INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Luis Guilherme Gomes Aguiar

SIMULATED HUMANOID ROBOT CONTROL WITH REINFORCEMENTE LEARNING

Final Paper
2018

Course of Electronics Engineering

CDU 621.38

Luis Guilherme Gomes Aguiar

SIMULATED HUMANOID ROBOT CONTROL WITH REINFORCEMENTE LEARNING

Advisor

Prof. Dr. Takashi Yoneyama (ITA)

Co-advisor

Prof. Dr. Marcos Ricardo Omena de A. Máximo (ITA)

ELECTRONICS ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2018

Cataloging-in Publication Data
Documentation and Information Division

Aguiar, Luis Guilherme Gomes
Simulated Humanoid Robot Control with Reinforcemente Learning / Luis Guilherme Gomes Aguiar.
São José dos Campos, 2018.
40f.

Final paper (Undergraduation study) – Course of Electronics Engineering– Instituto Tecnológico de Aeronáutica, 2018. Advisor: Prof. Dr. Takashi Yoneyama. Co-advisor: Prof. Dr. Marcos Ricardo Omena de A. Máximo.

1. Dinamica de robos. 2. Robos humanoides. 3. Controle de robos. 4. Inteligencia artificial. 5. Robotica. 6. Controle. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

AGUIAR, Luis Guilherme Gomes. **Simulated Humanoid Robot Control with Reinforcemente Learning**. 2018. 40f. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: Luis Guilherme Gomes Aguiar
PUBLICATION TITLE: Simulated Humanoid Robot Control with Reinforcemente Learning.
PUBLICATION KIND/YEAR: Final paper (Undergraduation study) / 2018

It is granted to Instituto Tecnológico de Aeronáutica permission to reproduce copies of this final paper and to only loan or to sell copies for academic and scientific purposes. The author reserves other publication rights and no part of this final paper can be reproduced without the authorization of the author.

Luis Guilherme Gomes Aguiar
Rua H8A, 131
12228-460 – SÃ£o JosÃ© dos Campos–SP

SIMULATED HUMANOID ROBOT CONTROL WITH REINFORCEMENTE LEARNING

This publication was accepted like Final Work of Undergraduation Study

---

Luis Guilherme Gomes Aguiar

Author

Takashi Yoneyama (ITA)

Advisor

Marcos Ricardo Omena de A. Máximo (ITA)

Co-advisor

Prof. Dr. Cairo Nascimento
Course Coordinator of Electronics Engineering

São José dos Campos: Junho 26, 2018.

To God, my parents and all my good friends.

Acknowledgments

make it later

Resumo

Abstract

List of Figures

List of Tables

List of Abbreviations and Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
ANN	Artificial Neural Network
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CL	Curriculum Learning
CoM	Center of Mass
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DDPG	Deep Deterministic Policy Gradients
DPG	Deterministic Policy Gradients
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
GPU	Graphics Processing Unit
KL	Kullback?Leibler
IK	Inverse Kinematics
MC	Monte-Carlo
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RPC	Remote Procedure Call
SS3D	Soccer Simulation 3D
Soccer3D	Soccer Simulation 3D
SGD	Stochastic Gradient Descent
TD	Temporal-Difference

List of Symbols

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
a_i	Element i of vector a, with indexing starting at 1
A^T	Transpose of matrix A
$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
∇y	Gradient of y
$\int_a^b f(x)dx$	Definite integral with respect to x over [a, b]
$\mathbb{I}^{\text{condition}}$	conditional unit function
$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f(x, \theta)$	A function of x parameterized by θ
$\mathbb{P}(x)$	Probability distribution over a continuous variable
$\mathbb{E}[X]$	Expectation of random variable X
$\text{KL}[P, Q]$	KL divergence between P and Q
$\log x$	Natural logarithm of x

Reinforcement Learning

$s, s\prime$	States
a	Action
r	Reward
t	Discrete timestep
A_t	Action at timestep t
S_t	State at timestep t
R_t	Reward at timestep t
G_t	Return (cumulative discounted reward) following time t
π	Policy, agent behavior rule
$\pi(s)$	Action taken in state s under policy π

$v_\pi(s)$	Value of state s under policy π (expected return)
$v_*(s)$	Value of state s under the optimal policy
$q_\pi(s, a)$	Value of taking action a in state s under policy π
$q_\pi(s, a)$	Value of taking action a state s under policy π
V, V_t	Estimate of state-value function v_π or v_*
Q, Q_t	Estimate of action-value function q_π or q_*

Contents

1 Introduction

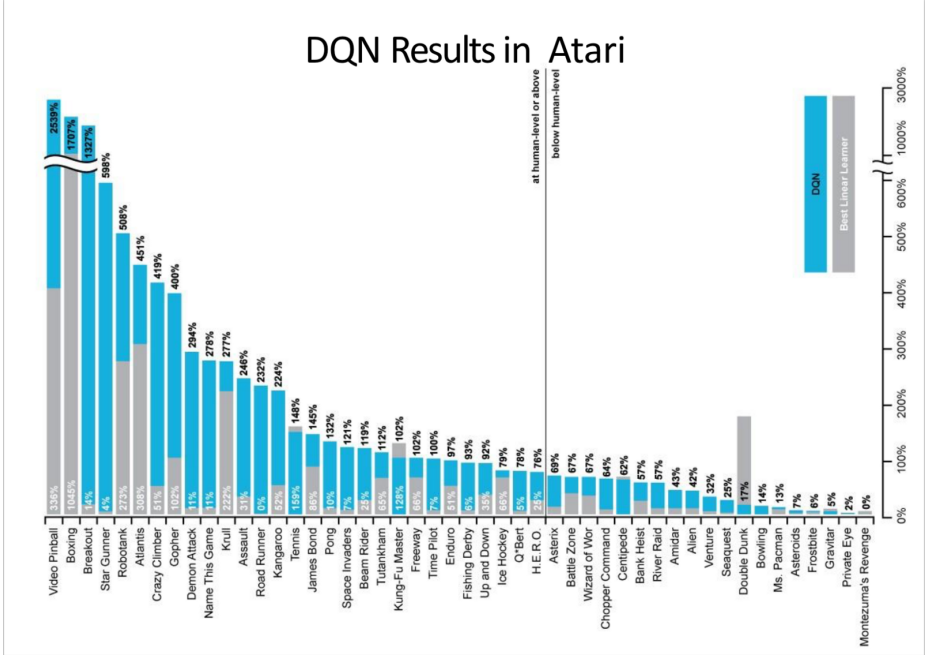
1.1 Motivation

In the last decades, the advances in computers architecture and computer science have pushed the research frontier to produce super intelligent algorithms, capable of dealing with difficult classification problems of subtle and inherently human concepts, or even hard decision making tasks in challenge situations. We can see intelligent algorithms applied to speech and image recognition, email spam classification, advertising, fraud detection, autonomous self-driving cars, virtual assistants, security, finance and several other applications in our lives. Never in the history we have witness such a great bet in the future of machines.

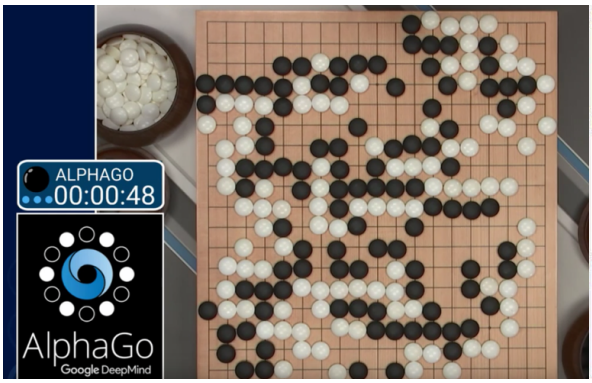
Besides, most recently, the cutting edge achievements in AI have shown algorithms capable of overcome top human intelligence in difficult problems. In 2015, Google DeepMind created an AI that learned how to play 49 Atari games using the same learning algorithm (including the same hyper parameters), using as the input only the pixels from the screen. The algorithm, called Deep Q-Learning (MNIH *et al.*, 2015), was a breakthrough achievement in the mission of accomplishing a general purpose machine learning agent in a wide variety of games.

In the same year, DeepMind left another big mark in the human history. A computer program called AlphaGo defeated the best human Go player for the first time. Go consists of a very complex board game, with more than $10^{1}70$ configurations, and represents one of the biggest challenges to human intelligence and an unconceivable problem to a computer until then. Going even further, in 2017, Silver *et al.* (2017) introduced AlphaGo Zero, an evolution of the previous version, capable of leaning how to play without any outside data from human games, but just playing with itself. Figure 1.1 illustrates these successful examples.

Another big field in Computer Science is mobile robotics, which plays a major role in the future of the industry and the forefront of academic research recently. Robotics can address state of the art challenges in different domains, such as Electronics Engineering, Mechanical Engineering, Control Theory and, of course, Machine Learning.



(a) DQN results in Atari games



(b) AlphaGo against Lee Sedol

(c) Different Atari games

FIGURE 1.1 – DeepMind recent achievements.

AI finds in Robotics several applications, as computer vision, path planning and even locomotion, and in this last one we can find one of its biggest challenges. In Atari and board games we can easily define a goal, win, but how can we define the agility and flexibility of a walk or a jump movement? In this sense, several works have been conducted in the problem of trying to reproduce human movements in humanoid robotics agents, and more specifically, in a simulated scenario.

One of the biggest initiatives of the research community to foster the study of robotics is RoboCup. RoboCup established itself as one of the main international robotics competition in the world, and a powerful scientific conference. It pushes state-of-the-art research in robotics by maintaining different technical challenges with one thing in common, making robots play soccer, with the mission of developing a humanoid robot team capable of win the champion of FIFA World Cup in 2050.

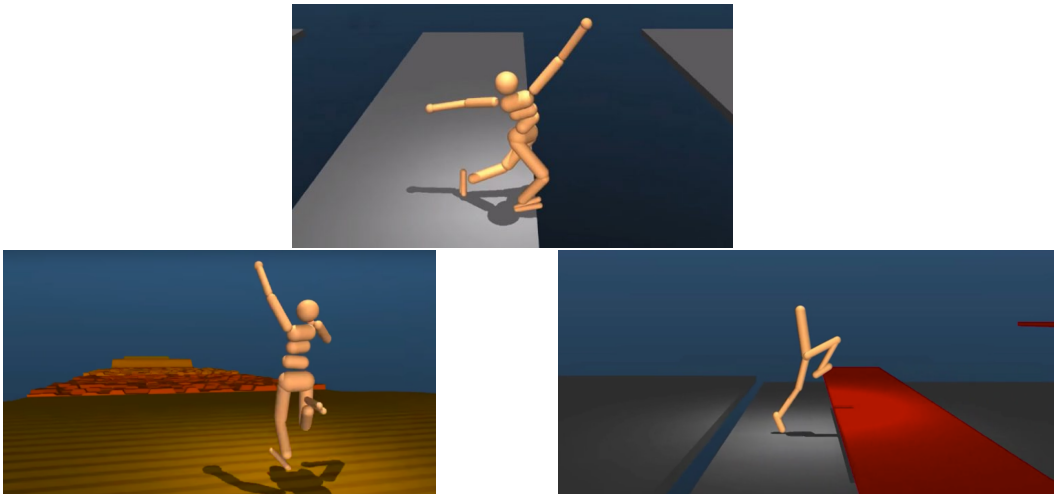


FIGURE 1.2 – Simulated humanoid agent movements and AI.

One RoboCup's particular league is the RoboCup 3D Soccer Simulation League, consisting of a soccer match between two teams, each one composed by up to 11 simulated NAO robots from Aldebaran Robotics. This league address both high level and low level robotics challenges, like path planning and locomotion, and greatly helped improving our understand of human movements like walk and kick the ball.

FIGURE 1.3 – Robocup symbol and Soccer 3D Simulation league match.

1.2 Problem Statement

Inspired by the RoboCup Soccer Simulation 3D (Soccer3D or SS3D) league, this dissertation's objective is to learn a high level soccer behavior for simulated humanoid robots, more specifically, the behavior of kicking the ball towards a planned final distance from the agent. In this sense, the algorithm should input the current game state, including agent's and ball's positions, and output a sequential movement for each robot's joint.

1.3 Approach

Instead of employing a deterministic and off-line built single movement, called Keyframe movement, we intend to use a model-free deep reinforcement learning based approach that tries to learn the desired behavior by interacting with an environment (the simulation server) and receiving rewards depending on which actions it chooses. The chosen strategy for this will be first learning a behavior that imitates the current kick movement, as described in the work (PENG *et al.*, 2018), and then improving this behavior, by learning with reinforce, targeting the desired position given as an input to the policy function.

1.4 Literature Review

Reinforcement Learning techniques have been in increasingly study in the past 30 years. In this time, some specific works have established famous breakthroughs in this field. For instance, Temporal-Difference (TD) learning algorithms (BARTO *et al.*, 1983) created the groundwork for many future RL algorithms, making use of value function estimation and policy and value iteration methods. In the same way, Q-Learning (WATKINS, 1989) introduced off-policy model-free learning with *Sarsa*, estimating action-value functions and making room to future DRL algorithms.

After these works, Wiliams (1992) created a new paradigm presenting the REINFORCE algorithm, which introduces policy search methods. This technique estimates the optimal-policy function π_* directly, without the need of value or action-value functions and works better in continuous action space, as we see in the robotics world. The most recent approaches developed make use and improve this method, such as the Deep Deterministic Policy Gradients (DDPG) algorithm, introduced by (LILLICRAP *et al.*, 2015), the Trust Region Policy Optimization (TRPO) algorithm, presented in (SCHULMAN *et al.*, 2015), and the Proximal Policy Optimization (PPO) algorithm, given in (SCHULMAN *et al.*, 2017).

Another recent breakthrough in the RL field was given by Deep Neural Networks, making possible to escalate the classical algorithms to high dimensional problems. This process was marked by the incredible results of the Deep Q-Networks (DQN) algorithm (MNIH *et al.*, 2015). This technique was able to learn 49 Atari games directly from raw pixels of the screen. It introduces the idea of modeling the action-value function as a neural network, and handle the inherently instability problem from value function approximation by introducing also two techniques: Experience Replay (LIN, 1992) and Target Networks Mnih *et al.* (2015).

Meanwhile, some recent works started to address humanoid movements in the RL

field. However, the main problem which comes with that endeavor is that simple rewards functions or naively selected ones, such as the score for the games problems, can lead to results that do not match the expectations. This is the common case in continuous control tasks, like locomotion. In this sense, (HEESS *et al.*, 2017) proposed that rich and robust behaviors can emerge from simple reward functions, if the environment itself contains sufficient richness and diversity. This work introduces scenarios with a lot of obstacles and varying levels of difficulty, which are presented to the agent as an implicit curriculum, making possible to overcome increasingly hard challenges. A similar idea was also introduced by (BENGIO *et al.*, 2009)

Other DeepMind's papers introduced methods to learn to imitate human movements, like (MEREL *et al.*, 2017) and (WANG *et al.*, 2017), which combines supervised learning and Generative Adversarial Imitation Learning (GAIL) ((HO; ERMON, 2016)), in a way that accentuates their individual strengths and address their limitations.

Another recent state-of-the-art work in imitation learning is (PENG *et al.*, 2018). This work makes possible for a motion capture actor to supply a set of reference motions for style, and then generate goal-directed and physically realistic behaviors from them. The approach used for this is designing a reward function which combines rewarding motions that resemble reference animation data, and also achieving additional task objectives.

The most famous work that address the specific mission of kicking the ball towards a planned final distance from the agent in the Soccer3D environment is (ABDOLMALEKI *et al.*, 2016). In this work, the authors used a policy search approach to determine the optimal parameters for the kicking behavior policy $\pi(\theta|s)$, proposing the use of contextual relative entropy policy search with covariance matrix adaptation (CREPS-CMA) algorithm for this task.

However, we can see some limitations of this work that is possible to tackle. In (ABDOLMALEKI *et al.*, 2016), the authors still make use of a Keyframe based movement, and the policy $\pi(\theta|s)$ only outputs the initial and final frame position given the desired distance s. We propose a neural network based policy which inputs and outputs all the joints positions in run time, performing a closed loop behavior. Besides, there are more recent and cutting-edge RL algorithms that can be used to learn this task, such as DDPG, TRPO and PPO, and also, by making use of the approach described in (PENG *et al.*, 2018), it is possible to overcome the initial challenge of learning a basic but complete behavior at first.

At last, we must cite the work (cite), which is one of the first works covering deep reinforcement learning and the SS3D, with similar tasks regarding humanoid locomotion. More specifically, this work handles the problem of developing a behavior for dribbling the ball against a single opponent, and provide successful solutions using DDPG, TRPO

and PPO.

1.5 Contributions

This work's major contribution is applying recent deep reinforcement learning (DRL) algorithms in the task of learning a complete behavior to kick a ball towards a planned final distance from the agent in the Soccer3D environment domain. To the best of our knowledge, this work is the first that makes use of a DRL approach to develop a complete soccer behavior for this specific task.

1.6 Outline of this Dissertation

This dissertation is organized as follows:

- **Chapter 1** introduces this dissertation by describing the motivation behind the problem we address, by the Literature review and by summarizing the contributions.
- **Chapter 2** describes a brief theoretical background of reinforcement learning and neural networks.
- **Chapter ??** describes some experiments using OpenAI gym frameworks

2 Reinforcement Learning

2.1 Model Introduction

Reinforcement Learning is a new branch and paradigm in the Machine Learning field. But different than the classical Supervised Learning, in RL there is no supervisor, only a reward signal. In this sense, we have an agent which interacts with the environment and receives instant rewards. Therefore, in RL, we can say that the agent's actions affect the subsequent data it receives, and that time really matters, since this data is sequential non i.i.d, and moreover, the total feedback, given by the sum of all rewards, is delayed, not instantaneous (SUTTON; BARTO, 1998). All these features make RL a quite unique research field, and it can also finds a wide range of applications, from playing board games to machines control.

The basic model for a RL approach makes use of a discrete state space for representing the agent's state S_t, its actions A_t and its full or partial environment observations O_t. Thus, at each timestep t, the agent receives an observation O_t, takes an action A_t and ends up in the next state S_{t+1}, receiving an scalar reward R_{t+1}. Therefore, the agent's final goal is to maximize the total future reward it gets. Figure 2.1 illustrates this dynamics representation.

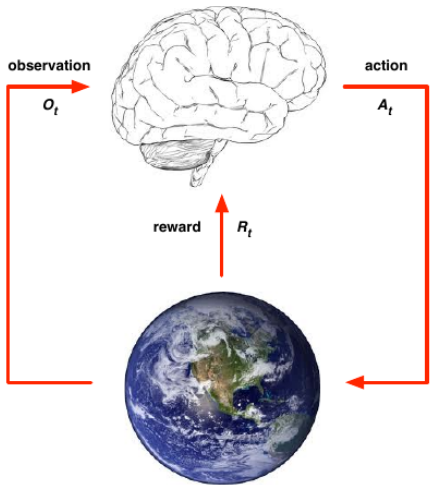


FIGURE 2.1 – Agent interacting with environment.

2.2 Markov Decision Processes

A classic formal description of an environment for reinforcement learning makes use of Markov decision processes, which inherits the Markov Property from the well known Markov chains. The Markov property establish that the state of the agent captures all the relevant information from the history, in other words, the probability update function only depends of the immediate previous state, as described in 2.1.

$$P(S_{t+1}|S_t) = P(S_{t+1} \mid S_1, ..., S_t) \tag{2.1}$$

Markov decision processes, however, has a more complete definition, it is a Markov process with rewards and actions. Basically, can be represented as a tuple $< \mathbf{S}, \mathbf{A}, P, R, \gamma >$, with each component defined as follows:

- **S** is a finite set of states the agent can assume in the environment.
- **A** is a finite set of actions the agent can take.
- P is a state transition probability function, represented as $P^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- R is a reward function, given by $R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor, such that $\gamma \in [0, 1]$, used to compute the cumulative total reward.

Besides this basic tuple representation, there are four more concepts very used in the RL literature that we must present.

- The **return** G_T is the total discounted reward from time-step t, given by:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

  Notice that by introducing the discount factor γ, it is possible to set the agent preference between short-term and long-term rewards.

  The **policy** $\pi(a \mid s)$ is a probability distribution function over actions given states, defined by:

$$\pi(a \mid s) = \mathbb{P}[A_t = a|S_t = s] \tag{2.3}$$

  The policy fully defines the behavior of the an agent, and can also be deterministic.

- The **state-value function** $v_\pi(s)$ is the expected return starting from state s and then following a policy π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{2.4}$$

  In other words, it can be seen as a measure of how good the current agent's state is.

- The **action-value function** $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi(G_t \mid S_t = s, A_t = a) \tag{2.5}$$

  It can also be understood as a measure of how good it is to take a given action at the agent's current state.

2.2.1 Optimality in Reinforcement Learning

The ultimate goal of RL is finding an optimal behavior that maximizes the total expected return given by $\mathbb{E}_{R_i, S_i \sim E, A_i \sim \pi}[G_i]$. Before starting presenting the methods for achieving this goal, we must define what is optimality for value functions and policies.

2.2.1.1 Optimal Value Function

A value function is optimal if it is the maximum over all policies for all states (or state-action pairs).

- Optimal State-value function: $v_*(s) = \max_\pi v_\pi(s)$
- Optimal Action-value function: $q_*(s, a) = \max_\pi q_\pi(s, a)$.

The optimal value functions specify the best possible performance in the MDP, which is only "solved" when we know the optimal value functions.

2.2.1.2 Optimal Policy

First, we must define a partial ordering over policies, given in equation 2.6.

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s \tag{2.6}$$

The optimal policy π_* is then defined as the best among all the others policies π, such that $\pi_* \geq \pi, \forall\pi$. Moreover, the following theorem introduces three fundamental properties about optimal policies.

- There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall\pi$.
- All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$.
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$.

Furthermore, a deterministic optimal policy always exists, and can be easily defined from the optimal action-value function, as described in equation 2.7.

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \underset{a'}{\text{argmax}}\ q_*(s, a') \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

2.3 RL Algorithms

In order to base our future approaches in the main task, we must start presenting the classical reinforcement learning algorithms in the literature and, first of all, categorizing its different natures. The following subsections will be responsible for that.

2.3.1 Categorizing RL

RL algorithms can be classified into several types related to their problems and solutions approaches.

Model based algorithms are used when the model of the environment is known, and therefore, the agent can performs computations without external interaction to improves its policy. Model free algorithms, however, do not use any environment model, and it improves its policy by only interacting with the environment. These two problems conditions are respectively classified as planning and reinforcement learning problems.

Moreover, RL algorithms can also be classified as value based, policy based or actor critic. Value based algorithms are methods based on computing optimal value functions, whereas policy based algorithms make use of optimal policy search approaches. Between them, we also have an hybrid solution which combines both value function and policy search approaches. Figure 2.2 illustrates this classical RL categories division.

Regarding their solution nature, we also have two more classifications. Prediction algorithms are basically policy evaluation algorithms, which does not obtain any optimal policy solution, but instead, given a policy computes value functions. Control algorithms, whereas, aims to achieve an optimal policy. Besides, we still have two other classifications regarding the solution approach, that are on-policy algorithms, which learns by interacting with the world with its own policy, and off-policy algorithms, which learns an optimal policy by interacting with the world with another policy.

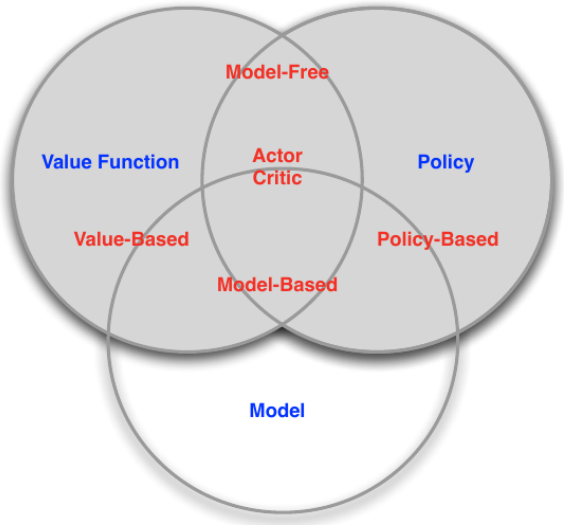


FIGURE 2.2 – Classical division among RL algorithms classifications.

2.3.2 Value Function Methods

As already mentioned, value function methods employ techniques that aims to compute or estimate value functions. These algorithms are the most classic in the RL literature, and their implementation differ basically in relation to concepts like bootstrapping, bias, variance, on-policy and off-policy. In this section, we will focus on presenting only model-free algorithms, since it is from the same nature of our problem.

2.3.2.1 Monte Carlo Methods

Monte Carlo or MC methods are model-free methods that learn directly from full episodes of experience. It computes value functions through its expectation definition given in equations 2.4 and 2.5. However, without a known model for the MDP, it estimates this expectation by sampling and computing the empirical mean return.

Therefore, for Monte Carlo prediction, the MC version for policy evaluation, we compute the state value $v_\pi(s)$ by averaging full episodic returns starting from state s and running on policy π for an arbitrarily large number of episodes in this condition.

For the control method, the main idea is to employ policy iteration, which is a basic framework for control problems which consists in an iterative solution that evaluates a given policy, by computing the value functions, and then updates this policy by acting greedy with respect to these computed value functions. Acting greedy would be choosing a deterministic policy given by equation 2.7. This loop repeats until convergence to optimal policy and optimal value functions.

In Monte Carlo model-free control, in particular, the usual approach is to use Monte-Carlo prediction to evaluate action-value functions $Q_\pi(s, a)$ under the policy π, then update the policy acting ϵ-greedy. ϵ-greedy is a simple policy update technique very similar to the greedy approach from 2.7, but it is able to ensure continual exploration. The solution for this is choosing the greedy action with $1-\epsilon$ probability, and all the others actions with the same non-zero probability. This calculation is presented in equation 2.8.

$$\pi_*(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \underset{a'}{\text{argmax}}\ q_*(s, a') \\ \epsilon/m, & \text{otherwise} \end{cases} \tag{2.8}$$

2.3.2.2 Temporal-Difference and Sarsa

Temporal-Difference methods are model-free methods that are able to learn from incomplete episodes, by bootstrapping. This idea consists in updating an estimate by using another estimate obtained after taking some decision steps. In its most simple version for policy evaluation, $TD(0)$, it is possible to learn v_π online while experiencing under π, by updating the estimate $V(S_t)$ toward the estimated return $R_{t+1} + \gamma V(S_{t+1})$, as given in equation 2.9.

$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{2.9}$$

For Temporal-Difference control, it is also commonly employed a policy iteration approach. We use TD to compute action-value function estimates $Q(S, A)$ given the policy π, and then use ϵ-greedy to improve the policy. This method is known as SARSA, due to the process of, given a state S, take an action A, receive a reward R, end in a next state S', and then choose another action A' in order to compute the TD target. The algorithm

1 fully describes this method.

Algorithm 1: Sarsa algorithm

Initialize $Q(s,a)$, $\forall s \in S$, $a \in A(s)$, arbitrarily, and Q(terminal-state,.)=0
for *episode* = 1, M **do**
  Initiliaze S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy).
  for t = 1, ...,T **do**
    Take action A, observe R, S'.
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy).
    $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$.
    $S \leftarrow S'$; $A \leftarrow A'$;
  end
end

The SARSA algorithm is most famous for another variation, that is able to bootstrap into more future steps. The SARSA(λ) method computes a different TD target Q_t^λ making use of a discount weight $(1-\lambda)\lambda^{n-1}$ over n steps action-values $Q_t^{(n)}$, as shown in equation 2.11.

$$Q_t^\lambda = (1-\lambda)\sum_{i=1}^{n} \lambda^{n-1} Q_t^{(n)} \tag{2.10}$$

$$Q(S,A) = Q(S,A) + \alpha(Q_t^\lambda - Q(S,A)) \tag{2.11}$$

The main difference between SARSA and MC control algorithms is that SARSA can learns before the end of the episode, and have low variance and some bias, whereas MC can only learn from complete episodes, and therefore has high variance and zero bias. Besides, MC has better convergence properties.

2.3.2.3 Q-Learning

For off-policy learning we have some variations of the previous algorithms. The most basic ones involve importance sampling, that corrects expectations given in one distribution to another, as shown in equation 2.12.

$$\mathbb{E}_{X\sim P}[f(X)] = \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)} f(X)\right] \tag{2.12}$$

The most famous off-policy algorithm though is Q-learning, that does not importance sample. Q-learning is able to improve both the behavior policy μ and the target policy π,

by acting greedy w.r.t. $Q(s,a)$ for π and acting ϵ-greedy w.r.t. $Q(s,a)$ for μ. The general algorithm is very similar to SARSA, with the difference that the next action is chosen using the behavior policy, and the alternative successor action is chosen using the target policy. Equation 2.13 describes the update step for $Q(S,A)$ in Q-learning.

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_{a'} Q(S',a') - Q(S,A)) \tag{2.13}$$

2.3.3 Policy Search Methods

Policy search methods are model-free control algorithms which directly compute the optimal value function, without the use of value functions estimates, as shown in previous algorithms. In this sense, the approach consists in using a parametrized function approximation $\pi_\theta(a|s)$ for the policy, and the goal is to reach the desired parameters θ^* that best estimate the optimal policy, in a way that $\pi_{\theta^*}(a|s) \approx \pi_*(a|s)$. Furthermore, the strategy for achieving this goal is solving an optimization problem, which comes from another perspective for the optimal policy, that is the policy that maximizes the expected total return running over several trajectories τ, as defined in equations 2.14 and 2.15.

$$P_\theta(\tau) = P_\theta(s_1, a_1, ..., s_T, a_T) = P(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t) \tag{2.14}$$

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\ J(\theta) = \underset{\theta}{\operatorname{argmax}}\ \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\sum_t R_t(s_t, a_t) \right] \tag{2.15}$$

This optimization problem is solved through gradient ascent methods, and therefore we must compute the gradient of the objective function $J(\theta)$. For this propose, we can rewrite $J(\theta)$ as in 2.16, and compute its gradient $\nabla_\theta J(\theta)$ using the manipulation described in 2.17, in order to work with the score function $\nabla_\theta \log \pi_\theta(a|s)$.

$$J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \underbrace{\left[\sum_t R_t(s_t, a_t) \right]}_{r(\tau)} = \int \pi_\theta(\tau) r(\tau) d\tau \tag{2.16}$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(\tau) r(\tau) \right] \tag{2.17}$$

Using the gradient $\nabla_\theta J(\theta)$, computed in 2.17, the main idea presented in (WILIAMS, 1992) is making use of stochastic gradient ascent to reach θ^*. The algorithm is therefore

presented in 2.

Algorithm 2: REINFORCE algorithm

Initialize θ arbitrarily
for *each episode* $\{ s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T \} \sim \pi_\theta$ **do**
  for $t = 1, ...,T$ **do**
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$
  end
end
return θ

In comparison with value function methods, the main advantages of policy-based RL algorithms are the effectiveness in high-dimensional or continuous action spaces, the possibility to learn stochastic policies and the better convergence properties. The main disadvantages, though, are the computational inefficiency, the high variance of these methods and the usual convergence to local optimum rather than the global optimum.

2.3.4 Actor-Critic

In algorithm 2, we used the total return G_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$, but we can also rewrite $\nabla_\theta J(\theta)$ as in equation 2.18. In this sense, we could also use an action-value function approximation to derive the objective function gradient, by making use of a *critic* function $Q_W(s, a) \approx Q^{\pi_\theta}(s_t, a_t)$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(\tau) Q^{\pi_\theta}(s_t, a_t) \right] \tag{2.18}$$

Therefore, Actor-Critic methods maintain two sets of parameters to be updated toward optimum policy and action-value functions: the parameters W for the *critic* action-value estimate $Q_W(s, a)$, and the parameters θ for the *actor* policy estimate $\pi_{\theta^*}(a|s)$.

3 Deep Reinforcement Learning

The previous chapter described a summarized background for RL concepts and classical algorithms. This chapter aims to go deeper towards the techniques we are actually going to use in this work. First, we must build the background for Neural Networks (NN) and Multi Layer Perceptrons (MLP), which are commonly used as policy and value function approximations in modern RL algorithms. Then, we will present two state of the art RL techniques which we plan to use in our problem.

3.1 Neural Networks

Neural Networks are mathematical structures that receive inputs and calculate outputs. In this sense, can be described as a mathematical fitter to very complex and non-linear functions.

Also known as Multi Layer Perceptron (MLP), it has a multiple layer architecture, where each layer is composed by several nodes called neurons. These nodes, in an analogy to human brain neurons, are still mathematical units which receives some numerical inputs and outputs one single number. All of these mathematical units make use of a specific set of parameters.

3.1.1 Representation

Regarding the representations and notations used for NN, we can describe its basic elements consisting of inputs, outputs, parameters and activate functions.

For each one of the m training examples, we define the input $x^{(i)}$, corresponding to the i example, as a n-dimensional feature column vector, so with dimensions $(n, 1)$. It is also called the input layer, with index 0, and also represented as $a^{[0](i)}$ with dimensions $(n^{[0]}, 1)$. Moreover, we also define the final expected output for each training example, given by the vector y with dimensions $(n_F, 1)$.

For each layer l, also called hidden-layers, with $1 < l < L$ in a total of L layers, we

have a specific structure:

- The layer l has a total of $n^{[l]}$ hidden units, each jth unit receives all the inputs $a^{[l-1](i)}$ from layer $l-1$, and output a scalar $a_j^{[l](i)}$. The total output of the layer is then composed by all hidden units outputs, given by the vector $a^{[l](i)}$, with dimensions $(n^{[l]}, 1)$.
- Each jth hidden unit has a row-vector linear parameter $w_j^{[l]}$ with dimensions $(1, n^{[l-1]})$. The total linear parameters $W^{[l]}$ of the layer is given by all hidden units parameters in a $(n^{[l]}, n^{[l-1]})$ matrix.
- Each jth hidden unit also has a scalar bias-parameter given by $b_j^{[l]}$, and all units bias-parameters compose the layer bias vector $b^{[l]}$, with dimensions $(n^{[l]}, 1)$
- The layer also has a specific activation function $g^{[l]}(z)$, which can assume different mathematical functions, such as the *sigmoid* function, the *tanh* function, and, the most common, the $ReLU$ function, shown in equation 3.1.

$$sigmoid(z) = \frac{1}{1+e^{-z}} \; ; \; tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \; ; \; Relu(z) = max(z, 0) \tag{3.1}$$

All these elements of the layer are related through the update equations 3.2 and 3.3, which, for each layer l, receives the previous layer output $a^{[l-1]}$ and generates the current layer output $a^{[l]}$. The figure 3.1 illustrates this representation for a shallow one hidden-layer NN.

$$z^{[l](i)} = W^{[l]} a^{[l-1](i)} + b^{[l]} \tag{3.2}$$

$$a^{[l](i)} = g^{[l]}(z^{[l](i)}) \tag{3.3}$$

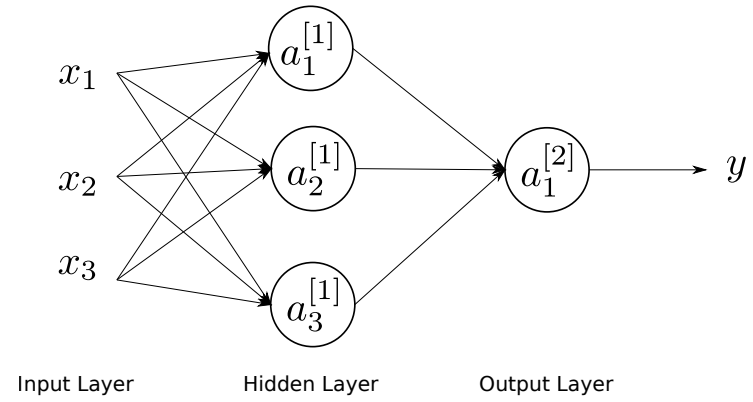


FIGURE 3.1 – Shallow Neural Network Architecture

One last basic definition for NN is the cost function $J(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]})$. The cost function basically measure how good our estimator is to our dataset. The lower the cost function, the less error we obtain when predicting the desired function. Therefore, it represents the target function to be minimized, with respect to all the parameters used in the NN $\{W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}\}$.

It is possible to employ several different cost functions according to each specific problem. However, we can mention the most common ones, that are the Quadratic Cost, given by equation 3.4, and the Cross-Entropy Cost, given by equation 3.5.

$$J(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n^{[L]}} (y_j^{(i)} - a_j^{[L](i)})^2 \tag{3.4}$$

$$J(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n^{[L]}} \left[y_j^{(i)} \log a_j^{[L](i)} - (1 - y_j^{(i)}) \log (1 - a_j^{[L](i)}) \right] \tag{3.5}$$

3.1.2 Vectorization

The previous equations are capable of fully defining update rules for any MLP, however, in order to avoid the naively computation over each training example, and therefore reach more efficient algorithms exploring the use of parallelization in modern GPUs architectures, we must introduce a vectorization notion.

Let's first define the X matrix as the concatenation of each training example $x^{(i)}$ composing its columns, as shown in equation 3.6. The matrix X now assumes the dimensions $(n^{[0]}, m)$.

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \tag{3.6}$$

In this sense, we can analogously define the matrices $A^{[l]}$ and $Z^{[l]}$, as a concatenation of the values for each example $a^{[l](i)}$ and $z^{[l](i)}$, described in equation 3.7. The matrices $A^{[l]}$ and $Z^{[l]}$ now assume the dimensions $(n^{[l]}, m)$.

$$A^{[l]} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ a^{[l](1)} & a^{[l](2)} & \dots & a^{[l](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \quad Z^{[l]} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ z^{[l](1)} & z^{[l](2)} & \dots & z^{[l](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \tag{3.7}$$

The update equations for each layer, in the vectorized representation, is therefore modified and given by the equations 3.8 and 3.9.

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]} \tag{3.8}$$

$$A^{[l](i)} = g^{[l]}(Z^{[l]}) \tag{3.9}$$

3.1.3 Forward Propagation

Given all the definitions of the basic structures of a NN and the main calculations involved, we can derive the algorithm for the forward propagation. The forward Propagation algorithm consists in: given all the dataset composed by the matrix X, compute all layers outputs $A^{[l]}$ and $Z^{[l]}$, including the final layer and the final output for the NN, which are $A^{[L]}$ and $Z^{[L]}$.

Algorithm 3: Forward Propagation

for $l = 1, ..., L$ **do**
  $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$
  $A^{[l]} = g^{[l]}(Z^{[l]})$
  cache $Z^{[l]}$ and $A^{[l]}$
end
return $A^{[L]}$

3.1.4 Backward Propagation

The ultimate goal in Supervised Learning is minimizing the cost function. For this purpose, one of the most famous optimization techniques used is gradient descent. However, in order to make use of gradient descent, we must compute the derivatives of the cost function with respect to all the parameters $W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}$. The notation used for these partial derivatives is given in equations from 3.10 to 3.13.

$$\frac{\partial J(\theta)}{\partial W^{[l]}} = dW^{[l]} \tag{3.10}$$

$$\frac{\partial J(\theta)}{\partial b^{[l]}} = db^{[l]} \tag{3.11}$$

$$\frac{\partial J(\theta)}{\partial A^{[l]}} = dA^{[l]} \tag{3.12}$$

$$\frac{\partial J(\theta)}{\partial Z^{[l]}} = dZ^{[l]} \tag{3.13}$$

Finding a closed expression for each partial derivative would be a massive analytical work, hence an elegant solution is to employ a iterative algorithm called Backward Propagation, or Backpropagation. The main idea involved in this algorithm is computing each partial derivative from the next layer output derivative $dA^{[l]}$ and the cached data $A^{[l-1]}$ and $Z^{[l]}$ from the forward propagation algorithm. The Backpropagation algorithm is then given in Algorithm 4.

Algorithm 4: Backward Propagation

$dA^L = \sum_{i=1}^{m} \frac{-y^{(i)}}{a^{[L](i)}} + \frac{(1-y^{(i)})}{(1-a^{[L](i)})}$

for $l = L, ..., 1$ **do**

  input $dA^{[l]}$ and cached values $A^{[l-1]}$ and $Z^{[l]}$

  $dZ^{[l]} = dA^{[l]} * g'^{[l]}(Z^{[l]})$

  $dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$

  $db^{[l]} = \frac{1}{m}$ sum over rows of $dZ^{[l]}$

  $dA^{[l-1]} = W^{[l]T} dZ^{[l]}$

end

return $(dW^{[1]}, db^{[1]}, ..., dW^{[L]}, db^{[L]})$

Finally, by making use of the partial derivatives already computed, we can employ the classical Gradient Descent, described in Algorithm 5. Notice that, until convergence, we update all the parameters by the derivatives computed from all the training examples in each step, consisting in the Batch Gradient Descent.

Algorithm 5: Gradient Descent

Initialize $W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}$ with random values near 0

while *An approximate minimum is not obtained* **do**

  Forward Propagation()

  Backward Propagation()

  for $l = 1, ..., L$ **do**

    $W^{[l]} = W^{[l]} - \alpha dW^{[l]}$

    $b^{[l]} = b^{[l]} - \alpha db^{[l]}$

  end

end

return $(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]})$

3.2 Trust Region Policy Optimization

Given all the RL and NN background necessary, we will introduce two state of the art techniques combining both fields.

The point of contact relies on the fact that we need complex non-linear functions

to efficiently estimate policies and value-functions. Therefore, the most common used structure for this approximation is MLP. Aiming also to fit stochastic policies distributions in continuous action spaces, we define our policy $\pi_\theta(s|a)$ by a normal distribution $\mathcal{N}(\mu, \sigma)$, where the mean μ and log standard deviation σ are outputs of a neural network (SCHULMAN *et al.*, 2015).

Moreover, in order to work with continuous action spaces, we must employ policy search or actor-critic methods. The classic vanilla policy gradient methods, however, bring some limitations that are handled in the recent techniques. We can cite the main limitation as the vulnerability to the step size in the optimization procedure. This fact brings even bigger impacts due to the non-stationary nature of the input data, which means that a small error in the policy update affects the visitation distribution and consequently the future policy updates.

The Trust Region Policy Optimization tries to address this limitation. The novel idea introduced by this algorithm is limiting the policy step size to lie within a so called trust region, and thus avoid divergence. This trust region is computed through the Kullback-Leibler (KL) divergence. The KL divergence is basically a measure of the deviation between two probability distributions, and it is defined as the relative entropy between two continuous random variables P, Q. Let $p(x)$ and $q(x)$ be the probability density functions of P and Q respectively, the KL divergence, $KL[P, Q]$, is then given by equation 3.14.

$$KL[P, Q] = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \tag{3.14}$$

TRPO therefore limits the policy gradient step size by applying a KL based constraint (Eq. 3.15) to an optimization problem on a slightly modified cost function (Eq. 3.16). The overall algorithm is iteratively, we run episodes and collect data from $\pi_{\theta_{old}}$, compute π_θ from the optimization problem and then update $\pi_{\theta_{old}} \leftarrow \pi_\theta$. This method is described in Algorithm 6.

$$\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_{s_t, a_t \sim \pi_{\theta_{old}}} \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] \tag{3.15}$$

$$\text{subject to}\ \hat{\mathbb{E}}_{s_t, a_t \sim \pi_{\theta_{old}}} \left[KL[\pi_{\theta_{old}}, \pi_\theta(.|s_t)] \right] \leq \delta \tag{3.16}$$

Algorithm 6: TRPO

for *iteration* = 1, 2, ... **do**

Run policy for N trajectories

Estimate advantage function $\hat{A(.)}$ at all n timesteps

$\underset{\theta}{\text{maximize}}\ \sum_{n=1}^{N} \frac{\pi_\theta(a_n|s_n)}{\pi_{\theta_{old}}(a_n|s_n)} \hat{A}_n$

subject to $\overline{KL}[\pi_{\theta_{old}}, \pi_\theta(.|s_t)] \leq \delta$

$\pi_{\theta_{old}} \leftarrow \pi_\theta$

end

return π_θ

In this algorithm, we must compute an estimate for the advantage function $\hat{A}_t$. The advantage function is given by $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$, and it is basically the standard critic function $Q(s_t, a_t)$ subtracted by a unbiased baseline $V(s_t)$ (state-value function) in order to reduce variance. For solving the constrained optimization problem, the conjugate gradient method is generally used.

TRPO presents a great improvement under data efficiency and convergence properties. However, it has a great computational cost, and its implementation through conjugate gradients can be relatively complicated (SCHULMAN *et al.*, 2017).

3.3 Proximal Policy Optimization (PPO)

The Proximal Policy Optimization (SCHULMAN *et al.*, 2017) is a recent technique that follows a similar approach used for TRPO, which consists in solving an optimization problem by computing an update at each step that minimizes the cost function while ensuring a relatively small deviation from the previous policy. However, the great breakthroughs introduced are enhancements in the ease of implementation and the ease of hyperparameters tuning, while achieving an even better performance.

The main idea behind TRPO is to use an adaptive KL penalty to control the change of the policy at each iteration. PPO introduces a novel objective function, which intrinsically address this constraint in a first-order optimization.

Let $r_t(\theta)$ be the ratio between the new and old policy, given by the equation 3.17, the PPO's objective function is then described by the equation 3.18.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3.17}$$

$$L(\theta) = \hat{\mathbb{E}}_t \left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{3.18}$$

In equation 3.18, θ is the policy parameter, $\hat{\mathbb{E}}_t$ is the empirical expectation over all t timesteps, $\hat{A}_t$ is the estimate for the advantage function, as already described, ϵ is a new hyperparameter, and finally the clip function is explained in equation 3.19.

$$clip(x, min, max) = \begin{cases} min, & \text{if } x < min \\ x, & \text{if } min \leq x \leq max \\ max, & \text{if } x > max \end{cases} \tag{3.19}$$

PPO does not need complex conjugate gradient techniques, and can be solved with classical stochastic gradient descents. Besides, it has displayed the best performance on continuous control tasks, even compared to TRPO, it also allows parallel implementations and it is very data efficient.

Bibliography

ABDOLMALEKI, A.; SIMOÌES, D.; LAU, N.; NEUMANN, L. P. R. andGerhard. Learning a humanoid kick with controlled distance. **20th RoboCup International Symposium, Leipzig, Germany, July 2016.**, July 2016.

BANSAL, T.; PACHOCKI, J.; SIDOR, S.; SUTSKEVER, I.; MORDATCH, I. Emergent complexity via multi-agent competition. **CoRR**, abs/1710.03748, 2017. Disponível em: <http://arxiv.org/abs/1710.03748>.

BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. 1983.

BENGIO, Y.; COURVILLE, A. C.; VINCENT, P. Unsupervised feature learning and deep learning: A review and new perspectives. **CoRR**, abs/1206.5538, 2012. Disponível em: <http://arxiv.org/abs/1206.5538>.

BENGIO, Y.; LOURADOUR, J.; COLLOBERT, R.; WESTON, J. Curriculum learning. In: **Proceedings of the 26th Annual International Conference on Machine Learning**. New York, NY, USA: ACM, 2009. (ICML '09), p. 41–48. ISBN 978-1-60558-516-1. Disponível em: <http://doi.acm.org/10.1145/1553374.1553380>.

BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. **OpenAI Gym**. 2016. Acesso em: 10 maio de 2018.

FLORENSA, C.; HELD, D.; WULFMEIER, M.; ABBEEL, P. Reverse curriculum generation for reinforcement learning. **CoRR**, abs/1707.05300, 2017. Disponível em: <http://arxiv.org/abs/1707.05300>.

GOODFELLOW, I. J.; VINYALS, O.; SAXE, A. M. Qualitatively characterizing neural network optimization problems. **arXiv preprint arXiv:1412.6544**, 2014.

GOODFELLOW, Y. B. I.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>.

HEESS, N.; TB, D.; SRIRAM, S.; LEMMON, J.; MEREL, J.; WAYNE, G.; TASSA, Y.; EREZ, T.; WANG, Z.; ESLAMI, S. M. A.; RIEDMILLER, M.; SILVER, D. Emergence of locomotion behaviours in rich environments. july 2017.

HO, J.; ERMON, S. Generative adversarial imitation learning. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2016. p. 4565â4573.

LEVNER, D. **Is Brute Force Backgammon Possible?** 1976. Disponível em: <www.bkgm.com/articles/Levner/BruteForceBackgammon/>.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. **CoRR**, abs/1509.02971, 2015. Disponível em: <http://arxiv.org/abs/1509.02971>.

LIN, H. W.; TEGMARK, M.; ROLNICK, D. Why does deep and cheap learning work so well? **Journal of Statistical Physics**, Springer, v. 168, n. 6, p. 1223–1247, 2017.

LIN, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. 1992.

MATIISEN, T.; OLIVER, A.; COHEN, T.; SCHULMAN, J. Teacher-student curriculum learning. **CoRR**, abs/1707.00183, 2017. Disponível em: <http://arxiv.org/abs/1707.00183>.

MEREL, J.; TASSA, Y.; TB, D.; SRINIVASAN, S.; LEMMON, J.; WANG, Z.; WAYNE, G.; HEESS, N. Learning human behaviors from motion capture by adversarial imitation. july 2017.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529–533, Feb 2015. ISSN 0028-0836. Letter. Disponível em: <http://dx.doi.org/10.1038/nature14236>.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: **Proceedings of the 27th International Conference on International Conference on Machine Learning**. USA: Omnipress, 2010. (ICML'10), p. 807–814. ISBN 978-1-60558-907-7. Disponível em: <http://dl.acm.org/citation.cfm?id=3104322.3104425>.

PENG, X. B.; ABBEEL, P.; LEVINE, S.; PANNE, M. V. D. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. April 2018.

SCHULMAN, J.; LEVINE, S.; MORITZ, P.; JORDAN, M. I.; ABBEEL, P. Trust region policy optimization. **CoRR**, abs/1502.05477, 2015. Disponível em: <http://arxiv.org/abs/1502.05477>.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. **CoRR**, abs/1707.06347, 2017. Disponível em: <http://arxiv.org/abs/1707.06347>.

SILVER, D.; SCHRITTWIESER, J.; SIMONYAN, K.; ANTONOGLOU, I.; HUANG, A.; GUEZ, A.; HUBERT, T.; BAKER, L.; LAI, M.; BOLTON, A.; CHEN, Y.; LILLICRAP, T.; HUI, F.; SIFRE, L.; DRIESSCHE, G. van den; GRAEPEL, T.; HASSABIS, D. Mastering the game of go without human knowledge. **Nature**,

Macmillan Publishers Limited, part of Springer Nature. All rights reserved., v. 550, out. 2017.

SUTTON, R. S.; BARTO, A. G. **Introduction to Reinforcement Learning**. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981.

SWIRSZCZ, G.; CZARNECKI, W. M.; PASCANU, R. Local minima in training of neural networks. **stat**, v. 1050, p. 17, 2017.

TROMP, J.; FARNEBÃ¤CK, G. Combinatorics of go. 2016. Disponível em: <https://tromp.github.io/go/gostate.pdf>.

WANG, Z.; MEREL, J.; REED, S.; WAYNE, G.; FREITAS, N. de; HEESS, N. Robust imitation of diverse behaviors. july 2017.

WATKINS, C. J. C. H. **Learning from Delayed Rewards**. Tese (Doutorado) — King's College, 1989.

WILIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. 1992.

XU, R. H. B.; LI, M. Revise saturated activation functions. 2016.

ZAREMBA, W.; SUTSKEVER, I. Learning to execute. **CoRR**, abs/1410.4615, 2014. Disponível em: <http://arxiv.org/abs/1410.4615>.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
TC	4 de marï¿½o de 2015	DCTA/ITA/DM-018/2015	40

5. TÍTULO E SUBTÍTULO:

Simulated Humanoid Robot Control with Reinforcemente Learning

6. AUTOR(ES):

Luis Guilherme Gomes Aguiar

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnolï¿½gico de Aeronï¿½utica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Cupim; Cimento; Estruturas

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Cupim; Dilema; Construï¿½ï¿½o

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA, Sï¿½o Josï¿½ dos Campos. Curso de Mestrado. Programa de Pï¿½s-Graduaï¿½ï¿½o em Engenharia Aeronï¿½utica e Mecï¿½nica. ï¿½rea de Sistemas Aeroespaciais e Mecatrï¿½nica. Orientador: Prof. Dr. Adalberto Santos Dupont. Coorientadora: Profª. Drª. Doralice Serra. Defesa em 05/03/2015. Publicada em 25/03/2015.

11. RESUMO:

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**