision process (MDP), a mathematic framework to model decision processes. Its main application fields are optimization and dynamic programming.

An MDP is defined by

$$< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$$

where $\mathcal{S}$ is a finite set of states
$\mathcal{A}$ a finite set of actions
$\mathcal{P}$ a state transition probability matrix $\mathcal{P}^a_{ss'} = \mathbb{P}[s_{t+1} = s'|s_t = s, a_t = a]$
$\mathcal{R}$ a reward function $\mathcal{R}^a_s = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$
γ a discount factor such that $\gamma \in [0,1]$

(2.4)

The main goal of an MDP is to select the best action to take, given a state, in order to collect the best reward.

In this quick overview of the main unit of RL, the components that may compose the agent, the brain of the RL problem can not be missing: they are the *model*, the *policy* and the *value function*.

A *model* is composed by information about the environment. These data must not be confused with the ones provided by *states* and *observations*: they make it possible to infer prior knowledge about the environment, influencing the behaviour of the agent.

A *policy* is the core of RL because it is the representation of the agent's behaviour. It is a function that describes the mapping from states to actions. The *policy* is represented by π and it may be deterministic $a_t = \pi(s_t)$ or stochastic $\pi(a_t|s_t) = \mathbb{P}[a_t|s_t]$.

In this perspective, it is evident that the central goal of RL is to learn an optimal policy π^*. The optimal policy is a policy which can show agent what the most profitable way to achieve the maximum return is, what is the best action to do in a specific situation. In order to learn the nature of the optimal policy, RL exploits value functions.

A *value function* represents what is the expected reward that the agent can presume to collect in the future, starting from the current state. The reward signal represents only a local value of the reward, while the value function provides a broader view of future rewards: it is a sort of prediction of rewards.

It is possible to delineate two main value functions: the *state value* function and the *action value* function.

- The *State Value Function* $V^\pi(s)$ is the expected return starting from the state s and always acting according to policy π.

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[g_t|s_0 = s] \tag{2.5}$$

- The *Action Value Function* $Q^\pi(s)$ is the expected return starting from the state s, taking an action a and then always acting according to policy π.

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[g_t|s_0 = s, a_0 = a] \tag{2.6}$$

2.1.2 Approaches of Reinforcement Learning

It is possible to explain the main strategies in RL to solve problems using *policy*, *model* and *value function* defined previously.

Every agent has a specific application field which depends on the different approach it supports. Understanding differences among these approaches is useful to adequately understand what type of algorithm satisfies better the needs of a specific context.

The distinctions presented in this part are just a part of the complete set because this section aims to describe the most crucial distinctions that are useful in the context of the thesis without claiming to be exhaustive.

Model-Free vs Model-Based

One of the most crucial aspects of an RL algorithm is the question of whether the agent has access to (or learns) a model of the environment. A model of the environment enables the agent to predict state transitions and rewards.

A method is *model-free* when it does not build a model of the environment. All the actions made by the agent results from direct observation of the current situation in which the agent is. It takes the observation, does computations on them and then select the best action to take.

This last representation is in contrast with *model-based* methods. In this case, the agent tries to build a model of the surrounding environment in order to infer information useful to predict what the next observation or reward would be.

Both groups of methods have strong and weak sides. Ordinarily, *model-based* methods show their potential in a deterministic environment (e.g. board game with rules). In these contexts, the presence of the model enables the agent to plan by reasoning ahead, to recognise what would result from a specific decision before taking action. The agent can extract all this knowledge and learn an optimal policy to follow. However, this opportunity is not always achievable: the model may be partially or entirely unavailable, and the agent would have to learn the model from its experience. Learning a model is radically complex and may lead to various hurdles to overcome: for instance, the agent can exploit the bias present in the model, producing an agent which is not able to generalise in real environments.

On the other hand, model-free methods tend to be more straightforward to train and tune because it is usually hard to build models of a heterogeneous environment. Furthermore, model-free methods are more popular and have been more extensively developed and tested than model-based methods.

Policy-Based vs Value-Based

The use of policy or value function as the central part of the method represents another essential distinction between RL algorithms.

The approximation of the policy of the agent is the base of *policy-based* methods. The representation of the policy is usually a probability distribution over available actions. This method points to optimise the behaviour of the agent directly and, because of its on-policy nature, may ask manifold observations from the environment: this fact makes this method not so sample-efficient.

On the opposite side, methods could be *value-based*. In this case, the agent is still involved in finding the optimal behaviour to follow, but indirectly. It is not interested anymore about the probability distribution of actions. Its main objective is to determine the value of all actions available, choosing the best value. The main difference from the policy-based method is that this method can benefit from other sources, such as old policy data or replay buffer.

On-Policy vs Off-Policy

It is possible to classify this method also by different types of policy usage.

An *off-policy* method can use a different source of valuable data for the learning

process instead of the direct experience of the current policy. This feature allows the agent to use, for instance, large experience buffers of past episodes. In this context, these buffers are usually randomly sampled in order to make the data closer to being independent and identically distributed (i.i.d): random extraction guarantees this fact.

On the other hand, *on-policy* methods profoundly depend on the training data to be sampled according to the current policy.

2.1.3 Bellman Equations

Both eqs. (2.5) and (2.6) on page 8 satisfy recursive relationships between the value of a state and the values of its successor states. It is possible to see this property deriving *Bellman equations* – shown in eq. (2.7) and demonstrated in appendix A.1 on page 25 – where $s_{t+1} \sim E$ means that the next state is sampled from the environment E and $a_{t+1} \sim \pi$ shows that the next action is taken following the policy π.

$$\begin{aligned} V^\pi(s_t) &= \mathbb{E}_{a_t\sim\pi, s_{t+1}\sim E}[r(s_t,a_t) + \gamma V^\pi(s_{t+1})] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} P(s',r|s,a)\Big[r + \gamma V^\pi(s')\Big] \\ Q^\pi(s_t,a_t) &= \mathbb{E}_{s_{t+1}\sim E}[r(s_t,a_t) + \gamma \mathbb{E}_{a_{t+1}\sim\pi}[Q^\pi(s_{t+1},a_{t+1})]] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} P(s',r|s,a)\Big[r + \gamma Q^\pi(s',a')\Big] \end{aligned} \tag{2.7}$$

$r(s_t, a_t)$ is a placeholder function to represent the reward given the starting state and the action taken. As discussed above, the goal is to find the optimal policy π^* to exploit. It can be done using *Bellman optimality equations* defined in eq. (2.8).

$$\begin{aligned} V^*(s_t) &= \max_a \mathbb{E}_{s_{t+1}\sim E}[r(s_t,a) + \gamma V^*(s_{t+1})] \\ &= \max_a \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} P(s',r|s,a)\Big[r + \gamma V^*(s')\Big] \\ Q^*(s_t,a_t) &= \mathbb{E}_{s_{t+1}\sim E}[r(s_t,a_t) + \gamma \max_{a'}[Q^*(s_{t+1},a')]] \\ &= \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} P(s',r|s,a)\Big[r + \gamma \max_{a'} Q^*(s',a')\Big] \end{aligned} \tag{2.8}$$

Therefore, value functions allow defining a partial ordering over policies such that

$$\pi \geq \pi' \text{ if } V_\pi \geq V_{\pi'}, \forall s \in \mathcal{S}$$

This definition is helpful to enounce the *Sanity Theorem*. It asserts that for any MDP there exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall\pi$, but also that all optimal policies achieve the optimal state value function and the optimal action-value function.

2.1.4 Dynamic Programming

Dynamic programming is one of the approaches used to solve RL problems calculation the optimal policy π^*. Formally, it is a general method to solve complex problems by breaking them into sub-problems that are more convenient to solve. After solving all sub-problems, it is possible to sum them up in order to obtain the final solution to the whole original problem.

This technique provides a practical framework to solve MDP problems and to observe what is the best result achievable from it, but it assumes to have full knowledge about the specific problem. For this reason, it applies primarily to model-based problems.

This thesis will not focus on this type of approaches, so this section aims to present only the basic concept of *policy iteration* and *value iteration* which are worth quoting: [8, Chapter 4] provides further details about them.

Policy Iteration

The *policy iteration* aims to find the optimal policy by directly manipulating the starting policy. However, before proceeding with this process, a proper evaluation of the current policy is essential. This procedure can be done iteratively following algorithm 2.1 on the next page where θ is the parameter that defines the accuracy: the more the value is closer to 0, the more the evaluation would be precise.

Policy improvement represents the second step towards policy iteration. Intuitively, it is possible to find a more valuable policy than the starting one by changing the action to take in a specific state with a more rewarding one. The key to check if the new policy is better than the previous one is to use the action-value function $Q_\pi(s, a)$. This function returns the value of taking action a in the current state s and, after that, following the existing policy π. If $Q_\pi(s, a)$ is higher than $V_\pi(s)$, so the action selected is better than the action chosen by the current policy, and consequently, the new policy would be better overall.

Policy improvement theorem is the formalisation of this fact: appendix A.2 on page 26 shows its demonstration. Thanks to this theorem, it is reasonable to act greedily to find a better policy starting from the current one iteratively selecting the action that produces the higher $Q_\pi(s, a)$ for each state.

The iterative application of policy improvement stops after an improvement step that does not modify the initial policy, returning the optimal policy found.

Value Iteration

The second approach used by Dynamic Programming to solve Markov Decision Processes is *value iteration*. Policy iteration is an iterative technique that alternate evaluation and improvement until it converges to the optimal policy. On the contrary, value iteration uses a modified version of policy evaluation to determine

Algorithm 2.1: Policy Iteration for estimating $\pi \sim \pi^*$

Input: π the policy to be evaluated; a small threshold θ which defines the accuracy of the estimation

Initialise $V(s)\ \forall s \in \mathcal{S}$ arbitrarily, except that $V(terminal) = 0$
$is\_policy\_stable \leftarrow true$
repeat
  /* Policy Evaluation */
  repeat
    $\Delta \leftarrow 0$
    for *each* $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} P(s', r|s, a)\Big[r + \gamma V(s')\Big]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
  until $\Delta < \theta$
  $V_\pi \leftarrow V(s)$
  /* Policy Improvement */
  while *true* **do**
    for *each* $s \in \mathcal{S}$ **do**
      $old\_action \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} P(s', r|s, a)\Big[r + \gamma V_\pi(s')\Big]$
      if $old\_action \neq \pi(s)$ **then**
        $is\_policy\_stable \leftarrow false$
      end
    end
  end
until $\neg\ is\_policy\_stable$

Output: V^* and π^*

$V(s)$ and then it calculates the policy. The pseudocode of this method is available algorithm 2.2 on the next page.

2.1.5 Model-Free Prediction and Control

As reported in the previous section, having a comprehensive knowledge of the environment is at the foundation of dynamic programming methods. However, this fact is not always accurate in the real world and owning a full understanding of surroundings is almost a utopia. In these cases, the agent has to infer the model using its experience, so it has to exploit model-free methods, based on the

Algorithm 2.2: Value Iteration, for estimating $\pi \sim \pi^*$

Input: A small threshold θ which defines the accuracy of the estimation

Initialise $V(s)\ \forall s \in \mathcal{S}$ arbitrarily, except that $V(terminal) = 0$

repeat

  $\Delta \leftarrow 0$

  for *each* $s \in \mathcal{S}$ **do**

    $\mathrm{v} \leftarrow V(s)$

    $V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} P(s', r|s, a)\Big[r + \gamma V(s')\Big]$

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

  end

until $\Delta < \theta$

Output a deterministic policy, $\pi \sim \pi^*$, such that

$$\pi(s) = \arg\max_a \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} P(s', r|s, a)\Big[r + \gamma V(s')\Big]$$

Output: V^* and π^*

assumption that there is no knowledge about state transitions and rewards. These methods can be used with any reinforcement learning problem because they do not have the model of the environment as a requirement.

Model-free approaches are divided into *value-function-based methods* and *policy-based methods*. The learning of the value function to approximate and infer an optimal policy characterises the first type, while the second one directly seeks the optimal policy using the space of policy parameters.

This section intends to provide a brief description of two model-free approaches to prediction and control: Monte Carlo (MC) methods and Temporal-Difference (TD) ones.

Monte Carlo learning

Monte Carlo methods [8, Chapter 5] can learn from episodes of experience using the simple idea that averaging sample returns provide the value. Generalised policy iteration (GPI) is the central concept underlying these methods: it is an iterative scheme borrowed by dynamic programming based on the interaction of two main steps. The first step, known as policy evaluation step, exploits the current policy to build an approximation of the value function. The second step, known as policy improvement step, tries to improve the policy starting from the current value function. Monte Carlo methods do not use a model to compute the value of each state: they average many returns that start in the state. This lead to the main caveat of these methods: they work only with episodic MDPs because the

episode has to terminate before it is possible to calculate any returns. The total reward accumulated in an episode and the distribution of the visited states is used to calculate the value function while the improvement step is carried out by making the policy greedy concerning the value function. Therefore, these approaches combine policy evaluation and policy improvement steps on an episode-by-episode basis: these two processes compete and cooperate to find the optimal value function and an optimal policy.

Unfortunately, even though Monte Carlo methods are simple to implement, they require a high number of iteration to converge, and they have a wide variance in their value function estimation.

Temporal Difference learning

The idea of the GPI is also at the base of Temporal Difference (TD) [8, Chapter 6] methods, but with proper differences in respect of Monte Carlo methods. The central distinction between these two methods is that TD methods calculate a temporal error instead of using the total accumulated reward. The temporal error is the difference between the new estimate of the value function and the old one. Furthermore, they calculate this error considering the reward received at the current time step and use it to update the value function: this means that these approaches can work with continuing (non-terminating) environments. This type of update reduces the variance compared to Monte Carlo one but increases the bias in the estimate of the value function because of bootstrapping: it means that these strategies use one or more estimated values in the update step for the same kind of estimated value, leading to results more sensitive results initial values.

The fundamental update equation for the value function is shown in eq. (2.9), where *TD error* and *TD target* are in evidence.

$$V(s_t) \leftarrow V(s_t) + \alpha\Big(\overbrace{\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{TD target}} - V(s_t)}^{\text{TD error } (\delta_t)}\Big) \tag{2.9}$$

Two TD algorithms for the control problem which are worth quoting because of their extensive use to solve RL problems are *SARSA (State-Action-Reward-State-Action)* and *Q-Learning*.

SARSA is an *on-policy* temporal difference algorithm whose first step is to learn an *action-value* function instead of a *state-value* function. This approach leads to focus not to estimate the specific value of each state, but to determine the value of transitions and state-action pairs. Equation (2.10) represents the update function of *SARSA*, while algorithm 2.3 on the following page summarise its pseudocode.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.10}$$

Algorithm 2.3: SARSA (on-policy TD control) for estimating $Q \approx q_*$

Input: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Initialise $Q(s,a)\ \forall\ s \in \mathcal{S}, a \in \mathcal{A}$ arbitrarily, except that $Q(\text{terminal},\cdot) = 0$
foreach *episode* **do**
  Initialise s_t
  Choose a_t from s_t using policy derived from Q (e.g. ϵ-greedy)
  repeat
    Take action $a_t \rightarrow$ obtain r_{t+1} and s_{t+1}
    Choose a_{t+1} from s_{t+1} using policy derived from Q (e.g. ϵ-greedy)
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
    $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
  until s_t *is terminal*
end

Q-learning [11] is an off-policy TD control algorithm which represents one of the early revolution and advance in reinforcement learning. The main difference from SARSA is the update rule for the Q-function: it selects the action in respect of an ϵ-greedy policy while the Q-function is refreshed using a greedy policy based on the current Q-function using a max function to select the best action to do in the current state with the current policy.

Equation (2.11) represents the update function of *Q-learning*, while algorithm 2.4 summarise its pseudocode.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)] \tag{2.11}$$

Algorithm 2.4: Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Input: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Initialise $Q(s,a)\ \forall\ s \in \mathcal{S}, a \in \mathcal{A}$ arbitrarily, except that $Q(\text{terminal},\cdot) = 0$
foreach *episode* **do**
  Initialise s_t
  Choose a_t from s_t using policy derived from Q (e.g. ϵ-greedy)
  repeat
    Take action $a_t \rightarrow$ obtain r_{t+1} and s_{t+1}
    Choose a_{t+1} from s_{t+1} using policy derived from Q (e.g. ϵ-greedy)
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$
    $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
  until s_t *is terminal*
end

Temporal Difference Lambda Learning

As reported previously, Monte Carlo and Temporal Difference learning perform updates in different ways. The first approach exploits the total reward to update the value function, while the second one, on the other hand, works with the reward of the current step. Temporal Difference Lambda, also known as TD(λ) [8, Chapter 7,12], represents a combination of these two procedures and it takes into account the results of each time step together with the weighted average of those returns. The idea of calculating TD target looking n-steps into the future instead of considering only a single step is the baseline of TD(λ). This lead to the formalisation of the λ-weighted return G_t^λ

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)} \tag{2.12}$$

TD(λ) implementation takes into account an additional variable called eligibility trace $e_t(s_t)$ which indicates how much learning should be carried out for each state for each timestep. It aims to describe how much the agent encountered a specific state recently and eq. (2.13) describes the updating rule of this value where the λ represents the trace-decay parameter.

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbb{1}(s = s_t) \tag{2.13}$$

2.1.6 Function Approximation

2.2 Deep Reinforcement Learning

TODO (Macaluso P.):

- Introduction and motivation behind function approximation
- Value-based methods and Policy Gradient methods
- Focus and detailed description of DDPG and SAC

2.2.1 Taxonomy of Deep RL Algorithm

After the quick overview of the basics of RL terminology and notation provided in the previous section, it is possible to explore more in-depth the universe behind the algorithms of modern Deep RL. Because of the nature of this work, the focus of this section will be on the types of algorithms used in the thesis, without leaving out a quick overview of other types of algorithms most used today in Deep RL.

2.2.2 Deep Deterministic Policy Gradient (DDPG)

2.2.3 Soft-Actor Critic (SAC)

2.3 Related Work

TODO (Macaluso P.):

- Explanation of the state-of-the-art focusing more on Reda's paper, its approach and the related bibliography
- Increasing interest in Reinforcement Learning applied to real-world situations, in contrast with simulated environments experiments

2.4 Summary

Chapter 3

Tools and Frameworks

TODO (Macaluso P.):

- Introduction to the chapter

3.1 OpenAI Gym

TODO (Macaluso P.):

- General description of the framework
- Discussion about the motivation and the importance of this framework, such as the necessity of a Reinforcement Learning Framework to test different algorithms with different environments providing the same interface.
- One contribution of the thesis: the creation and design of an OpenAI Gym environment for Anki Cozmo Environment with connection to chapter 4.

3.1.1 The importance of a Framework

3.1.2 Main features

3.1.3 How to create an environment

3.2 Anki Cozmo

TODO (Macaluso P.):

- General description of Cozmo and Anki
- General information about the mechanics and features of Cozmo (LINK)

- Discussion about the selection of Cozmo instead of other solutions
  - Amazon Deep Racer: not available at the start of the thesis. It provides a simulator to train the agent. Using AWS for computation which can be a benefit, but also a drawback because it is a lock-in solution.
  - Building a Car: one of the best path to follow because of the personalization available. Main drawbacks are the length of the car construction process but also the time to spend in the creation of interfaces between the car and Python.
  - In the end, Cozmo is the best trade-off between functionalities and fast-developing. It provides plain and straightforward control of the car and a rich Python SDK to use with OpenAI Gym.
- Discussion about the on-board or off-board computation

3.2.1 Features of Cozmo

3.2.2 Cozmo SDK

3.3 PyTorch

TODO (Macaluso P.):

- General description of Pytorch framework

3.3.1 Tensor and Gradients

3.3.2 Building a Convolutional Neural Network

3.3.3 Loss function and Optimizers

3.3.4 TensorboardX

Chapter 4

Design of the Control System

4.1 The main concept

4.1.1 Related Work

4.2 The track

The design and training of a good driving model cannot go beyond the construction and design of the road. For this reason, some time was spent searching the better way to build a path where to train Cozmo. This section aims to present the central concept and decisions made about the design of the track for the experiments, starting from the materials used, up to the description of the dimensional choices applied.

4.2.1 Track requirements

It is essential to explain the primary needs of the road before proceeding with the description of the choices made.

Firstly, the track needs to be easily transportable to allow various attempt with different locations and environmental conditions that could affect the training phase. In particular, it is necessary to use a material less reflective as possible to avoid problems during the learning process. Another crucial factor is the dimension of the lane, which must reproduce an environment similar to the real one. It can be done analyzing the ratio of the size of a vehicle to the width of a road. On average, a family car is about 160-170cm large, while a lane width can vary between 275cm and 375cm. Cozmo width is about 5.5cm, which results in a ratio of 1/30. Therefore, the scaled lane must be between 9cm and 12.5cm.

4.2.2 Track design and materials

The first choice to make is the one about the type of material to use as terrain for the track. The first choice was the black floor of the Data Science laboratory of Eurecom. It was useful only during the initial design and development of the control system to build small pieces of track in which testing functionalities. This solution had numerous drawbacks such as the impracticality to transport and high light reflection.

The following list provides a brief report of the various solutions taken into account during the thesis, together with an analysis of advantages and drawbacks.

- *Covering fabric*: this material is easily transportable, but it has a high light reflection, and its structure is prone to make wrinkles and dunes challenging to remove.
- *Tar paper*: this solution slightly diminished the reflection problem compared to the previous choice, but the material was fragile and with the same drawbacks of the covering fabric.
- *Cotton fabric*: this solution offers an easily transportable material with reduced light reflection where it is easy to remove wrinkles and dunes.

Summing up, it is noticeable from this analysis that the cotton fabric provides the right trade-off among all requirements reported before.

The structure of the road is also composed by the lane. The implementation of this part was done using a simple paper tape of width equal to 2.5cm. As described in the beginning of this section, the width of the lane must be between 9cm and 12.5cm to provide a context similar to the real one. Because of the narrow and limited angle of vision provided by Cozmo camera, 10cm-width was set: positioning the tape with a distance greater than 10cm would result in a great part of the tape outside the view of the camera.

4.2.3 Problems and solutions

4.3 Cozmo Control System

4.3.1 Formalization as an Markov Decision Process

4.3.2 Design of the OpenAI Gym Environment

4.3.3 Main setup of the system

Chapter 5

Experiments

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

5.1 Results of the experiments

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

5.1.1 Track A

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

5.1.2 Track B

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Chapter 6

Conclusions

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

6.1 Future Work

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Appendix A

Reinforcement Learning

A.1 Bellman Equation

TODO (Macaluso P.): Check correctness and completeness

The value function is decomposable in the immediate reward r_t and the discounted state value of the next state. It is possible to obtain the result in eq. (A.1) by writing expectations explicitly.

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}[g_t | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma g_{t+1} | s_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} P(s', r | s, a) \Big[r + \gamma \mathbb{E}[g_{t+1} | s_{t+1} = s'] \Big] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} P(s', r | s, a) \Big[r + \gamma V^{\pi}(s') \Big] \end{aligned} \tag{A.1}$$

This equation expresses the relationship between the value of a state and the values of its successor states. It is further possible to derive the Bellman Equation for Action-Value function using the same procedure described above.

The resulting formulas are shown in eq. (2.7) on page 10.

Furthermore, it is possible to obtain the Bellman Equation solution in eq. (A.2) working with matrix notation.

$$\begin{aligned} V^{\pi} &= \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} V^{\pi} \\ (I - \gamma \mathcal{P}^{\pi}) V^{\pi} &= \mathcal{R}^{\pi} \\ V^{\pi} &= (I - \gamma \mathcal{P}^{\pi})^{-1} \mathcal{R}^{\pi} \end{aligned} \tag{A.2}$$

A.2 Policy Improvement Theorem

Let π and π' be any pair of deterministic policy such that

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s) \; \forall s \in S \tag{A.3}$$

Then the policy π' leads to

$$V'_\pi(s) \geq V_\pi(s) \tag{A.4}$$

Therefore, the presence of strict inequality in eq. (A.3) for a state leads to a strict inequality of eq. (A.4).

The proof of this theorem is shown in eq. (A.5).

$$\begin{aligned}
V_\pi(s) &\leq Q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s, a_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s] \\
&\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma Q_\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s] \;\; \text{(by A.3)} \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \mathbb{E}_{\pi'}[r_{t+2} + \gamma V_\pi(s_{t+2}) | s_{t+1}, a_{t+1} = \pi'(s_{t+1})] | s_t = s] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_\pi(s_{t+2}) | s_t = s] \\
&\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V_\pi(s_{t+3}) | s_t = s] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots | s_t = s] \\
&= v_{\pi'}(s)
\end{aligned} \tag{A.5}$$

Acronyms

CNN Convolutional Neural Network. 4

DL Deep Learning. 4

i.i.d. indipendent and identically distributed. 6

MDP Markov decision process. 1, 7, 9

NN Neural Network. 4

RL Reinforcement Learning. 1, 2, 4, 5, 7, 8

Bibliography

[1] Teddy Ort, Liam Paull, and Daniela Rus. Autonomous vehicle navigation in rural environments without detailed prior maps. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2040–2047. IEEE, 2018.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[6] Anki. Github repository of Cozmo SDK written in python, 2019. https://github.com/anki/cozmo-python-sdk.

[7] David Silver. University college london course on reinforcement learning. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html, 2015.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] OpenAI. Open AI spinning up, 2019. https://spinningup.openai.com/.

[10] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[11] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.