INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Luckeciano Carvalho Melo

A DEEP REINFORCEMENT LEARNING METHOD FOR HUMANOID KICK MOTION

Final Paper
2018

Course of Computer Engineering

Luckeciano Carvalho Melo

A DEEP REINFORCEMENT LEARNING METHOD FOR HUMANOID KICK MOTION

Advisor

Prof. Dr. Adilson Marques da Cunha (ITA)

Co-advisor

Prof. Dr. Marcos R. O. de A. Máximo (ITA)

COMPUTER ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2018

Cataloging-in Publication Data
Documentation and Information Division

Carvalho Melo, Luckeciano
A Deep Reinforcement Learning Method for Humanoid Kick Motion / Luckeciano Carvalho Melo.
São José dos Campos, 2018.
41f.

Final paper (Undergraduation study) – Course of Computer Engineering– Instituto Tecnológico de Aeronáutica, 2018. Advisor: Prof. Dr. Adilson Marques da Cunha. Co-advisor: Prof. Dr. Marcos R. O. de A. Máximo.

1. Deep Reinforcement Learning. 2. Robotics. 3. Artificial Intelligence. I. Instituto Tecnológico de Aeronáutica. II. A Deep Reinforcement Learning Method for Humanoid Kick Motion.

BIBLIOGRAPHIC REFERENCE

CARVALHO MELO, Luckeciano. **A Deep Reinforcement Learning Method for Humanoid Kick Motion**. 2018. 41f. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

Luckeciano Carvalho Melo
H8A St., 113
12.228-460 – São José dos Campos–SP

A DEEP REINFORCEMENT LEARNING METHOD FOR HUMANOID KICK MOTION

This publication was accepted like Final Work of Undergraduation Study

---

Luckeciano Carvalho Melo

Author

Adilson Marques da Cunha (ITA)

Advisor

Marcos R. O. de A. Máximo (ITA)

Co-advisor

Prof^{a}.Dr^{a}. Cecília César
Course Coordinator of Computer Engineering

São José dos Campos: JUNE 12, 2018.

Abstract

Controlling high degrees of freedom for humanoid robot is acknowledged as one of the hardest problems in Robotics. Due to the lack of mathematical models, an approach frequently employed is to rely on human intuition to design keyframe movements by hand, usually aided by graphical tools. In this preliminary work, we propose a learning framework based upon neural networks, in order to mimic humanoid robot movements. The developed technique does not make any assumption about the underlying implementation of the movement. Therefore, both keyframe and model-based motions may be learned. The framework was applied in the RoboCup 3D Soccer Simulation domain and some promising results were obtained, by using the same network architecture for several motions, even when copying motions from another teams.

List of Figures

List of Tables

Contents

1 Introduction

1.1 Motivation

Robotics is understood as an important area of research within Engineering and Computer Science, optimizing and automating various areas of industry. One of the ways in which research is developed in this area is in robot soccer, since it comprises challenges of machine perception, environment modeling, planning and reasoning, control, and multiagent strategy.

Over the years, several techniques have been developed to address each of the problems related to robot soccer, based on the theory of Signal Processing, Control, Trajectory Planning, and classical Artificial Intelligence. These techniques have proved to be functional for maturing the challenge. However, such techniques still perform worse as compared to humans in these activities.

In recent years, however, with the development of processing and memory architectures, Machine Learning techniques have been able to achieve or even have surpassed the human performance in machine perception activities (Computer Vision (LU; TANG, 2014) and Speech Recognition (XIONG *et al.*, 2016)), by planning and reasoning (SILVER *et al.*, 2017a), as shown in Figure 1.1, and also by controlling agent locomotion (HEESS *et al.*, 2017), as shown in Figure 1.2. In this way, the learning field of combining techniques from Deep Learning and Reinforcement Learning appears as a great candidate in search of General Artificial Intelligence.

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 robots will be able to beat the human team champion of the World Cup until the year 2050 (KITANO *et al.*, 1998). Since this is a goal with a range of different challenges to be solved, there are several categories of competition within the community.

FIGURE 1.1 – AlphaGo Zero, learning model that beat the best players of Go, Chess and Shogi, learning to play without previous human knowledge (SILVER *et al.*, 2017a).

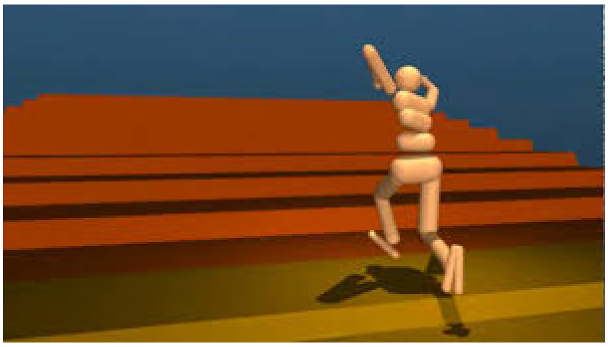


FIGURE 1.2 – Locomotion of Agent via Deep Reinforcement Learning (HEESS *et al.*, 2017).

In Robocup Soccer 3D Simulation League, as shown in Figure 1.3, there is a robot soccer competition in which each team has eleven Nao robots in a physical simulation environment called Simspark (OBST; ROLLMAN, 2005). The purpose in this case is to create and improve algorithms and physical models for the perception, locomotion, and strategy of the humanoid robot, before actually shipping them into hardware for real-world evaluation.

FIGURE 1.3 – A Snapshot from the RoboCup Soccer 3D Simulation League.

Over the last few years, team efforts have been seen in three different areas, layers, or strands. The first area is considered more basic, meaning the construction of the agent that can model the environment and interact with it – involving both the construction of a solid software architecture and the application of traditional techniques of localization and control of the humanoid robot (MACALPINE *et al.*, 2011).

The second layer, explored at the highest level, involves creating behaviors to perform actions within the game, given the modeled environment – from the creation of models and heuristics for navigation to the creation of multiagent strategies for positioning and marking (MACALPINE; STONE, 2016).

The third strand, to be approached in this work, is the creation and optimization of models based upon learning for activities such as robot walking and kicking. Simulated categories have great value for learning test, mainly because they provide a benchmark for comparison and do not involve physical robots. Historically, teams with the best performance in these two issues are usually the best positioned within category competitions, due to the fact that they are able to maintain greater possession of the ball and are more offensive to the opposing goal.

1.3 Objective

Inspired from the context of the RoboCup 3D Simulation League and based upon some results from the recent techniques of Deep and Reinforcement Learning, this work aims to develop and evaluate some new learning models, based on Deep Reinforcement Learning, for the task of making a humanoid robot to kick the soccer ball inside the RoboCup Soccer 3D Simulation environment.

1.4 Scope

In this work, Reinforcement Learning algorithms applied to models based upon Deep Neural Networks will be approached, in order to find, through gradient-based optimization techniques, optimal policies for kick control of the humanoid robot. These will be contrasted with classic control techniques coupled with evolutionary optimization strategies, widely used in the context of the RoboCup Soccer 3D Simulation League.

1.5 Organization of this work

This work is organized as follows: Chapter 2 will describe the RoboCup Soccer 3D Simulation League and the traditional methods used for the kick motion. Chapter 3 will cover the theory behind Deep Learning used in this work. Chapter 4 will explain all the methods and tooling used in experimentation. Chapter 5 describes the experimentation itself, detailing problem modeling, test scenarios, and the main results. Finally, Chapter 6 will share some conclusions and suggestions for future work.

2 Literature Review

2.1 The RoboCup Soccer3D Simulation League

2.1.1 Domain Description

The RoboCup Soccer 3D Simulation League (Soccer 3D) is a particularly interesting challenge concerning humanoid robot soccer. It consists of a simulation environment for a soccer match with two teams, each one composed by up to 11 simulated Nao robots (GOUAILLIER *et al.*, 2009), the official robot used for the RoboCup Standard Platform League since 2008. The Soccer 3D is interesting for robotics research, since it involves high level multi-agent cooperative decision making, while providing a physically realistic environment, which requires control and signal processing techniques for robust low level skills.

The RoboCup 3D simulation environment is based on SimSpark (XU; VATANKHAH, 2014) , a generic physical multi-agent system simulator. SimSpark uses the Open Dynamics Engine (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. The Nao robots has height of approximately 57 cm and 4.5 kilograms. The agent send torque commands to the simulator and receive perceptual data. Each robot has 22 joints with perceptors and effectors. Joints information are communication between agent and server, visual information and between agents happens in in the frequency of 50 Hz, 16.7 Hz and 25 Hz, respectively (MACALPINE *et al.*, 2012).

2.1.2 Kick Motion

In the current level of the Soccer 3D evolution, motion control is a key factor in team's performance. Indeed, controlling a high degrees of freedom for a humanoid robot is acknowledged as one of the hardest problems in Robotics. Much effort has been devised to humanoid robot walking, where researchers have been very successful in designing control algorithms which reason about reduced order mathematical models based upon the Zero Moment Point (ZMP) concept, such as the linear inverted pendulum model (KAJITA *et al.*, 2001). Nevertheless, these techniques restrict the robot to operate under a small region of its dynamics, where assumptions of the simplified models are still valid (COLLINS *et al.*, 2005; MUNIZ *et al.*, 2016).

Therefore, model-based techniques are hard to use for designing highly dynamic movements, such as long distance kicks and goalkeeper dives to defend goals from fast moving balls. In the robot soccer domain, a common approach for these movements is to employ keyframe movements, where motions are composed by sequences of robot postures. In this case, movements are designed off-line and executed in an open-loop fashion in execution time.

Due to the lack of mathematical models, an approach frequently employed is to rely on human intuition to design keyframe movements by hand, usually aided by graphical tools. However, this process is difficult, time consuming, and is often unable to obtain high performance motions given the high dimensionality of the search space. Other possible solution is to use motion to capture data from humans (SHON *et al.*, 2005), which has its own challenges due to the fact that the kinematic and dynamic properties of a humanoid robot differs greatly from those of a human.

2.1.3 Keyframe Movements

Definition 1 *A* keyframe $\mathbf{k} = [j_1, j_2, \ldots, j_n]^T \in K \subseteq \mathbb{R}^n$ *is an ordered set of joint angular positions, where K and n are the joint space and the number of degrees of freedom of the robot.*

Definition 2 *A* keyframe step *is an ordered pair* $\mathbf{s} = (\mathbf{k}, t) \in S = K \times \mathbb{R}$*, where* $\mathbf{k}$ *is a keyframe and t is the time when the keyframe must be achieved with respect to the beginning of the movement.*

Definition 3 *A* keyframe movement*, or simply a movement, is defined as* $\mathbf{m} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_\gamma, r) \in M = S^\gamma \times \mathbb{R}$*, where γ and r are the number of keyframe steps and the speed rate of the movement. In this representation, we assume the movement starts at time 0 and the first keyframe step represents the robot posture at the beginning of the movement. Therefore,* $t_1 = 0$ *and each time* $t_i, \forall i \geq 2$ *is a time since the beginning of the movement.*

Keyframe movements are executed in an open-loop fashion, where joint positions are computed through interpolation of keyframe steps based upon the current time. If the interface to the robot joints is not position-based, local controllers may be used to track the position references issued by the keyframe. For example, in the Simspark simulator, the simulated Nao Robot has speed-controlled joints, therefore, we use simple proportional controllers for each joint to track desired joint positions. To obtain smooth joint trajectories, we interpolate keyframe steps, by using cubic splines (BARTELS *et al.*, 1987), which are functions of class $\mathcal{C}^2$.

Finally, when a keyframe movement is requested, joint positions are often far away from movement's initial joint positions. Hence, by simply executing the keyframe movement in this case would result in high joints accelerations, which would probably make the robot to fall. To avoid this from happening, a transition movement based on linear interpolation is first employed to bring the joint positions to the initial joint positions required by the keyframe movement.

2.1.4 Optimization Techniques

In Soccer 3D environment, optimization techniques plays a important role to achieve competitive behaviors, because they are use to obtain faster and more robust motions.

Motion optimization means find the best values for each joint in the time instant during the whole motion. There's a lot of parameters involved, what turns manual tuning impossible. The common way to do this is modeling the motion and use optimization algorithms to find the best parameters for this model. For the kick motion, it means find the best keyframe and interpolator values. In the case of walking motion, we optimize the walk parameters, such as torso height, period and step size.

In the robotics simulation world, it's common to optimize using evolution strategies. (**??**) used Particle Swarm Optimization to optimize walk parameters. (URIELI *et al.*, 2011) compared the performance of several algorithms in Soccer3D context, such as Hill Climbing (HC), Cross-Entropy Method (CEM) (**??**), Genetic Algorithm and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (HANSEN, 2016). The last one turns out to be the best for this kind of task and therefore is the most common in this environment.

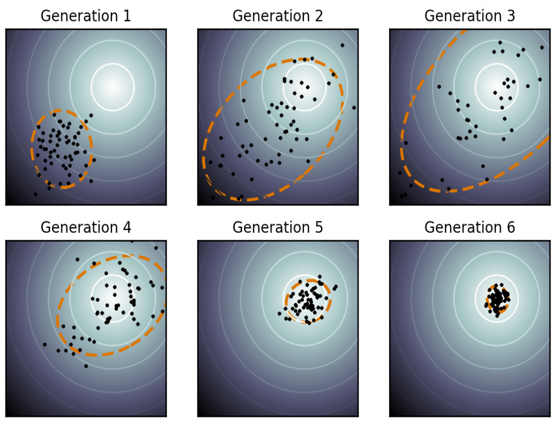


FIGURE 2.1 – Illustration of an actual optimization run with covariance matrix adaptation on a simple two-dimensional problem (KHAN, 2018).

The CMA-ES algorithm is a policy search algorithm that generates a population of parameter sets – also known as ”candidates” – sampled from a multivariate Gaussian distribution. These sets are evaluated with respect of a fitness measure. each candidate is evaluated with respect to a fitness measure. When all the candidates in the group are evaluated, the mean of the multivariate Gaussian distribution is recalculated as a weighted average of the candidates with the highest fitnesses. The covariance matrix of the distribution is also updated to bias the generation of the next set of candidates toward directions of previously successful search steps (URIELI *et al.*, 2011), as shown in Figure 2.1. Even the algorithm works well for kick and walk motions, it doesn't scale well for hundreds or thousands of parameters (MACALPINE; STONE, 2017).

2.2 Reinforcement Learning for Control

In this work, we will use more recent techniques that are able to optimize in this scale using policy gradient based algorithm instead of evolution strategies. These algorithms are based on Reinforcement Learning theory applied for control tasks.

Classic reinforcement learning algorithms are called tabular solution methods and uses dynamic programming for map state to actions. The most famous algorithms are Monte-Carlo methods, Temporal-Difference learning, SARSA (RUMMERY; NIRANJAN, 1994) and Q-Learning (WATKINS, 1989). These algorithms are good to solve toy problems, such as grid worlds and multi-armed bandits. However, they don't scale well for problems with greater state space.

However, these methods were improved using the idea of function approximation instead of tabular solutions. The new algorithms started to use other learning representations, since linear combination of features until neural networks. The last one was a huge improvement in the Reinforcement Learning field. With the arise of Deep Q-Networks (Figure 2.2), we were able to surpass human-level performance in several Atari games, as show in (MNIH *et al.*, 2015).

With the arise of Deep Learning and its new techniques and the development of new computer architectures such as GPUs and TPUs, reinforcement learning has been able to scale as well, surpassing top players from the best table games. For example, the AlphaGo (SILVER *et al.*, 2017c) won from the human top player of Go, considered the hardest tabular game. It used a Monte-Carlo Tree Search with a Deep Neural Network to find a optimal policy through self-play and without any human knowledge. The evolution of AlphaGo, called AlphaZero (SILVER *et al.*, 2017b) was able to learn not just Go, but also Chess and Shogi within a couple of hours.

On the other side, all the tasks mentioned before are discrete control. The problem of

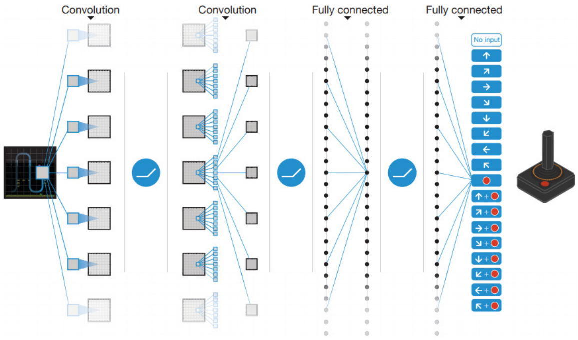


FIGURE 2.2 – Human level control through deep reinforcement learning in atari games (MNIH *et al.*, 2015).

kick motion is related to continuous control. In this kind of problem, there's no discrete actions, but a interval of them. Therefore, the algorithms are different and must satisfact new optimization challenges.

In the last years, several algorithms arised in order to improve the performance in control tasks. These algorithms are commonly applied to MuJoCo environment (TODOROV *et al.*, 2012), the most famous benchmark for continuous control tasks. Among such methods, we highlight A2C (MNIH *et al.*, 2016), ACER (WANG *et al.*, 2016), DDPG (LILLICRAP *et al.*, 2015), GAIL (HO; ERMON, 2016), HER (ANDRYCHOWICZ *et al.*, 2017), TRPO (SCHULMAN *et al.*, 2015) and PPO (SCHULMAN *et al.*, 2017). The TRPO and PPO algorithms will be covered in deep later, and the last one will be used in the learning framework proposed in this work, since it has proved to perform better in MuJoCo benchmark.

In terms of applications, (HEESS *et al.*, 2017) applied a distributed variation of PPO in MuJoCo and were able to learn parkour movements and run without in a model free way. These motions, however, are weird, asymmetric and, sometimes, unstable. (PENG *et al.*, 2018) applied the same algorithm but with key modifications in the reward function and optimization process, resulting in better and more human motions.

The most recent and impressive application of PPO was in OpenAI Five (BROCKMAN *et al.*, 2018). It consists in five independent neural networks that form a team to play the online game Dota 2. This team was trained in the equivalent of 180 years of continuous self-play, using 256 GPUs and 128 thousand of CPUs, and won from the 99.95^{th} percentile human player team in a restricted environment. This is impressive not just because the environment is challenging – due to the partial observability, long time horizon and high-dimensional, continuous spaces – but also because of the multi-agent strategy layer learned. The network architecture used by OpenAI Five is shown in Figure 2.3.

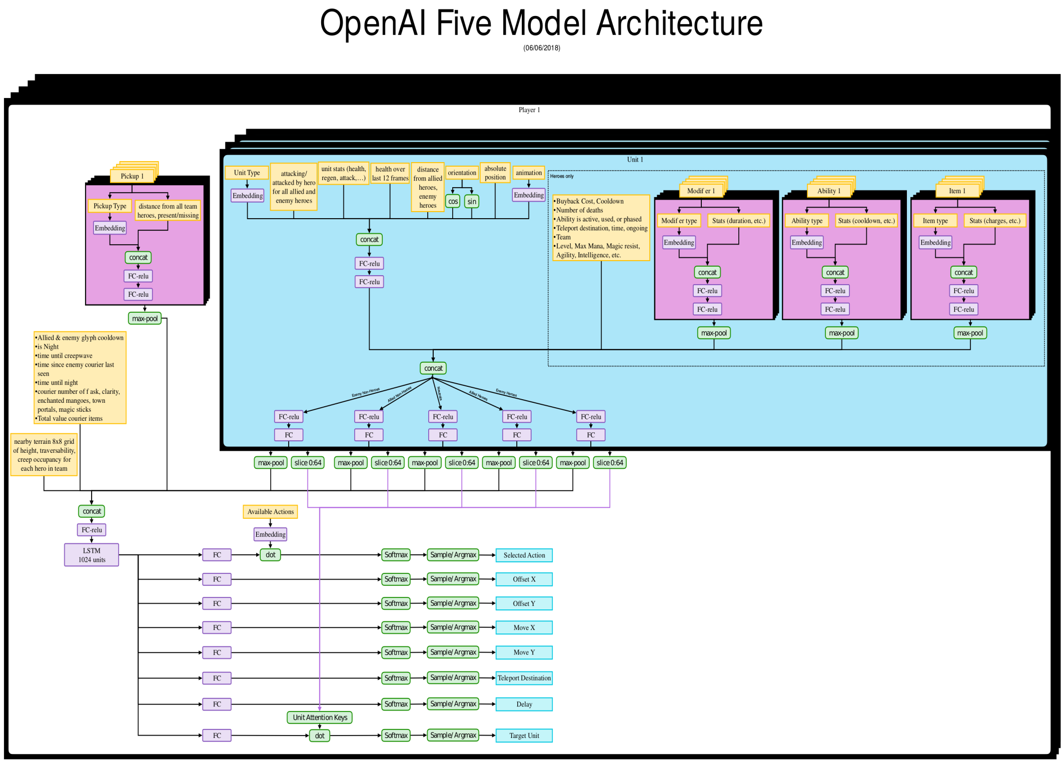


FIGURE 2.3 – OpenAI Five network architecture (BROCKMAN *et al.*, 2018).

3 Deep Learning

3.1 Neural Networks

Neural Networks are a learning representation which the goal is to approximate some function f^*. Data collected from an environment encodes an underlying function $\mathbf{y} = f^*(\mathbf{x})$ that maps an input $\mathbf{x}$ to an output $\mathbf{y}$, which may be a category from a classifier or a continue value in regression problems. The neural network defines an approximate mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, by learning the values of the parameters $\boldsymbol{\theta}$, which result in the best function approximation. Figure 3.1 shows a neural network and an artificial neuron in detail.

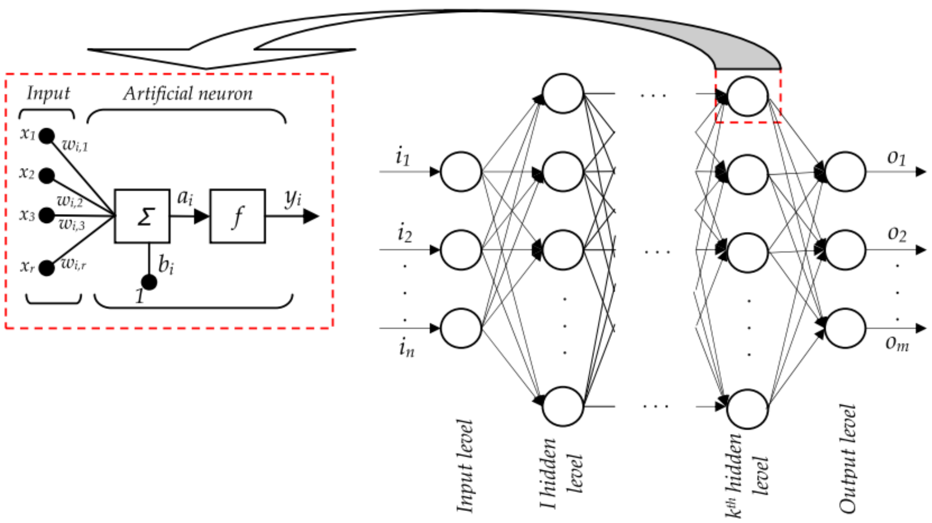


FIGURE 3.1 – An artificial neuron within a feed forward artificial neural network (TANIKIC; DESPOTOVIC, 2012).

These networks are typically represented by composing together many different functions, which are associated with a directed acyclic graph, by describing a computational model. For example, we might have three layers (each of them representing a function $f^{(1)}, f^{(2)}$, and $f^{(3)}$), connecting in a chain and resulting in a final representation $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$.

During a neural network training, the objective is to adjust $f(\mathbf{x})$ to match $f^*(\mathbf{x})$, by using the training dataset, which provides noisy examples of $f^*(\mathbf{x})$ evaluated in different points. The training examples directly specify what the output layer must do at each point $\mathbf{x}$, but the learning algorithm must decide how to use all layers to produce this desired output (GOODFELLOW *et al.*, 2016).

Additionally, we must also choose a learning algorithm to tune this function approximation. In the context of neural networks, gradient-based algorithms are broadly used, especially those based on the backpropagation idea (RUMELHART *et al.*, 1988). The purpose of these algorithms are to propagate the gradient of a cost function through the whole network, in order to minimize the cost function. Most modern neural networks perform this optimization strategy, by using maximum likelihood, i.e. the cross-entropy between the training data and the model distribution:

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{data}} \log p_{model}(\mathbf{y}|\mathbf{x}) \tag{3.1}$$

In this work, we used the mean squared error loss function, in order to fit the dataset. Indeed, we may show that both cost functions are closely related. Let us consider normally distributed errors:

$$p_{model}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; f(\mathbf{x};\boldsymbol{\theta}), \sigma^2\mathbf{I}) \tag{3.2}$$

where $f(\mathbf{x};\boldsymbol{\theta})$ and $\sigma^2\mathbf{I}$ are the mean and covariance of this distribution. By substituting Eq. (3.2) in Eq. (3.1):

$$J(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{data}} \|\mathbf{y} - f(\mathbf{x};\boldsymbol{\theta})\|^2 + const \tag{3.3}$$

The constant term does not depend on $\boldsymbol{\theta}$ and may be dropped. By explicitly evaluating the expectation in Eq. (3.3), we arrive at the mean squared error cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i}^{m} \|y_i - f(\mathbf{x};\boldsymbol{\theta})\|^2 \tag{3.4}$$

Lastly, the gradient of the loss function is taken and propagated through the hidden layers by the chain rule. For example, given $\mathbf{Y} = g(\mathbf{X})$ and $z = f(\mathbf{Y})$, then the chain rule states:

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j)\frac{\partial z}{\partial Y_j} \tag{3.5}$$

This equation is recursively taken, until the gradient is propagated to all layers of the neural network.

4 Methodology

4.1 Experimentation Setup

In order to find the best method for learn the Kick Motion, we tested several techniques to compare them and find out that one which converges faster and results in the better and more robust kick. In the next subsections, we will describe each of the methods evaluated in this work.

4.1.1 Hybrid Learning Model – HLM

In the Hybrid Learning Model (HLM), the training process occurs in two phases. The first one, a supervised learning phase, we learn the keyframe used by using a neural network, reproducing almost the same motion that we already had previously. The intuition behind this is if the agent starts from a good initial point in the optimization problem, it could be easier to get better results applying the gradient in the neighborhood of that point. The reward starts with a good value and the starting motion is well defined. Otherwise, if the optimization starts in a random point, it could be impossible to reach a good motion and probably the agent will get stuck in a local optima. This idea is shown in Figure 4.1.

4.1.2 Reinforcement with Naive Reward - RNR

In this learning model, we just use reinforcement learning with a simple and directly reward. As our task is related to kick the ball, the "naive" reward is composed by its velocities after the kick motion, as shown in 4.1, where s, v and u are actual state, the vector of velocity and weight parameters, respectively. We call it naive because we don't pass to the learning model any idea of how the motion have to be performed - just what we expect to have as final objective.

It is important to highlight that the techniques described could be used jointly as

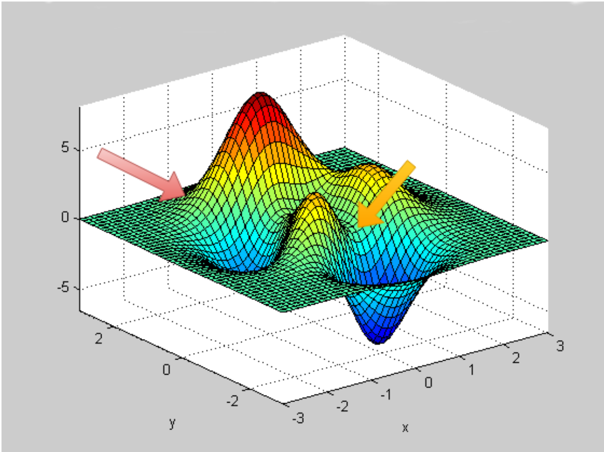


FIGURE 4.1 – In the hybrid model, we can ensure the starting point is near of the optimal solution (orange arrow); otherwise, the starting point can be bad and harder to optimize (red arrow).

building blocks of a more labored model.

$$R(s) = u^T v \tag{4.1}$$

4.1.3 Reinforcement with Reference Reward - RRR

In this learning model we add a term that compares the actual performance with a reference motion, as shown in 4.2:

$$R(s) = w_{ref}^T r \tag{4.2}$$

In this equation, w_{ref} are weight parameters for each joint and r is the absolute value of the difference between joint values in that state and the reference (equation 4.3).

$$r(\theta, \theta_0) = \|\theta - \theta_0\| \tag{4.3}$$

The intuition behind this weight parameters is because we have some joints more important than others. For example, some joints from the leg that kicks the ball themselves could collapse the whole motion if the error is high. On the other hand, joints from the neck are not so important to the kick. Therefore, it's natural to penalize differently in each case.

Lastly, the reference motion used was the keyframe kick previously mentioned and also used in HLM.

4.1.4 Reinforcement with Initial State Distribution - RISD

The Initial State Distribution is a technique related to where the episode starts to collect data to use in the learning. Without this technique, all episodes start in the beginning of the motion. However, the problem of this strategy is because the policy is forced to learn the motion in a sequential manner, first learning the early phases of the motion, and then incrementally progressing towards the later phases. This can be problematic for the kick motion. Another disadvantage of a fixed initial state is the resulting exploration challenge. The policy only receives reward retrospectively, once it has visited a state. Therefore, until a high-reward state has been visited, the policy has no way of learning that this state is favorable (PENG *et al.*, 2018).

To implement Initial State Distribution, we used a uniformly random variable that could assume any value in the set of states from the reference motion previously described. Suppose we pick the value s. Then, we start the episode running reference motion, from its first state until s. Finally, we start to collect data and perform the actions from the running policy.

4.1.5 Reinforcement with Early Termination - RET

The Early Termination technique is related to where the episode ends. Without early termination, all motions stops in the same state, which means that the episode length is fixed. The problem of this is because if the motion fails somehow during its execution, all the data collected after the failure will harm learning and evaluation of later stages.

In the kick motion task, the early termination will be triggered when the robot falls during the motion. This behavior will ensure two things: first, that we don't collect wrong data, as mentioned before; and second, to gain more reward and have longer episodes, the agent will be forced to learn a kick motion that doesn't fall – which will be very good in game situation.

4.2 Supervised Learning Setup

4.2.1 The Dataset

In order to use supervised learning for learning keyframe motions using neural networks in the HLM model, we first need to construct a dataset. A dataset consists of samples of keyframe steps. Samples were collected within the Soccer 3D environment with a frequency of 50 Hz. We acquired these samples in two different ways.

In the first one, we commanded an agent of our team to execute specific motions and sampled the reference joint positions computed by our code. In this case, we sampled the kick and get up keyframe motions (MUNIZ *et al.*, 2016). Notice that, for this approach to be successful, one needs access to the source-code.

The second approach involved changing the Soccer 3D server source-code to provide current joint positions of a given robot, in a similar way as described in (MACALPINE *et al.*, 2013). This allowed us to acquire motion datasets from other teams, without any knowledge of how these movements are implemented. In this case, we collected two types of kicks based on keyframes and sampled joint values of the walking engine (MACALPINE *et al.*, 2012).

4.2.2 Neural Network Architecture and Hyperparameters

The neural network has to be able to learn how to interpolate between samples, which actually happens. The architecture that performed best – in terms of mean absolute error minimization and simplicity – is shown in Figure 4.2. A deep neural network with 2 hidden, fully connected layers of 75 and 50 neurons was used. The output layer has 23 regression neurons, which represent the 22 joint angles and a neuron which output indicates if the motion has ended or not. The neurons in each hidden layer use the LeakyReLU activation function (XU *et al.*, 2015):

$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

where α is a small constant. This activation function was used to improve the representation capacity of the neural network, adding support for non-linear functions.

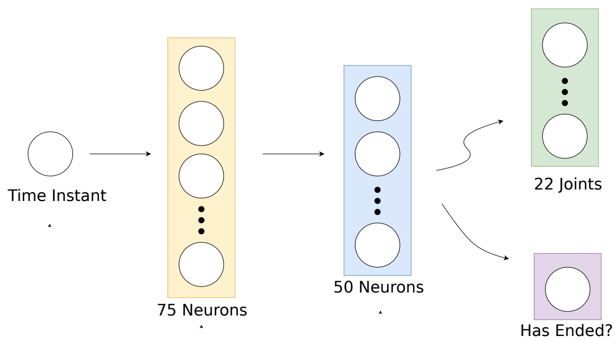


FIGURE 4.2 – The architecture of a neural network designed to learn motions.

This architecture resulted in thousand of parameters to optimize, as exposed in Table 4.1. A very high number, when compared to more traditional optimization approaches (MACALPINE *et al.*, 2012). Notice that, by increasing the number of parameters usually allows representing better movements.

TABLE 4.1 – The Network Summary

Layer	**Neurons**	**Activation**	**Parameters**
Dense	75	LeakyReLU	150
Dense	50	LeakyReLU	3800
Dense	23	Linear	1173

Total Parameters	5123

4.2.3 The Training Procedure

Since keyframe motions are executed in an open-loop fashion, the sequence of joint positions are always the same for different repetitions, independently of robot's state. Therefore, by adding samples of multiple executions of the same motion would not make our dataset richer. So, we decided to use only one repetition for each movement for faster training. In the case of the walking motion, we collected samples within one walking period.

During the training, we used 50 thousands epochs divided into 5 training phases, where the learning rate was decreased between phases, in order to achieve better performance. First, we executed 30000 epochs, by using the learning rate of 0.001. The other phases had 5000 epochs each, and we decreased the learning rate by 0.0002 in each phase.

Furthermore, we used Adam optimization (KINGMA; BA, 2014), during the whole training. The loss function used was the mean squared error, as explained in Subsec. 3.1. We decided this loss function is adequate for this problem, mainly because it strongly penalizes large errors, which can collapse the whole motion.

4.2.4 The Deployment in the Soccer 3D Environment

In order to perform the network design and the training procedure, we used the Keras (CHOLLET *et al.*, 2015) framework coupled with Tensorflow (ABADI *et al.*, 2015) as backend. After training, the weights were frozen and converted to a specific format, which was readable, by using the Tensorflow C++ API integrated within agent's code. Hence, the training was performed outside the environment, but the agent actually has computed network inferences, during the simulation execution.

4.3 Reinforcement Learning Setup

To use a Reinforcement Learning model, we need to define a policy representation and a task which the agent will follow during the process.

4.3.1 Policy Representation

The policy is represented by a neural network similar to the supervised learning model. A state is described as the stage of kick motion. An action is a set of joint values that has to be applied by the agent in that state.

In HLM, the architecture from both value function and policy networks is that described by section 4.2. On the other cases, the network is composed by three building blocks: an input normalization Filter, the neural networks themselves and a Gaussian action space noise.

4.3.1.1 Input Normalization Filter

The input normalization filter, also know as feature scaling, has the objective of normalize the input to avoid the problem of vanishing and exploding gradients by applying updates through all dimensions in equal proportions, as illustrated in Figure 4.3. This normalization happens at each pass in the network, re-calculating mean and standard deviation using the new examples. Hence, this method helps during the convergence.

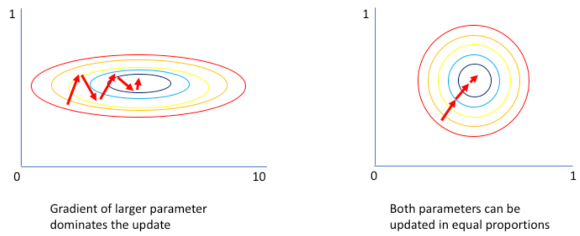


FIGURE 4.3 – Intuition behind input normalization

4.3.1.2 Neural Network

As an actor-critic model, we have two networks: one for the value function and other for the policy itself. They have the same architecture: two layer, fully connected, with 64 neurons in each hidden layer and hyperbolic tangent as activation function. The architecture is illustrated in Figure 4.4. Table 4.2 summarizes the parameters in this new architecture, also considering the parameters from the Gaussian Action Space Noise, described in 4.3.1.3.

4.3.1.3 Gaussian Action Space Noise

The last idea explored in this policy representation is the Gaussian action space noise for better exploration. It adds adaptive noise to the action space of the neural network.

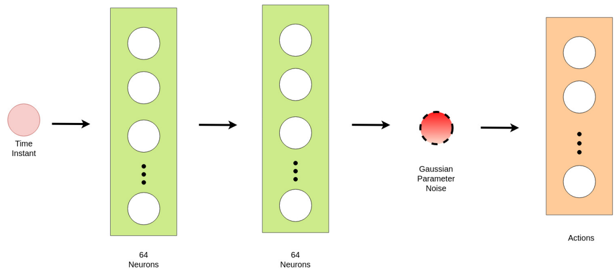


FIGURE 4.4 – Architecture used by pure Reinforcement Learning models.

TABLE 4.2 – The Reinforcement Learning Network Summary

Layer	Neurons	Activation	Parameters
Dense	64	*tanh*	128
Dense	64	*tanh*	4160
Output	23	Linear	1495
Noise	23	Linear	23

Total Parameters	5806

Discrete RL uses ϵ-greedy (WATKINS, 1989) to confer exploration. Gaussian action space noise injects normal randomness directly into the actions of the agent, altering the types of decisions it makes and helping algorithms explore continuous environments more effectively. The Figure 4.5 illustrates this technique.

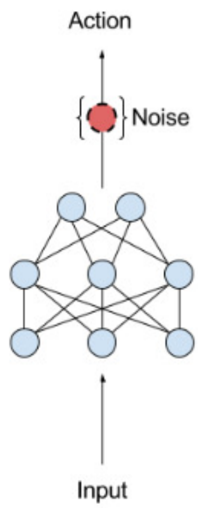


FIGURE 4.5 – Gaussian noise applied to action space to ensure better exploration in continuous environments. (PLAPPERT *et al.*, 2017)

4.3.2 Task Description

5 Results' Analysis and Discussion

5.1 Training Results

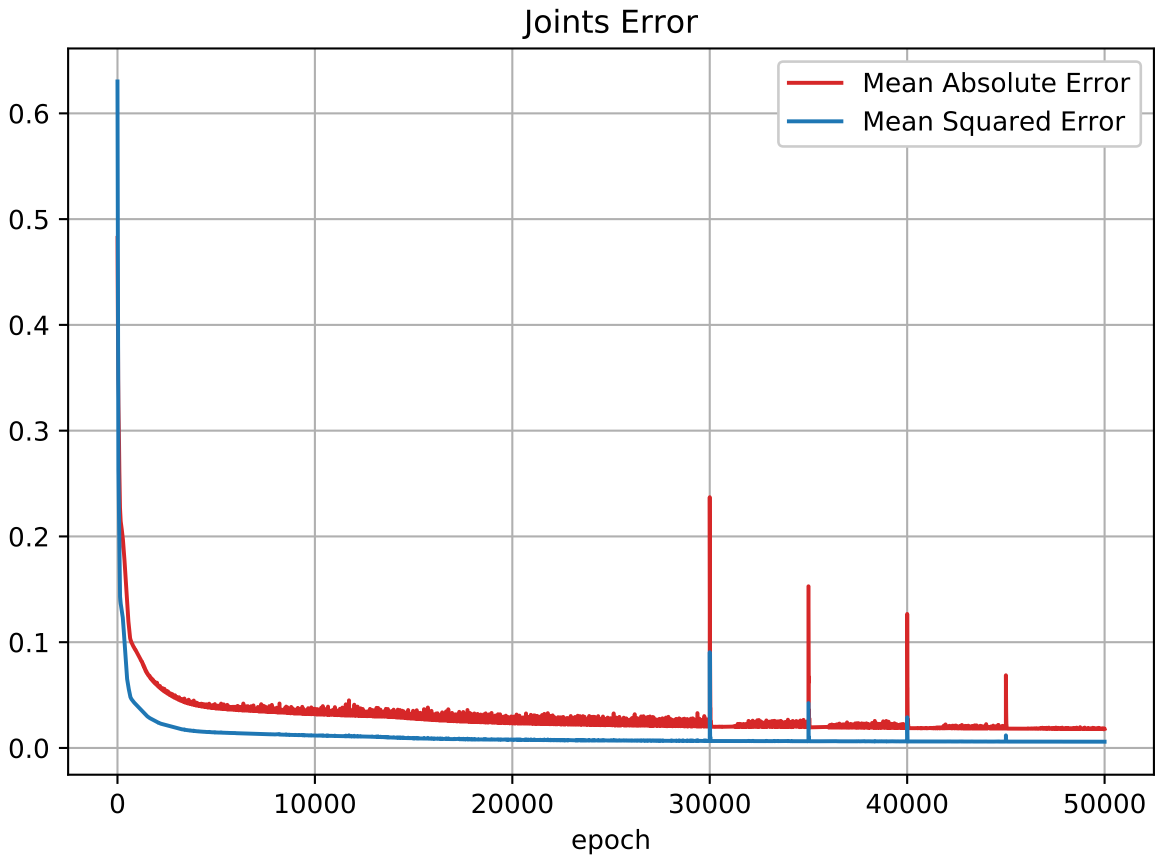


FIGURE 5.1 – Plots of mean squared error and mean absolute error, during training.

The initial results came from the training procedure, outside the simulation environment. Figure 5.1 presents training curves for the kick keyframe dataset. In this case, plots show the mean squared error and the mean absolute error metrics. In both metrics, the value drastically decreases in the first epochs. This same behavior was present in other training procedures as well. However, only after thousands of epochs, the network has achieved a low error that successfully reproduced the motion, which has shown how sensible to small joint errors keyframes were, given that they were open-loop motions.

The peaks, during the training has happened at the learning rate transition instants, but they did not hurt the training procedure itself.

5.2 The Learned Kick Motion

The final mean absolute error was **0.018** radians and the motion was visually indistinguishable from the original one, as can be seen in Figure 5.2. In this figure, snapshots from both motions were taken. Figure 5.3 shows several plots of joint angles, by comparing the original and learned kick motions. As we may see, the learned motion has fitted the movement with minor errors[1].

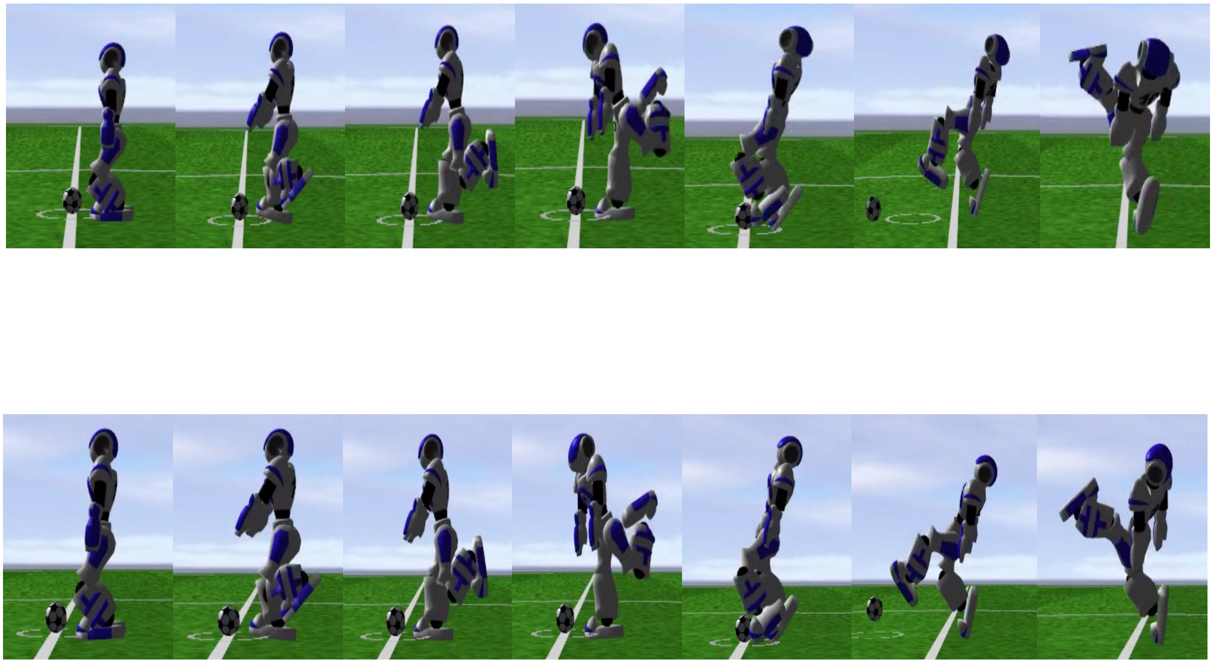


FIGURE 5.2 – The kick motion. The first row of figures shows the original kick motion. The second row shows the learned kick motion. Both motions are visually indistinguishable.

In order to evaluate the learned kick motion in the RoboCup Soccer 3D domain, we created a statistical test. Inside the test scenario, the ball was initially placed in the center of the field with an agent near to it. The only action of the agent was to kick the ball in the goal direction. After the kick, the agent run until reaching the ball and kicked it again, repeating this process till scoring a goal. When the goal has occurred, this same scenario was repeated. The whole test was conducted, during thirty minutes in clock time and the following data was collected: total number of kicks, number of successful kicks, mean distance that the ball has traveled, and the standard deviation of this measure. The results from the original and learned kicks is shown in Table 5.1.

[1] Kick results video: https://youtu.be/UAbqQLUnvDo

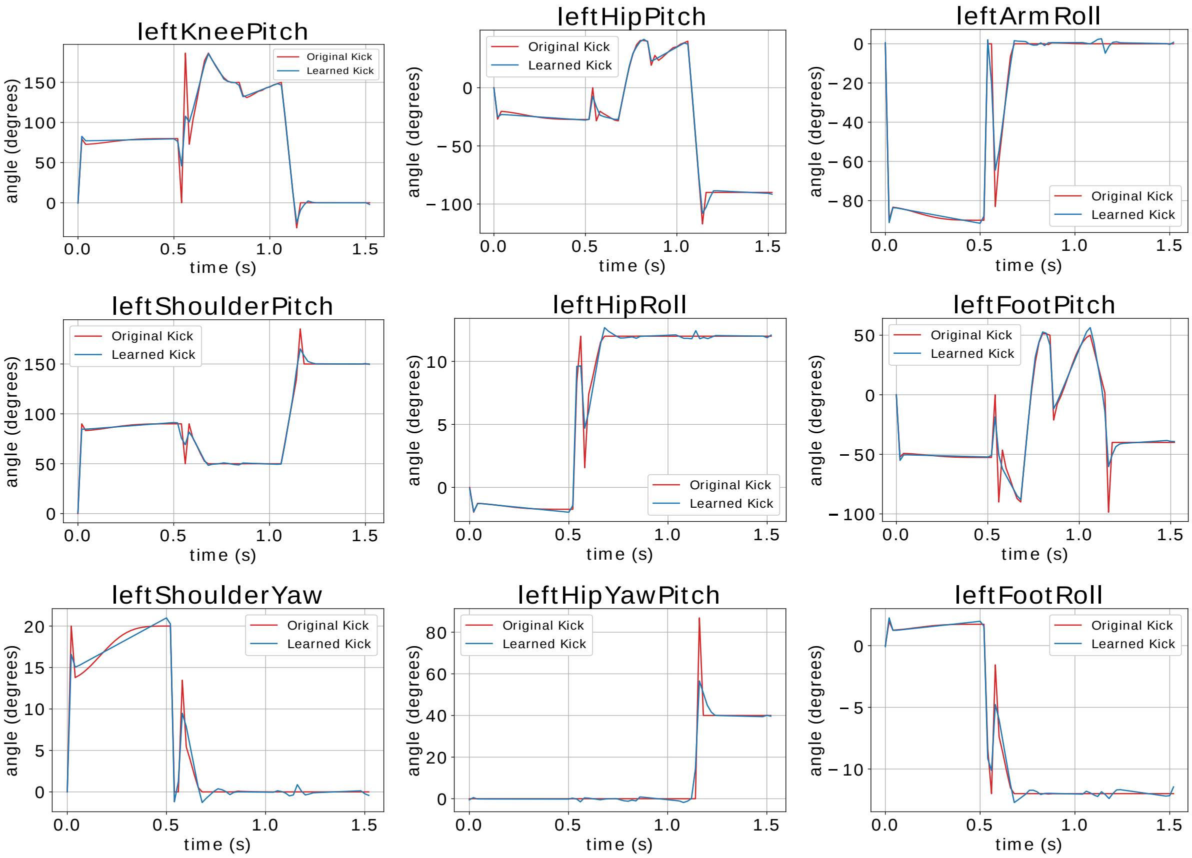


FIGURE 5.3 – Joint values for comparing original and learned kicks. The neural network was able to fit the joint trajectories with small errors.

TABLE 5.1 – The Kick Comparison

Kick Type	**Statistics**		
	Accuracy (%)	**Distance** (m)	
		Mean	***Std***
Original Kick	64.5	8.92	3.82
Neural Kick	52.6	7.16	4.06

Although both kicks had similar results, the original kick was slightly better in this scenario. By confronting Figure 5.3, we can conclude that even with an almost equal representation, the kick lost part of its efficiency and this fact has shown how sensible were movements based on keyframe data.

5.3 The Learned Walk Motion

By using the modified server described in Subsec. 4.2.1, a dataset with samples of the UT Austin Villa's walking motion (MACALPINE *et al.*, 2013) was acquired. This team is the current champion of the RoboCup Soccer 3D competition (MACALPINE; STONE, 2018).

The objective was to mimic the walk motion as a keyframe and has used that in our agent. The previously described framework for learning our own kick motion was used in this training, by including the neural network architecture and its hyperparameters.

The results from this training are shown in Figure 5.4. Similarly to Figure 5.3, it shows the joint angles throughout the walking motion period for the original and learned walk. Additionally, it shows the real joints values from the movement in the server. These joints were chosen because they were the most dynamic in the walk motion and, therefore, the hardest to learn.

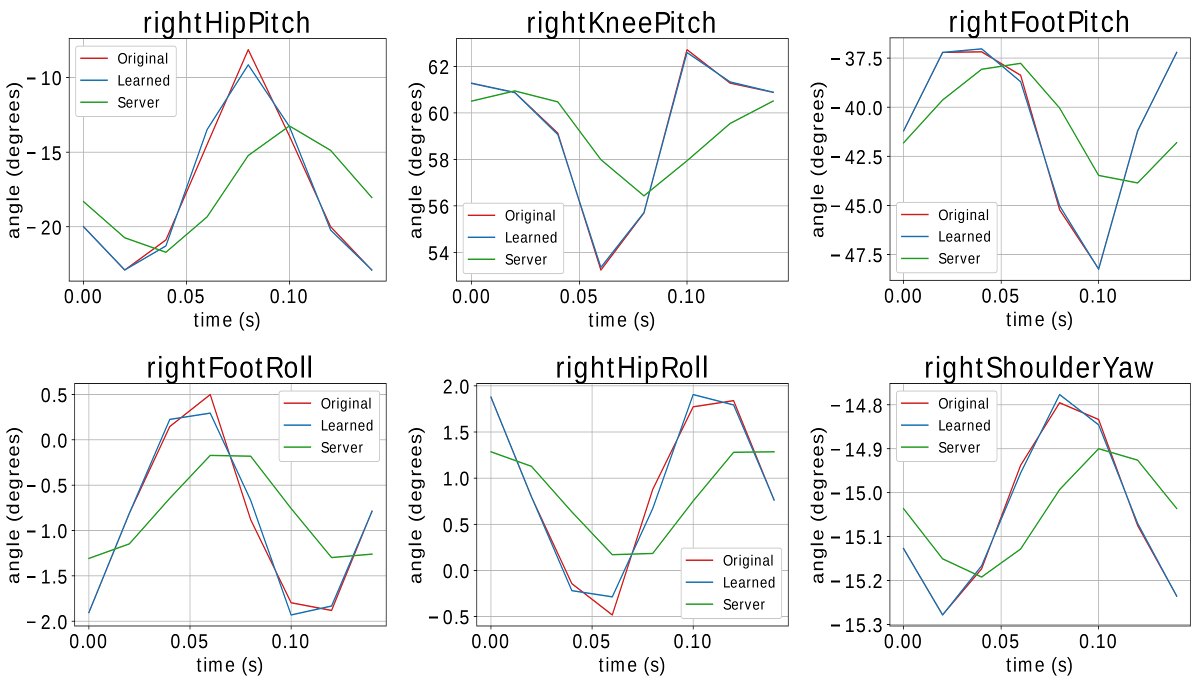


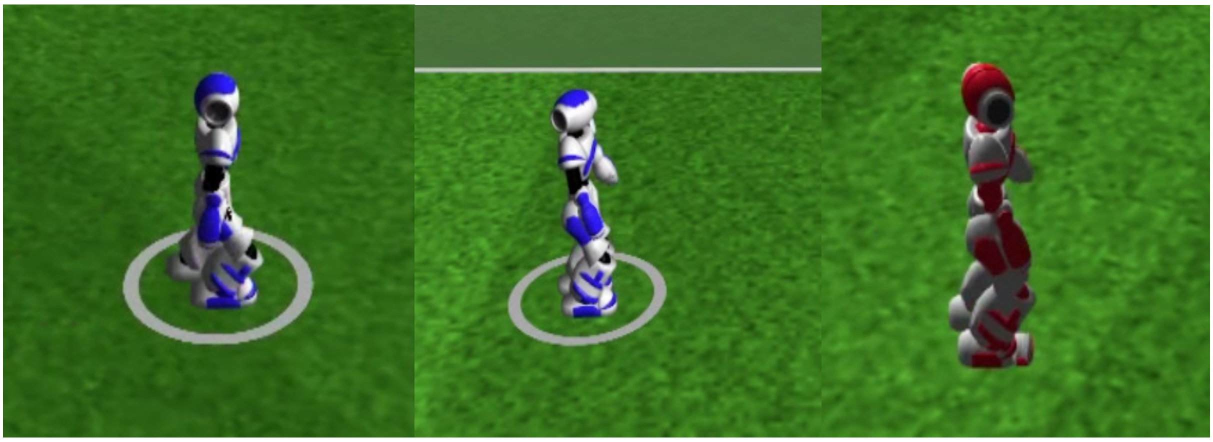
FIGURE 5.4 – Joints positions, during a period of the walking motion for the original walk, and the learned walk and the joints positions effectively attained, during the learned walking motion.

The learned motion has fitted the dataset very well. However, these values were just desired joints. In fact, these values were used as references to joint controllers and were also attenuated due to joint dynamics. Furthermore, this motion was operated in a open-loop fashion, so the agent was not able to correct its own trajectory, and this walks got biased within the simple task of walking straight forward.

Despite the facts previously described, the motion has worked well in a non-competitive scenario[2], which was shown in the metrics collected from the Forward Walk test scenario – agent walking forward from the goal post until the center line of the field – in Table 5.2 and the visual representation in Figure 5.5.

TABLE 5.2 – Walk Comparison - Forward Walk

Walk Type	**Statistics**			
	Velocity (m/s)		**Y Error** (m)	
	Mean	***Std***	***Mean***	***Std***
Original Walk	0.87	0.01	-	-
Learned Walk	0.23	0.01	0.96	2.63



(a) (b) (c)

FIGURE 5.5 – The walking motions comparison. Figure (a) shows our agent in its regular walk, Figure (b) shows the same agent mimicking UT Austin Villa walk, and Figure (c) shows the UT Austin Villa agent itself performing his own walking motion.

5.4 Other motions

This same framework was used to learn other keyframe motions originated from our agent itself, such as the get up motion. As the cases previously described, the resultant

[2] Walk results video: https://youtu.be/-pHxTrxllyY

neural network was capable of mimicking the keyframe, by including its interpolation. Hence, all of our keyframe motions could be replaced by neural motions with similar performance.

However, the huge improvement of this method was about mimicking other teams motions. In the Soccer 3D environment, movements like kick and walking have giant impact in team's performance. With this learning framework, our agent was able to mimic multiples movements from several teams.

As an example, we have collected data from UT Austin Villa kick, which was originally optimized by using Deep Reinforcement Learning techniques (MACALPINE; STONE, 2017). Our agent has learned this kick without any additional optimization strategy: we just have used samples collected from the modified server.

6 Conclusions, Recommendations, and Future Works

6.1 Preliminary Conclusions and Future Works

In this work, we presented a method for learning humanoid robot movements using datasets composed of joint values at each time instant. The provided learning framework was capable to learn several types of motion, including walk and kick, without any change in network architecture or hyperparameters.

Moreover, the learned motions had similar performance to the original ones. Furthermore, this framework was able to learn from other teams motions, without any knowledge about the underlying implementation – only by using the joints values provided by a modified version of the server. This was a huge improvement, in terms of getting improved motions, as our agent was able to mimic other teams motions, by using this machine learning technique.

As future works, we plan to apply Deep Reinforcement Learning algorithms to obtain faster and more robust kicks, by using as a ”seed” the neural network obtained from this work.

Another track to be followed is to transfer the learning of this obtained network to a new network that represents the motion policy itself (i.e a network which has as inputs the current state of the robot, by including joint and link states, besides the current time instant), optimizing this motion policy, in order to get a closed-loop walking and kicking motion that can correct itself.

Finally, as a long term goal, we intend to create some model-free walking and kicking engines.

6.2 The Activities Plan

As next steps of this work, we plan to:

1. Import our supervised policy model into Deep Reinforcement Learning algorithms – Expected to be finished until the end of **June, 2018**;

2. Iterate over objective function construction and optimization, in order to improve the kick motion – Expected to be finished until mid **August, 2018**;

3. Execute the same test scenarios previously applied to compare results – Expected to be finished until the end of **August, 2018**;

4. Test in 11 x 11 game to compare team performance – Expected to be finished until the end of **August, 2018**; and

5. Complement this work with some novel background, methodology, results, and conclusions – Expected to be finished until **November, 2018**.

Bibliography

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

ANDRYCHOWICZ, M.; WOLSKI, F.; RAY, A.; SCHNEIDER, J.; FONG, R.; WELINDER, P.; MCGREW, B.; TOBIN, J.; ABBEEL, P.; ZAREMBA, W. Hindsight experience replay. **CoRR**, abs/1707.01495, 2017. Disponível em: <http://arxiv.org/abs/1707.01495>.

BARTELS, R. H.; BEATTY, J. C.; BARSKY, B. A. **An Introduction to Splines for Use in Computer Graphics &Amp; Geometric Modeling**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987. ISBN 0-934613-27-3.

BROCKMAN, G.; DENNISON, C.; ZHANG, S.; JÓZEFOWICZ, R.; TANG, J.; WOLSKI, F.; SIDOR, S.; DĘBIAK, P.; FARHI, D.; PONDÉ, H.; RAIMAN, J.; PETROV, M.; CHAN, B.; PACHOCKI, J.; CHRISTIANO, P.; RADFORD, A.; SIGLER, E.; CLARK, J.; GRAY, S.; YOON, D.; SUTSKEVER, I.; SCHIAVO, L.; SCHULMAN, J.; FISCHER, Q.; HASHME, S.; LUAN, D.; HESSE, C.; BERNER, C.; SCHNEIDER, J. Openai five. In: . [s.n.], 2018. Disponível em: <https://blog.openai.com/openai-five/>.

CHOLLET, F. *et al.* **Keras**. 2015. Https://keras.io.

COLLINS, S.; RUINA, A.; TEDRAKE, R.; WISSE, M. Efficient Bipedal Robots Based on Passive Dynamic Walkers. **Science Magazine**, v. 307, p. 1082–1085, February 2005.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. `http://www.deeplearningbook.org`: MIT Press, 2016.

GOUAILLIER, D.; HUGEL, V.; BLAZEVIC, P.; KILNER, C.; MONCEAUX, J.; LAFOURCADE, P.; MARNIER, B.; SERRE, J.; MAISONNIER, B. Mechatronic design of nao humanoid. In: **2009 IEEE International Conference on Robotics and Automation**. Kobe, Japan: IEEE, 2009. p. 769–774. ISSN 1050-4729.

HANSEN, N. The CMA evolution strategy: A tutorial. **CoRR**, abs/1604.00772, 2016. Disponível em: <http://arxiv.org/abs/1604.00772>.

HEESS, N.; TB, D.; SRIRAM, S.; LEMMON, J.; MEREL, J.; WAYNE, G.; TASSA, Y.; EREZ, T.; WANG, Z.; ESLAMI, S. M. A.; RIEDMILLER, M. A.; SILVER, D. Emergence of locomotion behaviours in rich environments. **CoRR**, abs/1707.02286, 2017. Disponível em: <http://arxiv.org/abs/1707.02286>.

HO, J.; ERMON, S. Generative adversarial imitation learning. **CoRR**, abs/1606.03476, 2016. Disponível em: <http://arxiv.org/abs/1606.03476>.

KAJITA, S.; KANEHIRO, F.; KANEKO, K.; YOKOI, K.; HIRUKAWA, H. The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation. In: **In Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems**. Hawaii, USA: IEEE, 2001.

KHAN, N. A parallel implementation of the covariance matrix adaptation evolution strategy. **CoRR**, abs/1805.11201, 2018. Disponível em: <http://arxiv.org/abs/1805.11201>.

KINGMA, D. P.; BA, J. Computer science > learning adam: A method for stochastic optimization. **arXiv**, Dec 2014.

KITANO, H.; ASADA, M.; KUNIYOSHI, Y.; NODA, I.; OSAWAI, E.; MATSUBARA, H. Robocup: A challenge problem for ai and robotics. In: KITANO, H. (Ed.). **RoboCup-97: Robot Soccer World Cup I**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. p. 1–19. ISBN 978-3-540-69789-3.

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>.

LU, C.; TANG, X. Surpassing human-level face verification performance on LFW with gaussianface. **CoRR**, abs/1404.3840, 2014. Disponível em: <http://arxiv.org/abs/1404.3840>.

MACALPINE, P.; BARRETT, S.; URIELI, D.; VU, V.; STONE, P. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In: **Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)**. Toronto, Ontario, Canada: AAAI, 2012.

MACALPINE, P.; COLLINS, N.; LOPEZ-MOBILIA, A.; STONE, P. Ut austin villa: Robocup 2012 3d simulation league champion. In: CHEN, X.; STONE, P.; SUCAR, L. E.; ZANT, T. van der (Ed.). **RoboCup 2012: Robot Soccer World Cup XVI**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 77–88. ISBN 978-3-642-39250-4.

MACALPINE, P.; STONE, P. Prioritized role assignment for marking. In: BEHNKE, S.; LEE, D. D.; SARIEL, S.; SHEH, R. (Ed.). **RoboCup 2016: Robot Soccer World Cup XX**. Berlin: Springer Verlag, 2016, (Lecture Notes in Artificial Intelligence). p. 306–18.

MACALPINE, P.; STONE, P. UT Austin Villa 3D Simulation Soccer Team 2017. In: . Nagoya, Japan: RoboCup Symposium, 2017.

MACALPINE, P.; STONE, P. UT Austin Villa: RoboCup 2017 3D simulation league competition and technical challenges champions. In: SAMMUT, C.; OBST, O.; TONIDANDEL, F.; AKYAMA, H. (Ed.). **RoboCup 2017: Robot Soccer World Cup XXI**. Japan: Springer, 2018, (Lecture Notes in Artificial Intelligence).

MACALPINE, P.; URIELI, D.; BARRETT, S.; KALYANAKRISHNAN, S.; BARRERA, F.; LOPEZ-MOBILIA, A.; STIURCA, N.; VU, V.; STONE, P. **UT Austin Villa 2011 3D Simulation Team Report**. Austin, Texas, December 2011.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T. P.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. **CoRR**, abs/1602.01783, 2016. Disponível em: <http://arxiv.org/abs/1602.01783>.

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, February 2015. ISSN 00280836. Disponível em: <http://dx.doi.org/10.1038/nature14236>.

MUNIZ, F.; MAXIMO, M. R. O. A.; RIBEIRO, C. H. C. Keyframe movement optimization for simulated humanoid robot using a parallel optimization framework. In: **2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)**. Recife, Brazil: IEEE, 2016. p. 79–84.

OBST, O.; ROLLMAN, M. Spark – a generic simulator for physical multiagent simulation. **Computer Systems Science and Engineering**, v. 5, p. 347–356, 2005.

PENG, X. B.; ABBEEL, P.; LEVINE, S.; PANNE, M. van de. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. **ACM Transactions on Graphics (Proc. SIGGRAPH 2018 - to appear)**, v. 37, n. 4, 2018.

PLAPPERT, M.; HOUTHOOFT, R.; DHARIWAL, P.; SIDOR, S.; ABBEEL, P.; ANDRYCHOWICZ, M.; CHEN, R.; CHEN, X.; ASFOUR, T. Better exploration with parameter noise. In: . [s.n.], 2017. Disponível em: <https://blog.openai.com/better-exploration-with-parameter-noise/>.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 533, 1988.

RUMMERY, G. A.; NIRANJAN, M. **On-Line Q-Learning Using Connectionist Systems**. [S.l.], 1994.

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>.

SHON, A. P.; GROCHOW, K.; RAO, R. P. N. Robotic imitation from human motion capture using gaussian processes. In: **In Proceedings of the IEEE/RAS International Conference on Humanoid Robots (Humanoids**. Tsukuba, Japan: [s.n.], 2005.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T.; LILLICRAP, T. P.; SIMONYAN, K.; HASSABIS, D. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. **CoRR**, abs/1712.01815, 2017. Disponível em: <http://arxiv.org/abs/1712.01815>.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T.; LILLICRAP, T. P.; SIMONYAN, K.; HASSABIS, D. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. **CoRR**, abs/1712.01815, 2017. Disponível em: <http://arxiv.org/abs/1712.01815>.

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, p. 354–, October 2017. Disponível em: <http://dx.doi.org/10.1038/nature24270>.

TANIKIC, D.; DESPOTOVIC, V. Artificial intelligence techniques for modelling of temperature in the metal cutting process. **IntechOpen**, September 2012.

TODOROV, E.; EREZ, T.; TASSA, Y. Mujoco: A physics engine for model-based control. In: **IROS**. [S.l.]: IEEE, 2012. p. 5026–5033. ISBN 978-1-4673-1737-5.

URIELI, D.; MACALPINE, P.; KALYANAKRISHNAN, S.; BENTOR, Y.; STONE, P. On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer. In: TUMER, K.; YOLUM, P.; SONENBERG, L.; STONE, P. (Ed.). **Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)**. [S.l.]: IFAAMAS, 2011. v. 2, p. 769–776. ISBN 978-0-9826571-5-7.

WANG, Z.; BAPST, V.; HEESS, N.; MNIH, V.; MUNOS, R.; KAVUKCUOGLU, K.; FREITAS, N. de. Sample efficient actor-critic with experience replay. **CoRR**, abs/1611.01224, 2016. Disponível em: <http://arxiv.org/abs/1611.01224>.

WATKINS, C. J. C. H. **Learning from Delayed Rewards**. Tese (Doutorado) — King's College, Cambridge, UK, May 1989. Disponível em: <http://www.cs.rhul.ac.uk/˜chrisw/new_thesis.pdf>.

XIONG, W.; DROPPO, J.; HUANG, X.; SEIDE, F.; SELTZER, M.; STOLCKE, A.; YU, D.; ZWEIG, G. Achieving human parity in conversational speech recognition. **CoRR**, abs/1610.05256, 2016. Disponível em: <http://arxiv.org/abs/1610.05256>.

XU, B.; WANG, N.; CHEN, T.; LI, M. Empirical evaluation of rectified activations in convolutional network. **arXiv**, Nov 2015.

XU, Y.; VATANKHAH, H. Simspark: An open source robot simulator developed by the robocup community. In: BEHNKE, S.; VELOSO, M.; VISSER, A.; XIONG, R. (Ed.). **RoboCup 2013: Robot World Cup XVII**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 632–639. ISBN 978-3-662-44468-9.

FOLHA DE REGISTRO DO DOCUMENTO			
1. CLASSIFICAÇÃO/TIPO TC	2. DATA June 12th, 2018	3. DOCUMENTO N DCTA/ITA/DM-018/2015	4. N DE PÁGINAS 41

5. TÍTULO E SUBTÍTULO:

A Deep Reinforcement Learning Method for Humanoid Kick Motion

6. AUTOR(ES):

Luckeciano Carvalho Melo

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

Aeronautics Institute of Technology – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Deep Reinforcement Learning; Robotics; Artificial Intelligence

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Deep Reinforcement Learning; Robotics; Artificial Intelligence

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

ITA, São José dos Campos, 2018. Trabalho de Graduação.

11. RESUMO:

Controlling a high degrees of freedom humanoid robot is acknowledged as one of the hardest problems in Robotics. Due to the lack of mathematical models, an approach frequently employed is to rely on human intuition to design keyframe movements by hand, usually aided by graphical tools. In this paper, we propose a learning framework based on neural networks in order to mimic humanoid robot movements. The developed technique does not make any assumption about the underlying implementation of the movement, therefore both keyframe and model-based motions may be learned. The framework was applied in the RoboCup 3D Soccer Simulation domain and promising results were obtained using the same network architecture for several motions, even when copying motions from another teams.

12. GRAU DE SIGILO:

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