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.
42f.

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. 42f. 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

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

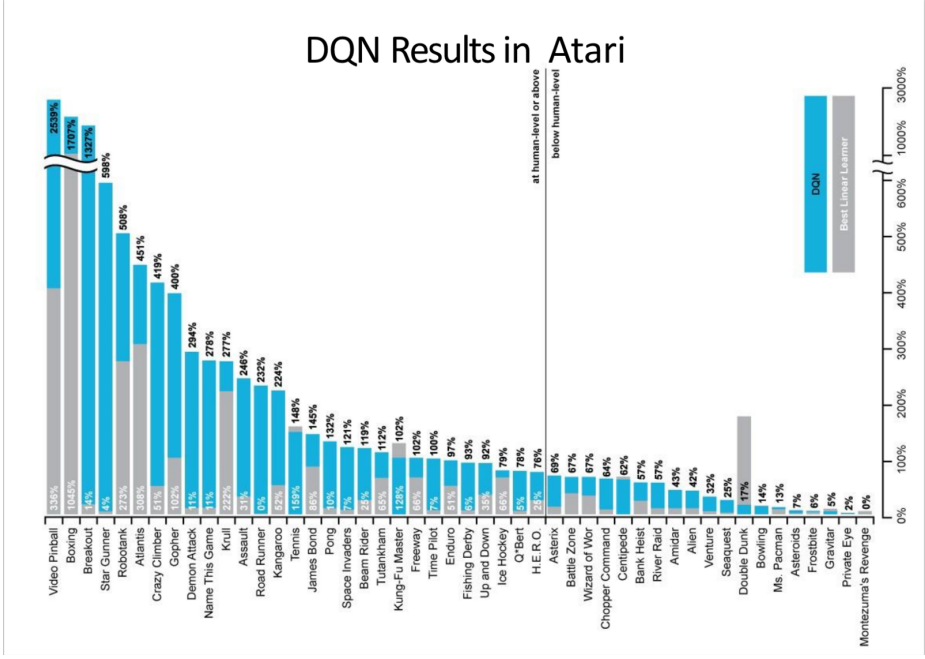
1.1 Motivation

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.

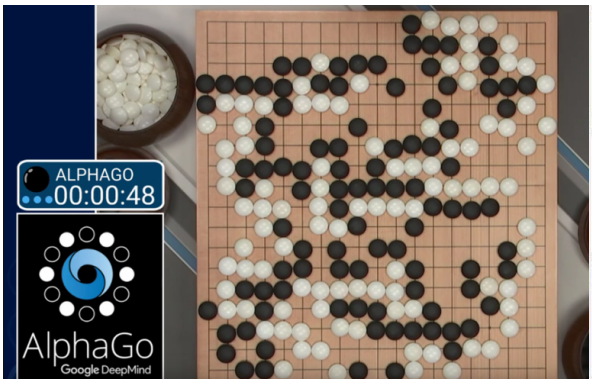
Besides, most recently, the cutting edge achievements in AI have shown algorithms capable of overcome top human intelligence in difficult problems. In 2015, Google DeepMind created an AI that learned how to play 49 Atari games using the same learning algorithm (including the same hyper parameters), using 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.

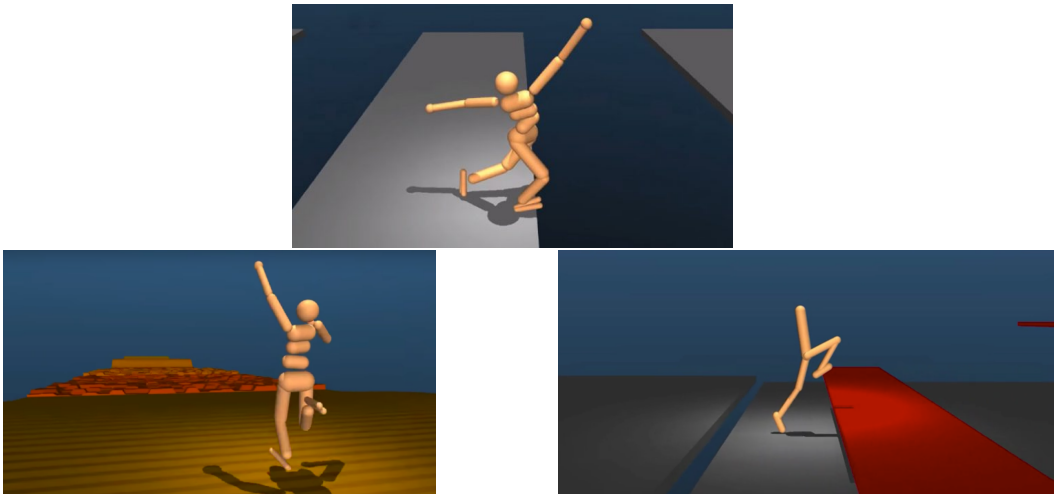


FIGURE 1.2 – Simulated humanoid agent movements and AI.

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.

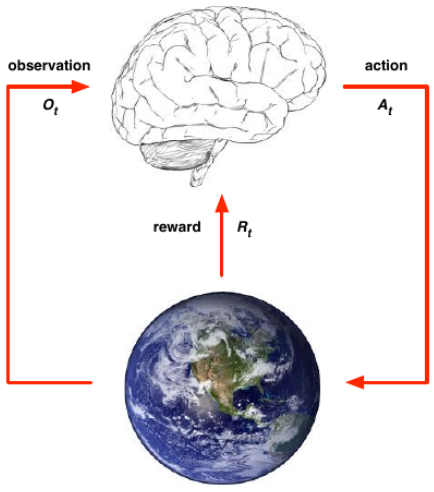


FIGURE 2.1 – Agent interacting with environment.

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.

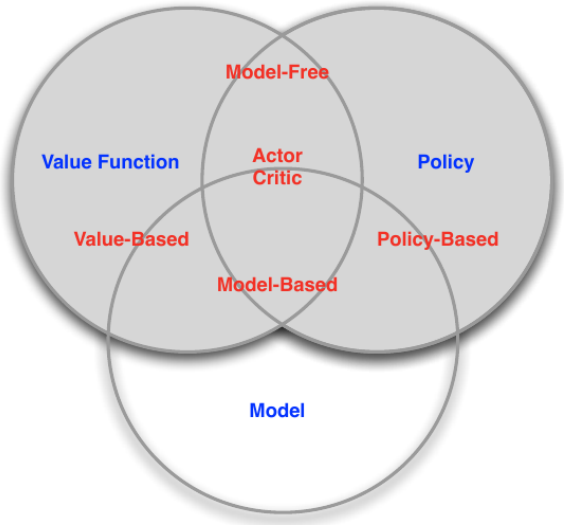


FIGURE 2.2 – Classical division among RL algorithms classifications.

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 ??.

$$Q_t^\lambda = (1-\lambda)\sum_{i=1}^{n} \lambda^{n-1} Q_t^{(n)}$$
$$Q(S, A) = Q(S, A) + \alpha(Q_t^\lambda - Q(S, A))$$

2.3.2.3 Q-Learning

2.3.3 Policy Search Methods

3 Deep Learning

In the last chapter we described the mathematical background behind reinforcement learning and how a problem can be modeled with it. We also saw that for problems with continuous state and/or action spaces, it can be necessary to use function approximators such as neural networks. In this chapter we will describe the theoretical background of NNs, as function approximators.

Deep learning is a field of Machine Learning that in part studies deep neural networks and how they learn.

3.1 History

Today, Deep learning is a very significant topic for the scientific community yet its history goes back since the 1940s (GOODFELLOW; COURVILLE, 2016).

There has been a major resurgence of deep learning mainly because computers today became faster. Training large deep learning models are computationally very expensive and could be sped up with the use of graphics processing units (GPUs).

Artificial Neural Networks are one of the earliest learning algorithms. The novel concept behind it was to create mathematical models that tried to mimic the brain. NNs were based on the human brain's neurological structure, more specifically neurons and how they connect with other neurons. Figure 3.1 illustrates a single neuron cell.

A neuron cell is capable of transmitting information by sending electrical and chemical signals through axons and they connect to other neurons forming neural networks. The human brain, for example, has billions of connected neurons. NNs, therefore, are a simplified mathematical representation of neurons: essentially, a neuron can be viewed as a cell that receives one or more inputs and produces one or more outputs.

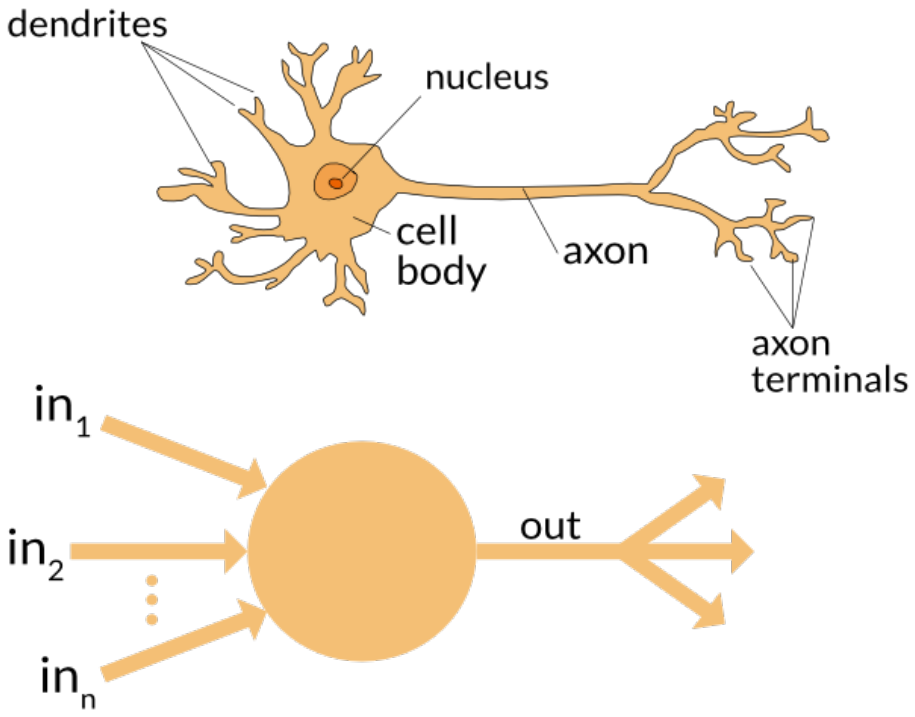


FIGURE 3.1 – Neuron unit cell structure.

3.2 Neural Networks

3.2.1 Representation

In simple terms, a neural network is a mathematical structure that receives an input and calculates an output. We will describe a simple NN known as feedforward neural network or Multi Layer Perceptron (MLP).

A neural network is composed of multiple layers where each layer has one or more nodes (neurons). The NN receives an input and computes the output according to the network architecture and parameters θ. Each input, called sample or example, is a n-dimensional feature vector x, and is represented as a column vector:

$$x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & \dots & x_{n-1} & x_n \end{bmatrix}^\intercal \tag{3.1}$$

x is known as the input layer, and each coordinate represents each of the sample's features. The output is a k-dimensional vector y, known as the output layer. Every other layer that is not the input nor the output layer is known as a hidden layer.

In this work, note that every time we refer to an element $z^{[l]}$ we are referring to an element of the l-th layer.

Figure 3.2 illustrates a very simple NN with a 3-dimensional feature vector, 1-dimensional

output vector and only one hidden layer.

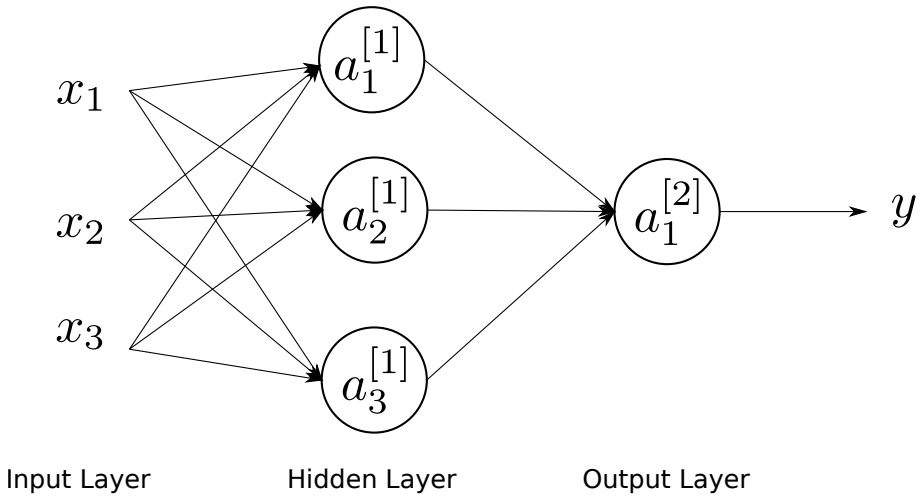


FIGURE 3.2 – Simple neural network.

Every node of the network is computed using the values from the previous layer. For the MLP, the node computation model is done through a linear model $f(x;\theta)$, with θ consisting of internal parameters w (weights) and b (bias). w has the same dimensions as the input and b is a scalar. In this work, we treat w as a column vector.

$$w = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & \dots & w_{n-1} & w_n \end{bmatrix}^\intercal \tag{3.2}$$

The model is defined as

$$z = w^T x + b \tag{3.3}$$

The output of the node a, also known as the activation, is calculated with x, b and a function $\sigma : \mathbb{R} \to \mathbb{R}$:

$$a = \sigma(z) = \sigma(w^T x + b) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.4}$$

Figure 3.3 illustrates the output of a single neuron.

σ is called activation function, applied to the model output (z). The activation function normally is a non-linear function and its most used common choice today is the **rectified linear unit** (NAIR; HINTON, 2010) or ReLU and is defined as $\sigma(z) = max\{z, 0\}$, shown in 3.4.

Other examples of popular activation functions are $\sigma(z) = \frac{1}{1+e^{-z}}$, known as the sigmoid function and $tanh(z)$ (XU; LI, 2016).

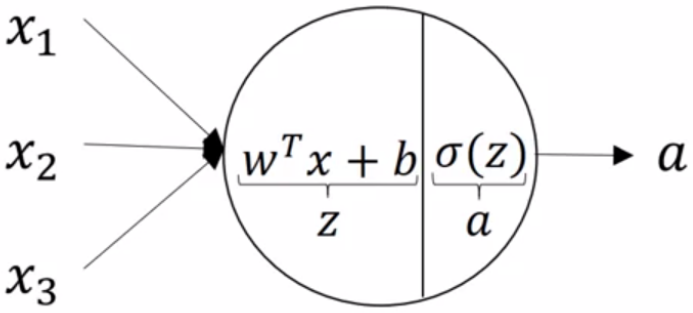


FIGURE 3.3 – Node computation.

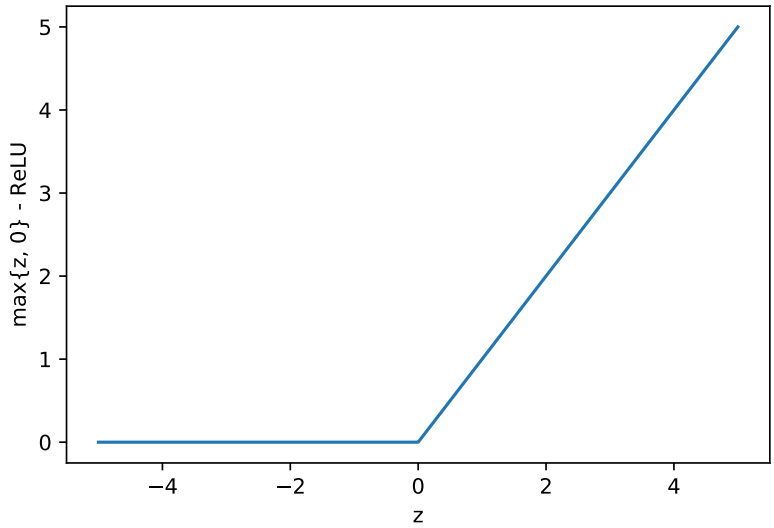


FIGURE 3.4 – Linear rectified unit plot.

A neural network is therefore is a connection of multiple neurons as inputs to other neurons. Let us return our attention to the network from Figure 3.2. We denote $(W, b) = (W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$ as the network parameters, where $W^{[l]}$ is the matrix formed by concatenating the individual parameters of the l-th layer such that W_{ij} denotes the parameter associated with the connection between neuron j from the l-th layer and neuron i from layer $l+1$. Similarly, $b^{[l]}$ is the concatenation of the biases b of each neuron of layer $l+1$ and $b_i^{[l]}$ the bias of the i-th neuron of layer $l+1$.

For the network depicted in Figure 3.2, we are now capable of calculating the output y.

For the first layer:

$$z^{[1]} = W^{[1]}x + b^{[1]} \tag{3.5}$$

$$a^{[1]} = \sigma(z^{[1]}) \tag{3.6}$$

And for the second layer:

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \tag{3.7}$$

$$y = a^{[2]} = \sigma(z^{[2]}) \tag{3.8}$$

We can think of the neural network as a recursive structure where each activation is calculated with the past's layers activation. $a^{[l]}$ can be computed as:

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \tag{3.9}$$

$$a^{[l]} = \sigma(z^{[l]}) \tag{3.10}$$

Note also that $a^{[1]} = x$.

3.2.2 Vectorization

In the previous section, we focused on computing the output of the neural network given a single sample x. In modern days, however, we are normally interested in calculating the output of a neural network for thousands or even millions of samples.

For m samples, we could naively compute the output of the neural network for each example $x^{(i)}$ with Algorithm 2.

Algorithm 2: Naive algorithm for computing NN output of m samples.

Result: Output of NN for m samples.

for *i = 1 to m* **do**
  $a^{[0]} = x^{(i)}$
  for *j = 1 to L* **do**
    $z^{[j](i)} = W^{[j]}a^{[j-1](i)} + b^{[j]}$
    $a^{[j](i)} = \sigma(z^{[j](i)})$
  end
  $y^{(i)} = a^{[l](i)}$
end

In practice this algorithm runs relatively slow since it computes the output of the

network sequentially for each sample and we will apply vectorization to achieve a much faster algorithm.

Firstly, Vectorization is a technique that transforms a set of computations done sequentially in a for loop into matrix operations. We will now vectorize our initial problem.

Let us define the following matrices

- X is a matrix where column i is the i-th sample $x^{(i)}$ and $W^{[l]}$. We can analogously define matrices A and Z. $X = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix}$.
- $W^{[l]}$ is the weight matrix for the l-layer, as defined in Section 3.2.1.
- $b^{[l]}$ is the bias vector for layer l, as defined in Section 3.2.1.

For the vectorized version of Algorithm 3.

Algorithm 3: Vectorized algorithm for computing NN output of m samples.

Result: Output of NN for m samples.

for $j = 1$ *to* L **do**

$Z^{[j]} = W^{[j]} A^{[j-1]} + b^{[j]}$

$A^{[j]} = \sigma(Z^{[j]})$

end

The vectorized version is much better since these matrices operations can be greatly sped up on GPUs or hardware that have support for Streaming SIMD Extensions (SSE). The great majority of popular deep learning libraries use as much vectorization as they can.

3.2.3 Deep Neural Networks

We can extend the architecture we saw on Section 3.2.1 to a higher number of layers called deep neural networks. Figure 3.5 illustrates a neural network with multiple hidden layers.

The activation of layer $l + 1, l > 1$, can be calculated according to Equation 3.9.

3.3 Learning

As described in the previous section, feedforward networks defines a mapping $y = f(x; \theta)$, or, equivalently, serve as general function approximators.

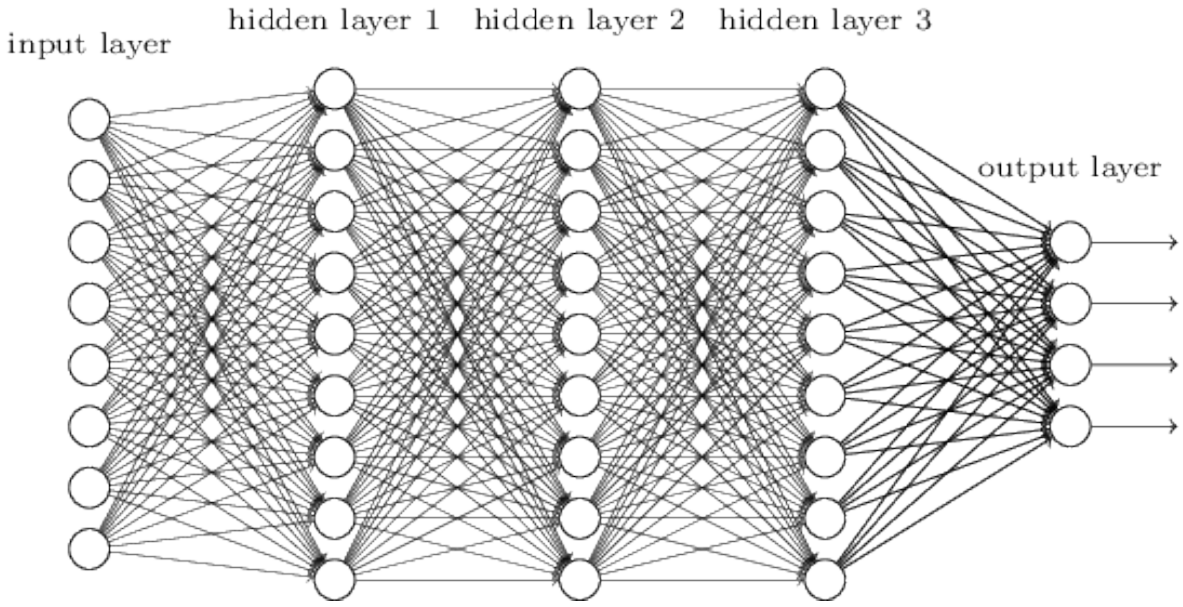


FIGURE 3.5 – Neural network with multiple layer.

This section describes deep learning algorithms that learn the value of the parameters θ that best approximates f.

Most deep learning algorithms need to describe a cost function and optimization algorithm.

3.3.1 Cost Function

The cost function $J(\theta)$ is a metric of how good our estimator is to our dataset: the lower the cost function, the less error the model has when predicting y. It describes the function that we wish to minimize and normally it envolves an average of the errors between the the target value of a sample and the predicted value of the estimator for the sample.

For example, for the problem of linear regression where X are the sample values and y are the target values for our dataset, we wish to learn the parameters θ from our function approximator f. The most used cost function for this problem is

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(x_i, \theta))^2 \tag{3.11}$$

Another example is the logistic regression problem. The most common cost function

for this problem is

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log f(x_i, \theta + (1 - y_i) \log 1 - f(x_i, \theta)] \tag{3.12}$$

Choosing a good cost function greatly impacts how good the design of the deep neural network is.

3.3.2 Optimization Algorithms

After defining a cost function, finding the parameters that effectively minimize the cost function is not trivial. The non linearity of NN causes most cost functions to be non-convex and therefore, neural networks are usually trained by iterative, gradient-based optimizers. In practice, however, this is very unlikely (SWIRSZCZ *et al.*, 2017; GOODFELLOW *et al.*, 2014; LIN *et al.*, 2017).

Section 3.4 describes how neural networks gradients can be calculated.

3.3.2.1 Gradient Descent

One of the most common optimization algorithm is Gradient Descent. It is an iterative first-order method that tries to find a local minimum for a function f by moving in the negative direction of its gradient. Specifically for neural networks, our goal is to minimize the cost function $J(\theta)$. The algorithm's update step is Equation (3.13) done for each θ_i

$$\theta_i = \theta_i - \alpha\frac{\partial J}{\partial \theta_i} \tag{3.13}$$

The learning rate, α measures how big the update step in the direction of the gradient will be.

For example, let $g : \mathbb{R}^2 \to \mathbb{R}$ given by $g(x, y) = y\sin(x) - x\cos(y)$. Figure 3.6 illustrates 5 steps of the gradient descent algorithm. Notice how the red circle is the starting point and after the last iteration, the algorithm arrives approximately at a local minimum.

3.3.2.2 Stochastic Gradient Descent

Another popular algorithm is Stochastic Gradient Descent (SGD) that is a stochastic approximation of Gradient Descent. Instead of calculating $J(\theta)$ as an average between all the samples, the gradient update step is done once per sample. It calculates an approximation of the gradient and updates the parameters θ_i with every sample. Algorithm 4

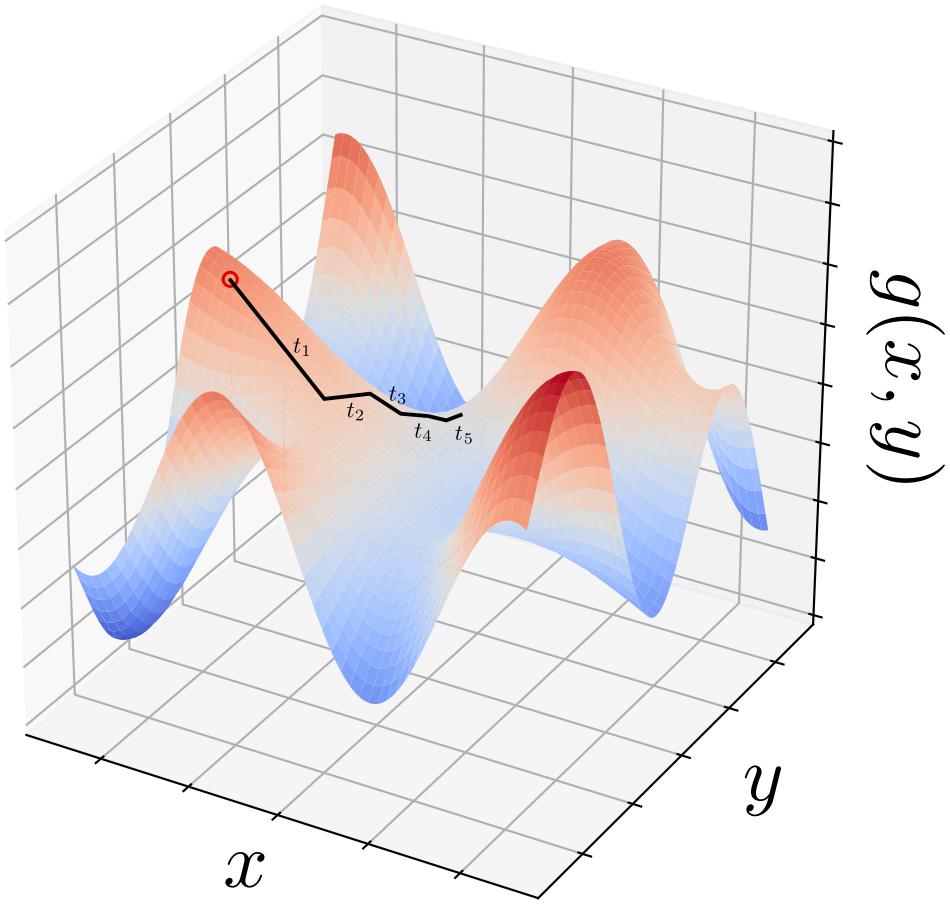


FIGURE 3.6 – Gradient descent on function g.

describes SGD

Algorithm 4: SGD algorithm.

Result: Function minimum

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

  Randomly shuffle examples in the training set

  for $i = 1,2,\ldots, m$ **do**

    $\theta_i = \theta_i - \alpha \nabla J_i(\theta)$

  end

end

The gradient approximation for SGD is much faster to compute, however, since it is a noisy estimate of the gradient, SGD may need more steps to converge to a local minimum.

3.3.2.3 Mini-Batch Gradient Descent

Finally, Mini-Batch Gradient Descent combines the idea from Gradient Descent and SGD. It computes the gradient using more than one training example, called a *mini-batch*, at each step. It may result in smoother convergence and the code can be accelerated by

making use of vectorization.

3.4 Back Propagation

We have seen how a few gradient-based optimization algorithms work. To minimize the cost function we need to compute the partial derivatives of the cost function w.r.t. to each weight of the neural network. Calculating these partial derivatives manually would be computacionally infeasible for large networks. In deep neural networks, by using the chain rule, we can calculate the partial derivatives in an efficient and compact way.

Backpropagation is an algorithm that computes the gradients of the loss function w.r.t. each input from the network.

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>.

Appendix A - Tï¿½picos de Dilema Linear

A.1 Uma Primeira Seção para o Apêndice

A matriz de Dilema Linear M e o vetor de torques inerciais b, utilizados na simulação são calculados segundo a formulação abaixo:

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \tag{A.1}$$

FIGURE A.1 – Uma figura que está no apêndice

Annex A - Exemplo de um Primeiro Anexo

A.1 Uma Seção do Primeiro Anexo

Algum texto na primeira seção do primeiro anexo.

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	42

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**