Dissertação apresentada à Pró-Reitoria de Pós-Graduação e Pesquisa do Instituto Tecnológico de Aeronáutica, como parte dos requisitos para obtenção do título de Mestra em Ciências no Programa de Pós-Graduação em Engenharia Aeronáutica e Mecânica, Área de Sistemas Aeroespaciais e Mecatrônica.

André Marcello Soto Riva Figueira

REINFORCEMENT LEARNING APPLIED TO THE SOCCER 2D KEEPER

Dissertação aprovada em sua versão final pelos abaixo assinados:

Prof. Dr. Manga

Orientador

Prof[a]. Dr[a]. Doralice Serra

Coorientadora

Prof. Dr. John von Neumann

Pró-Reitor de Pós-Graduação e Pesquisa

Campo Montenegro
São José dos Campos, SP - Brasil
2019

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Soto Riva Figueira, André Marcello
Reinforcement Learning applied to the Soccer 2D keeper / André Marcello Soto Riva Figueira. São José dos Campos, 2019.
43f.

Dissertação de Mestrado – Curso de Engenharia Aeronáutica e Mecânica. Área de Sistemas Aeroespaciais e Mecatrônica – Instituto Tecnológico de Aeronáutica, 2019. Orientador: Prof. Dr. Manga. Coorientadora: Prof[a]. Dr[a]. Doralice Serra.

1. Cupim. 2. Dilema. 3. Construção. I. Instituto Tecnológico de Aeronáutica. II. Título.

REFERÊNCIA BIBLIOGRÁFICA

SOTO RIVA FIGUEIRA, André Marcello. **Reinforcement Learning applied to the Soccer 2D keeper**. 2019. 43f. Dissertação de Mestrado – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DA AUTORA: André Marcello Soto Riva Figueira
TITULO DO TRABALHO: Reinforcement Learning applied to the Soccer 2D keeper.
TIPO DO TRABALHO/ANO: Dissertação / 2019

André Marcello Soto Riva Figueira
Av. Cidade Jardim, 679
12.233-066 – São José dos Campos–SP

REINFORCEMENT LEARNING APPLIED TO THE SOCCER 2D KEEPER

André Marcello Soto Riva Figueira

Composição da Banca Examinadora:

Prof. Dr.	Alan Turing	Presidente	-	ITA
Prof. Dr.	Manga	Orientador	-	ITA
Prof[a]. Dr[a].	Doralice Serra	Coorientadora	-	OVNI
Prof. Dr.	Linus Torwald		-	UXXX
Prof. Dr.	Richard Stallman		-	UYYY
Prof. Dr.	Donald Duck		-	DYSNEY
Prof. Dr.	Mickey Mouse		-	DISNEY

ITA

Aos amigos da Graduação e Pós-Graduação do ITA por motivarem tanto a criação deste template pelo Fábio Fagundes Silveira quanto por motivarem a mim e outras pessoas a atualizarem e aprimorarem este excelente trabalho.

Agradecimentos

Primeiramente, gostaria de agradecer ao Dr. Donald E. Knuth, por ter desenvolvido o TEX.

Ao Dr. Leslie Lamport, por ter criado o LATEX, facilitando muito a utilização do TEX, e assim, eu não ter que usar o Word.

Ao Prof. Dr. Meu Orientador, pela orientação e confiança depositada na realização deste trabalho.

Ao Dr. Nelson D'Ávilla, por emprestar seu nome a essa importante via de trânsito na cidade de São José dos Campos.

Ah, já estava esquecendo... agradeço também, mais uma vez ao TEX, por ele não possuir vírus de macro :-)

"If I have seen farther than others,
it is because I stood on the shoulders of giants."
— Sir Isaac Newton

Resumo

Aqui começa o resumo do referido trabalho. Não tenho a menor idéia do que colocar aqui. Sendo assim, vou inventar. Lá vai: Este trabalho apresenta uma metodologia de controle de posição das juntas passivas de um manipulador subatuado de uma maneira subótima. O termo subatuado se refere ao fato de que nem todas as juntas ou graus de liberdade do sistema são equipados com atuadores, o que ocorre na prática devido a falhas ou como resultado de projeto. As juntas passivas de manipuladores desse tipo são indiretamente controladas pelo movimento das juntas ativas usando as características de acoplamento da dinâmica de manipuladores. A utilização de redundância de atuação das juntas ativas permite a minimização de alguns critérios, como consumo de energia, por exemplo. Apesar da estrutura cinemática de manipuladores subatuados ser idêntica a do totalmente atuado, em geral suas caraterísticas dinâmicas diferem devido a presença de juntas passivas. Assim, apresentamos a modelagem dinâmica de um manipulador subatuado e o conceito de índice de acoplamento. Este índice é utilizado na sequência de controle ótimo do manipulador. A hipótese de que o número de juntas ativas seja maior que o número de passivas ($n_a > n_p$) permite o controle ótimo das juntas passivas, uma vez que na etapa de controle destas há mais entradas (torques nos atuadores das juntas ativas), que elementos a controlar (posição das juntas passivas).

Abstract

Soccer 2D is a simulated league, two teams of 11 intelligent autonomous agents play a soccer game in two dimensions. Each agent represents a player, receives limited information about the game situation and under them has to decide which action upon many to take. This simulation has it's own version of penalty and the aim of this work is to develop an optimized behavior for the goalkeeper under this situation utilizing Deep Reinforcement Learning techniques.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CTq	computed torque
DC	direct current
EAR	Equação Algébrica de Riccati
GDL	graus de liberdade
ISR	interrupção de serviço e rotina
LMI	linear matrices inequalities
MIMO	multiple input multiple output
PD	proporcional derivativo
PID	proporcional integrativo derivativo
PTP	point to point
UARMII	Underactuated Robot Manipulator II
VSC	variable structure control

Lista de Símbolos

a	Distância
$\mathbf{a}$	Vetor de distâncias
$\mathbf{e}_j$	Vetor unitário de dimensão n e com o j-ésimo componente igual a 1
$\mathbf{K}$	Matriz de rigidez
m_1	Massa do cumpim
δ_{k-k_f}	Delta de Kronecker no instante k_f

Sumário

1 Introduction

1.1 Motivation

In an afternoon of 2017, humanity had an unlikely champion. Ke Jie, the number one player in GO, was to battle against one of the world's most intelligent machines, AlphaGo; a powerhouse of artificial intelligence backed by one of the world's top technology company: Google. Since the number of possible positions on a Go board exceeds the quantity of atoms in the universe, defeating the world champion was the biggest deed possible for AI researchers at the time. Romantics said it was impossible for a machine to do so as it lacked spiritual power only a human could possess, engineers would agree but by a different reason, they pointed out for the fact that there was just too many possibilities for a computer to evaluate; these "theories" fell apart though. On this day, what was witnessed was not a clash between two equal titans, AlphaGo completely dominated Ke Jie. Over the course of three marathon matches of more than three hours each, the top ranked player tried every single approach: conservative, aggressive, defensive and unpredictable. Nothing worked, there were no openings, not a single sign of hope.

1.1.1 AI Super Powers

As described by (LEE, 2018), this event is the equivalent of Sputnik for China. Less than two months after Ke Jie resigned his last game to AlphaGo, the central government of China issued an ambitious plan to build artificial intelligence capabilities. China designed a clear strategy, set benchmarks for progress by 2020 and 2025 aiming to be the center of global innovation in artificial intelligence. Private money answered that call, Chinese venture-capital investors had already responded to that call, unleashing record amounts into AI startups and making up 48 percent of all AI venture funding globally, surpassing the United States for the first time.

Less than two months after Ke Jie resigned his last game to AlphaGo, the Chinese central government issued an ambitious plan to build artificial intelligence capabilities. It called for greater funding, policy support, and national coordination for AI development.

FIGURA 1.1 – Ke Jie faces AlphaGo

It set clear benchmarks for progress by 2020 and 2025, projected that by 2030 China would become the center of global innovation in artificial intelligence, leading in theory, technology, and application. By 2017, Chinese venture-capital investors had already responded to that call, pouring record sums into artificial intelligence startups and making up 48 percent of all AI venture funding globally, surpassing the United States for the first time.

The fact that the current global economic power and the one aspiring to be are massively investing on AI technology points at first hand that the topic is worth studying for. The State of AI in the Enterprise 2nd Edition conducted by Deloitte (LOUCKS DAVID SCHATSKY, 2018) gives some numbers on how AI is making the industry grow: "To obtain a cross-industry view of how organizations are adopting and benefiting from cognitive computing/AI, Deloitte surveyed 1,100 IT and line-of-business executives from US-based companies in Q3 2018. All respondents were required to be knowledgeable about their company's use of cognitive technologies/artificial intelligence, and 90 percent have direct involvement with their company's AI strategy.". According to it, 82 per cent of the survey participants claim a positive financial return on their AI investment and 88 per cent plan to increase spending in 2019; 79 per cent agree that AI technologies empower people to make better decisions and 73 per cent believe AI will increase job satisfaction. Furthermore, the graphic 1.3 shows that around 75 per cent reported positive results when comparison to direct competitors was made.

What are the results AI can bring to justify this current fever? (AGRAWAL; GOLDFARB, 2018) puts it simply, AI is a technology that optimizes and facilitates decisions and

FIGURA 1.2 – Ke Jie is ultimately defeated

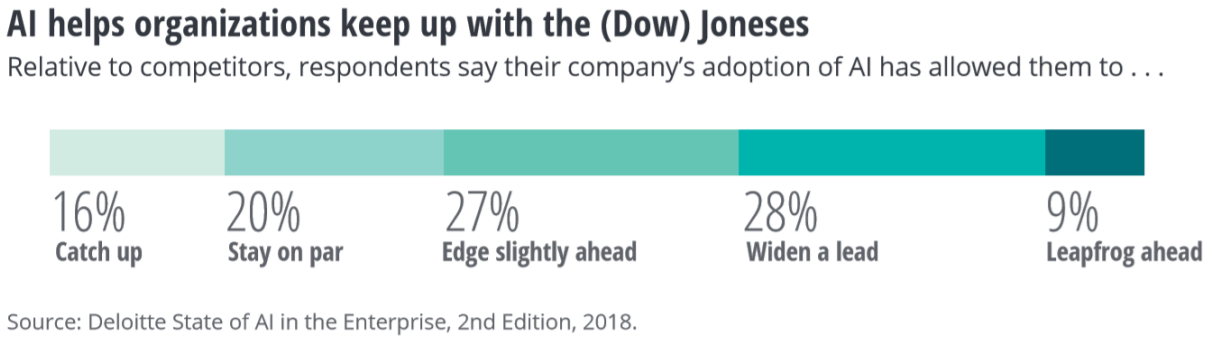


FIGURA 1.3 – Distribution of answers of 1,100 candidates from Deloitte's survey about AI in the Enterprise on how the technology levered their business production in comparison to their competitors

predictions. For example, the article (LENTINO, 2019) talks about the Shanghai-based YITU Technology that has gained wide recognition for its Dragonfly Eye System, a facial scanning platform that can identify a person from a database of at least 2 billion people in a matter of seconds .(CHAPMAN, 2019) talks about Zymergen, a company that uses machine learning to navigate the genomic search space in order to guide scientists to the precise set of genetic changes required to engineer cells to make a product of interest or to do it more efficiently. The applications are too numerous to cite them all but these two examples represent how broad the technology can go and how the future is being passed by it. The graphic 1.4 is from (LOUCKS DAVID SCHATSKY, 2018) and it shows more specifically how AI generate

1.1.2 Deep Learning Revolution

Machine learning , despite being this new driver of the future, is lucky to have survived a tumultuous half-century of research. For each great promising new lead there were big voids too, when weak practical results led to investments retraction.

There were 2 camps in the field: the "rule-based" approach and the "neural networks" approach. Researchers of the first attempted to teach computers to think by encoding a series of logical rules (If Then and Else). This method worked well for simple and well defined problems for example a game of Sudoku but when the universe of options was too large it did not respond well. Their insight was too question experts on the problem at hand and try to embody their wisdom into the machine (Hence why this method is also called the "expert systems").

The "neural network" approach goes a different route by trying to reconstruct the brain itself, taking inspiration on the one and only source of intelligence we could perceive. This approach mimics the neurological architecture of the biological brain and instead of trying to transfer knowledge by force like the first one, this approach just feeds the neural networks with a plenitude of data and lets it find patterns on it.

(LEE, 2018) gives a simple illustration on how the techniques differ, if it is desired for the computer to recognize the picture of a cat, the "rule based" approach would work by trying, for starters, to embed the machine with the idea that a cat has 2 triangles above a circle. In the "neural network" approach, the machine would be fed with tones of pictures of cats and non-cats for it to build the features of what is a cat on its own.

To make things short, promising results were achieved during 1950s and 1960 by artificial neural networks but its core methodology found 2 problems that could not be solved at the time, first it needed a lot of data and second a lot of computational power. It quickly went out of fashion then and AI entered into one of its first "winters" during the 1970s.

It was not just the increased computing force that came with time which reanimated neural networks, a huge chunk of credit must be given to the optimization in the algorithm itself discovered by leading researcher Geoffrey Hinton and his Convolution Neural Networks (CITAR PAPER ORIGINAL). This new empowered neural network-now rebranded as "deep learning"- could outperform older models at a variety of tasks. However, the real demonstration occurred in 2012, when a neural network built by Hinton's team overpowered the competition in an international computer vision contest.

1.1.3 Reinforcement Learning

Although companies have been using AI, they are mostly using them to make predictions as written in (AGRAWAL; GOLDFARB, 2018), in other words, they do things like predict which product a costumer might want given an history of purchases or which size of clothing best suits the client. These are not necessarily and often not done by Reinforcement Learning(A branch of Machine Learning), which is the precise topic of this thesis.

Although still not fully incorporated into corporations, Reinforcement Learning is not to be neglected for it is paving the road of more powerful innovations, here are some recent papers showing the technique applications.

- **Resources Management in Computer Clusters**: The paper (MAO MOHAMMAD ALIZADEH; KANDULA, 2016) Learning) showed how to use RL to automatically learn to allocate and schedule computer resources to waiting jobs, with the objective to minimize the average job slowdown.
- **Traffic Signal Control** In the paper (WEI GUAMJIE ZHENG; LI, 2018), researchers built a traffic light controller to solve congestions problems. In simulations, their methods showed great improvement from the state-of-art techniques; they also accomplished decent results in real scenarios.
- **Optimizing Chemical Reactions** In the paper (ZHOU; N.ZARE, 2018), researchers built a mode that iteratively records the results of a chemical reaction and chooses new experimental conditions to improve the reaction outcome. outperforming a state-of-the-art black box optimization algorithm by using 71% fewer steps on both simulations and real reactions.

1.2 Contextualization

RoboCup is an international scientific community aiming to advance the state of the art of intelligent robots. Its mission is that a team of androids will be able to beat the human team champion of the World Cup until the year 2050 (ROBOCUP,). To achieve this goal, there is actually a plenitude of tasks to be solved, if you actually enumerate what a professional player must know, the difficulty stacks up.

For example, to kick a ball, a thing we humans almost take for granted, there is a complex mixture of muscle movements and positioning that a robot would have to mimic. Of course when you think about soccer you think about kicking but there is way more: to

carry the ball a certain distance, to pass, to tackle, to receive a pass; these are all complex set of movements that requires good hardware and software coordination.

Soccer is not a pure athletic sport though, that means victory is not only decided by being more able, stronger or faster. It is a versus sport, the players must decide how to act in different conditions, when to run, where to position themselves and what movement to make according to the enemy same actions.

1.2.1 Simulated Soccer 2D

Directly from the RoboCup official site (ROBOCUP,): "In the 2D Simulation League, two teams of eleven autonomous software programs (called agents) each play soccer in a two-dimensional virtual soccer stadium represented by a central server, called Soccer-Server. This server knows everything about the game, i.e. the current position of all players and the ball, the physics and so on. The game further relies on the communication between the server and each agent. On the one hand each player receives relative and noisy input of his virtual sensors (visual, acoustic and physical) and may on the other hand perform some basic commands (like dashing, turning or kicking) in order to influence its environment."

In other words, this category focus on the behavior of a soccer player and disregard body-physics implementations.

1.3 Objective

The objective of this bachelor's thesis is to use Deep Learning in order to optimize the behavior of the goalkeeper agent in Simulated Soccer 2D, under the more constricted penalty scenario in which the goalkeeper faces in a one-on-one dispute against an attacker that starts with the ball and is fairly distanced from the goalkeeper whose initial position lies in the goal area. Currently, unlike real soccer game, the rules accept that the goalkeeper can move freely during the penalty.

1.4 Scope

The scope of this work is Reinforcement Learning, to use *Actor-Critic Method* with *PPO*(Proximal Policy Optimization).

1.5 Organization of this work

- On chapter 2 a brief overview of the soccer 2D and the task at hand is made.
- On chapter 3 the theory of Deep Learning is presented.
- On chapter 4 a plan of execution is scheduled.

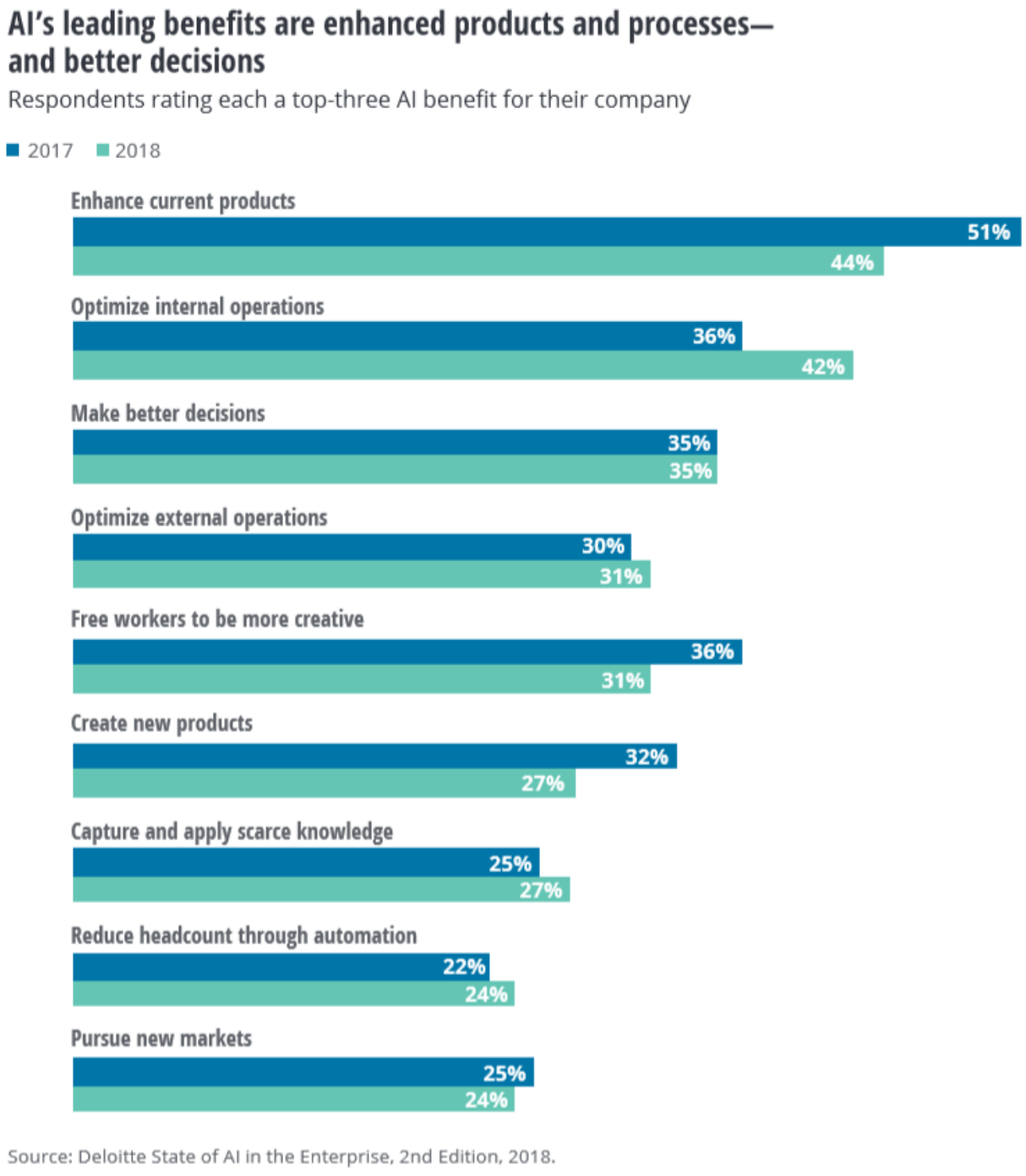


FIGURA 1.4 – How AI brings benefit according to the surveyed candidates from Deloitte report

FIGURA 1.5 – Simulated Soccer 2D League

2 Soccer 2D

Here the important aspects of the problem at hand will be described briefly. Since the penalty event is a more closed scenario than the whole game, only a subset of the game will be mentioned, that is, the subset that is relevant to the penalty shootouts.

2.1 How the Simulator Works

In the football simulator, the program which is in charge of the actual simulation is *rcssserver*.In the RoboCup soccer simulator, distributed multi agent simulation is realized by server client method. The soccer agent operating with the RoboCup soccer simulator is a client program that communicates with the rcssserver. The agents receives information from the server about it's state and the agent under light of this information chooses an action and sends his choice to the serve, the server will then update the agent state and repeat the process.

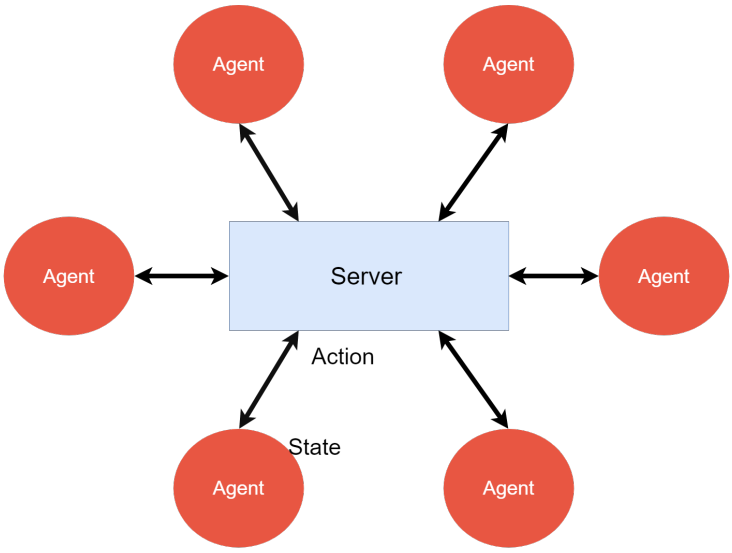


FIGURA 2.1 – Simple server-agent interaction diagram

2.2 Physical model of rcssserver

Here is described how the physical model of the environment operates.

2.2.1 Movement of the object

When rcssserver's simulation cycle is updated, information on object movement is updated in the following order.

1. The acceleration of the agents are corrected so they do not overflow their respective maximum acceleration value.
2. Add the acceleration vector to velocity vector.
3. The velocity of the agents are corrected so they do not overflow their respective maximum velocity value.
4. Add noise to velocity vector.
5. Check collision with goal post.
6. Add velocity vector to position vector.
7. Decay speed
8. Set the acceleration to zero.
9. After updating all objects position, check for collisions.

The position and speed will be updated in this procedure for both the ball and the player.

2.3 Player Model

2.3.1 Available Action Commands

When the player agent makes a decision, it sends the action command necessary to realize the action to rcssserver, rcssserver accepts the following commands as player's actions.

These commands move the body of the player agent.The command to move the body can be exclusively executed once per cycle, for example, when the rcssserver accepts the

TABELA 2.1 – Agent possible actions

Action	Description
kick	Accelerate the ball in any direction.
dash	Accelerate yourself toward the body.
turn	Rotate the direction of the body.
tackle	Accelerate the ball toward the body.
catch	(Only keeper) Catch the ball by hand.
move	Move instantaneously to the specified position.

TABELA 2.2 – Agent possible auxiliary actions

Auxiliary Action	Description
turn neck	Rotate the neck and change the direction of vision.
change view	Change the viewing mode.
turn	Generate a communication message.
say	Point to a specific position.
point to	Pay attention to communication from specific players.

kick command from the player, the cycle ends That player will not be able to execute the new body moving command until you can not move it diagonally because you can not use dash and turn at the same time.In addition the following auxiliary action command can be used.

2.4 The Penalty Event

In the case of a tie, the match may be decided by penalty shootouts. In this event, an agent is set at a certain distance from the goal under possession of the ball while the keeper start at the goal line. After a trigger start, the attacker and the keeper may move and act freely, different from the real soccer game where in the penalty shootout event the shooter may only shoot and the keeper may only move above the goal line. So the possible actions the keeper may take are: *dash* , *turn* and *catch* and the auxiliary action *turn neck*

3 Deep Reinforcement Learning

Here a survey of the theory behind the work will be exposed but in a briefly matter. The Markov Decision Process section was based on the book (SUTTON, 2018) and on (UC..., 2014). The Neural Network section was derived from (NIELSEN, 2015).

3.1 Markov Decision Processes

A MDP (Markov Decision Process) is a mathematical model of the reinforcement learning problem. It is composed by an agent, the decision maker, and the thing it interacts with, the environment. These two interact continually, with the environment presenting a situation to the agent in the form of a state, and the agent selecting an action in regards to this state. This same environment also presents the agent with numerical rewards for each state accessed, the goal of the agent is to maximize the sum of these rewards by making good choices of actions.

In short, the MDP is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function $T(s, a, s')$
  - Probability that a from s leads to s', i.e., $P\left(s' \mid s, a\right)$
  - Also called the model of the dynamics
- A reward function $R(s)$
- A start state
- Maybe a terminal state

For each time step, the environment returns its current representation in the form of a state s. The agent then selects an action and the transition function $T(s, a, s')$ returns

the probability of reaching state s' from state s given the action chosen. On the next time step, the agents finds itself on a new state.

3.1.1 Important Concepts

Here a list of important concepts of a MDP is shown.

3.1.1.1 Reward Function

At each reached state, the agent accumulates a reward returned by a reward function $R(s)$. The purpose of this functions is to guide the agent, good states are scored with positive values, while bad states are scored with negative values. By making the agent act on the desire of maximizing his reward score, the agent will act optimally as it is intentioned.

3.1.1.2 Transition Function

The MDP is not necessarily deterministic, after an agent chooses an action the next state is ruled by a probability distribution. The transaction function $T(s, a, s')$ return the probability that an agent in state s performing action a will land in state s'.

3.1.1.3 Discounting

In a MDP, it's reasonable to maximize the sum of rewards but it's also reasonable to make preference over rewards now than later. One solution is to make rewards decay exponentially in value. So for each episode, all rewards available are multiplied by a γ factor as illustrated by 3.1. This factor is called discount factor.

FIGURA 3.1 – Illustrating of the discount factor, image from http://ai.berkeley.edu

3.1.1.4 Utility

Utility is, by definition, the sum of the discounted rewards for a given trajectory of the agent.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.1}$$

In recursive form:

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{3.2}$$

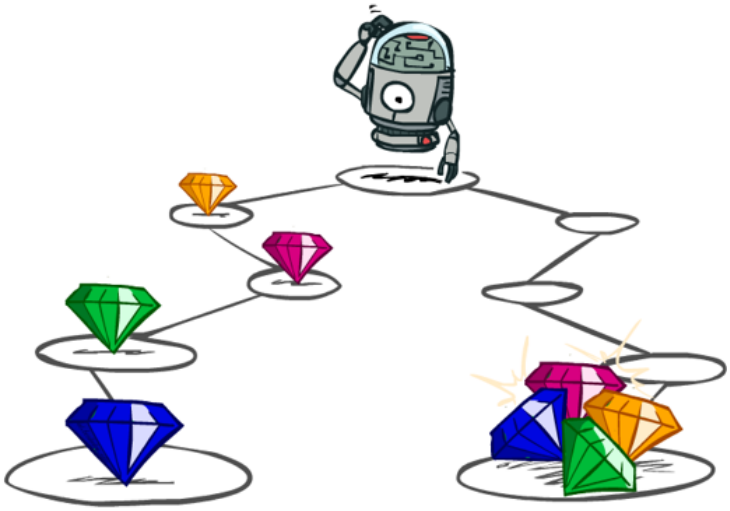


FIGURA 3.2 – Illustration of a Reward and Utility, image from http://ai.berkeley.edu

3.1.1.5 Policy

For MDPs, an *optimal policy* $\pi^* : S \rightarrow A$ is desired. To put it simply, a *policy* π works as a map of instructions. Given a state s, the policy return which action a to take. Therefore, an optimal policy has an action for each state that maximizes the expected utility when followed.

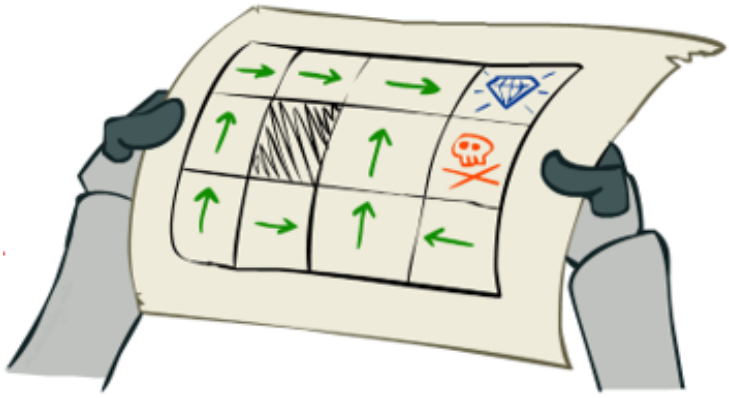


FIGURA 3.3 – Illustration of a policy, image from http://ai.berkeley.edu

3.1.1.6 Value Function

The *value function* of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. Formally:

$$v_\pi(s) \doteq \mathbf{E}_\pi\left[G_t \mid S_t = s\right] \tag{3.3}$$

where $\mathbf{E}_\pi$ represents the expected value of a random variable given that the agent follows policy π.

3.1.1.7 Q-State

A Q-State is a tuple formed by a state s and an action a.

3.1.1.8 Action Value Function

Just like the Value Function, the *action value function* defines value for the Q-States. The value of taking action a in state s under the policyπ, denoted $q_\pi(s,a)$, as the expected return starting from s, taking the action a, and thereafter following policy π. Formally:

$$q_\pi(s,a) \doteq \mathbf{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \tag{3.4}$$

3.1.2 The Bellman Equations

3.1.2.1 Optimal Quantities

- $V^*(s)$ is the expected utility starting in s and acting optimally.
- $Q^*(s,a)$ is the expected utility starting out having taken action a from state s and (thereafter) acting optimally.
- $\pi^*(s)$ returns the optimal action from state s

3.1.2.2 The Equations

$$V^*(s) = \max_a Q^*(s,a) \tag{3.5}$$

$$Q^*(s,a) = \tag{3.6}$$

3.1.3 About Solving It

To make things clear, the solution required is the optimal policy, the set of instructions that maximizes the utility. If the transition function is know and well defined than the solution is easily found by means of dynamic programming, no reinforcement learning involved. Of course, in the real world, this transition function cannot be know a priori, that's where reinforcement learning comes.

The most basic reinforcement learning algorithms have the agent updates states values in a series of trainings, multiple sessions of sampling substitutes the transition function. The idea is to have the agent explore the environment by doing, at first, random actions; but as time passes and almost all states are decently evaluated, it progressively abandons the random nature of exploration in favor of a greedy behavior in which the agent chooses the action that will return the best benefit, ergo the best reward.

In Q-Learning algorithm, for example, the Q-States values are updated by the following expression 3.7:

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q\left(s', a'\right)\right] \tag{3.7}$$

where α is a factor that averages the current value with the next value. The expression is very intuitive, the Q value is updated by an average between its current value and the reward the state gives plus the discounted maximum Q-Value accessible by the current state.

By doing a series of training, the Q-Values will converge and then the optimal policy may be extracted by using 3.8:

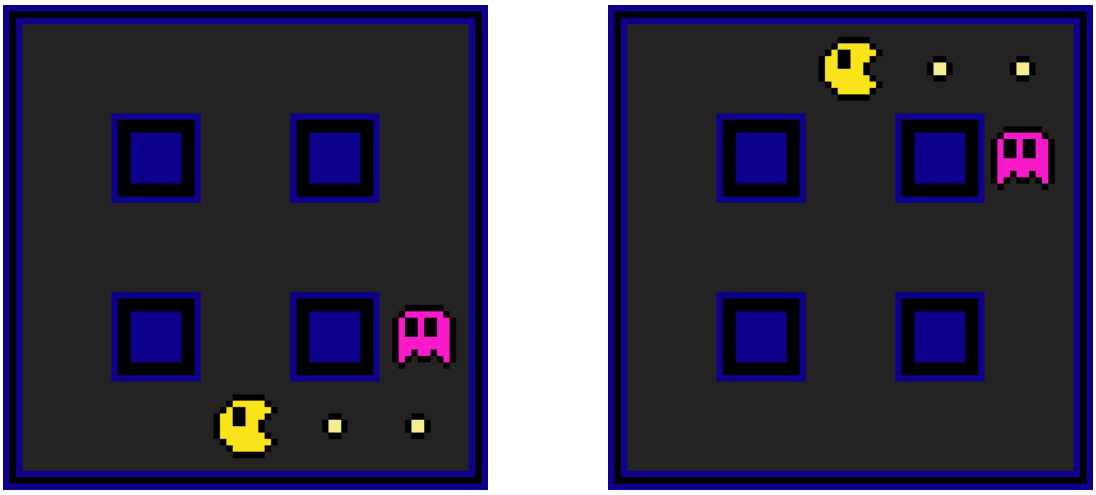
$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{3.8}$$

3.1.4 The Problem With The Tabular Solution

Q-Learning solves the MDP but it's solution is brute force styled (Without optimizations that is). That's because the algorithm contemplates all states and it's possible actions without any scrutiny. For real world problems, that's not computational feasible. For instance, Go has a number of possible situations greater than the number of atoms in the whole observable universe.

A great improvement is to make state approximations. For example, if you look at a Pac-Man game, each configuration on the board is a different state, move a ghost or Pac-Man just a bit and a whole new state is reached. The insight here is that some

states are actually very close to each other. All states that has Pac-Man moving close to a ghost (which results in PacMan's death) are equally bad states to be avoided 3.4. So there is actually no need to register all these similar states if you can extract crucial information from them and hereby decide if they are bad or not. What is wanted is that the machine, by looking at these relevant informations, learns what they mean in order to classify states.



(a) 1a (b) 1b

FIGURA 3.4 – The two Pac-Man states are different yet for what it matters they are both states to be avoided in equal magnitude.

The role of function approximation belongs to Neural Networks in Deep Learning.

3.2 Neural Networks

A Neural Network is a learning model whose goal is essentially function approximation. For example, for a classifier, $y = f^*(x)$ maps an input x to a category y. A neural network defines a mapping $y = f(x, \theta)$ and learns the value of the parameters θ that result in best function approximation. They are called networks because they are commonly represented by composing together many different functions. The model is illustrated by a directed acyclic graph describing how the functions are composed together. For example, there could be three functions f_1, f_2 and f_3 connected in a chain, to form $f(x) = f_3\Big(f_2\big(f_1(x)\big)\Big)$.

3.2.1 Neurons

Neurons are the single unit from a Neural Network. They are an abstraction of a vector of *weights* $\mathbf{w}$ and a number b called *bias*. Upon receiving an input $\mathbf{x}$, the neuron

outputs:

$$z = \mathbf{w} \cdot \mathbf{x} + b \tag{3.9}$$

Actually, because the values of $\mathbf{w}$ and b are supposed to be changed by a learning process, it is important that the output changes in linear form in relation to them so these changes may actuate smoothly. To achieve it, the *sigmoid function* is often used, the final output is therefore:

$$\text{Neuron output} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.10}$$

where z is extracted from 3.9.

3.2.2 The Network

In graph representation, a neuron is a vertex and each incoming edge is a weight in the weight vector $\mathbf{w}$. Networks are composed of layers of neurons, the first layer receives the initial input while the subsequent layers receive input from their predecessor layer. The last layer returns the final output.

3.2.3 Cost Function and Gradient Descent

After a final output is received from the last layer, there is need to check how adequate it is and at the some time use this result to better balance the weights and biases. For that, it is defined a *cost function*. The simplest example is the *quadratic cost function*:

$$C(w, b) \equiv \frac{1}{2n} \sum_{\mathbf{x}} \left\| y(\mathbf{x}) - \mathbf{a} \right\|^2 \tag{3.11}$$

Here,$y(x)$ denotes the desired output, w the collection of all weights in the network, b all the biases, n is the total number of training inputs, $\mathbf{a}$ is the vector of outputs from the network when $\mathbf{x}$ is input, and the sum is over all training inputs, $\mathbf{x}$.

The more the desired output is different from the received output, the higher the cost function value will be.Therefore, the true objective is to manipulate the values of weights and biases in order to shrink the cost function returning value. This is achieved by *gradient descent* (). By calling $\mathbf{W}$ the collection of weights and biases, the update rule would be:

$$\mathbf{W}' = \mathbf{W} - \epsilon \frac{\nabla C(\mathbf{W})}{\left\| \nabla C(\mathbf{W}) \right\|} \tag{3.12}$$

where ϵ is a step size.

The algorithm that applies this update rules efficiently are called *Back Propagation*

Algorithms

3.2.4 Back Propagation

Back Propagation allows the calculation of $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$ of a single training sample. Then $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$ is recovered by averaging over the samples.

3.2.4.1 Notation

- w^l_{jk} denotes the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.
- b^l_j is the bias of the j^{th} neuron in the l^{th} layer.
- z^l_j is the output of the j^{th} neuron in the l^{th} layer **before** being processed by the sigmoid function.
- a^l_j is the output of the j^{th} neuron in the l^{th} layer **after** being processed by the sigmoid function, ergo $\sigma(z)$

3.2.4.2 Neuron Error Definition

A change Δz^l_j in z^l_j propagates through the layers, causing the overall cost to change by an amount $\frac{\partial C_x}{\partial z^l_j}\Delta z^l_j$. The higher the value of $\frac{\partial C_x}{\partial z^l_j}$, the more the cost function can be adjusted, if it is low than the neuron cannot improve the cost function by much. Therefore, in an heuristic sense, $\frac{\partial C_x}{\partial z^l_j}$ works as a measure of error in the neuron. The *neuron error* is than defined as:

$$\delta^l_k \equiv \frac{\partial C_x}{\partial z^l_j} \tag{3.13}$$

$\delta^{\mathbf{l}}$ will denote the vector of errors associated with layer l. Back propagation allows the computing of $\delta^{\mathbf{l}}$ for every layer, and then relating those errors to the quantities of real interest, $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$.

3.2.4.3 Back Propagation Equations

First there is need for an equation for the error of the nodes in the last layer, that will serve as a base for the propagation. **Under the definition from 3.13 and by applying the chain rule from Calculus, the neuron error for the last layer nodes is:**

$$\delta^L_k = \frac{\partial C_x}{\partial a^L_j}\,\sigma'\left(z^L_j\right) \tag{3.14}$$

where L represents the last layer number. It may be important to note that $\frac{\partial C_x}{\partial a_j^L}$ should be easy to compute which is the case for the quadratic cost function, other cost functions should maintain this property as well. A better representation of 3.14 is in matrix form:

$$\delta^{\mathbf{L}} = \nabla_a C_x \odot \sigma'\left(\mathbf{z^L}\right) \tag{3.15}$$

where $\odot$ represents the Hadamart Product (CITAR).

Next, an equation for the error $\delta^{\mathbf{l}}$ in terms of the error in the next layer, $\delta^{\mathbf{l+1}}$:

$$\delta^{\mathbf{l}} = \left(\left(w^{l+1}\right)^T \delta^{\mathbf{l+1}}\right) \odot \sigma'\left(z_j^L\right), \tag{3.16}$$

where $\left(w^{l+1}\right)^T$ is the transpose of the weight matrix $\left(w^{l+1}\right)$ for the $(l+1)^{th}$ layer. Using these two equations, 3.15 and 3.16, the algorithm can compute all neuron errors. Now what is left is how to calculate $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$ using these errors.

An equation for the rate of change of the cost with respect to any bias in the network:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \tag{3.17}$$

An equation for the rate of change of the cost with respect to any weight in the network:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l. \tag{3.18}$$

So by using equations 3.15, 3.16, 3.17 and 3.18; you can compute $\nabla\mathbf{W}$ which represents the vector that contains all partial derivatives of weights and biases together. Finally, by using 3.12 you can update all weights and biases for a single training.

3.2.4.4 Neural Networks in Reinforcement Learning

This section is to point out briefly that in reinforcement learning there is no actually correct output $y(x)$ used in the cost function 3.11. The *reward* concept substitutes the error one. So in reinforcement learning, the neural network tuning algorithms take in another form which will be presented next.

(citar michal neuron netowrks)

3.3 Policy Gradient Methods, PPO

Q-Learning is an *action-value* method, the actions are chosen by evaluating the action value in a given state. A *policy gradient* method learns a parameterized policy that cant

select actions without consulting a value function. A value function may still be used to learn the policy parameter, but is not required for action selection.

3.3.1 Notation and Concepts

Here are some new basic concepts about policy learning.

3.3.1.1 The parameters $\boldsymbol{\theta}$

The policy parameter vector is $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ thus $\pi\left(a \mid s, \theta\right) = \mathbb{P}\left\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\right\}$ represents the probability that action a is taken at time t given that the environment is in state s at time t with parameter $\boldsymbol{\theta}$. This parameter $\boldsymbol{\theta}$ vector is the weights and biases of a neural network.

3.3.1.2 The Score Function

Learning the policy parameter is based on the gradient of some scalar performance measure $J\left(\boldsymbol{\theta}\right)$ with respect to the policy parameter. This function is called *score function* as the aim is to maximize it (unlike the cost function). Therefore, their updates approximate gradient ascent in J:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha\widehat{\nabla J\left(\boldsymbol{\theta_t}\right)} \tag{3.19}$$

where $\widehat{\nabla J\left(\boldsymbol{\theta_t}\right)}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument $\boldsymbol{\theta_t}$.

3.3.2 Policy Approximation and its Advantages

Policy gradient offer advantages that justify its use, in general these methods are often better than value based ones. Here are the main advantages:

- Q-Learning cannot learn stochastic policies, a necessity in some environments. This also means that it needs its own exploration strategy which makes the process more inefficient.
- Q-Learning cannot handle continuous actions while policy gradient can.
- Policy gradient work directly at the desired answer and not implicitly like Q-Learning, the result is that Policy Gradient turns out to be faster to execute.

3.3.3 Policy Gradient Vanilla

Sparing the details that can be found on (SUTTON, 2018), here is the update algorithm for Policy Gradient Method in its most basic form:

Algorithm 1 Policy Gradient Vanilla

procedure REINFORCEMENT(π, α)
  Initialize policy parameter $\boldsymbol{\theta_{t+1}}$
  while $True$ **do**
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following the policy π
    while t is not T **do**
      $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma G_t \nabla_\theta \ln \pi \left(A_t \mid S_t, \boldsymbol{\theta_t}\right)$
      $t \leftarrow t + 1$

where γ is the discount factor and α is the step size.

3.3.3.1 Issues with the Vanilla Version

- **Full episodes are needed:** The first step on the algorithm proposed requires that an episode be first generated, that is a problem as these generations might be costly.
- **High gradient variance:** The episode generated is stochastic leading to high variance in the results, the vanilla version does nothing to account for that problem.
- **Exploration** Even with the policy represented as probability distribution, there is a high chance that the agent will converge to some locally-optimal policy and stop exploring the environment.

The high variance issue is addressed in the Actor Critic family of algorithms.

3.3.4 Actor Critic

The problem of high variance in the vanilla version comes from the fact that the value G is used in respect to the current episode with no regards to the other G values calculated on other episodes. The Actor-Critic method adds the idea of an *advantage function*:

$$\delta = G_{t:t+1} - \gamma \hat{v}\left(S_t, \boldsymbol{w}\right) \tag{3.20}$$

where $\hat{v}\left(S_t, \boldsymbol{w}\right)$ represents the current estimative of the utility of state s approximated by the parameters vectors $\boldsymbol{w}$. Now there is no need to generate the whole episode to start the training, the updates are executed by each step of the episode. Furthermore, the

variance is mitigated as the updates are applied under a current prediction of the state utility. Substituting the G value by the recursive formula:

$$\delta = R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \boldsymbol{w}\right) - \hat{v}\left(S_t, \boldsymbol{w}\right) \tag{3.21}$$

Now there is a presence of two neural networks, one for the policy and one for the reward values for each state. Of course both needs to be updated. Here follows the whole algorithm:

Algorithm 2 One-Step Actor Critic

procedure ACTOR CRITIC($\pi, \alpha^{\boldsymbol{w}}, \alpha^{\boldsymbol{\theta}}$)
  Initialize policy parameter $\boldsymbol{\theta_{t+1}}$ and state-value weights $\boldsymbol{w}$
  while $True$ **do**
    Initialize S
    $I \leftarrow 1$
    while S is not terminal **do**
      Extract action A from the policy $\pi\left(\cdot \mid S, \boldsymbol{\theta}\right)$
      Take action A, observe S', R
      $\delta = R + \gamma \hat{v}\left(S_{t+1}, \boldsymbol{w}\right) - \hat{v}\left(S_t, \boldsymbol{w}\right)$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \delta \nabla_w \hat{v}\left(S, \boldsymbol{w}\right)$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_\theta \ln \pi\left(A_t \mid S_t, \boldsymbol{\theta_t}\right)$
      $I \leftarrow \gamma I$
      $S \leftarrow S'$

3.3.5 Proximal Policy Optimization (PPO)

Having a fixed step-size is not optimal, if the size of the step is too small than the training is too slow but if the sizes are too large the algorithms becomes unstable, changing the policy too much in each step. PPO aims to optimize this concept.

3.3.5.1 Replacing the log

So far in the Actor Critic we had:

$$J_{\boldsymbol{\theta}} = \mathbb{E}_t\left[\ln \pi\left(A_t \mid S_t, \boldsymbol{\theta_t}\right)\delta_t\right] \tag{3.22}$$

The PPO paper (REFERENCIAR PAPER) suggests:

$$J_{\boldsymbol{\theta}} = \mathbb{E}_t\left[\frac{\pi\left(A_t \mid S_t, \boldsymbol{\theta_t}\right)}{\pi_{old}\left(A_t \mid S_t, \boldsymbol{\theta_t}\right)}\delta_t\right] \tag{3.23}$$

so we define:

$$r_t(\boldsymbol{\theta}) \equiv \frac{\pi\left(A_t \mid S_t, \boldsymbol{\theta_t}\right)}{\pi_{old}\left(A_t \mid S_t, \boldsymbol{\theta_t}\right)} \tag{3.24}$$

to simplify the notation.

3.3.5.2 The Clipped Surrogate Objective

By using $r_t(\boldsymbol{\theta})$ the samples become more powerful in updating the values but this new potential may be exaggerated, if the probability of the new policy is way higher than the probability of the old policy than the size step may be too large. What is needed is a way to bound this steps:

$$J(\boldsymbol{\theta}) = min\left(r_t(\boldsymbol{\theta})\,\delta_t, clip\left(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon\right)\delta_t\right) \tag{3.25}$$

3.3.5.3 The Final Score Function

This algorithm is an optimization of the Actor Critic model, hence it still estimates $v(s)$ values and is constituted of 2 neural networks, one to compute the policy parameters and the other for the critic values. The score function of the algorithm is:

$$J^{CLIP+VF+S}(\boldsymbol{\theta}) = \mathbb{E}_t[J^{CLIP}(\boldsymbol{\theta}) - c_1 J^{VF}(\boldsymbol{\theta}) + c_2 S[\pi_{\boldsymbol{\theta}}](s_t)]. \tag{3.26}$$

$J^{CLIP+VF+S}$ denotes the clipped surrogate loss detailed earlier. The second term, J^{VF} is the squared loss from prediction of value function and its value computed by the samples collected. Thus, this term aims to train the Critic network. The third term, denotes a entropy term to ensure sufficient exploration (CITAR ARTIGO Q NEM O LUCK FEZ)

3.3.5.4 Multiple epochs for policy updating

Unlike vanilla policy gradient methods, and because of the Clipped Surrogate Objective function, PPO allows you to run multiple epochs of gradient ascent on your samples without causing destructively large policy updates. This allows you to squeeze more out of your data and reduce sample inefficiency.

PPO runs the policy using N parallel actors each collecting data, and then it samples mini-batches of this data to train for K epochs using the Clipped Surrogate Objective function.

The new part is that they are able to run K epochs of gradient ascent on the trajectory samples. As they state in the paper, it would be nice to run the vanilla policy gradient

Algorithm 3 PPO

1: **procedure** PPO
2: **for** `iteration` = 1,2,... do **do**
3: **for** `iteration` = 1,2,...,N **do**
4: Run policy $\pi_{\boldsymbol{\theta}_{old}}$ in environment for T time steps
5: Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
6: Optimize surrogate J with respect to $\boldsymbol{\theta}$ with K epochs and minibatch size $M \leq NT$
7: $\boldsymbol{\theta}_{old} \leftarrow \boldsymbol{\theta}$

optimization for multiple passes over the data so that you could learn more from each sample. However, this generally fails in practice for vanilla methods because they take too big of steps on the local samples and this wrecks the policy. PPO, on the other hand, has the built-in mechanism to prevent too much of an update

4 To Do List

Having acquired the theory of reinforcement learning to be used in this work, now we make plans on how to use it. The PPO algorithm is already implemented in the OpenAI website (citar), so the work required is to build the bridge between the RoboCup server program and the already constructed implementation of the algorithm. For the neural network, we plan to use TensorFlow. Here are the intended objectives with deadlines:

1. Build the bridge between OpenAI and RoboCup program.
2. Build the model for the learning algorithm.
3. Start training.
4. Analise results and iterate the model until good results appear.

Referências

AGRAWAL, J. G. A.; GOLDFARB, A. **Prediction Machines**. 1st. ed. Boston: Harvard Business Review Press, 2018.

CHAPMAN, L. **SoftBank Plows 400 Million Into Synthetic Bio Startup Zymergen**. 2019. Disponível em: <https://www.bloomberg.com/news/articles/2018-12-13/softbank-plows-400-million-into-synthetic-bio-startup-zymergen>. Acesso em: 23 may. 2019.

LEE, K. fu. **AI Superpowers**. 1st. ed. New York: Houghton Mifflin Harcout Publishing Company, 2018.

LENTINO, A. This chinese facial recognition start-up can identify a person in seconds. **CNBC**, 2019. Disponível em: <https://www.cnbc.com/2019/05/16/this-chinese-facial-recognition-start-up-can-id-a-person-in-seconds.html>. Acesso em: 23 may. 2019.

LOUCKS DAVID SCHATSKY, T. D. J. State of ai in the enterprise, 2nd edition. **Deloitte Insights**, 2018. Disponível em: <https://www2.deloitte.com/content/dam/insights/us/articles/4780_State-of-AI-in-the-enterprise/DI_State-of-AI-in-the-enterprise-2nd-ed.pdfl>. Acesso em: 23 may. 2019.

MAO MOHAMMAD ALIZADEH, I. M. H.; KANDULA, S. Resource management with deep reinforcement learning. **HotNets 16 Proceedings of the 15th ACM Workshop on Hot Topics in Networks**, p. 50–56, November 09-10 2016. Disponível em: <https://dl.acm.org/citation.cfm?id=3005750>. Acesso em: 4 jun. 2019.

NIELSEN, M. A. **Neural Networks and Deep Learning**. 1st. ed. [S.l.]: Determination Press, 2015.

ROBOCUP. Disponível em: <https://www.robocup.org/objective>. Acesso em: 4 junl. 2019.

SUTTON, A. G. R. S. **Reinforcement Learning - An Introduction**. 2nd. ed. [S.l.]: MIT Press, 2018.

UC Berkley Intro to AI: Home site to the introduction to ai from uc berkley. 2014. Disponível em: <http://ai.berkeley.edu/home.html>. Acesso em: 4 junl. 2019.

WEI GUAMJIE ZHENG, H. Y. H.; LI, Z. Intellilight: A reinforcement learning approach for intelligent traffic light control. **KDD 18: The 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, p. 2496–2505, August 19 - 23 2018. Disponível em: <https://dl.acm.org/citation.cfm?doid=3219819.3220096>. Acesso em: 4 jun. 2019.

ZHOU, X. L. Z.; N.ZARE, R. Optimizing chemical reactions with deep reinforcement learning. **ACS Cent. Sci.**, v. 3, n. 12, p. 1337–1344, December 15 2018. Disponível em: <https://doi.org/10.1021/acscentsci.7b00492>. Acesso em: 4 jun. 2019.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
DM	07 de junho de 2019	DCTA/ITA/DM-018/2019	43

5. TÍTULO E SUBTÍTULO:

Reinforcement Learning applied to the Soccer 2D keeper

6. AUTORA(ES):

André Marcello Soto Riva Figueira

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

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELA AUTORA:

Cupim; Cimento; Estruturas

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Cupim; Dilema; Construção

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

ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Aeronáutica e Mecânica. Área de Sistemas Aeroespaciais e Mecatrônica. Orientador: Prof. Dr. Adalberto Santos Dupont. Coorientadora: Profª. Drª. Doralice Serra. Defesa em 05/03/2015. Publicada em 25/03/2015.

11. RESUMO:

Aqui começa o resumo do referido trabalho. Não tenho a menor idéia do que colocar aqui. Sendo assim, vou inventar. Lá vai: Este trabalho apresenta uma metodologia de controle de posição das juntas passivas de um manipulador subatuado de uma maneira subótima. O termo subatuado se refere ao fato de que nem todas as juntas ou graus de liberdade do sistema são equipados com atuadores, o que ocorre na prática devido a falhas ou como resultado de projeto. As juntas passivas de manipuladores desse tipo são indiretamente controladas pelo movimento das juntas ativas usando as características de acoplamento da dinâmica de manipuladores. A utilização de redundância de atuação das juntas ativas permite a minimização de alguns critérios, como consumo de energia, por exemplo. Apesar da estrutura cinemática de manipuladores subatuados ser idêntica a do totalmente atuado, em geral suas caraterísticas dinâmicas diferem devido a presença de juntas passivas. Assim, apresentamos a modelagem dinâmica de um manipulador subatuado e o conceito de índice de acoplamento. Este índice é utilizado na sequência de controle ótimo do manipulador. A hipótese de que o número de juntas ativas seja maior que o número de passivas ($n_a > n_p$) permite o controle ótimo das juntas passivas, uma vez que na etapa de controle destas há mais entradas (torques nos atuadores das juntas ativas), que elementos a controlar (posição das juntas passivas).

12. GRAU DE SIGILO:

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