INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Thiago Filipe de Medeiros

REINFORCEMENT LEARNING APPLIED TO IEEE'S VSS SOCCER STRATEGY

Bachelor's Thesis
2019

Course of Computer Engineering

Thiago Filipe de Medeiros

REINFORCEMENT LEARNING APPLIED TO IEEE'S VSS SOCCER STRATEGY

Advisor

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

Co-advisor

Prof. Dr. Takashi Yoneiama (ITA)

COMPUTER ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2019

Abstract

Designing the behavior of an agent for an environment with uncertainty and unfeasible number of configurations as in robot soccer is a difficult problem in Robotics. Due to the lack of mathematical models, a typical approach is to rely on heuristics and human intuition to design specific actions for specific states of the environment. In this preliminary work, we propose a learning framework for IEEE's Very Small Size Soccer competition based on neural networks in order to train and improve the behavior of agents within the game strategy. The developed technique aims to be model free, relying, in principle, only on the virtual interaction of the agent and the environment. The framework will be applied to the IEEE's VSS Soccer domain, intending to be a scalable way of developing strategies.

List of Figures

Contents

1 Introduction

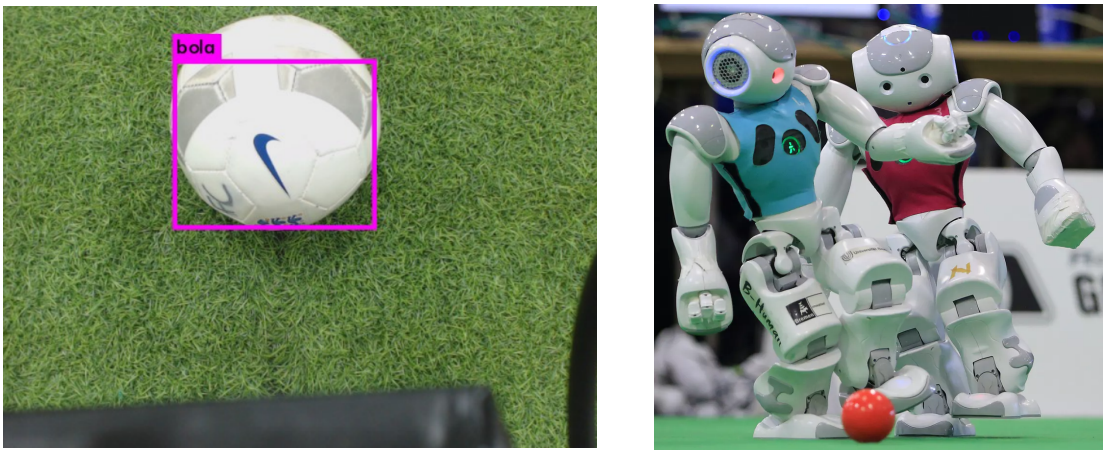
1.1 Motivation

In the context of Computer Engineering, Robotics is an interdisciplinary area with intense research and interesting practical applications, especially in industry (as exemplified on **Figure. 1.1**), that frequently comes across open problems and, as a result, "state of the art" techniques are further advanced to overcome those barriers.

FIGURE 1.1 – Industrial robots in a car assembly line (WINCHESTER, 2017).

A typical controlled environment for developing on this area is robot soccer, which encompasses challenges in computer vision (as in **Figure. 1.2a**), path planning, control (as shown in **Figure. 1.2b**), signal processing, reasoning and multi-agent strategy.

Recently, due to advances on Artificial Intelligence (AI) research, in particular on the fields of Deep Learning and Reinforcement Learning, such techniques could then been used to achieve superhuman performance on complex activities, such as: reconnaissance and detection of objects on images (LU; TANG, 2014), planning and reasoning into environments with a prohibitive number of configurations for an exhaustive search ((SILVER



(a) (b)

FIGURE 1.2 – Techniques applied in robot soccer: (*a*) Object detection used for ball detection (SUSANTO *et al.*, 2017) and (*b*) Control theory for controlling robot joints (POLANI, 2016).

et al., 2017), as seen in **Figure. 1.3**), and control of a humanoid agent locomotion (HEESS *et al.*, 2017). Hence, those new Machine Learning techniques seen like good candidates as tools to overcome current problems in robotics.

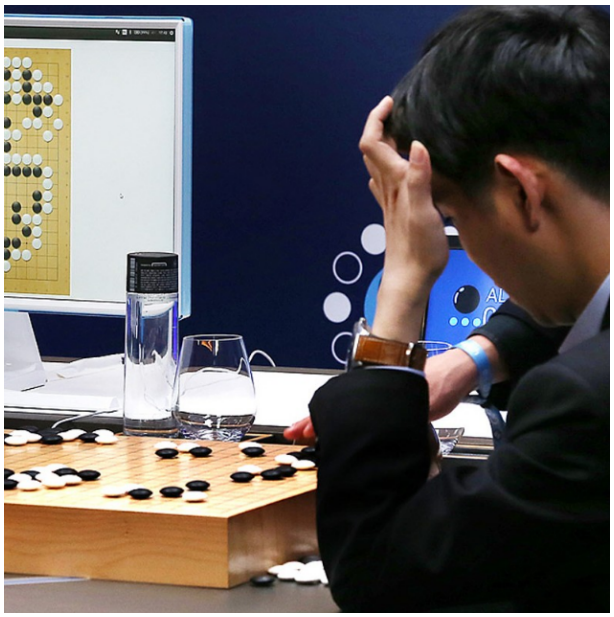
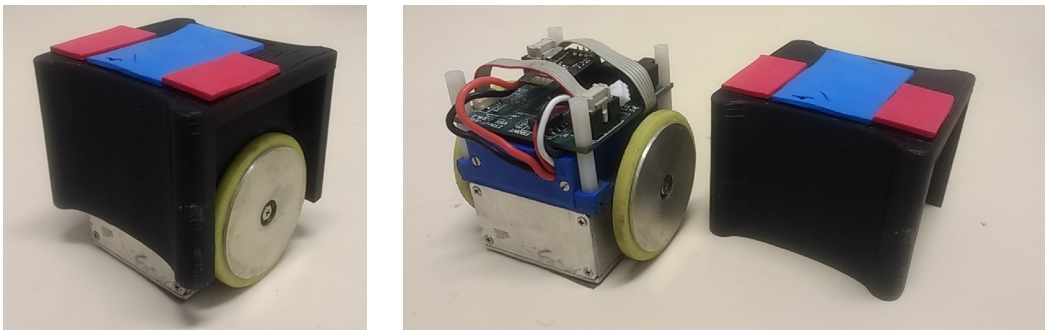


FIGURE 1.3 – 2016's Go match between world champion Lee Sedol and AlphaGo program, developed by Google's DeepMind branch, where the computer program won by a 4 to 1 score against the human player (LAKE, 2017).

1.2 Contextualization

IEEE stands for Institute of Electrical and Electronic Engineers and it is a large technical professional organization dedicated to advancing technology for the benefit of humanity (IEEE, 2019a). In particular, IEEE Robotics & Automation Society (RAS) is a branch of IEEE which, besides conferences and publications on those subjects, sponsors robot competitions (RAS, 2019).

IEEE's Very Small Size Soccer (VSSS), is a competition where two teams of small differential robots (of dimensions up to 7.5 $cm \times 7.5\ cm \times 7.5\ cm$, as shown in **Figure. 1.4**) play a soccer game in a field on the ground (of dimensions around $1.5\ m \times 1.3\ m$, as depicted in **Figure. 1.5**). The robots are remotely controlled by a computer, but without human intervention. The computer processes the image of a video camera placed above the field and commands the robots via radio. Usually the competitions are, with respect to the number of players, on the 3×3 format, but recently there has been an expectation for 5×5 games (IEEE, 2019b).



(a) (b)

FIGURE 1.4 – Exemplary of VSS Soccer robot: (a) with top cover, as used during the match and (b) without cover, with exposed electronic circuit.

The teams face many challenges when designing and building the robots. Aside Mechanic and Electronic Engineering challenges, the Computer Engineering challenges are:

- Embedded software: design the embedded software which is responsible for receiving and decoding radio messages and controlling the wheels speeds;
- Object detection: design an algorithm to recognize the robots given the image captured by the camera and the predetermined colored patterns on the top cover of the robots;
- Estimation: design an algorithm to compute and predict positions and velocities, given current position and previous positions of the players, opponents and ball;

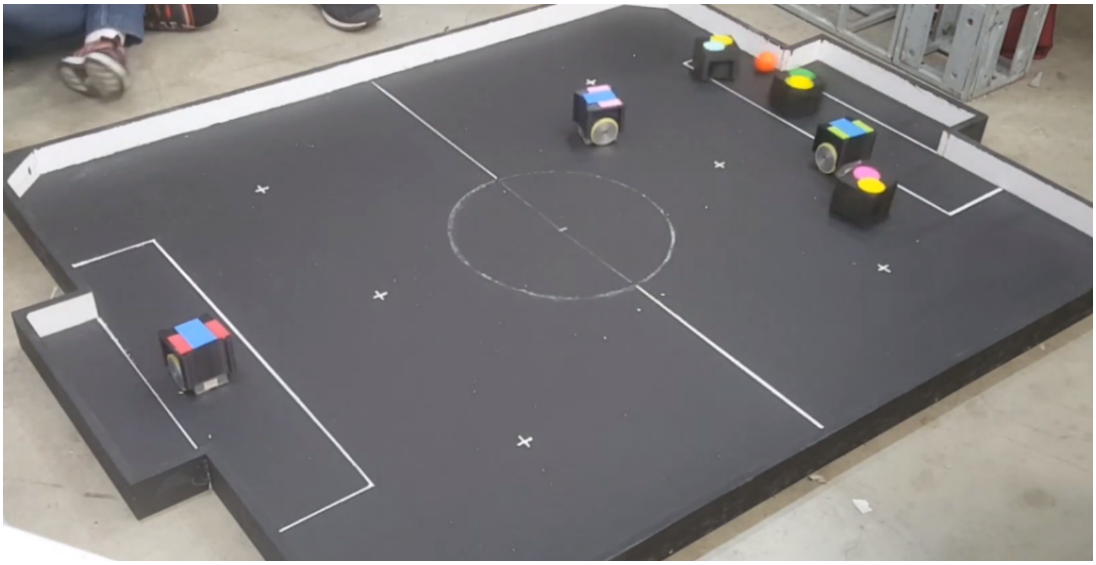


FIGURE 1.5 – A frame of a VSS Soccer game. Each team has a camera fixed on the top of the field in a, not shown, rigid structure (ITANDROIDS, 2018).

- Strategy and Reasoning: design an algorithm to decide the next configuration of the robots, given the current positions and velocities;
- Path planning: design an algorithm that, given the high level decision of the strategy, plans a suitable path to achieve the goal;
- Control: design an algorithm that, given the path to be followed, determines the wheel speeds to be sent to the robots;
- Radio Communication: design an algorithm that, given the wheel speed of each robot, sends this information via radio to the robots' receivers, using a prestablished protocol.

Also, it is usual for the teams to have a virtual simulator, since extensive tests can easily deteriorate the hardware. Hence, there is an extra challenge of designing and creating a simulator that mimetizes the game physics, and it is able to run the in-game strategy.

The challenges to be focused on this work are the ones of Strategy, Path Planning and Control. More specifically, to apply Reinforcement Learning techniques to train and create improved agents to support the other players in offensive and defensive moves.

1.3 Objective

In the context of IEEE's VSS Soccer competitions, we aim to use state-of-the-art Deep Reinforcement Learning techniques for continuous spaces such as Proximal Policy

Optimization (PPO) to train and improve agents, in a virtual simulator, with respect to two scenarios:

- A support player for offensive strategies, concerned with the maintenance of the ball on the opponent's side of the field while other players try to lead the ball to the goal;
- A support player for defensive strategies, concerned with taking the ball from the player's side of the field and lead it to the opponent's so as to engage into a offensive strategy.

The result will be contrasted in the simulated environment with the hand-coded Behavior Tree based strategy used by ITAndroids' VSS Soccer team, winner of the 2018 Latin America Robot Competition (LARC) (MAXIMO *et al.*, 2019).

1.4 Outline of this Work

The thesis is organized as follows: **Chapter 2** describes the simulated environment ITAndroids' VSS Soccer currently uses to evaluate the game strategy. **Chapter 3** will cover the theoretical background of Supervised Learning and Reinforcement Learning. **Chapter 4** outlines how the problem is modeled and will be attacked, as well as the core tools to be used. Finally, **Chapter 5** concludes with the intended next steps of this work.

2 VSS Simulated Environment

2.1 The virtual simulator

ITAndroids' simulated environment (depicted in **Figure. 2.1**) it is an application built on C++ language that runs on Linux Operational System and it is based on ODE and OGRE libraries. Open Dynamics Engine (ODE) was designed for high fidelity simulation of rigid body dynamics (SMITH, 2019), and it is used to create the physics of the environment. Objected-Oriented Graphics Rendering Engine (OGRE) it is a 3D rendering engine (GOLDBERG *et al.*, 2019), and it is used to generate 3D graphics to allow the visualization of the simulation.

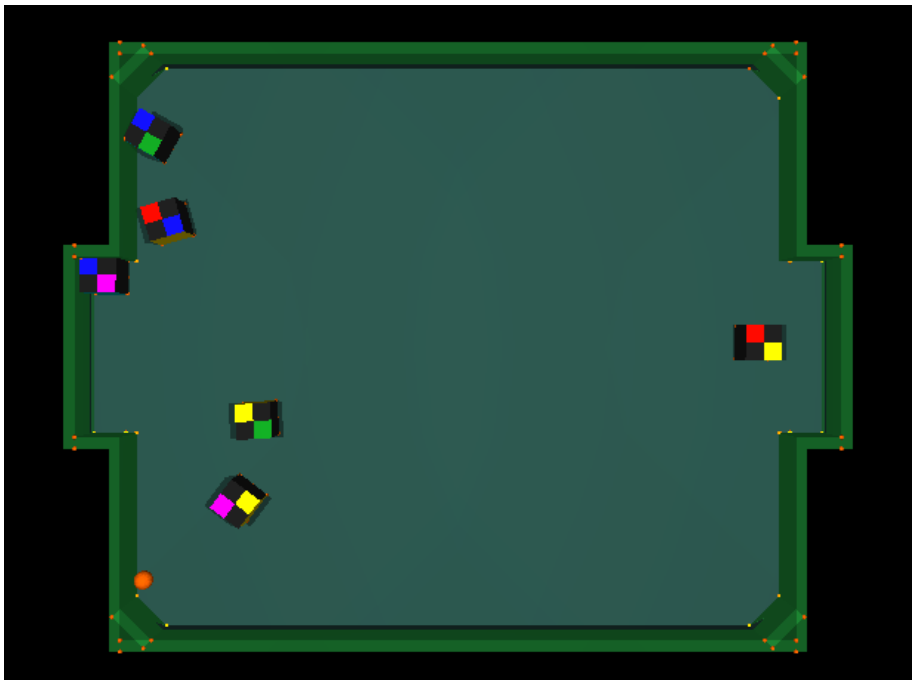


FIGURE 2.1 – A frame of the ITAndroids' VSS Soccer simulator.

The simulator supports games on 3×3 format, as well as the 5×5. It is also able to run matches between teams with different strategies, because the simulator and the teams are distinct processes that communicates via sockets. Hence, distinct strategies can be compiled into different binaries and be concurrently executed with the simulator, which

simulates the match, sending messages with the current environment state to both teams and receiving the wheel speed commands for each robot from both teams, as schematized in **Figure. 2.2**.

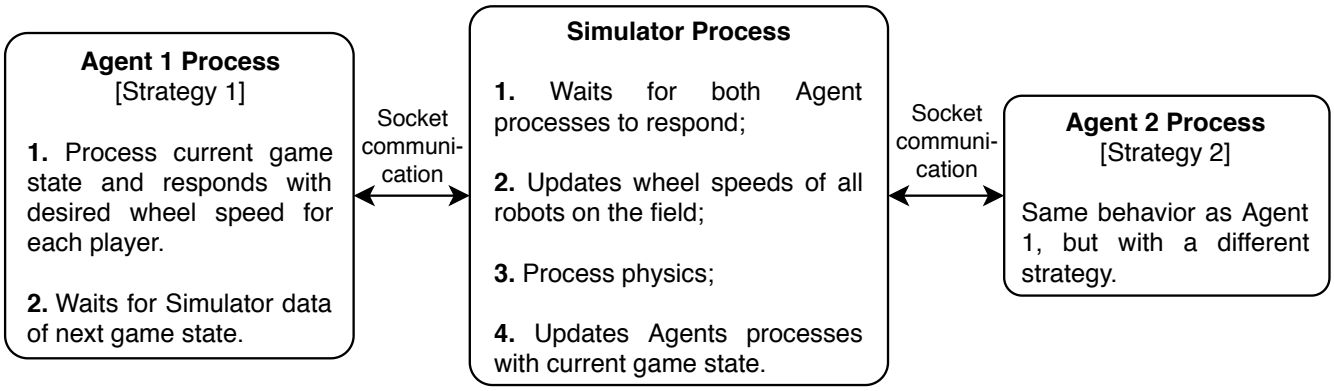


FIGURE 2.2 – Processes communication schema.

2.2 The Agents

The agents' application, which is the same program as the one used during real matches in the competition, captures data from the camera and estimates the current state of the game, then performs a strategic decision that culminates on the determination of the velocities of each wheel of the robots, which are send, via radio, to the robots, as schematized in **Figure. 2.3**.

In the case of the simulation, the data captured from the camera and the sending of wheel speeds via radio are replaced with socket messages between the agents' applications and the simulator.

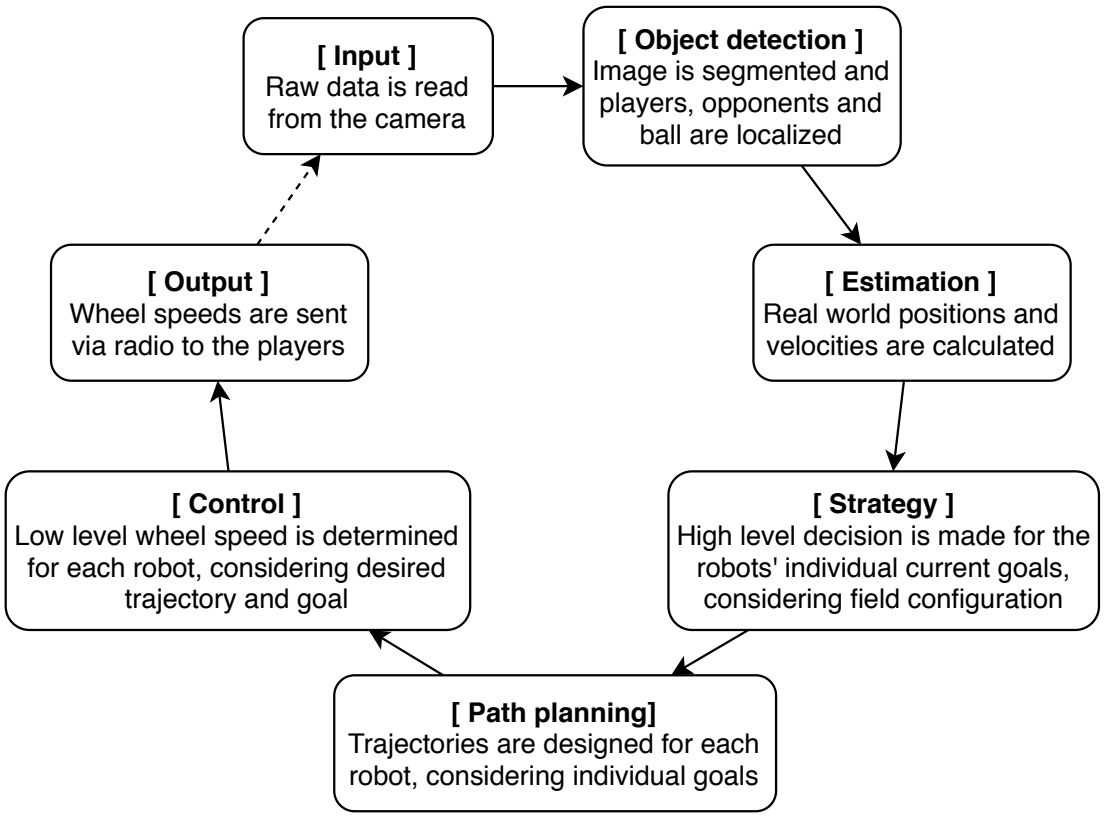


FIGURE 2.3 – Agent processing cycle schema. After the current wheel speed command is sent to the robots via radio, a new cycle begins.

3 Machine Learning

3.1 Feedforward Neural Networks

3.1.1 Definitions

Capable of approximating functions with arbitrary precision, given enough data, Feedforward Neural Networks are commonly used to approximate an unknown function f^*, from which input-output pairs (x, y) are sampled, by learning parameters θ such that the mapping $y = f(x; \theta)$ is accurate (GOODFELLOW *et al.*, 2017).

The building blocks of Feedforward Neural Networks are the so called *perceptrons*, as seen in **Figure. 3.1**, where the w_i are the input weights, b is the bias, ϕ is the activation function and y is the output of the *perceptron*.

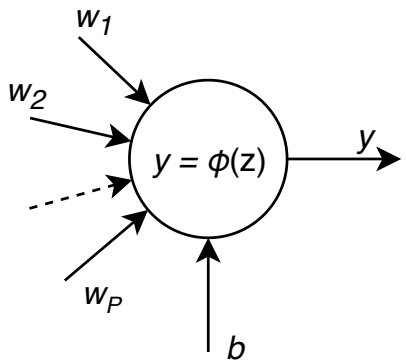


FIGURE 3.1 – The classical artificial neuron model, the *perceptron*.

The inputs of the *perceptron* are connected to the output of other *perceptrons*, and the value z is the weighted sum of such inputs, as shown in **Equation** (3.1),

$$z = \left(\sum_{i=1}^{P} w_i y_i \right) + b \tag{3.1}$$

where y_i are the outputs of other *perceptrons* connected to the corresponding weight w_i, and P is the number of inputs of the *perceptron*. The output y of the *perceptron* can then

be computed by **Equation** (3.2).

$$y = \phi(z). \tag{3.2}$$

Equation (3.1) can also be written in a more compact form, known as *vectorized* form, as shown in **Equation** (3.3),

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

where w is a column vector with entries w_i corresponding to the *perceptron* weights, a is a column vector with entries $a_i = y_i$ corresponding to the *perceptron* inputs.

By stacking many *perceptrons*, a layer is formed, as shown in **Figure. 3.2**. Usually, the *perceptrons* in a layer all have the same activation function ϕ. K is the number of *perceptrons* in the layer.

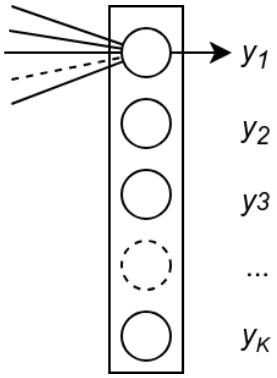


FIGURE 3.2 – A layer of *perceptrons*. Only the top *perceptron* connections are shown, and the bias is omitted.

The weight vector $w^{(i)}$ of each *perceptron* is, then, stacked in a matrix $W^{(l)}$ such that $W_i^{(l)} = \left(w^{(i)}\right)^T$, i. e. the i^{th} row of $W^{(l)}$ represents the transposed weight vector of the i^{th} *perceptron*. The values of the z_i (the weighted sums of each *perceptron*) can, then, be summarized as in **Equation** (3.4),

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{3.4}$$

where $z^{(l)}$ is a column vector with entries $z_i^{(l)}$ corresponding to the weighted sum of the i^{th} *perceptron* of the layer l, $W^{(l)}$ is the weight matrix of the layer, $a^{(l-1)}$ is the output the another *perceptron* layer (i. e. is the same as in **Equation** (3.3) and is shared by all *perceptrons* in the layer l), and $b^{(l)}$ is a column vector with entries $b_i^{(l)}$ corresponding to the bias of the i^{th} *perceptron* of the layer l. The output of $a^{(l)}$ the layer can, then, be written as in **Equation** (3.5),

$$a^{(l)} = \phi_{(l)}\left(z^{(l)}\right) \tag{3.5}$$

where the activation function $\phi_{(l)}$ of the layer l is element-wise applied to the vector $z^{(l)}$.

Then, by stacking many layers, the Feedforward Network is formed, as shown in **Figure. 3.3**. Usually all *perceptrons* in a layer are connected to all *perceptrons* on the previous layer, and there are no connections on *perceptrons* in the same layer. On the input layer there is no processing, the data is just redirected to the first *perceptron* layer. Layers from 1 to $Q-1$ are called "hidden layers", and the layer L_Q is the output layer, from which the output of the processing is retrieved. Q is the number of layers in the network (the input layer is not counted).

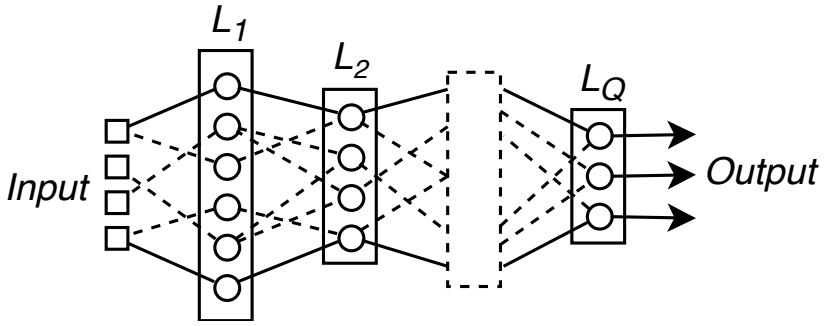


FIGURE 3.3 – A Feedforward Neural Network. Connections are suggested but not fully depict for clarity reasons. Each layer is fully connected to the immediately previous layer.

Each layer can have a different number of *perceptrons*, and the size of input and output layers are problem dependant. Also, layers in a Feedforward Network can have different activation functions $\phi_{(l)}$. Typical activation functions are: sigmoid, hyperbolic tangent and ReLU (Rectified Linear Unit). The parameter θ encompasses all weights of all *perceptrons* in the network as well as their biases, i. e. θ stands for all the entries of all the matrices $W^{(l)}$ and vectors $b^{(l)}$, with $1 \leq l \leq Q$.

3.1.2 Processing

Feedforward Networks have this name because of the way data flow when processing an input: The layer L_1 receives the input vector $a^{(0)}$ and the output $a^{(1)}$ is computed using **Equations** (3.4) and (3.5); This output, then, serves as input to the layer L_2 and so on, until the layer L_Q computes the output $a^{(Q)}$, which is then treated as the network output.

3.1.3 Training

For the training process, a cost function $J(\theta)$ must be chosen to measure how the predicted values $\hat{y} = a^{(Q)}$, given the inputs $x = a^{(0)}$, are far from the measured values y. The parameters in θ are optimized by minimizing $J(\theta)$. Modern neural networks are

trained using Maximum Likelihood Estimation (MLE), meaning the cost function takes the form in **Equation. 3.6** (GOODFELLOW *et al.*, 2017),

$$J(\theta) = -\mathbb{E}_{x,y \sim \hat{p}_{data}} log\left(p_{model}(y|x)\right) \tag{3.6}$$

where x and y are sampled from the available data $\hat{p}_{data}$, and $p_{model}(\cdot)$ is an approximation for distribution of the sampled data which must be chosen by the network designer. A typical approach, is to use the Gaussian distribution, as in **Equation** (3.7),

$$p_{\text{model}}(y|x) = \mathcal{N}\left(y; \hat{y}(x), \sigma^2 I\right) = C \cdot exp\left(-\frac{1}{2\sigma^2}\left|\left|y - \hat{y}(x)\right|\right|^2\right) \tag{3.7}$$

where C is a normalization factor and σ is the variance. Substituting **Equation** (3.7) into **Equation** (3.6), ignoring terms that doesn't depend on the parameters θ and picking a convenient value for σ, yields **Equation** (3.8),

$$J(\theta) = \left(\frac{1}{2}\right)\mathbb{E}_{x,y \sim \hat{p}_{\text{data}}}\left|\left|y - \hat{y}(x)\right|\right|^2 \tag{3.8}$$

which is the same as evaluating the Mean Squared Error (MSE) of the predicted values $\hat{y}(x)$, given target values y. The $1/2$ factor is introduced as a convenience for the computations of derivatives of $J(\theta)$. In practice, $J(\theta)$ is evaluated as the mean value given by **Equation** (3.9),

$$J(\theta) = \left(\frac{1}{2m}\right)\sum_{i=1}^{m}\left|\left|y_i - \hat{y}(x_i)\right|\right|^2 \tag{3.9}$$

where m is the number of samples (x_i, y_i) in the dataset. The *Gradient Descent* technique relies on the update of the parameters θ by applying **Equation** (3.10),

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta) \tag{3.10}$$

where α is the *learning rate*, a hyper-parameter of the training process. For the *perceptron* weights $W^{(l)}$, gradient of $J(\theta)$ can be expanded using the chain rule, as in **Equations** (3.11), (3.12) and (3.13),

$$\frac{\partial J(\theta)}{\partial W^{(l)}} = \frac{\partial J(\theta)}{\partial a^{(Q)}}\frac{\partial a^{(Q)}}{\partial a^{(Q-1)}}\frac{\partial a^{(Q-1)}}{\partial a^{(Q-2)}}\left(\ldots\right)\frac{\partial a^{(l+1)}}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial W^{(l)}} \tag{3.11}$$

$$\frac{\partial a^{(l+1)}}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial a^{(l-1)}} = \frac{\partial a^{(l+1)}}{\partial z^{(l+1)}}\frac{\partial z^{(l+1)}}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial a^{(l-1)}} \tag{3.12}$$

$$\frac{\partial z^{(l+1)}}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}} = \left(W^{(l+1)}\right)^T \odot \phi'_{(l)}\left(z^{(l)}\right) = \phi'_{(l)}\left(z^{(l)}\right) \odot \left(W^{(l+1)}\right)^T \tag{3.13}$$

where $\odot$ is the *Hadamard product*. Finally, by defining $\delta^{(l)}$ on the recursive **Equations** (3.14) and (3.15),

$$\delta^{(l)} = \phi'_{(l)}\left(z^{(l)}\right) \odot \left\{\left(W^{(l+1)}\right)^T \delta^{(l+1)}\right\} \tag{3.14}$$

$$\delta^{(Q)} = \left(a^{(Q)} - y\right) \odot \phi'_{(Q)}\left(z^{(Q)}\right) \tag{3.15}$$

the *Backpropagation* algorithm is established. The gradient of $J(\theta)$ with respect to $W^{(l)}$ can be computed as shown in **Equation** (3.16),

$$\frac{\partial J(\theta)}{\partial W^{(l)}} = \delta^{(l)}\left(a^{(l-1)}\right)^T \tag{3.16}$$

With a similar reasoning, it is possible to derive the biases' update equation, as shown in **Equation** (3.17),

$$\frac{\partial J(\theta)}{\partial b^{(l)}} = \delta^{(l)} \tag{3.17}$$

The training processes can then be summarized as follows:

- **1.** Sample m input-output pairs (x_i, y_i) from the dataset;
- **2.** For each sample, compute the predicted value $\hat{y}(x_i)$ using **Equations** (3.4) and (3.5). This is the forward propagation step;
- **3.** Compute the MSE by using **Equation** (3.9). If acceptable, stop training;
- **4.** For each predicted value, compute the error $\hat{y}(x_i) - y_i$ and propagate through the network with **Equations** (3.14), (3.15), (3.16) and (3.17). This is the backpropagation step;
- **5.** Repeat from step **1**.

3.2 Deep Learning

Neural networks with more layers have greater representational power, but for many years were too hard to train. By observing **Equation** (3.14), in which it is intuitive to see that the $\delta^{(l)}$ behave as the product of the weight matrices $W^{(i)}$ from layer $l+1$ onwards, it is possible to understand one of the reasons behind the difficulty of training deeper networks: the more layers a network has, the easier is for the $\delta^{(l)}$ to diverge to infinity or converge to zero. Also, a bigger number of layers implies more parameters in θ, which make the training process slower.

It was only recently that Artificial Intelligence research lead to new techniques that allowed faster and better training of deeper neural networks. Techniques such as ℓ_2-Regularization, *Early Stopping*, *Dropout* and *Adaptive Momentum Optimization* (ADAM), as well as the use of GPU, improved the speed and quality training of deep networks and allowed the ascension of Dense, Convolutional, Recurrent and Deep Belief Neural Networks (GOODFELLOW *et al.*, 2017).

3.3 Reinforcement Learning

3.3.1 Definitions

In contrast to Neural Networks in Supervised Learning, which have a model closer to the cognitive capabilities of the human brain by relying on artificial neurons, Reinforcement Learning (RL) follows a behavioral approach. The concepts of agent and environment are formalized by the definition of the Markov Decision Process (MDP), which constitutes of a state space $\mathcal{S}$ with elements s_t, an action space $\mathcal{A}$ with elements a_t, a reward function $\mathcal{R}$ (which maps triples (s_t, a_t, s_{t+1}) into scalar numbers r_{t+1}, the rewards), the dynamic of the environment (modeled as transition probabilities $p\left(s_{t+1}|s_t, a_t\right)$) and the *discount factor* $\gamma \in [0, 1]$, which conditions the agent to be more concerned with immediate rewards, when γ is close to 0, or long term rewards, when γ is close to 1 (SUTTON; BARTO, 2018).

For each observed state s_t, the actions of the agent are condition by the so called *policy function* $\pi\left(a_t|s_t\right)$, which is a probability distribution that maps states s_t into possible actions. The agent's interaction with the environment works as follows:

- **1.** At time-step t, the agent observes the current environment state s_t;
- **2.** Based on the state observation, an action $a_t \sim \pi\left(a_t|s_t\right)$ is sampled from the policy π;
- **3.** The agent's action a_t changes the environment;
- **4.** A reward r_{t+1} is received, and the time-step is updated as $t \leftarrow t + 1$;
- **5.** Repeat from step **1**.

Equation (3.18) summarizes the accumulated rewards of the agent while interacting with the environment,

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.18}$$

where the sum is weighted by the γ factor, R_{t+k+1} are the gathered rewards during the trajectory, and the interaction starts at time-step t. Since the agent's policy π is, in general, non-deterministic, the **Equation** (3.19) evaluates the quality of the policy (when starting from the state s_t) as the expected accumulated reward when following the policy,

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left\{G_t | s_t\right\}. \tag{3.19}$$

Equation (3.19) is also known as the *value function*, and measures how good for the agent is to be at state s_t, when following π. Another quality parameter of π is the so called *action-value function*, given by **Equation** (3.20),

$$Q^{\pi}(a_t, s_t) = \mathbb{E}_{\pi}\left\{G_t | a_t, s_t\right\}. \tag{3.20}$$

which measures how good is for the agent to be at state s_t and take action a_t, and is related to $V^{\pi}(s_t)$ by the **Equation** (3.21),

$$Q^{\pi}(a_t, s_t) = r(s_t, a_t, s_{t+1}) + \gamma V^{\pi}(s_{t+1}). \tag{3.21}$$

Early Reinforcement Learning algorithms explored such functions to devise methods for improving the starting policy π and reach an optimal policy π^* that maximizes $V^{\pi}(s_t)$ (or equivalently, $Q^{\pi}(a_t, s_t)$), but failed to scale when the state space $\mathcal{S}$ or the action space $\mathcal{A}$ were too big or continuous.

3.3.2 Policy Gradient

By parametrizing the policy $\pi_{\theta}(a_t|s_t)$, it is possible extend Reinforcement Learning to continuous state spaces $\mathcal{S}$ and/or continuous action spaces $\mathcal{A}$ (SUTTON; BARTO, 2018). Considering the cost function $J(\theta)$ given by **Equation** (3.22),

$$J(\theta) = \mathbb{E}_{a\sim\pi_{\theta}}\left\{G_t\right\} \tag{3.22}$$

it is possible to apply *Gradient Ascent* (since we are interested in maximizing the value of $J(\theta)$) on the parameters θ, in a Supervised Learned fashion, using **Equation** (3.23),

$$\theta_{t+1} \leftarrow \theta_t + \alpha^{\theta}\nabla_{\theta}J(\theta) \tag{3.23}$$

where α^{θ} is the learning rate for the parameters θ. The gradient of $J(\theta)$ is given by **Equation** (3.24),

$$\nabla_{\theta}J(\theta) = \int \nabla_{\theta}\pi_{\theta}(\tau)G_t(\tau)d\tau \tag{3.24}$$

where the integral evaluates over to all possible trajectories τ starting at time-step t. Using the transformation shown in **Equation** (3.25),

$$\nabla_\theta f(\theta) = \frac{f(\theta)\nabla_\theta f(\theta)}{f(\theta)} = f(\theta)\nabla_\theta \, log\left(f(\theta)\right) \tag{3.25}$$

where *log* is the natural logarithm, it is possible to change **Equation** (3.24) into a more tractable form, as in **Equation** (3.26),

$$\nabla_\theta J(\theta) = \int \pi_\theta(\tau)\nabla_\theta \log\left(\pi_\theta(\tau)\right) G_t(\tau) d\tau = \mathbb{E}_{a\sim\pi_\theta}\left\{\nabla_\theta log\left(\pi_\theta\right) G_t\right\}. \tag{3.26}$$

which allows for estimating the gradient by sampling. Unfortunately, *Policy Gradient* suffers from high variance and slow convergence during training.

3.3.3 Actor-Critic

It is possible to achieve a quality estimator of the policy π_θ that reduces *Policy Gradient* variance without altering the value of $\nabla_\theta J(\theta)$, by combining **Equations** (3.19) and (3.20) into the so called *advantage function*, shown in **Equations** (3.27) and (3.28),

$$A^\pi(a_t, s_t) = Q^\pi(a_t, s_t) - V^\pi(s_t) \tag{3.27}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi_\theta}\left\{\nabla_\theta log\left(\pi_\theta\right) A_t^\pi\right\} \tag{3.28}$$

which evaluates, overall, the gain of taking action a_t at state s_t. Moreover, it is possible to use **Equation** (3.21) to compute Q^π (and hence, A^π) from V^π, as in **Equation** (3.29),

$$A^\pi(s_{t+1}, s_t) = R_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \tag{3.29}$$

Finally, estimates $\hat{V}_W^\pi(s_t)$ of $V^\pi(s_t)$ are made by a neural network with parameters W, the *critic*, so that the policy neural network with parameters θ, the *actor*, can compute an estimate $\hat{A}^\pi$ of A^π, using **Equation** (3.30) (SUTTON; BARTO, 2018),

$$\hat{A}^\pi\left(s_{t+1}, s_t\right) = R_{t+1} + \gamma\hat{V}_W^\pi(s_{t+1}) - \hat{V}_W^\pi(s_t) \tag{3.30}$$

and update the parameters θ with **Equation** (3.31),

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi_\theta}\left\{\nabla_\theta log\left(\pi_\theta\right)\hat{A}_t^\pi\right\}. \tag{3.31}$$

Hence, the actual cost function that is being maximized is given by **Equation** (3.32),

$$J(\theta) = \mathbb{E}_{a\sim\pi_\theta}\left\{\hat{A}_t^\pi\right\}. \tag{3.32}$$

The *critic* network can be trained, in parallel with the *actor* network, by Supervised Learning with cost $J(W)$ and update rule as shown in **Equations** (3.33), (3.34) and (3.35),

$$J(W) = \frac{1}{2}\left(R_{t+1} + \gamma\hat{V}_W^\pi(s_{t+1}) - \hat{V}_W^\pi(s_t)\right)^2 \tag{3.33}$$

$$\nabla_W J(W) = R_{t+1} + \gamma\hat{V}_W^\pi(s_{t+1}) - \hat{V}_W^\pi(s_t) \tag{3.34}$$

$$W_{t+1} \leftarrow W_t - \alpha^W \nabla_W J(W) \tag{3.35}$$

where α^W is the learning rate for the parameters W. An online training with the *Actor-Critic* can be summarized as follows:

- **1.** Actor and Critic observe the environment, getting the state s_t;
- **2.** An action a_t is sampled from the Actor policy $\pi_\theta(a_t|s_t)$. The Critic takes note of a_t;
- **3.** Actor and Critic observe the environment, getting the state s_{t+1} and the reward r_{t+1};
- **4.** Actor updates parameters θ by consulting the Critic for the estimate $\hat{A}_t^\pi$ and using **Equations** (3.31) and (3.23);
- **6.** An action a_{t+1} is sampled from the Actor policy $\pi_\theta(a_{t+1}|s_{t+1})$. The Critic takes note of a_{t+1};
- **7.** Actor and Critic observe the environment, getting the state s_{t+2} and the reward r_{t+2};
- **8.** Critic updates parameters W by using **Equations** (3.34) and (3.35), considering observations of s_{t+1} and s_{t+2};
- **9.** Time-step is updated as $t \leftarrow t + 2$;
- **10.** Repeat from step **2**.

Even with reduced variance when compared with *Policy Gradient*, *Actor-Critic* is hard to train: when updating θ, due to the inaccuracy in the estimates of $\hat{A}^\pi$, the new policy $\pi_{\theta'}$ can end up in a region of the parameters space where it performs worse and it is not able to return to the previous region of parameters space. The currently adopted solution is to update θ in a more conservative way.

3.3.4 Proximal Policy Optimization

One technique used for conservative updates of the parameters θ that is gaining notoriety for its simplicity and efficiency is the so called *Proximal Policy Optimization* (PPO) (SCHULMAN *et al.*, 2017). By defining the ratio $r_t(\theta')$ as in **Equation** (3.36),

$$r_t(\theta') = \frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \tag{3.36}$$

where $\pi_\theta(a_t|s_t)$ is a fixed policy and $\pi_{\theta'}(a_t|s_t)$ is the resulting policy after one update of the parameters θ, the following cost function is used,

$$J(\theta) = \mathbb{E}_{a\sim\pi_{\theta'}}\left\{min\left[r_t\left(\theta'\right)\hat{A}_t^\pi, clip\left(r_t(\theta'), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t^\pi\right]\right\} \tag{3.37}$$

where the left value of the $min(\cdot,\cdot)$ function reduces to **Equation** (3.32) when $\theta' = \theta$. The $clip(\cdot,\cdot,\cdot)$ function saturates the value of $r_t(\theta')$ inside the range $[1-\varepsilon, 1+\varepsilon]$, preventing the policy to follow too positive or too negative values of $\hat{A}_t^\pi$. The value ε is a hyperparameter, usually taken as 0.2.

PPO was used to reach *state-of-the-art* AI performance in difficult tasks, such as winning a match of Dota 2 (depicted in **Figure. 3.4**), a multi-agent strategy game with continuous state and action spaces, against professional human players (BROCKMAN *et al.*, 2018).

FIGURE 3.4 – 2019's Dota 2 match between world champions OG team and AI team OpenAI Five developed by OpenAI (BROCKMAN *et al.*, 2018).

4 Methodology

4.1 Simulator

As explained in **Chapter 2**, ITAndroids' VSS Soccer simulated environment is capable of simulating the physics in a VSS match, as well as to run two different strategies against each other in a simulated game. For our purposes, we'll hand-code the behavior of adversaries, like quantity of opponents on the field, their positions, velocities and trajectories, in such a way that our Reinforcement Learning agent can experience different episodes/scenarios in its training process.

Also, for the sake of simplicity in the management of the agent, opponent and environment, the three processes needed to run the simulation (agent process, adversary process and simulator process) will be united in a single process.

4.2 Markov Decision Process

As seen in **Chapter 3**, the building blocks of Reinforcement Learning are grounded on the concepts that constitutes a MDP. For our scenario:

- The state space $\mathcal{S}$ are the positions and velocities of the players, opponents and the ball. Hence, a continuous state space;
- The action space $\mathcal{A}$, in principle, is taken as the desired positions in the field and final orientations at those positions. Again, a continuous space.
- The reward function $\mathcal{R}$ is yet to be decided, but will be the main focus of attention during the training process, along with the specific episodes/scenarios of training;
- The discount factor γ will be taken, as it is commonly used, to be a value between 0.90 and 0.99;
- The dynamic of the environment is encompassed in the simulator and doesn't need to be explicitly known by the agent.

4.3 OpenAI and Intel AI DevCloud

OpenAI is a non-profit AI research organization engaged into promote and develop AI in such a way as to benefit humanity (OPENAI, 2019a). OpenAI created a GitHub repository with high-quality implementations of reinforcement learning algorithms, including PPO, called OpenAI Baselines (OPENAI, 2019b).

OpenAI Baselines runs as a process in the Operational System (OS), and can be communicated with via gRPC, an open source remote procedure call developed at Google (GOOGLE, 2019). It communicates with the process were the RL agent resides, and the message's nature are the MDP information, such as: current state s_t, current action a_t, next state s_{t+1}, gained reward r_{t+1}, and so on.

For speeding up the training process by using distributed computation, we intend to use Intel AI DevCloud, a free cluster of Intel Processors to assist Machine Learning and Deep Learning training and inference.

4.4 Training

The training process, established by the interaction between the simulation process (which is a union of the simulator, the agent and the opponents) and OpenAI baselines via gRPC, works as follows:

- **1.** The simulation process sends the current state s_t to the baselines process;
- **2.** The baselines process responds to the simulation process with the sampled action a_t from the actor policy;
- **3.** The simulation process updates the environment after the action a_t was taken, then responds with the next state s_{t+1} and reward r_{t+1};
- **4.** The time step is updated as $t \leftarrow t + 1$;
- **5.** Repeat from step **2**.

4.5 Success criteria

After training the agent, it will be integrated on the current ITAndroids VSS Soccer strategy as the support player and confronted against the strategy with the hand-coded support player, via matches played on the game simulator, and the ball possession will be

measured. The objective of the training is to improve the capability of the support player in maintaining the ball possession of the team.

4.6 Activities plan

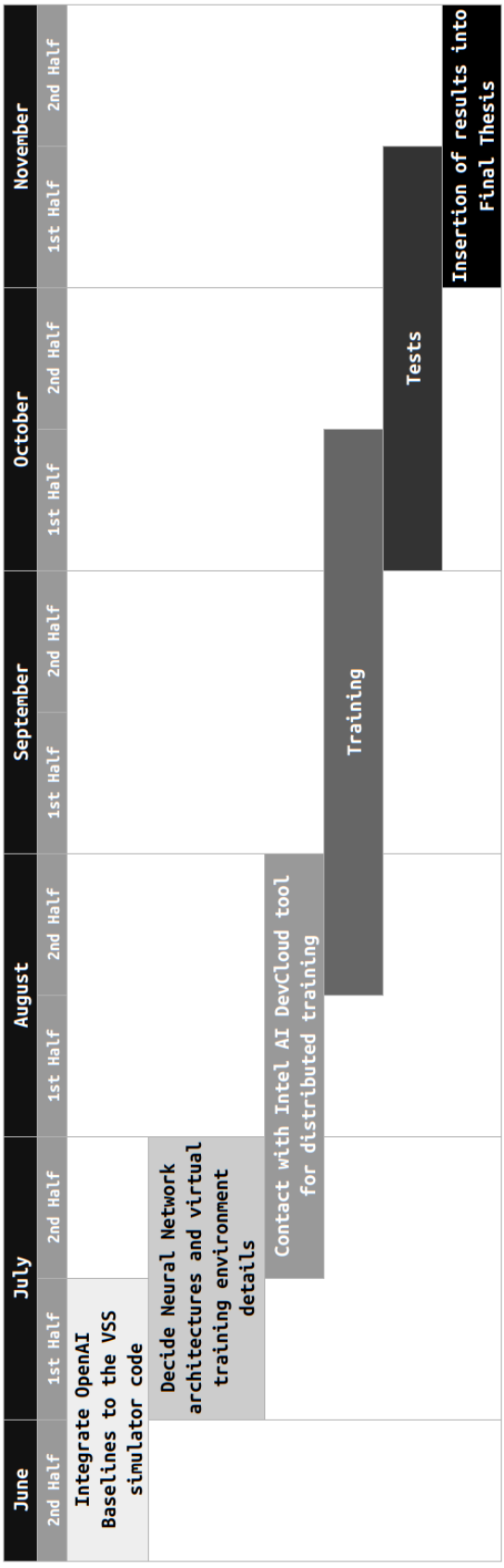


FIGURE 4.1 – Activities plan for the 2nd semester of 2019.

5 Preliminary Conclusions and Future Work

In this work we outlined the possibility of using the OpenAI Baselines project to train RL agents in a robot soccer environment, more specifically the VSS Soccer competition, in the particular situation of the support player behavior, to maintain the ball possession. The chosen RL techniques were Actor-Critic and PPO, as they have successfully been used in scenarios with continuous state and actions spaces and attained superhuman performance in difficult tasks.

As the next steps of this work, we plan to:

- Finish the integration of the Baselines program with the ITAndroids' VSS Soccer simulator code;
- Create training scenarios to improve the support player behavior within the simulator environment;
- Use Intel AI DevCloud tool to speed up training time by using distributed computation;
- Test the trained agent within the current ITAndroids VSS Soccer strategy against the hand-coded support strategy.

Bibliography

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/>.

GOLDBERG; GOLUSHKOV; ROJTBERG; ROGERS; SARI; ALLGAIER; RAMAN *et al.* Website, **Object-Oriented Graphics Rendering Engine**. 2019. Disponível em: <https://www.ogre3d.org/>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. `http://www.deeplearningbook.org`: MIT Press, 2017.

GOOGLE. Website, **About gRPC**. 2019. Disponível em: <https://grpc.io/>.

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>.

IEEE. Website, **About IEEE**. 2019. Disponível em: <https://www.ieee.org/about/>.

IEEE. Website, **About IEEE's VSS Soccer**. 2019. Disponível em: <http://www.cbrobotica.org/?page_id=81>.

ITANDROIDS. Video, **IEEE VSSS - ITAndroids - LARC 2018**. 2018. Disponível em: <https://www.youtube.com/watch?v=Cw3VOU1YcLU>.

LAKE, B. Online Article, **Finding Solace in Defeat by Artificial Intelligence**. 2017. Disponível em: <https://www.technologyreview.com/s/604273/finding-solace-in-defeat-by-artificial-intelligence/>.

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>.

MAXIMO; JUNIOR; ARRUDA; OLIVEIRA de; RIBEIRO; BALDAO; MEDEIROS de; CYSNE; QUEIROZ de; GUIMARAES. Article, **ITAndroids Very Small Size Soccer Team Description Paper for LARC 2019**. 2019. Disponível em: <https://drive.google.com/open?id=1lx1W9Z-EvUy2psRgzPdY1uISuE01lvwv>.

OPENAI. Website, **About OpenAI**. 2019. Disponível em: <https://openai.com/about/>.

OPENAI. Code Repository, **OpenAI Baselines**. 2019. Disponível em: <https://github.com/openai/baselines>.

POLANI, D. Online Journal, **Why football, not chess, is the true final frontier for robotic artificial intelligence**. 2016. Disponível em: <http://theconversation.com/why-football-not-chess-is-the-true-final-frontier-for-robotic-artificial-intelligence-62296>.

RAS, I. Website, **About IEEE RAS**. 2019. Disponível em: <https://www.ieee-ras.org/about-ras>.

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.; 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>.

SMITH, R. Website, **Open Dynamics Engine**. 2019. Disponível em: <https://www.ode.org/>.

SUSANTO, S.; JAMZURI, E.; ANALIA, R.; PAMUNGKAS, D.; SOEBHAKTI, H. The deep learning development for real-time ball and goal detection of barelang-fc. In: . [S.l.: s.n.], 2017.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. MIT Press, 2018. Disponível em: <http://incompleteideas.net/book/the-book-2nd.html>.

WINCHESTER, C. Online Article, **Industrial Robots set to Generate USD$45 Billion by 2025**. 2017. Disponível em: <https://www.engineering.com/AdvancedManufacturing/ArticleID/14090/Industrial-Robots-set-to-Generate-USD45-Billion-by-2025>.