UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Mateus Gonçalves Machado

DyLam: A Dynamic Reward Weighting Method for Reinforcement Learning Policy Gradient Algorithms

Recife

2022

Mateus Gonçalves Machado

DyLam: A Dynamic Reward Weighting Method for Reinforcement Learning Policy Gradient Algorithms

A M.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Concentration Area: Computer Intelligence

Advisor: Hansenclever de França Bassani

Recife

2022

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

M149d

Machado, Mateus Gonçalves
DyLam: a dynamic reward weighting method for reinforcement learning policy gradient algorithms / Mateus Gonçalves Machado. – 2022.
59 f.: il., fig., tab.

Orientador: Hansenclever de França Bassani.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computção, Recife, 2022.

Inclui referências.

1. Engenharia da computação. 2. Aprendizagem. I. Bassani, Hansenclever de França (orientador). II. Título.

621.39 CDD (23. ed.) UFPE - CCEN 2022-126

Mateus Gonçalves Machado

“DyLam: A Dynamic Reward Weighting Method for Reinforcement Learning Policy Gradient Algorithms”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Inteligência Computacional

Aprovado em: 07/06/2022.

To my wife, family, friends, and advisor. I would not be so happy researching and this work probably would not exist without you. Love you guys.

ACKNOWLEDGEMENTS

I think being a researcher requires a lot of courage and a little madness, especially when you are doing it in Brazil. You are courageous and resilient when your experiments fail during a year, but you are still trying to make it work. It is brave to face the harsh reality of frustration after frustration for so long, having no idea if your method will work or not. But when it does work, the first feeling you take is not happiness or relief. It is anger. "How could I be so dumb that I did not try that?". Then, you feel the whole wave of good feelings.

The good part about this trajectory is that you share it with people. Family and friends are the anchors of every researcher. Your coworkers are the most brilliant people alive, be always by their side, and you will have incredible ideas. Thank you, Felipe (so much). Your advisor is your Master Yoda and beyond. I could not tell in words how Hansenclever is a fantastic man and how he calmed me so many times. "In your advisor, you must trust."

Never, I repeat, never think of leaving your friends. You will need them more than ever. As I said, you are a bit mad. Your troop is your antipsychotic pill. You need them to live, laugh, and enjoy the time. Thank you, Ana, Anna, Cristiano, Jailson, Thainá, and RobôCIn; you are the best.

Love your family like you never did it before. They are support, happiness, and love. A father/mother hug never felt so good. A brother/sister call never felt so laughable. Thank you for always being there for me, Alexandre, Selma, Gabriel, Juliana, Marinalva, Lívia, and Maria. At last, but most important, your partner, your love, your everything. Love them. If they support your research and choices, just love them and show that you do. Samara, I cannot believe you are by my side and loving me so much every day. Thank you for every day's laugh, hug, and love. I love you and always will.

Unfortunately, without money, we cannot live in this capitalist system. You will need a scholarship program or something like that. I would like to acknowledge the Fundação de Amparo a Ciência e Tecnologia de Pernambuco (FACEPE) and Fundação de Apoio ao Desenvolvimento da Universidade Federal de Pernambuco (FADE). I also thank Alexandre Costa, Tsang Ing Ren, Edna Barros, and Victor Sabino for trusting in my work and giving me the opportunities during my research.

ABSTRACT

Reinforcement Learning (RL) is an emergent subfield of Machine Learning in which an agent interacts with an environment and leverages their experiences to learn, by trial and error, which actions are the most appropriate for each state. At each step the agent receives a positive or negative reward signal, which is the main feedback used for learning. RL finds applications in many areas, such as robotics, stock exchange, and even in cooling systems, presenting superhuman performance in learning to play board games (Chess and Go) and video games (Atari Games, Dota2, and StarCraft2). However, RL methods still struggle in environments with sparse rewards. For example, an agent may receive very few goal score rewards in a soccer game. Thus, it is hard to associate rewards (goals) with actions. Researchers frequently introduce multiple intermediary rewards to help learning and circumvent this problem. However, adequately combining multiple rewards to compose the unique reward signal used by the RL methods frequently is not an easy task. This work aims to solve this specific problem by introducing DyLam. It extends existing policy gradient methods by decomposing the reward function used in the environment and dynamically weighting each component as a function of the agent's performance on the associated task. We prove the convergence of the proposed method and show empirically that it overcomes competitor methods in the environments evaluated in terms of learning speed and, in some cases, the final performance.

Keywords: reinforcement learning; reward shaping; sparse rewards; deep learning.

RESUMO

Aprendizagem por Reforço (AR) é um subcampo emergente de Aprendizagem de Máquina no qual um agente interage com um ambiente e aproveita suas experiências para aprender, por tentativa e erro, quais ações são as mais adequadas para cada estado. A cada passo o agente recebe um sinal de recompensa positivo ou negativo, que é o principal *feedback* utilizado para o aprendizado. A AR encontra aplicações em diversas áreas, como robótica, bolsa de valores e até mesmo em sistemas de refrigeração, apresentando desempenho sobre-humano no aprendizado de jogos de tabuleiro (Xadrez e Go) e videogames (jogos de Atari, Dota2 e StarCraft2). No entanto, os métodos AR ainda lutam em ambientes com recompensas escassas. Por exemplo, um agente pode receber poucas recompensas por gols em um jogo de futebol. Assim, é difícil associar recompensas (gols) com ações. Os pesquisadores frequentemente introduzem várias recompensas intermediárias para ajudar no aprendizado e contornar esse problema. No entanto, combinar adequadamente várias recompensas para compor o sinal de recompensa único usado pelos métodos AR frequentemente não é uma tarefa fácil. Este trabalho visa resolver este problema específico através da introdução do DyLam. Ele estende os métodos de gradiente de política existentes, decompondo a função de recompensa usada no ambiente e ponderando dinamicamente cada componente em função do desempenho do agente na tarefa associada. Provamos a convergência do método proposto e mostramos empiricamente que ele supera métodos concorrentes nos ambientes avaliados em termos de velocidade de aprendizado e, em alguns casos, desempenho final.

Palavras-chaves: aprendizagem por reforço; *reward shaping*; recompensas esparsas; aprendizagem profunda.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ANN	Artificial Neural Network
DDPG	Deep Deterministic Policy Gradient
DPG	Deterministic Policy Gradient
DQN	Deep Q-Networks
DyLam	Dynamic λ weighting function
HER	Hindsight Experience Replay
IM	Intrinsically Motivated Reinforcement Learning
KL	Kullback–Leibler
MDP	Discrete Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean-Squared Error
PG	Policy Gradient
PPO	Proximal Policy Optimization
RB	Replay Buffer
RL	Reinforcement Learning
TRPO	Trust Region Policy Optimization
VSS	IEEE Very Small Size League

CONTENTS

1 INTRODUCTION

Reinforcement Learning (RL) is a subfield of Machine Learning (ML) in which an agent faces the problem of some environment and it learns to solve it by trial-and-error (KAELBLING; LITTMAN; MOORE, 1996). It combines psychology, neuroscience, and computer science, where an agent learns from an environment which action to take in a particular situation. We call this process "train". During training, the agent receives signals that advise what to do, called "rewards" (SUTTON; BARTO, 2018). We can see the training process as training to kick a penalty in soccer. If the agent scores a goal, the crowd will applaud. Otherwise, they will heckle. In the last years, the research on RL has grown due to Deep Learning research advances (HENDERSON et al., 2018). These advances have opened the opportunity to apply RL to video games such as Atari 2600 with Deep Q-Networks (DQN) (MNIH et al., 2013). Researchers recently developed cooperative agents to play DOTA2 against players and even compete in international events (OPENAI, 2018).

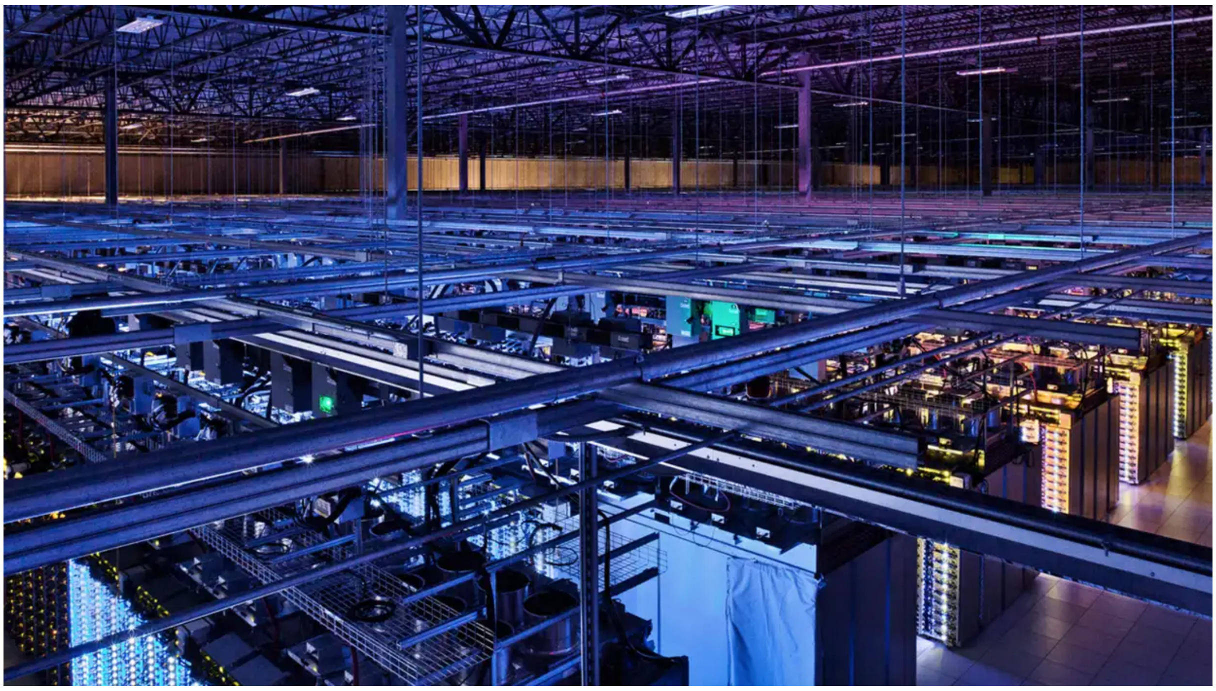
Figure 1 – The OpenAI Five team in action in a sample game.

Source: (OPENAI, 2018)

These results influence the development of other areas, such as cooling a data centre. Deepmind improved its energy efficiency in around 30 percent of its cooling system with an RL agent. Considering that specialized people have already optimized the system, this agent shows the potential of an RL intelligence (GAMBLE; GAO, 2018).

Although presenting exciting results, RL still struggles with certain problems. The most

Figure 2 – First data centre with the RL cooling system.



Source: (GAMBLE; GAO, 2018)

significant difficulty in applying RL in general problems is the need for a great number of observations to create efficient agents (SUTTON; BARTO, 2018). For example, in the Lunar Lander environment, a DQN agent needs over a million observations to efficiently complete the objective (MNIH et al., 2013). In general ML problems, this is an unreal number of samples in a single dataset. Another known obstacle in RL is the sparsity of reward signals (SUTTON; BARTO, 2018). A human can evaluate and point out what he did wrong to create a situation. For example, when we kick the ball in a soccer match, we only know if we score when the ball enters the goal. This sparse-reward situation takes N steps to occur, and meanwhile, the agent knows nothing about the problem.

The current literature usually addresses the problem of sparse rewards with a technique known as reward shaping (NG; HARADA; RUSSELL, 1999). A poorly shaped reward can create undesired behaviors like reward hacking or Pareto conflicting rewards. Reward hacking is a behavior in which the agent exploits a local minimum and does not complete the main objective. E.g., an environment in which the objective is to reach a particular position, but the agent has to open a door first. Ng, Harada and Russell (1999) show that only rewarding when opening the door create a hacking agent that keeps opening the door forever. A Pareto conflicting reward shaping is defined by two or more rewards that impact each other. E.g., a reward to go up and a punishment to go up (BRYS et al., 2014). Besides these problems, weighting each reward component in the environment is an exhausting task once the developer has to train an agent every time there is a problem with the reward.

Reward shaping can be related to sub-objectives or sub-tasks of the environment. Sub-objectives are known objectives of the environment that the agent must complete before or along with the main objective. Subtasks are tasks that help the agent complete the objective of the environment. An environment can have both subobjectives and subtasks (SUTTON; BARTO, 2018). E.g., in a grid world environment where the agent has to open a door, there is the subtask of reaching the objective and the subtask to reach the door, while opening the door is the sub-objective. Curriculum learning is a research field in machine learning that aims to facilitate the learning process by learning to solve a sequence of problems easier but correlated with the main objective, thus climbing a mountain of challenges of increasing difficulty (WANG; CHEN; ZHU, 2020). Each challenge can be seen as sub-objective or sub-task. In RL, reward shaping is applied with a similar objective.

In this work, we propose to improve the reward shaping technique to solve the sparse reward problem, introducing **Dy**namic **Lam**bda weighting function or "**DyLam**". We use intuitive reward signals that compose a general reward when designing an environment to create an automated curriculum learning method for RL. Our method dynamically prioritizes reward components, automatically identifying and learning to accumulate the easiest rewards first, then moving to the more difficult ones until learning to achieve the final goal.

In this thesis, we aim to:

- Remove the weights of the reward shaping from the environment.
- Automate the weighting of reward components.
- Provide a mathematical proof of the convergence of our methods under certain conditions.

We organized this work as follows: Chapter 2 provides the theoretical background as the mathematical and algorithmic basis for our proposed methods; Chapter 3 discusses the ideas of prior and recent research on which we have based our work; Chapter 4 presents the mathematical proof of convergence of the methods proposed; Chapter 5 reports the results achieved using our methods and discusses these results when compared to baseline methods; Finally, in Chapter 6 we take the final considerations of our work, analyzing the methods in a general way and seeking future works and applications.

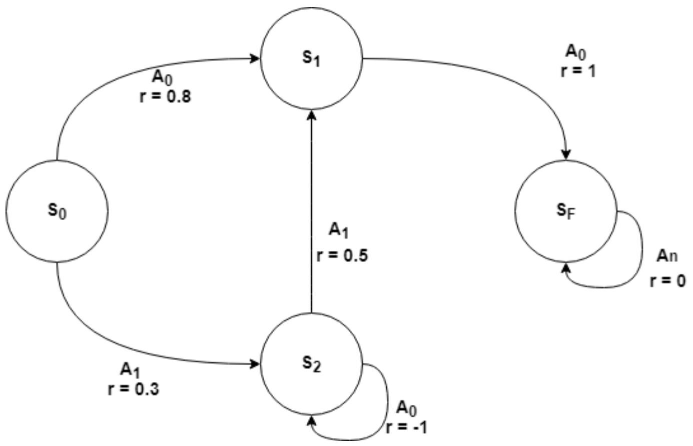
2 THEORETICAL BACKGROUND

This chapter presents some concepts of RL necessary to understand this thesis. We divided into four sections as follows: **Discrete Markov Decision Process**, the basics of RL, defining the Agent-Environment interaction; **Value Functions**, the basic math which defines the functions that RL uses; **Q-learning methods**, basic methods of RL and Deep RL; **Policy Gradient and Actor-Critic**, the family of methods on which we based our work; **Deterministic Policy Gradient Methods** and **Proximal Policy Policy Optimization**, methods we adapted to our reward-shaping strategy.

2.1 DISCRETE MARKOV DECISION PROCESS

In RL, MDPs represent the Agent-Environment interactions, being the **state space** S the perceptions of the agent over the environment and the **action space** A actions that the agent can perform in the environment. An MDP M can be described as a 4-tuple (S, A, P, R) where P is the **transition probability** model of going from **state** s to **next state** s' choosing **action** a and R is the **reward distribution** of taking action a on state s leading to the next state s'. In Figure 3 we illustrate an example of MDP.

Figure 3 – Example of MDP. There is four states which the agent can reach and two actions which it can take. In S_F, any action leads to S_F. We denominate S_F as terminal state.



Source: Author

The agent which "walks" through the MDP has a **policy** π that, in the ideal scenario, maps each state to an optimal action that will lead to the **maximum cumulative expected future reward** (SUTTON; BARTO, 2018).

2.2 VALUE FUNCTIONS

The value function of a state s under a policy π, denoted $V_\pi(s)$, is the expected future reward (SUTTON; BARTO, 2018). The value functions look toward maximizing a function G_t, which is the **reward accumulated** from the steps. This function sums the rewards of the episode weighted by a factor γ, which is called **discount factor**, that tells how significant the future rewards are (SUTTON; BARTO, 2018). A value function $V_\pi(s)$, **state-value** function for policy π, on a MDP M is defined as:

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \tag{2.1}$$

where $\mathbb{E}_\pi[\cdot]$ is the expected value of the random variable G_t given that the agent follows policy π.

As we have a function that estimates the value of being in each state s under a policy π, we can similarly have a value function to describe how valuable it is taking an action a on a state s under a policy π, denoted by $Q_\pi(s, a)$. The **action-value** function for policy π, $Q_\pi(s, a)$, is defined as:

$$Q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]. \tag{2.2}$$

Both value functions can be approximated over several iterations following the policy on the environment. These functions have a fundamental property for reinforcement learning and dynamic programming that satisfy particular recursive relationships. Applying the recursive relationships on state-value function, for example, we end up with the **Bellman Equation** (SUTTON; BARTO, 2018), which expresses the relationship between the value of a state and the value of the next states, shown in Equation 2.3.

$$\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}[G_t|S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}\middle|S_t = s\right] \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r + \gamma\,\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}\middle|S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \forall s \in S.
\end{aligned} \tag{2.3}$$

For finite MDPs we can state that a policy is better than another, $\pi \geq \pi'$, if and only if $V_\pi(s) \geq V_{\pi'}(s), \forall s \in S$. There is always at least one policy better than or equal to all other policies, denominated as **optimal policies** (SUTTON; BARTO, 2018). This set of optimal policies is denoted by π^* and the optimal state-value and action-value functions shared by π^* are denoted $V^*(s)$ and $Q^*(s)$, where:

$$V^*(s) \doteq \max_\pi V_\pi(s), \tag{2.4}$$

$$Q^*(s,a) \doteq \max_\pi Q_\pi(s,a), \tag{2.5}$$

for all $s \in S$ and $a \in A(s)$. We can rewrite Q^* in terms of v^* since Q^* returns the expected future reward for taking action a in state s over an optimal policy:

$$Q^*(s,a) = \mathbb{E}[R_t + \gamma V^*(S_{t+1})|S_t = s, A_t = a], \tag{2.6}$$

2.3 Q-LEARNING METHODS

Q-learning is the technique that modern Reinforcement Learning uses as the base (SUTTON; BARTO, 2018). This method represents the learning policy as a table that maps states to actions and updates those tables with the optimal Bellman function (Equation 2.6). They denominate the table as Q-table and the state-action values as Q-values. We use the same denomination in this work. Q-learning also uses an exploration strategy called ϵ-greedy. This strategy introduces exploration in early training but removes it when the agent already knows the Q-values of the state-action pairs (SUTTON; BARTO, 2018).

As the tables used in Q-learning can only represent discrete state and action spaces, Mnih et al. (2013) present the DQN. The DQN was inspired by the TD-gammon (TESAURO, 1994), which was able to learn how to play the Backgammon game using an Multilayer Perceptron (MLP) with one hidden layer to approximate the value function of TD-lambda (SUTTON; BARTO, 2018). DQN comes with a breakthrough framework that revolutionized Deep Reinforcement Learning. They present three base ideas: State representation of images, Experience Replay, and Target Networks (MNIH et al., 2013). Mnih et al. (2013) propose using Convolutional Neural Networks (GOODFELLOW et al., 2013) to extract the state features when the agent's perceptions are images.

Experience Replay is a method to represent the agent memory. It has a queue of transitions denominated as Replay Buffer (RB) or Replay Memory representing an MDP M (s, a, r, s'). The RB samples these transitions into mini-batches and then the policy is updated with Q-learning.

The Target Network aims to stabilize the learning of the algorithm. It is a delayed copy of the policy network used on the bootstrap step of the Bellman function to replace the target Q-value (MNIH et al., 2015). Before this change, the learning policy could only increase its Q-values and create a superestimation behavior. The target network reduces this superestimation of Q values. The network syncing rate is a hyperparameter defined by the agent designer and can change depending on the problem.

2.4 POLICY GRADIENT AND ACTOR-CRITIC

PG methods use a parameterized policy to select actions without consulting an action-value function. We denote the policy parameters as $\theta \in \mathbb{R}^n$. Those parameters are adjusted during the training to maximize the accumulated future reward. The value function is used to learn θ^*, but, generally, it is not consulted to select the actions. Thus the probability of choosing an action a in state s at time t with the parameters vector θ is written as $\pi(a|s, \theta) \doteq Pr\{A_t = a | S_t = s, \theta_t = \theta\}$ (SUTTON; BARTO, 2018).

PG methods learn the policy parameters based on the gradient of some performance measure $J(\theta)$ with respect to the policy parameters. The objective is to maximize $J(\theta)$, so their updates approximate gradient ascent in J:

$$\theta_{t+1} \doteq \theta + \lambda \widehat{\nabla J(\theta_t)}, \tag{2.7}$$

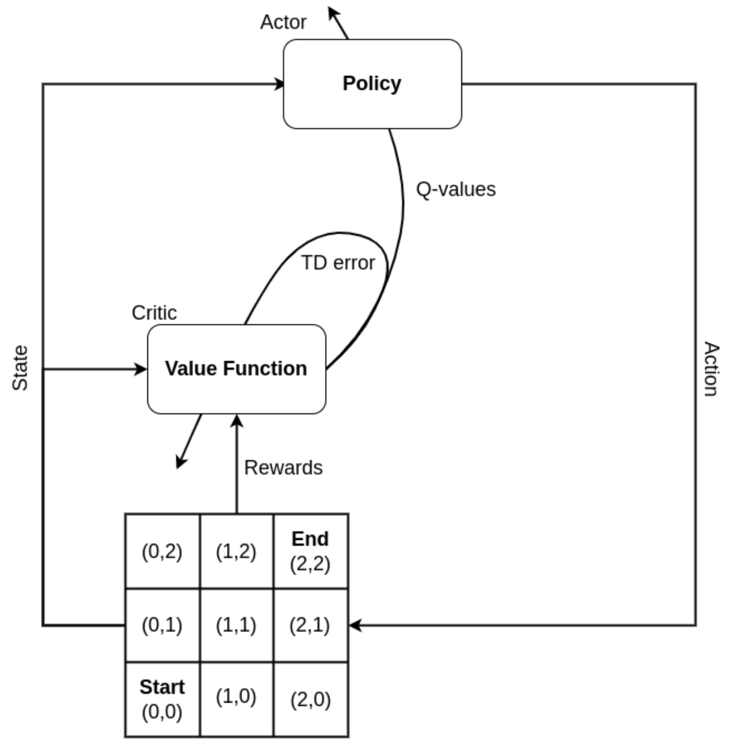
where $\widehat{\nabla J(\theta_t)}$ is a stochastic estimate the gradient of the performance measure with respect to θ_t. All methods that follow this general schema are PG. The adjustment of θ is made through iterations of the gradient in Therorem 2.4.1.

Theorem 2.4.1 (Policy Gradient theorem)**.**

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q_\pi(s,a)\ \delta a\ \delta s \\ &= \mathbb{E}_{S\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta log \pi_\theta(a|s) Q_\pi(s,a)] \end{aligned} \tag{2.8}$$

In this work, we use an **Actor-Critic** method, a PG method that approximates the policy and value functions. The term **actor** references the policy learned by 2.8, and the term **critic** refers to the value functions. The Critic is also a parameterized function with parameters w, and it tries to approximate $Q_w(s,a) \approx Q_\pi(s,a)$ (SUTTON; BARTO, 2018). See in Figure 4 the representation of a general Actor-Critic training.

Figure 4 – Example of Actor-Critic training loop. The Q-values of the critic module are used to train the actor module.



Source: Author

The main difference between Actor-Critic and other PG methods is that the value function is used for bootstrapping. That is, substituting the true action-value $Q_\pi(s,a)$ for a function approximator $Q_w(s,a)$.This change, generally, introduces bias and an asymptotic dependence on the quality of the function approximation. However, if the function approximator follows that $Q_w(s,a) = \nabla_\theta log\pi_\theta(a|s)^T w$ and the parameters w are minimized by the Mean-Squared Error (MSE) $\epsilon^2(w) = \mathbb{E}_{s\sim p,a\sim\pi_\theta}[(Q_w(s,a) - Q_\pi(s,a))^2]$, there is no bias (SUTTON; BARTO, 2018).

2.5 DETERMINISTIC POLICY GRADIENT METHODS

The Deterministic Policy Gradient (DPG) was introduced as an **off-policy** Actor-Critic algorithm with the promise to learn a deterministic target policy from an exploratory behavior policy (SILVER et al., 2014). That is, they adapted the method introduced in (DEGRIS; PILARSKI; SUTTON, 2012), where a behavior policy $\beta(a|s)$ different than the learning policy $\pi_\theta(a|s)$ is sampled from trajectories to update the parameters θ, to learn a deterministic target policy.

The DPG algorithm generally maintains the parameterized actor function, denoted by $\mu(s|\theta_\mu)$, which deterministically maps states to a specific action. The critic $Q_w(s,a)$ updates w using the Bellman Equation (2.3) and the actor is updated to optimize the expected return from the objective function J concerning the actor parameters (SILVER et al., 2014):

$$\begin{aligned}\nabla_{\theta\mu} J &\approx \mathbb{E}_{s_t\sim p^\beta}[\nabla_{\theta\mu} Q(s,a|\theta_Q)|_{s=s_t,a=\mu(s_t|\theta_\mu)}] \\ &= \mathbb{E}_{s_t\sim p^\beta}[\nabla_a Q(s,a|\theta_Q)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta_\mu}\mu(s|\theta_\mu)|_{s=s_t}],\end{aligned} \tag{2.9}$$

where p^β is the probability of transitioning using the behavior policy β.

Although DPG works in many state spaces, the larger the state space, the harder it is to approximate a linear function to Q. For that, Lillicrap et al. (2019), inspired by the DQN, using an Artificial Neural Network (ANN) as a function approximator and the Experience Replay, proposed the Deep Deterministic Policy Gradient method.

Besides those additions, Deep Deterministic Policy Gradient (DDPG) introduced a solution similar to Target Networks (MNIH et al., 2015) to reduce the learning instability and uses the Ornstein-Uhlenbeck process to generate temporally correlated exploratory actions(UHLENBECK; ORNSTEIN, 1930). The Target Networks are updated softly, instead of copying the whole weights vector of the original network. Both the actor and the critic networks have target networks, and the weights are updated with $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, i.e., the target weights θ' receive a linear combination from the actual and target weights. The idea is to give a higher weight τ that to θ but still maintaining a fraction of θ', introducing a non-optimistic approximation of the target values (LILLICRAP et al., 2019). Algorithm 1 presents the complete DDPG method.

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|w)$ and actor $\mu(s|\theta)$ with weights $w$ and $\theta$.
Initialize target network $Q'$ and $\mu'$ with weights $w' \leftarrow w$, $\theta' \leftarrow \theta$
Initialize replay buffer R
for episode=1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t=1, T do
        Select action $a_t = \mu(s_t|\theta) + \mathcal{N}_t$ with the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in R
        Sample transition a random minibatch of N transition $(s_i, a_i, r_i, s_{i+1})$ from R
        $A \leftarrow \mu'(s_{i+1}|\theta')$
        $y_i \leftarrow r_i + \gamma Q'(s_{i+1}, A|w')$
        Update critic by minimizing the loss:
            $$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|w))^2 \qquad (2.10)$$
        Update the actor policy using the sampled policy gradient:
            $$\nabla_\theta J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|w)|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s|\theta)|_{s_i} \qquad (2.11)$$
        Update the target networks:
        $w' \leftarrow \tau w + (1 - \tau)w'$
        $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
    end for
end for
```

2.6 POLICY OPTIMIZATION ALGORITHMS

Deep Reinforcement Learning methods based on PG can become noisy due to the learning value function estimation during the policy's training. To solve this problem, Trust Region Policy Optimization (TRPO) comes with the idea to limit the policy gradient step, so the policy updates smoothly (SCHULMAN et al., 2015).

Firstly they call the action values estimated by the Value Function Network as Advantage $\hat{A}$ values and then define the likelihood of the new policy over the behavior policy as r_t. The loss function of TRPO uses this probability ratio to weight $\hat{A}$, so it maximizes the action distribution over the states (see Equation 2.12).

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2.12}$$

To constrain the policy update, they use a Kullback–Leibler (KL) divergence term shown in Equation 2.13. It checks the divergence of the new policy to the old and clips the new policy if KL is higher than a hyperparameter δ.

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathsf{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \end{aligned} \tag{2.13}$$

The PPO came with the same idea as TRPO but is more straightforward and faster (SCHULMAN et al., 2017). PPO's loss function also uses the probability ratio, but the constraining of the new policy is slightly different. The KL term of TRPO is known to introduce overhead in the optimization process. PPO uses a faster method by clipping the probability ratio and then using $\min(\pi_\theta, \pi_{\theta_{old}})$ (See equation 2.14).

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, \mathsf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.14}$$

We use and adapt the Algorithm 2 on discrete action space environments in this work.

Algorithm 2 PPO algorithm

```
Randomly initialize critic network $Q(s, a|w_Q)$ and actor $\mu(s|\theta_\mu)$ with weights $w_Q$ and $\theta_\mu$.
Initialize episode buffer R
for episode=1, M do
    Receive initial observation state $s_1$
    for t=1, T do
        Select action $a_t = \mu(s_t|\theta_\mu)$
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in R
    end for
    Create random minibatches from R and run the training loop:
    for t=1, Epochs do
        for mb=1, MiniBatches do
            Get minibatch mb
            Compute $\hat{A}_t$ for mb
            Compute $r_t(\theta)$
            Update the actor using Equation 2.14
            Update the critic like DDPG method
        end for
    end for
end for
```

2.7 REWARD SHAPING

Reward shaping is a technique formally discussed in Ng, Harada and Russell (1999) used in RL to guide the agent to the target of the environment efficiently. NG; HARADA; RUSSELL show that we can modify the reward function of an MDP $M = (S, A, P, R)$ to $R'(s, a, s'|S_t = s, A_t = a, S_{t+1} = s') = R(s, a, s') + F(s, a, s')$ turning it into an MDP $M' = (S, A, P, R')$, where F is a potential-based function and proves that every policy optimal in M' will be optimal in M. Using potential-based functions, their experiments on discrete spaces (state and action) environments show better sample efficency and cumulative reward (NG; HARADA; RUSSELL, 1999).

Brys et al. (2014) use the premise that since Ng, Harada and Russell (1999) proved that a single potential-based function $F(s, a, s'|S_t = s, A_t = a, S_{t+1} = s')$ on reward function $R' = R + F$ of MDP $M' = (S, A, P, R')$ is helpful, we can describe the reward function as $R' = R + \sum_{i=1}^{N} F_i$, where F_i is a potential-based reward function for a sub-objective i of the environment. Their experiments were also done in environments with discrete spaces and showed better performance when adding a reward function for reaching those subobjectives (BRYS et al., 2014). Mannion et al. (2017) state that Brys et al. (2014) experiments were not on multi-objective environments and showed better results with the method of Brys et al. (2014) in other environments.

In this chapter, we have presented the foundations of this work. The first three sections introduce the basis of Reinforcement Learning that we use in our method. The following sections describe the Policy Gradient methods and Reward shaping idea that we use to prove and adapt to Dynamic λ weighting function (DyLam).

3 RELATED WORK

The sparse-rewards problem can be understood as an exploration problem. The agent must explore well enough to reach the objective several times in a sparse reward environment. While exploring, the RL method keeps updating the value function to approximate the optimal one (SUTTON; BARTO, 2018). This chapter discusses fundamental and state-of-the-art works in RL to solve problems with different types of exploration. We use the ideas and premises of the works presented in this chapter to prove and base this thesis.

We present three possible research lines to solve the exploration problem: **reforming the value function or decomposing the policy**, methods related to the training method; **intrinsically motivated reinforcement learning**, methods that add reward functions related to events of the environment; **Hindsight Experience Replay**, methods that modify the experience buffer every episode.

3.1 VALUE FUNCTION AND POLICY DECOMPOSITION

Shelton (2001) presents a mixture of PG and importance sampling applied to the multi-objective domain. The gradients of the policy are calculated individually for each objective and then combined, constructing a weighted gradient. By varying these weights, an exploratory policy is reached, and the agent is rewarded for all objectives of the environment (SHELTON, 2001). Their experiments were on a simple financial environment. We use similar ideas in our work, such as varying a weight vector using a PG method. Due to the lack of specification of the environment and parameters of the method, we could not reproduce this work.

Russell and Zimdars (2003) proposed stratifying the Q-values into N components, using an ensemble method to train the policy. They used the Q-learning method to approximate $\pi^*(s)$ but with multiple Q-tables, one for each subobjective of the environment and one global. Each sub-objective table is updated concerning its reward component, and the global table receives the maximum Q from all updated tables. The actions are chosen according to the global Q-table (RUSSELL; ZIMDARS, 2003). This work shows that stratifying the Q-value and ensembling it can become handy. We use this fact as a critical step in our work.

Perny and Weng (2010) propose an adaptation of the **value-iteration** method to train the policy. They also stratify the state-value function and discuss that only a sum of the components with the same weights is not a correct approach and present a new weighting mechanism that uses the Tchebycheff normalization function. Then an ensemble method is used to gather all state-value components and the training follows the value-iteration method. They used a grid-world problem with given models in their experiments (PERNY; WENG, 2010). We also normalize the stratified value function but use a different function, and we use the sum of weights found by our adaptive function to train our policy.

Brys et al. (2017) propose a policy ensemble method for RL. They combined four different policies, each for one reward component, with simple ensembles: Linear, Majority Voting, Rank Voting, and Confidence-based. The ablation study shows that training an ensemble policy that gathers different policies or chooses one at a time, in a general way, has better results than even with the linear ensemble (BRYS et al., 2017). An useful detail, which we use in our work, is that they normalized the rewards when possible.

Hu et al. (2020) work is the most recent and similar to ours. The authors used a different training procedure, but employ the same idea of weighting multiple value functions. Their weighting function is an ANN that weights the rewards with values $W \in (0, \infty)$. Their experiments were on Atari and Mujoco environments and demonstrated a better performance than standard methods. We tried to run the available code of this work according to the instructions provided, but the agents did not reach the same performance presented in the paper[1].

3.2 INTRINSICALLY MOTIVATED REINFORCEMENT LEARNING

SINGH; BARTO; CHENTANEZ present the prior work of Intrinsically Motivated Reinforcement Learning (IM) considering not only the environmental reward, also called as extrinsic reward, but intrisic rewards (SINGH; BARTO; CHENTANEZ, 2005). Their work aims to reward the agent when a salient event is reached and learn first how the agent got to that salient event. The policies to reach each salient event are called options defined in (SUTTON; PRECUP; SINGH, 1999). The global policy converges to an optimal policy that satisfies all options and reaches the environment's objective (SINGH; BARTO; CHENTANEZ, 2005). This thesis focuses on reward shaping methods due to their ease of application to state-of-the-art Policy Gradient methods and working community repositories (BRITZ, 2019; HUANG et al., 2021). Although it is a different method, we believe that multiple value functions for different rewards provide

[1] The repository is provided in <https://proceedings.neurips.cc/paper/2020/file/b710915795b9e9c02cf10d6d2bdb688c-Supplemental.zip>

better learning for the behavior policy as shown by (RUSSELL; ZIMDARS, 2003; BRYS et al., 2017; HU et al., 2020).

3.3 HINDSIGHT EXPERIENCE REPLAY

It can be difficult to specify auxiliary rewards for reward shaping methods (POPOV et al., 2017; MARTINS et al., 2022). Hindsight Experience Replay (HER) comes up with the idea of rewarding the agent when it reaches alternative goals (ANDRYCHOWICZ et al., 2018). The algorithm rewards not only for the current episode but for past episodes too. For example, the objective of the current episode is different from the last one; The agent does not reach the current objective, but it gets very close to the last; It is rewarded according to the current and last episode. The method creates a non-sparse reward once the RB contains more information around the rewards. Note that this method was experimentally shown to work only for *off-policy* algorithms (ANDRYCHOWICZ et al., 2018).

This chapter shows state-of-the-art methods to solve the exploration problem. Section 3.1 presents the most similar works to this one, and it is where we took most of our ideas. The methods show that decomposing the reward function and selecting what is significant at a certain point of the training loop can improve the sample efficiency of the methods. Another well-pointed idea in (BRYS et al., 2017) and (HU et al., 2020) is to normalize the rewards to reach a more stable training of the value function. We also bring IM and HER as working methods but not necessarily as inspiration nor part of our solution. Especially HER-based methods have the constraint to work only on off-policy methods (FANG et al., 2018; FANG et al., 2019). We want to apply and adapt our method as much as possible, so it is not a research line that we follow.

4 PROPOSED METHOD

This chapter presents the method proposed in this thesis Dynamic λ weighting function (***DyLam***). In summary, it automatically weights the Q-values during training, removing the need for manually adjusting the weights of the reward functions of the environment and letting the algorithm decide what is crucial at that moment.

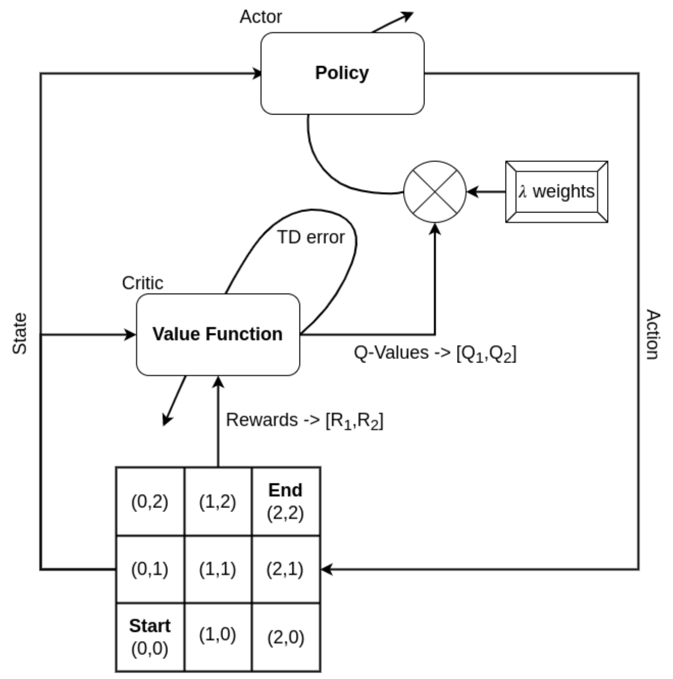
Firstly, the reward function is decomposed into components which should vary in similar ranges. For instance, we can decompose the reward function of a grid world environment into two functions of x and y distances to the objective (BRYS et al., 2014), each one ranging from zero to one. This decomposition is an intuitive human process that can be introduced by the environment designer. It is recommended to have all reward components varing in similar scales so that the method apply similar levels of importance for them, differing only in their learning difficulty, thus prioritizing learning the easiest ones first.

The second step is to decompose the value function into a list of Q-values, each associated with one reward component. Russell and Zimdars (2003)'s proof of decomposition of value functions (Section 3.1) is crucial to this part of our work since it ensures it will work for every MDP. This value function is a regression model which fits each expected reward component accumulated without applying any weights.

The last step is to rearrange the policy training process to multiply the value function by the weights λ found using a weight adjustment method to be presented in Section 4.2. This $\lambda \in \mathbb{R}^n$ is the vector of weights to be applied to each reward component, where n is the number of reward components. We illustrate and exemplify the whole process in Figure 5.

The sections below describe how our method works and present proof that it can be adapted into any PG method. In Section 4.1, we introduce the method and prove its validity for static weighting functions, and in Section 4.2, we do the same, but now considering a dynamic weighting function. In Section 4.3 we present the algorithm adaptation of DDPG and PPO methods.

Figure 5 – Illustration of our method on a Grid World problem. We decompose the reward function into two potential-based functions of $X(R1)$ and $Y(R2)$. We train the Critic using the PG method and use the λ weights combined with the Q-values to train the Actor. Comparing with the original Actor-Critic training in Figure 4, we notice the subtle change on training the policy module.



Source: Author

4.1 STATIC WEIGHTING FUNCTION

The static weighting function is the first step to proof and understanding DyLam. This method extends the usual reward shaping by transferring the weights from the environment to the agent. It also gives freedom to the agent designer to choose which reward component is more related to the objective in that training session.

The proposed critic uses the regular training scheme of the PG methods, but for multiple outputs. We want to prove that we can state the Policy Gradient Theorem even when weighting the Q-values of the actor's gradient $\nabla_\theta J$ instead of weighting the reward components in the environment and combining them into a single value, as usually done.

Theorem 4.1.1 (Extended Policy Gradient theorem)**.**

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q_\pi(s,a)\ \delta a\ \delta s \\ &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(s) \sum_i^N \lambda_i Q_i^\pi(s,a) \delta a\ \delta s\end{aligned} \tag{4.1}$$

Proof. Using the first form of Equation 2.8, we substitute the term of the value function by a weighted sum of Q-values, $\sum_i^N \lambda_i Q_i^\pi(s,a)$, where λ_i and Q_i^π are, respectively, the weight and

Q-value for the i-th reward component, and reach the Theorem 2.4.1.

$$\nabla_\theta J(\pi_\theta) = \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(s) \sum_i^N \lambda_i Q_i^\pi(s,a) \delta a \ \delta s. \tag{4.2}$$

Substituing each $\lambda_i Q_i^\pi(s,a)$ by $Q_i'^\pi(s,a)$ we obtain:

$$\nabla_\theta J(\pi_\theta) = \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(s) \sum_i^N Q_i'^\pi(s,a) \delta a \ \delta s. \tag{4.3}$$

As Russell and Zimdars (2003) show, we can decompose a value function $Q(s,a)$ into N components $Q_i(s,a)$, where $Q(s,a) = \sum_i^N Q_i(s,a)$. Therefore, we can rewrite $\nabla_\theta J$ as:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(s) Q'^\pi(s,a) \delta a \ \delta s, \\ &= \mathbb{E}_{S \sim p^\pi, a \sim \pi_\theta} [\nabla_\theta log \pi_\theta(a|s) Q'^\pi(s,a)]\end{aligned} \tag{4.4}$$

which is the the Policy Gradient theorem (Theorem 2.4.1). $\square$

4.2 DYNAMIC WEIGHTING FUNCTION

Using the static weighting function, we proved that the Critic becomes a regressor of Q^* for each reward component and trained the Actor with a weighted sum of Q-values. Using a dynamic weight adjustment method, we do the same for training the estimators, but we change the function that weights the Q-values. Now the vector λ is a function of the actual performance of the agent in the environment:

$$\overline{R_t^i} = R_t^i + \tau(R_{t-1}^i - R_t^i) \tag{4.5}$$

$$\zeta_i(\overline{R_t^i}) = \frac{R_{max}^i - \overline{R_t^i}}{R_{max}^i - R_{min}^i} \tag{4.6}$$

$$\lambda_i(\overline{R_t^i}) = \frac{e^{\zeta_i} - 1}{[\sum_i^C (e^{\zeta_i} - 1)] - \epsilon} \tag{4.7}$$

where R_t^i is the moving average accumulated return for the i-th component, τ is a smooth update factor, ζ is the inverse score of the component according to its maximum reward, R_{max} and R_{min} define the ranges for the reward component, and ϵ is a small value to prevent division by zero.

The main idea of the method is to reduce each weight λ_i as the agent learns the associated skill, prioritizing on the harder ones after learning the easier ones. However, this must be done smoothly to prevent instabilities. Therefore, we use smooth updates similar to the τ-updates presented in DDPG (LILLICRAP et al., 2019). The smooth λ updates lead to a low variance of $\sum_i^N \lambda_i Q_i(s,a)$ for the same state-action pairs along with the training. We update our theorem in Theorem 4.2.1.

Theorem 4.2.1 (Dynamic λ Policy Gradient theorem).

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(s) \sum_i^N \lambda_i(\overline{R_t^i}) Q_i^\pi(s,a) \delta a \; \delta s \\ &\iff E \ggg 1 \text{ and } |\tau(R_t^i - R_{t-1}^i)| \approx 0,\end{aligned} \tag{4.8}$$

where E is the number of episodes to take the average sum.

Proof. As the dynamic weighting function varies with time, we cannot prove the Theorem 4.2.1 as we proved the Theorem 4.1.1. We use the constraint of gathering many episodic returns to measure the agent's performance ($E \ggg 1$). This condition leads to $R_t^i \approx R_{t-1}^i$, removing the time dependency of Equation 4.7. Therefore, we can substitute $\lambda_i(\overline{R_t^i})$ with λ_i to reach the Theorem 4.1.1 and then extend Theorem 2.4.1:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(s) \sum_i^N \lambda_i(\overline{R_t^i}) Q_i^\pi(s,a) \delta a \; \delta s \\ &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(s) \sum_i^N \lambda_i Q_i^\pi(s,a) \delta a \; \delta s \\ &= \int_S p^\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q_\pi(s,a) \; \delta a \; \delta s\end{aligned} \tag{4.9}$$

□

4.3 ALGORITHMIC ADAPTATION

Our algorithmic adaptation is uncomplicated and straightforward. The static weighting is very similar to the dynamic one. It only changes when calculating new weights to the Advantage function. Therefore we present general algorithms for both proposed methods for DDPG and PPO on Algorithms 3 and 4.

This chapter introduced our proposed methods to enhance the reward shaping method. We first showed how the methods work in general lines and then proved that both extend the Policy Gradient theorem. The static weighting is the first step that removes the weights of the reward components from the environment and puts them only to train the actor module. The dynamic weighting is the automation of finding the ideal weights for each component in a dynamic way In the following chapter, we present the experiments for both methods using two different Actor-Critic methods in two different environments.

Algorithm 3 DDPG with reward weighting algorithm

```
**Let $NR$ be the number of reward components of the environment**
Randomly initialize $NR$ critic networks $Q(s,a|w_{Q_i})$ and actor $\mu(s|\theta_\mu)$ with weights $w_{Q_i}$ and $\theta_\mu$
Initialize $NR$ target networks $Q'_i$ and $\mu'$ with weights $w_{Q'_i} \leftarrow w_{Q_i}$, $\theta_{\mu'} \leftarrow \theta_\mu$
Initialize the replay buffer D
if Using Satic weighting Function then
    Initialize $\lambda$ weight vector with the desired weights
else
    Initialize reward queue $RQ$ empty
    Initialize $\lambda$ weight vector with the same weights
end if
for episode=1, $M$ do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    Run policy $\mu(s|\theta_\mu)$ in environment for $T$ steps and store the transitions in $D$
    Store the accumulated reward of each component in $RQ$
    $A \leftarrow \mu'(s_{i+1}|\theta'_\mu)$
    for j=1, $NR$ do
        $y_{i,j} \leftarrow r_{i,j} + \gamma Q'_j(s_{i+1}, A|w_{Q'_j})$
        Update critic by minimizing the loss:
            $$L_j = \frac{1}{N}\sum_i (y_{i,j} - Q_j(s_i, a_i|w_{Q_j}))^2 \quad (4.10)$$
    end for
    if Using DyLam then
        for i=1, $NR$ do
            $R^i_t \leftarrow mean(RQ_i)$
        end for
        Update $\lambda$ weight vector according to Equation 4.7
        $R^i_{t-1} \leftarrow R^i_t$
    end if
    Update the actor policy using the sampled policy gradient:
        $$\nabla_{\theta_\mu} J \approx \frac{1}{N}\sum_i \sum_j^{NR} \lambda_j \nabla_a Q_j(s,a|w_{Q_j}) \quad (4.11)$$
    Update the target networks:
    for i=1, $NR$ do
        $w_{Q'_i} \leftarrow \tau w_{Q_i} + (1-\tau) w_{Q'_i}$
    end for
    $\theta_{\mu'} \leftarrow \tau\theta_\mu + (1-\tau)\theta_{\mu'}$
    if  Size of $RQ > NR$ then
        Remove the oldest reward of $RQ$
    end if
end for
```

Algorithm 4 PPO with reward weighting algorithm

```
**Let $NR$ be the number of reward components of the environment**
Randomly initialize $NR$ critic networks $Q(s, a|w_{Q_i})$ and actor $\mu(s|\theta_\mu)$ with weights $w_{Q_i}$ and $\theta_\mu$
Initialize episode buffer $D$
if Using Static weighting Function then
    Initialize $\lambda$ weight vector with the desired weights
else
    Initialize reward queue $RQ$ empty
    Initialize $\lambda$ weight vector with the same weights
end if
for episode=1, $M$ do
    Receive initial observation state $s_1$
    Run policy $\mu(s|\theta_\mu)$ in environment for T steps and store the transitions in $D$
    Store the accumulated reward of each component in $RQ$
    if Using DyLam then
        for i=1, $NR$ do
            $R^i_t \leftarrow mean(RQ_i)$
        end for
        Update $\lambda$ weight vector according to Equation 4.7
        $R^i_{t-1} \leftarrow R^i_t$
    end if
    Create random minibatches from R and run the training loop:
    for t=1, Epochs do
        for mb=1, MiniBatches do
            Get minibatch mb
            Compute $\hat{A}_t$ for mb
            Compute $r_t(\theta)$
            Update the actor with the adpated Loss function:
                $$\hat{R} = \sum_{i}^{NR} \lambda_i \hat{A_{t,i}} \quad (4.12)$$
                $$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{R}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{R})] \quad (4.13)$$
            Update the critic like DDPG method
        end for
    end for
    if Size of $RQ > NR$ then
        Remove the oldest reward of $RQ$
    end if
end for
```

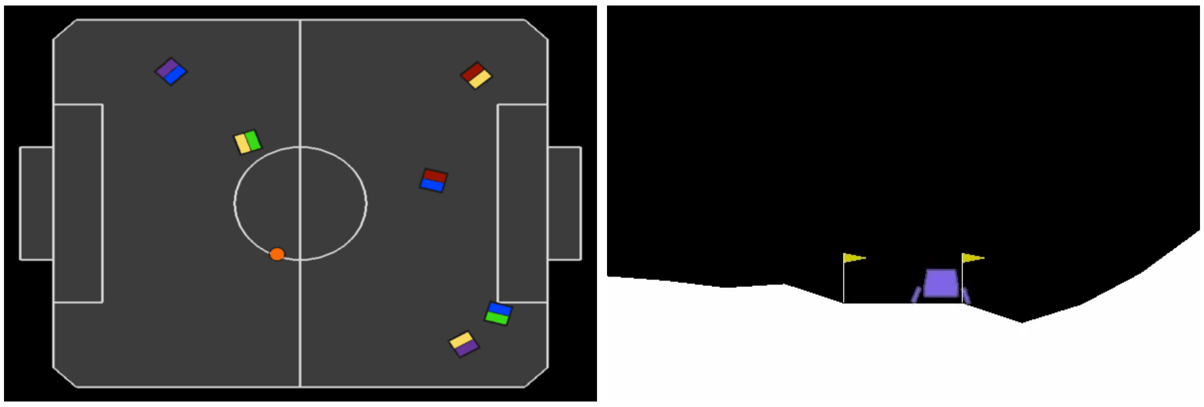
5 EXPERIMENTS AND RESULTS

This chapter aims to validate our proposed method with experiments on Reinforcement Learning environments. We first show the environments adapted to our agents (Section 5.1). In Section 5.2, we show the results achieved in each environment using the adaptation presented in Section 4.3 for DDPG and PPO methods and comparing with the original ones.

5.1 EXPERIMENTS

This section describes the environments and experiments that validated our methods. We divided our experiments into continuous and discrete action spaces for DDPG and PPO, respectively. For the DDPG experiments, we created a robot soccer's framework called rSoccer (MARTINS et al., 2022) during the development of this work. We executed the PPO experiments on the classic Lunar Lander environment (BROCKMAN et al., 2016). Both environments are represented in Figure 6. It is important to point that we executed three times each experiment shown in this chapter and each figure shows the running average of them using 100 steps to smooth the curve. Considering that each experiment lasts about three hours, we were not able to execute more experiments with the same setup.

Figure 6 – Environments we used in our experiments. The rSoccer environment is represented in the left and the Lunar Lander, in the right.



Source: Author

5.1.1 rSoccer Environment

We adapted the VSS-v0 environment of rSoccer only for the normalization step, maintaining the general reward shaping. We call our environment VSSStrat-v0. We perform a grid search on the vector λ for static weights using rSoccer as a basis. We executed the experiments in this environment with the parameters shown in Table 1. The idea is to conduct experiments with different priorities for each component, enabling us to understand their impact. After this study, we applied DyLam by adjusting λ online. Our DDPG hyperparameters are described in Table 2[1].

We follow the same goal score metric as defined in Martins et al. (2022), a rate of scoring or taking a goal. We show the behavior of each reward component during training and how it affects our metric. The components of this environment are: moving towards the ball, ball's move towards the enemy's goal, energy spent of the robot, and scoring a goal. We planned the grid search experiments in Table 1 to be: double impact, half impact and no impact over the other reward components. For the DyLam experiments, we set the range of the components as follows:

- Move - $[0, 0.5]$
- Ball - $[0, 1]$
- Energy - $[-2, -1]$
- Goal - $[0, 1]$

Regarding that these components range are used only in the Equation 4.7. The hyperparameters of the DDPG and DyLam are presented in Table 2.

5.1.2 Lunar Lander Environment

We adapted the reward shaping of the LunarLander-v2 environment to a straightforward one (BROCKMAN et al., 2016). The original shape uses euclidian distances of ideal x and y positions and speeds. We separate these two distances into four as our idea is to decompose as much as possible the reward components and analyze them separately. We describe the

[1] Code available on https://github.com/goncamateus/rl-agents-pytorch.

Table 1 – Grid Search table for the static λ vector.

	Component weight			
Experiment	**Move**	**Ball**	**Energy**	**Goal**
Balanced	0.2500	0.2500	0.2500	0.2500
Move_double	0.4000	0.2000	0.2000	0.2000
Move_half	0.1428	0.2857	0.2857	0.2857
Move_null	0.0000	0.3333	0.3333	0.3333
Ball_double	0.2400	0.4400	0.2400	0.2400
Ball_half	0.2857	0.1428	0.2857	0.2857
Ball_null	0.3333	0.0000	0.3333	0.3333
Energy_double	0.2400	0.2400	0.4400	0.2400
Energy_half	0.2857	0.2857	0.1428	0.2857
Energy_null	0.3333	0.3333	0.0000	0.3333
Goal_double	0.2400	0.2400	0.2400	0.4400
Goal_half	0.2857	0.2857	0.2857	0.1428
Goal_null	0.3333	0.3333	0.3333	0.0000

Source: Author

reward components of this adapted environment with the ranges of DyLam experiments as follows:

1. X distance to the center of the objective - X component - $[-1, 0]$
2. Y distance to the center of the objective - Y component - $[-1, 0]$
3. X speed of the agent - Speed_X component - $[-0.8, -0.03]$
4. Y speed of the agent - Speed_Y component - $[-0.5, -0.02]$
5. Angle difference to the ideal landing angle - Angle component - $[-1, 0]$
6. Power spent on X-axis thruster - Power_X component - $[-1, -0.2]$
7. Power spent on Y-axis thruster - Power_Y component - $[-1, -0.2]$
8. If the left leg contacted the ground - Leg_left component - $[-1, 1]$
9. If the right leg contacted the ground - Leg_right component - $[-1, 1]$
10. If the agent reached the final objective - Land component - $[-1, 1]$

We maintain the components as potential-based functions as the original implementation of the environment. This environment has the advantage of having the reward components in

Table 2 – Hyperparameters for the runs in the rSoccer environment.

Actor - Layers	6 Hidden Linear Layers
Actor - Units	256 x 256 x 512 x 512 x 256 x Actions
Actor - Activations	5 ReLU and 1 Tanh
Actor - α	0.0001
Critic - Layers	6 Hidden Linear Layers
Critic - Units	512 x 512 x 1024 x 512 x 512 x NR
Critic - Activations	5 ReLU
Critic - α	0.0001
DDPG - Soft update	0.9950
Noise - σ initial	0.8000
Noise - σ Decay	0.9900
Noise - σ min	0.1500
Noise - σ update steps	3000
Noise - μ	0.0000
Noise - θ	0.1500
γ	0.9500
DyLam - Reward buffer length	10
DyLam - Soft update	0.99999
Replay Size	5000000
Init trainning	100000
Batch Size	256
Concurrent Environments	4
Number of steps per policy update	20

Source: Author

the same scaling interval, which is well suited for our method, as described in Chapter 4. We call this environment with decomposed rewards LunarLanderStrat-v0.

We performed the experiments with a different setup in this environment. First, we ran the DyLam to find the mean of each λ component, then reran the experiment with the vector λ fixed with the values found. The hyperparameters of PPO are described in Table 3[2]. The original environment's objective is to obtain 200 accumulated points. We use this metric to evaluate the methods. Additionally, we show the rate of landing successfully or not. These graphs show that some agents do not land and stay still besides accumulating the points.

[2] Code available on https://github.com/goncamateus/agents.

Table 3 – Hyperparameters for the runs in the Lunar Lander environment.

Actor - Layers	3 Hidden Linear Layers
Actor - Units	64 x 64 x Actions
Actor - Activations	2 Tanh and 1 Categorical
Actor - α	0.00025
Critic - Layers	3 Hidden Linear Layers
Critic - Units	64 x 64 x NR
Critic - Activations	2 Tanh
Critic - α	0.00025
GAE - λ	0.95
Clipping coefficient	0.2
Entropy coefficient	0.01
Value function coefficient	0.5
Maximum norm of gradient clipping	0.5
γ	0.99
DyLam - Reward buffer length	10
DyLam - Soft update	0.995
Replay Size	5000000
Init trainning	100000
Concurrent Environments	4
Number of steps per policy update	128
Number of minibatches	4
Batch size	516
Minibatch size	128
Update epochs	4

Source: Author

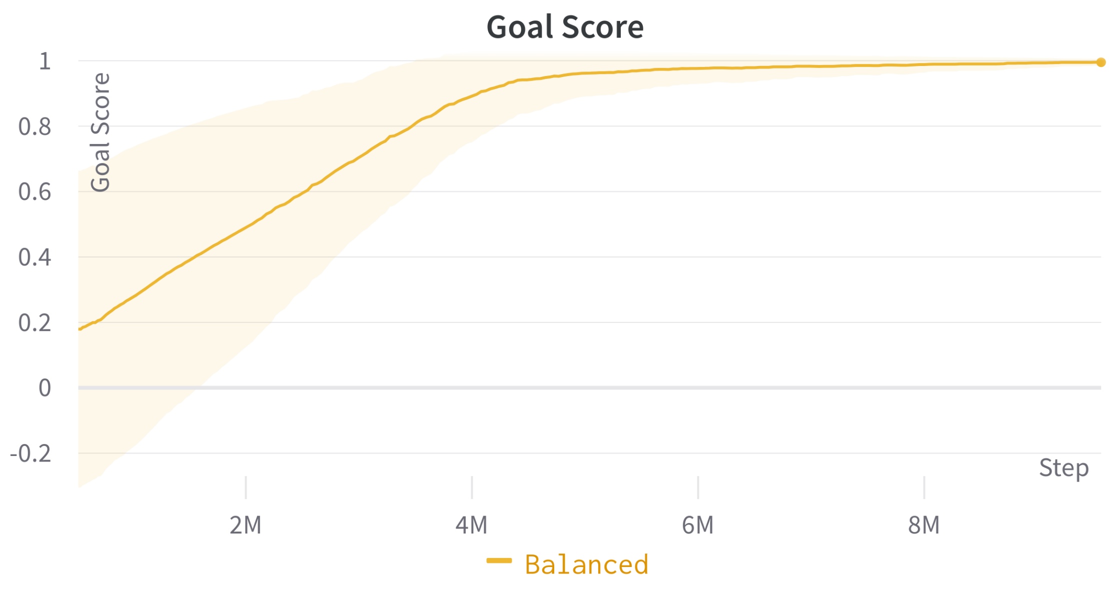
5.2 RESULTS

This section discusses the results achieved on the experiments designed in Section 5.1. We present a first and detailed analysis of the results of DDPG with the static and dynamic weighting function on the rSoccer environment, results on Lunar Lander using PPO, and analyze the impact of the λ smoothing method introduced in Section 4.2.

5.2.1 Static Weighting Function Performance on rSoccer

Figure 7 shows the mean and standard deviation that the DDPG follows in every experi-

Figure 7 – Mean and stand error of the Balanced experiment. The mean values are represented by the highlighted color and the standard error, the shaded one.



Source: Author

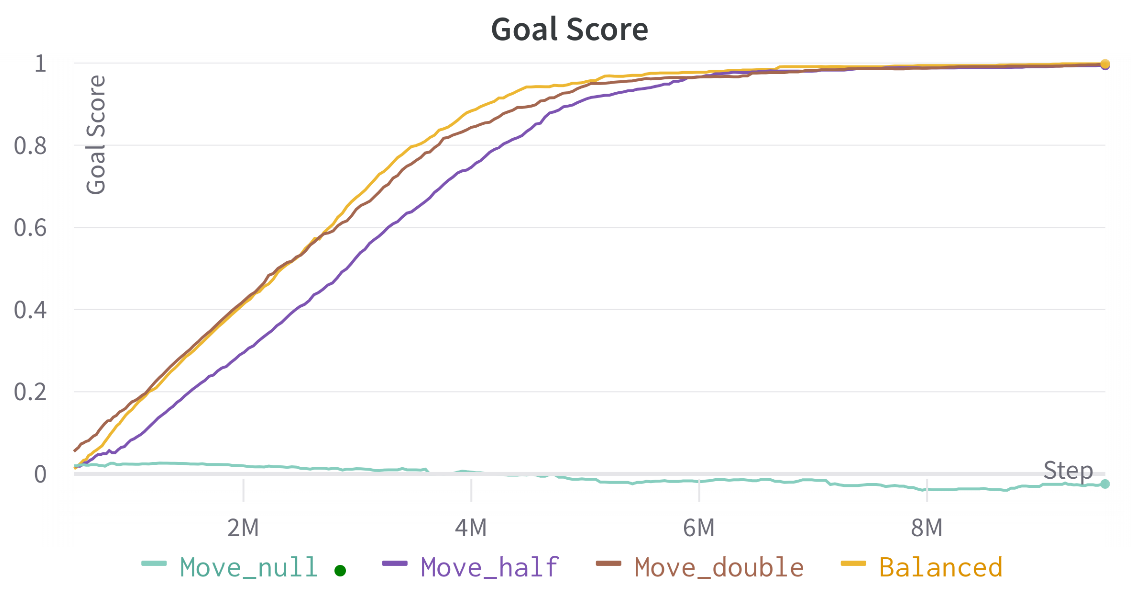
ment in this subsection. Note that, besides the standard deviation reaching around 0.5 at the beginning of the train, it converges to 0.

In Figures 8, 9, 10, and 12, we show the results of the experiments proposed in Table 1, each figure focusing on one component. The experiment named ‘Balanced’ with equal weights (0.25 for each component) presented excellent results compared to the other experiments. As the best results (e.g., experiment Ball_half) did not differ much in the sample efficiency, we will use the Balanced as the basis of our experiments in this section.

Analyzing Figure 8, the importance of the Move component in this environment is evident. When looking only at the Balanced and Move_double experiments, we note that increasing the weight of this component does not affect the sample efficiency. The Move_half experiment shows that the component is important for the early training but, once learned, does not impact the result (second half of the training). The Move_null experiment is the most interesting once it shows that the algorithm learns nothing without the component.

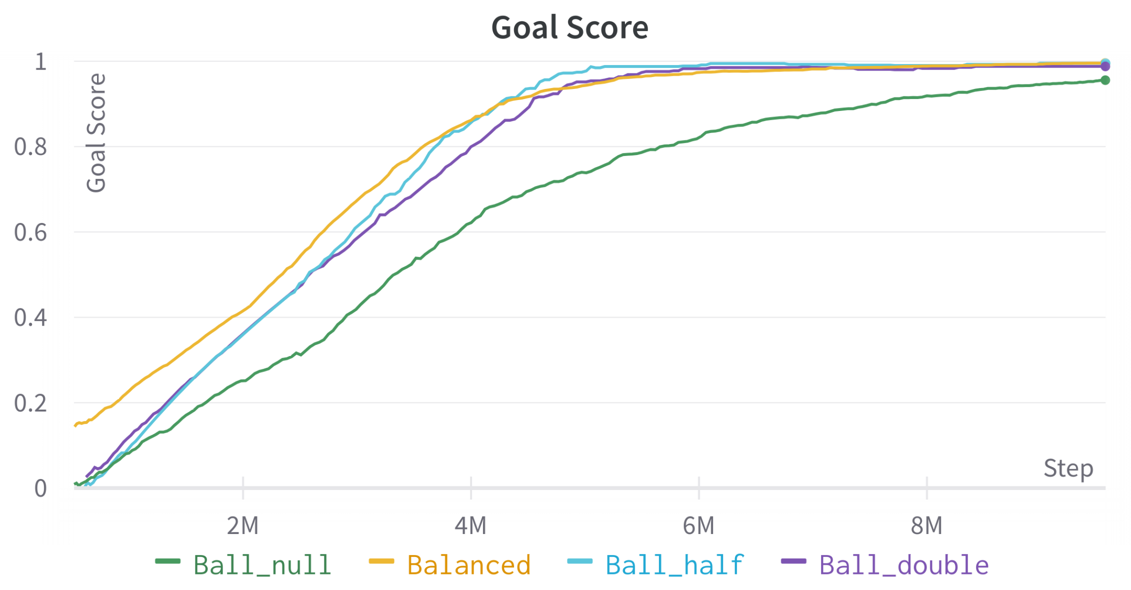
In particular, for the Ball component, we notice an interesting behavior in Figure 9: Ball_double performance is worse than Ball_half. The Ball_double has a lower weight on the first component, and we show on experiment Move_half that it influences the early per-

Figure 8 – Results of experiments of Table 1 for the first component, related to movement towards the ball.



Source: Author

Figure 9 – Results of experiments of Table 1 for the second component, related to the movement of the ball towards the enemy's goal.

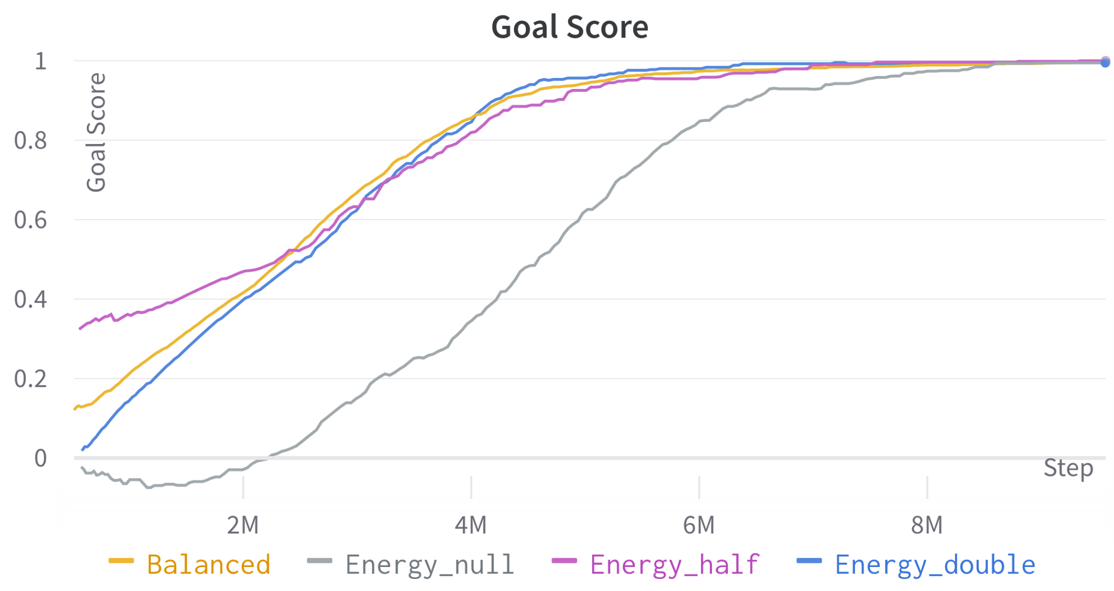


Source: Author

formance of the agent. The inverse occurs for Ball_half, which we can explain with experiment Move_double. Comparing the other results in Figure 9 with the experiment Ball_null, we check that the Ball component is a handful but not essential when combining the other components.

Figure 10 shows the experiments concerning the Energy component. This component was designed to reduce the wear of the motors in simulated robots, but we note a correlation of

Figure 10 – Results of experiments of Table 1 for the second component, related to the energy spent by the robot.



Source: Author

this penalty with the goal score. We created the experiment Energy_null expecting a similar behavior to the Balanced, but with some longer episodes. The obtained behavior was an early random agent but a later expert one, see Figure 11. The other experiments show that this component is supportive as the Ball component, accelerating the training but not very impactful if removed in this setup.

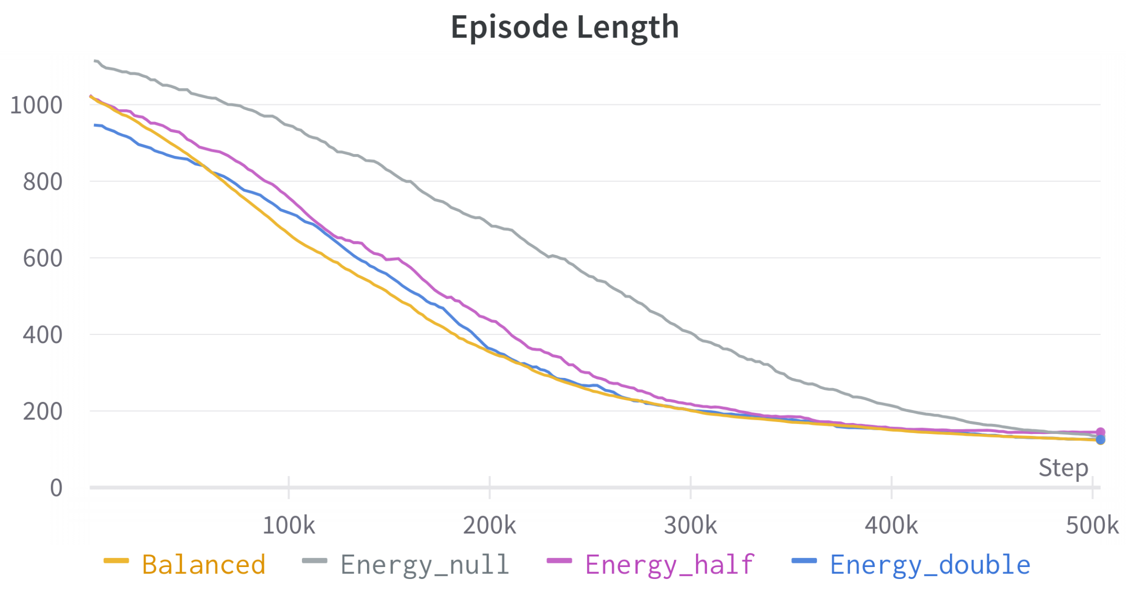
The Goal component is the most sparse one. Figure 12 shows that it does not impact the training of the learning algorithm. Our theory is that this task decomposition, for this specific environment, removes the necessity of this component.

This search for an adequated lambda vector clarified the impact of each reward component, but it is an exhaustive process. We want to automate the process with a DyLam and let the method learn what is important during the training.

5.2.2 DyLam Performance on rSoccer

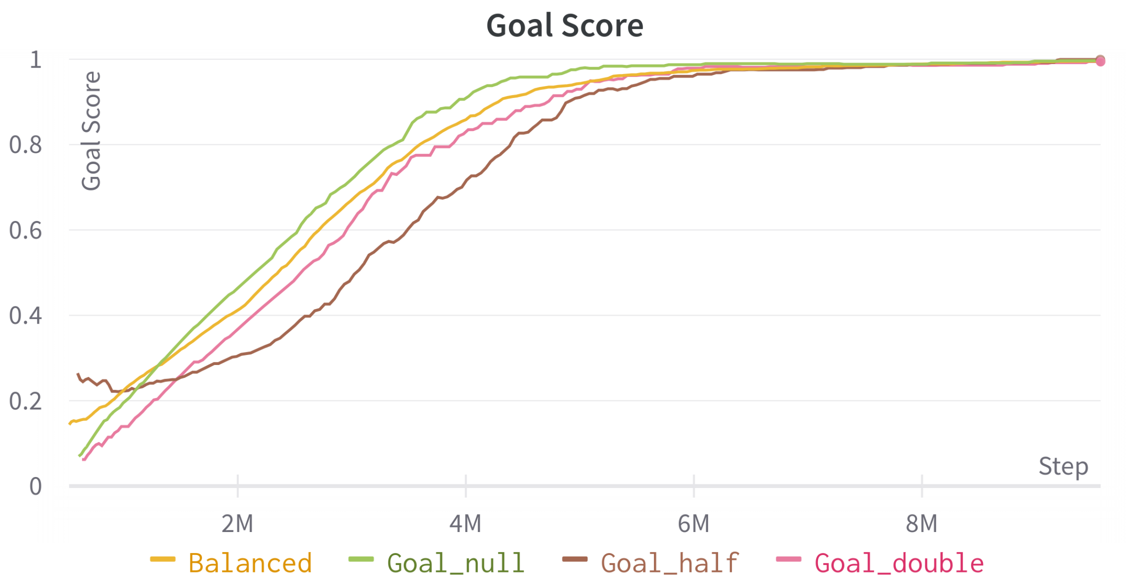
We show in Figure 13 the performance of DyLam compared with the Balanced. We note a better sample efficiency on the Balanced experiment, but the dynamic pondering did not

Figure 11 – Episode length along time of experiments related to the third component compared to the standard.



Source: Author

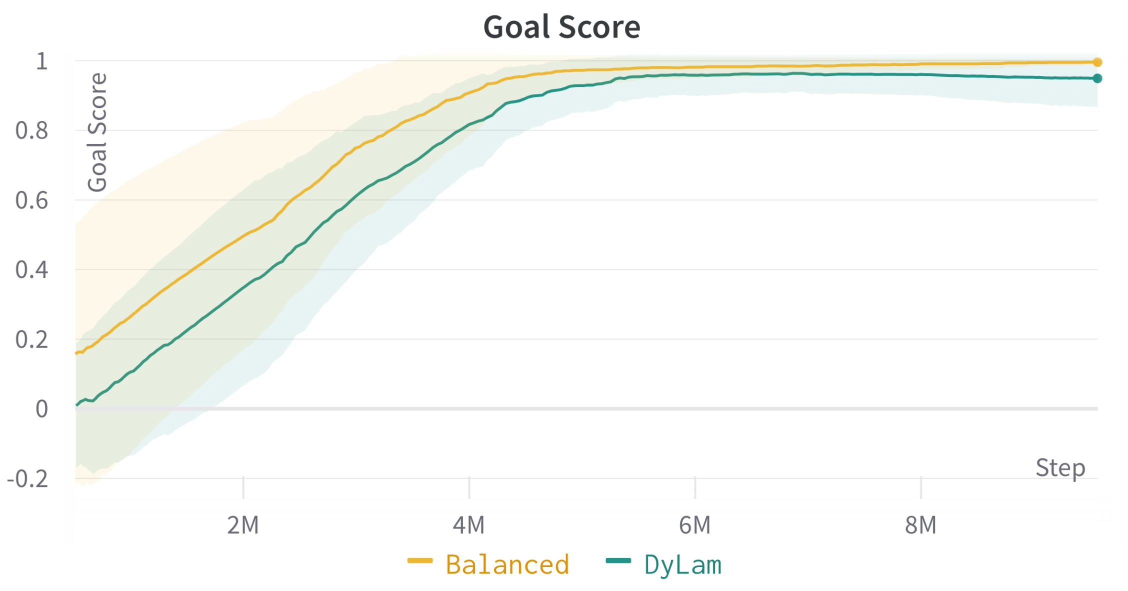
Figure 12 – Results of experiments of Table 1 for the second component, related to the goal score.



Source: Author

get far behind it. This performance gives hope in the sense that some hyperparameter used in the training sections might have caused this slightly worse efficiency. We show on further experiments that DyLam even helps find reward hacking weights on the Lunar Lander environment.

Figure 13 – Comparison of our Dynamic method compared to the basis experiment for rSoccer environment. The mean values are represented by the highlighted color and the standard error, the shaded one.



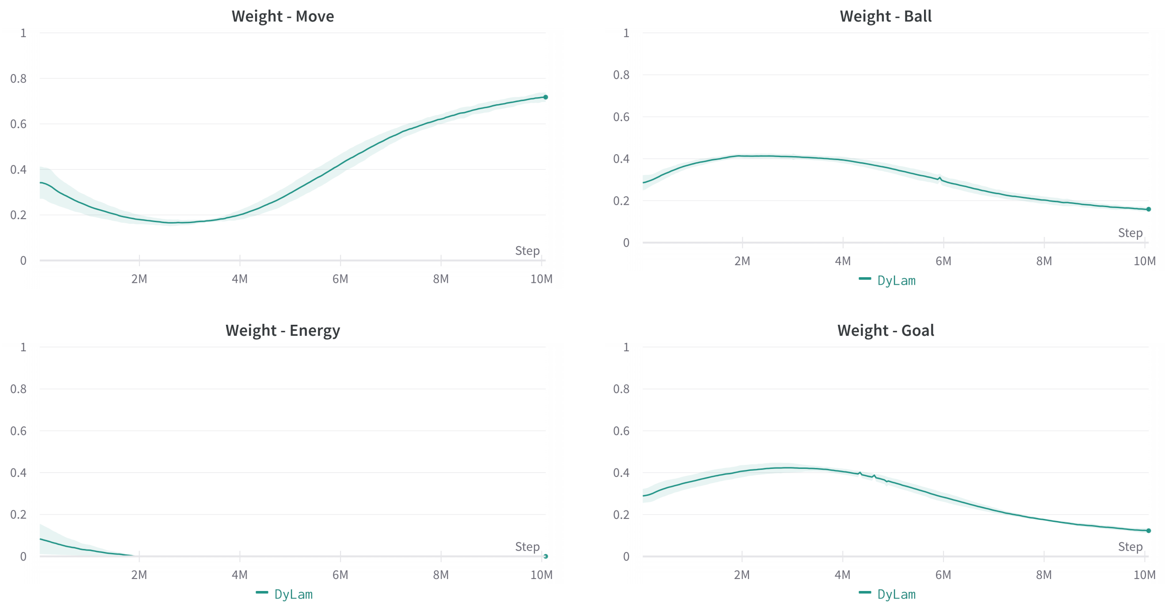
Source: Author

In Figure 14 we present the adaptation of the lambda vector during training. When analyzing the weight adaptation of the Move component, we observed that it begins with relatively high importance. The following fast decrease of the λ weight indicates that the agent learns the component. Although the agent already knows how to go to the ball, the agent begins to score fast enough that the component's weight increases again. This behavior probably happens because the agent learned how to 'kick' the ball.

We have discussed the impact of the Energy component in this environment. We note that it starts with some influence, but it significantly decayed by the beginning of the training and is kept null. Unlike the Move component, the Energy reward exceeds the maximum range we set, so DyLam annuls it.

The weights of the Ball and Goal components have similar behavior. The agent first learns to score by pushing the ball until they reach the goal. Then it learns to 'kick' the ball, as mentioned before. This behavior explains the subtle decrease at half the training and the increasing goal score result. At the end of the training, the agent learns to score as fast as possible, becoming an ideal football attacker.

Figure 14 – Behavior of each λ weight component using the Dynamic experiment in VSS environment. The mean values are represented by the highlighted color and the standard error, the shaded one.



Source: Author

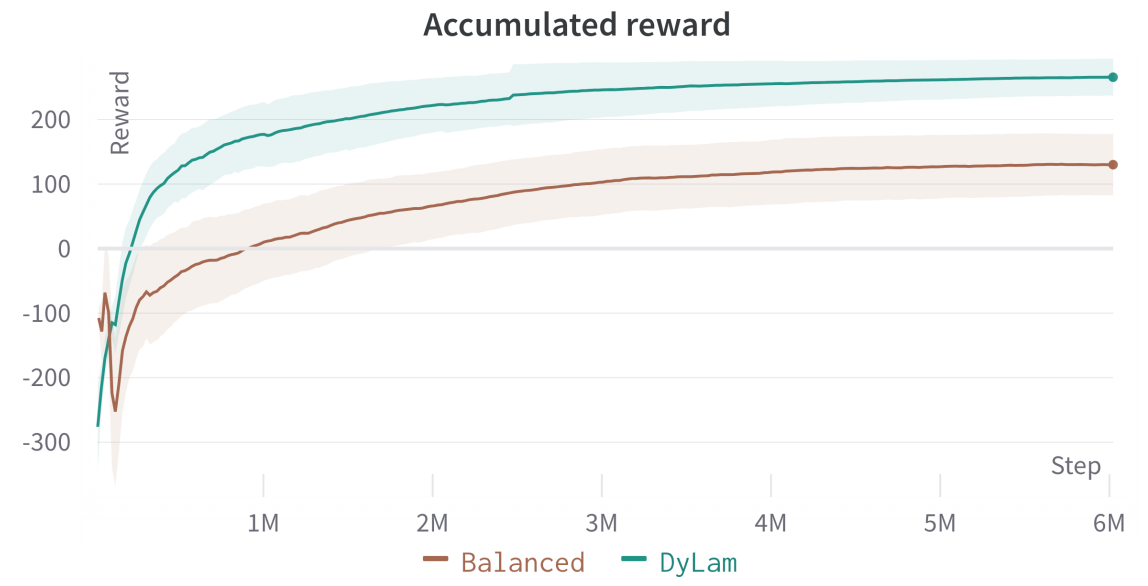
5.2.3 DyLam Performance on Lunar Lander

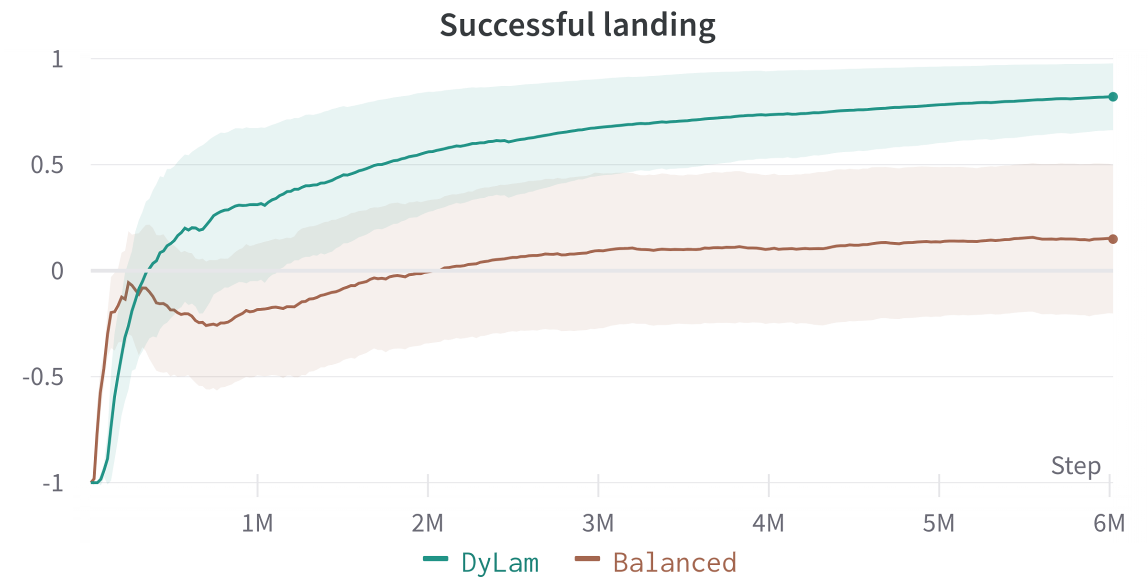
Figure 15 shows the comparison between the dynamic method and the original PPO. We see that around 1 million steps, the dynamic weighting method solves the environment (reachs 200 of cumulated reward) and keeps improving the agent. Meanwhile, the experiment using the original PPO does not even solve it. We suppose that the hyperparameters of PPO are not ideal for the environment, but they are the same for both methods. Therefore, our method outperformed the baseline in this situation.

We show in Figure 16 the behavior of each component of λ weight. We note that some rewards are easier to learn at the beginning of the training, so their priority is higher but becomes less important than others at the end, as the Angle component.

An important observation in this experiment is that we set the soft update parameter of the λ vector to 0.995, instead of 0.9999 used in DDPG, what causes noiser updates. This was required because, a slower update hinders learning in this environment using PPO. We hipothesize that this is related to the fact that PPO is an on-policy method, requiring faster λ soft updates.

Figure 15 – Results of PPO with and without our dynamic λ method. The mean values are represented by the highlighted color and the standard error, the shaded one. We show that on both proposed metrics our metric outperformed the baseline method.





Source: Author

5.2.4 Static Weighting Function Performance on Lunar Lander

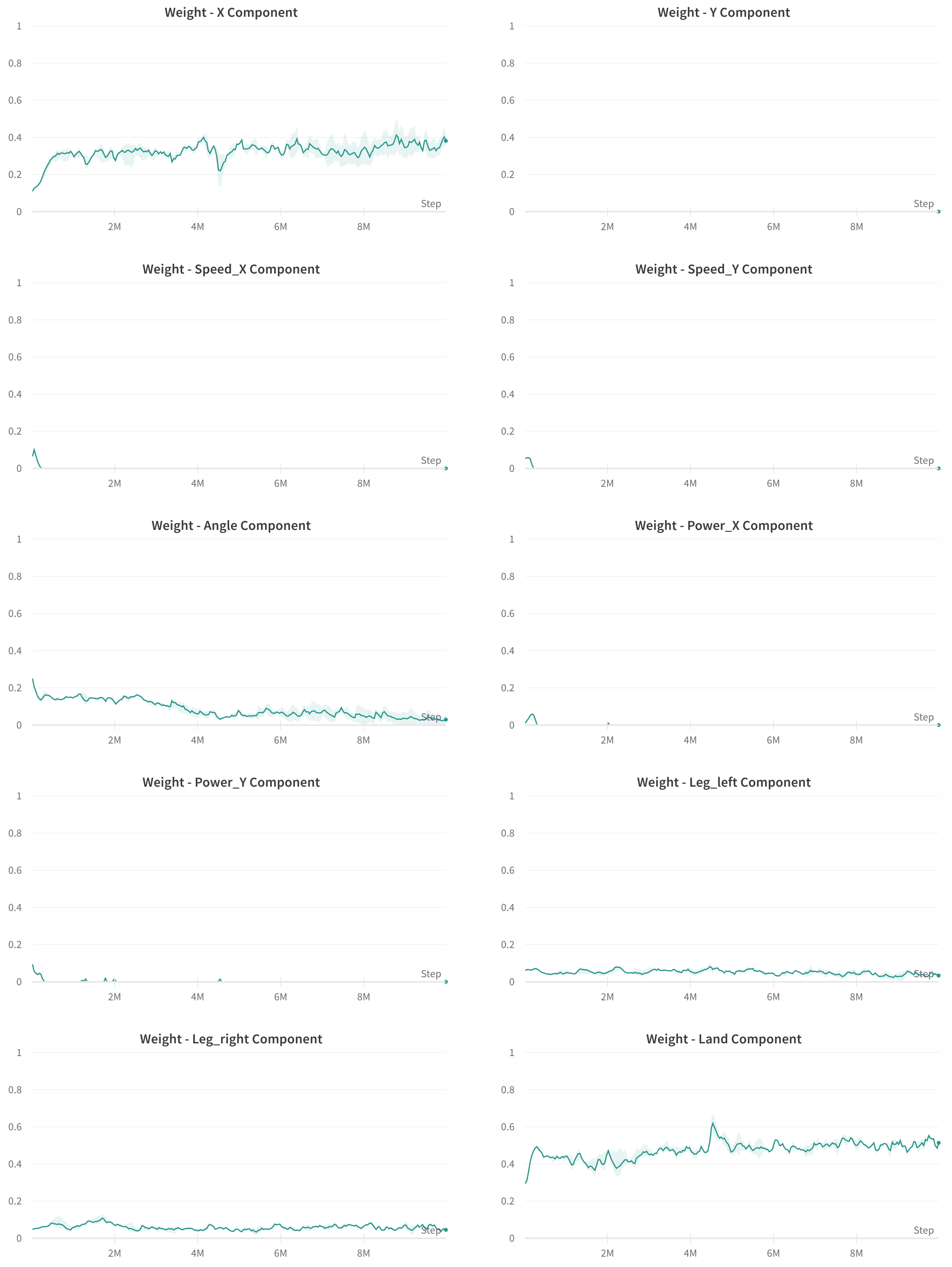
The result of the mean vector λ found by DyLam is presented in Figure 17. We use DyLam as the baseline method. The results show that the agent found a way to exploit the reward system using these weights. Note that even achieving a higher reward, the agent did not complete the objective as well as DyLam did. This experiment suggests that there is one or more components that do not follow the potential-based function suggestion of Ng, Harada and Russell (1999).

5.2.5 Policy Loss Analysis

To show the impact of the λ smooth update parameter, we experimented in the rSoccer and Lunar Lander environments with the parameter set to 0. In Figures 18 and 19 we compare this experiment with the dynamic method presented earlier. We note in Figure 19a and Figure 18a that there is not much difference between the agents' performance with and without the λ smooth update. These results indicate a theoretical need for this parameter but not a practical one. Figure 18b and Figure 19b show the high variance of policy loss in both experiments without smooth updates. We suppose that smooth updates are more impactful in more complex value function environments. E.g., a rSoccer environment that uses images as inputs instead of described features.

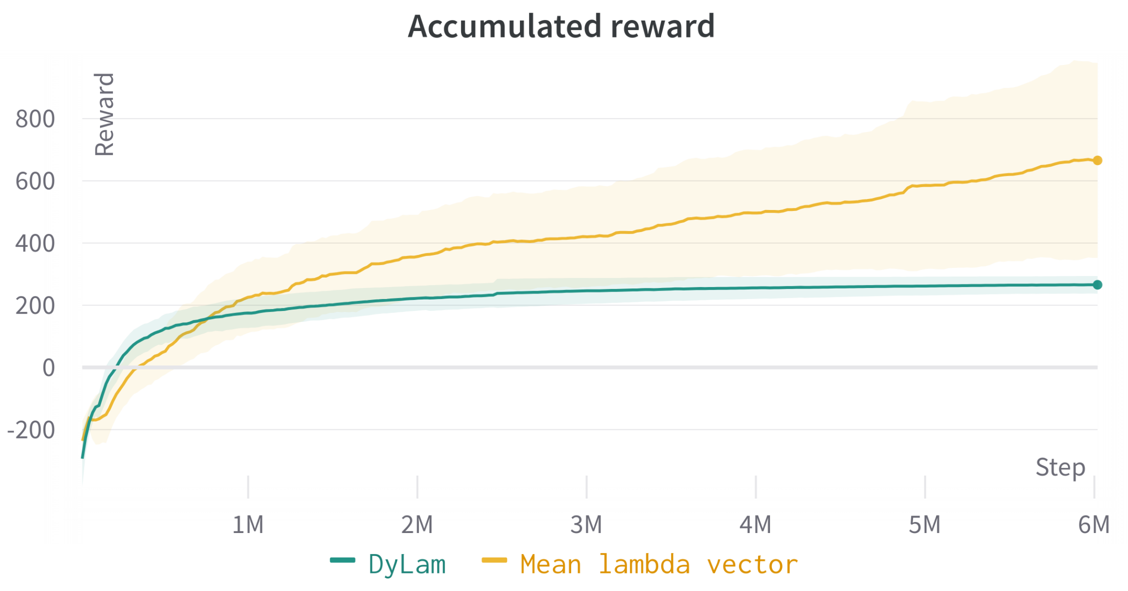
This chapter presented the experimental methodology and results obtained in the experiments with the methods proposed in this work and their respective baselines. We show the applicability of our methods and the impact of each part of them experimentally. The next chapter discusses our final considerations, regarding the contributions to science, the limitations of the method, its applicability, and future directions of research.

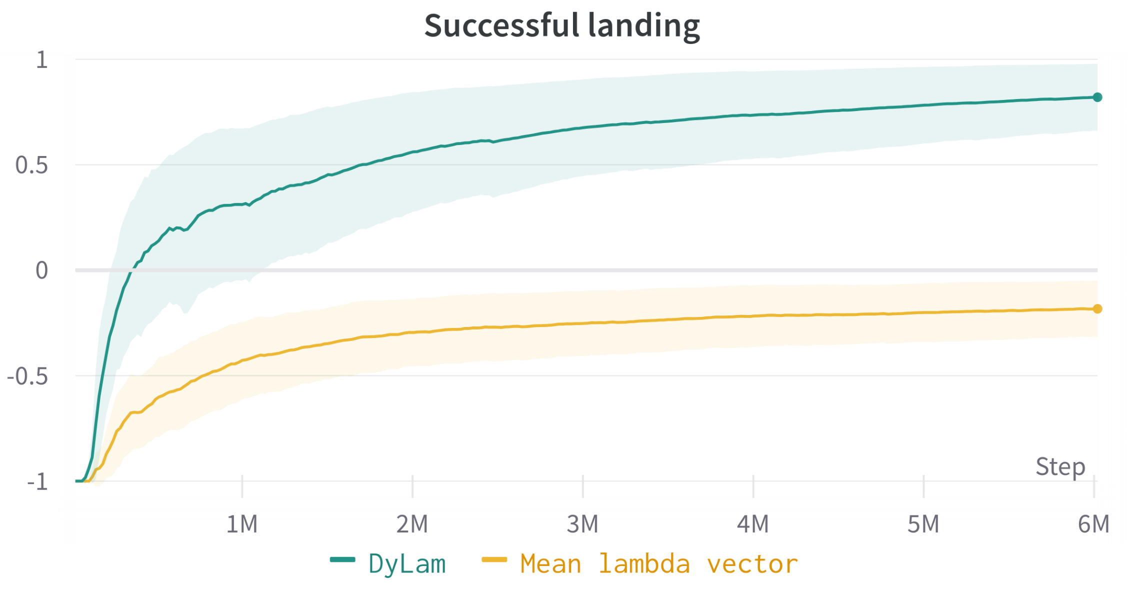
Figure 16 – Behavior of each λ weight component using the Dynamic experiment in Lunar Lander environment. The mean values are represented by the highlighted color and the standard error, the shaded one.



Source: Author

Figure 17 – Results of PPO on Lunar Lander environment comparing the best λ vector to our dynamic method. The mean values are represented by the highlighted color and the standard error, the shaded one.

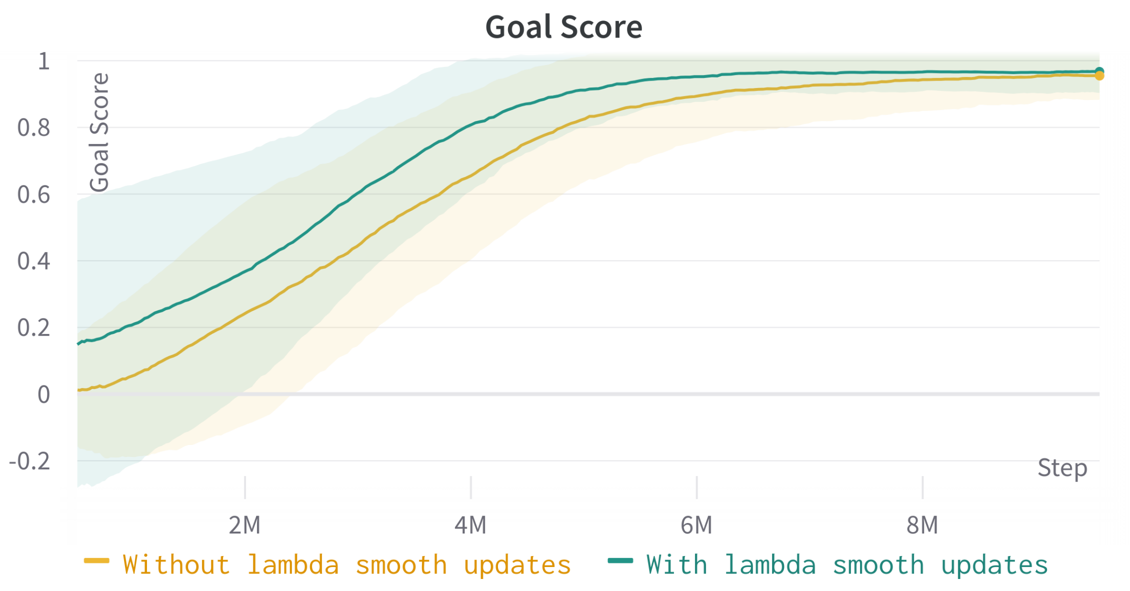




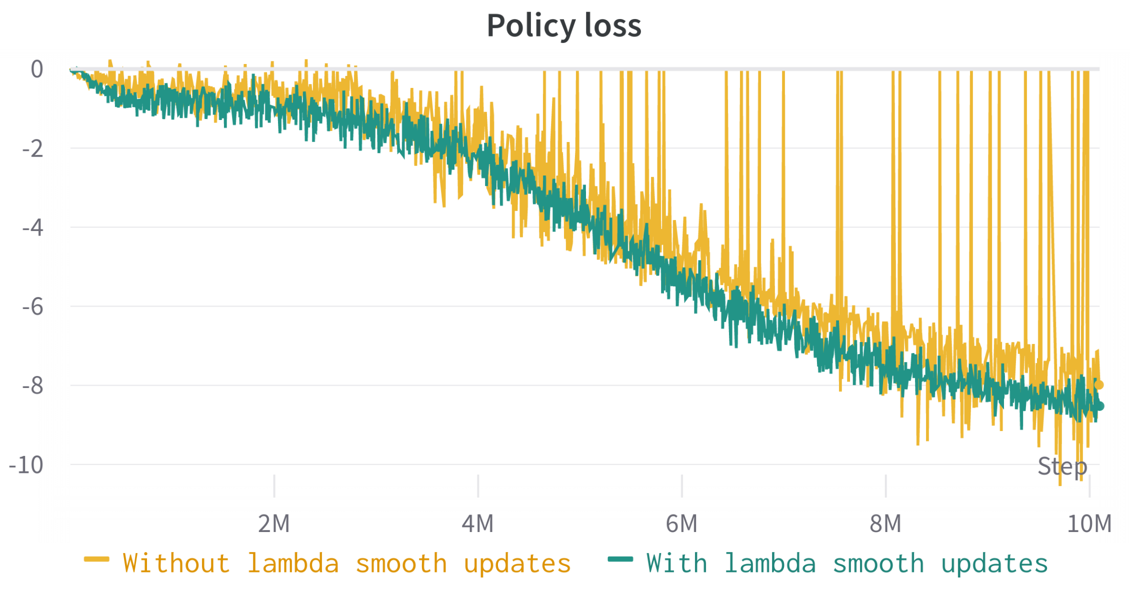
Source: Author

Figure 18 – Comparison of Goal Score results and policy loss in the VSS environment using our dynamic method with and without smooth updates.

(a) Comparing the results of DDPG using our dynamic method with and without smooth updates.



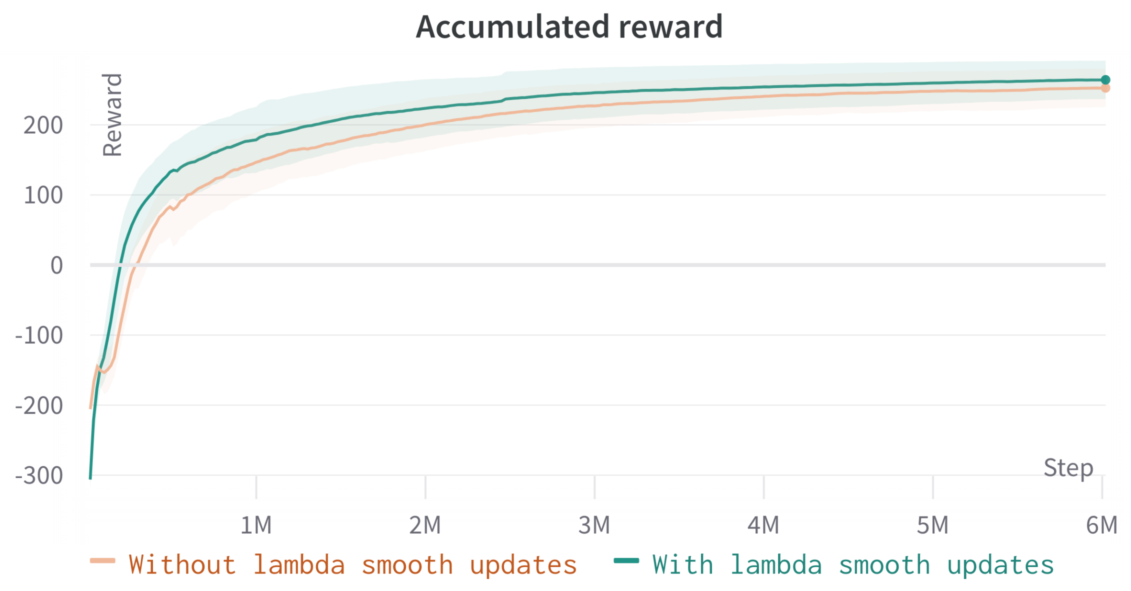
(b) Analysis of DDPG's policy loss using our dynamic method with and without smooth updates.



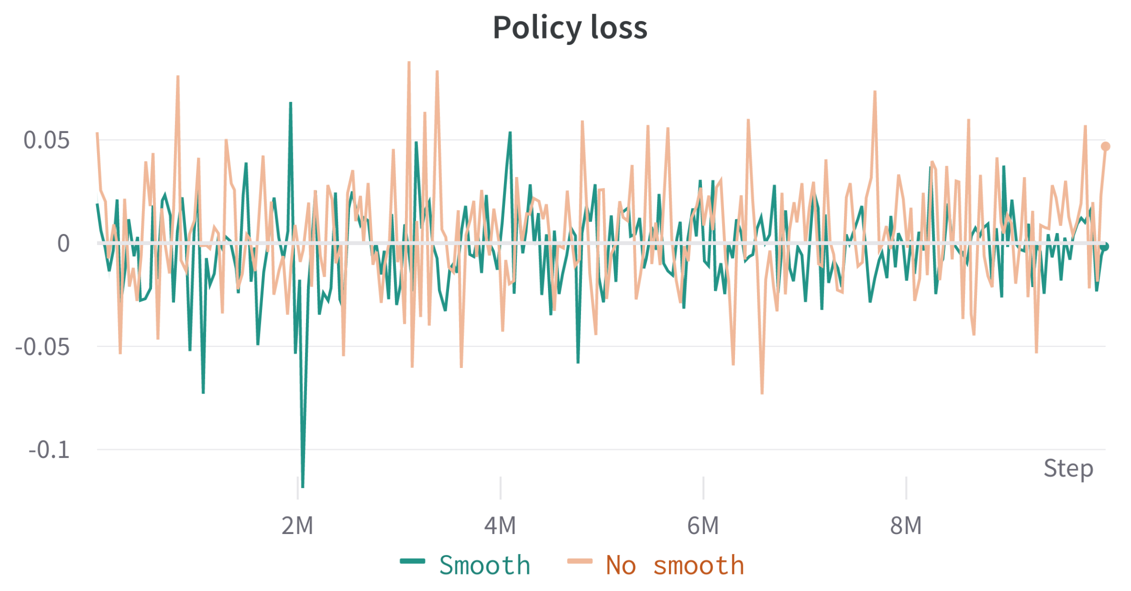
Source: Author

Figure 19 – Comparison of reward acumulated and policy loss in Lunar Lander environment using our dynamic method with and without smooth updates.

(a) Comparing the reward acumulated of PPO using our dynamic method with and without smooth updates.



(b) Analysis of PPO's policy loss using our dynamic method with and without smooth updates.



Source: Author

6 FINAL CONSIDERATIONS

This thesis presented a novel reward-shaping weighting method for Reinforcement Learning Actor Critic methods. We propose to automate the process of weighting potential-based reward functions. Our method uses the accumulated reward from the previous N episodes to evaluate the agent performance for each reward component. As the agent improves on a specific component, the weight of this component is decreased, so it prioritizes other reward components. This work also incorporates the idea of smooth updates presented in (LILLICRAP et al., 2019) into the weight updates. We analyze the importance of this addition when comparing the policy's loss with and without the smooth updates.

In this chapter, we draw the conclusions of this work. Section 6.1 discusses how well the objectives defined were achieved. Section 6.2 analyzes the cientific contributions of this Thesis. Section 6.3 discusses the limitations of the proposed method. Section 6.4 we present some ideas to apply our method. Finally, Section 6.5 presents future work that could naturally follow this research.

6.1 ANALYSIS OF THE PROPOSED MODEL

In the first chapter of this thesis, three main objectives were defined: 1: remove weights from the environment and decompose the reward, 2: automate the process of finding the perfect weighting for each component, and 3: smooth the policy updates due to the weight updates.

Theorem 4.1.1 proves that we can adapt the value function to remove the reward weighting from the environment, while mantaining the properties of the policy gradient theorem. This could be achieved by taking advantage of the proof presented by Russell and Zimdars (2003), that the decomposed form of value functions is equivalent to the composed one. Although encouraging results were obtained using grid search for finding adequate fixed weights for the reward components (Section 5.2.1), this method is exhaustive and in certain environments, a fixed set of weights may not be adequate througthout all the learning phases.

With Theorem 4.2.1, we prove that, under certain restrictions of the algorithm parameters, the dynamic weighting method is also compatible with the policy gradient theorem. The experiments using rSoccer and Lunar Lander environments, presented in Sections 5.2.2 and

5.2.3, show that our method can outperform the original methods. Specifically, in the Lunar Lander environment, we showed that using the dynamic method to find an adequated weighting can show leaks on supposed potential-based reward functions.

In a subsequent study (Section 5.2.5) we show the impact of the λ smooth updates on DyLam. Analyzing the impact of smoothing the λ updates for the loss functions of policies, we observe a significant reduction in the standard deviation of the loss values in DDPG, around 80%. In the experiments with PPO, the smoothing effect on the loss was not as clear, probably due to the Clipping and Normalization hyperparameters used.

6.2 CONTRIBUTIONS TO SCIENCE

The following list describes the main contributions of this Thesis:

- We introduced the idea of removing the reward's weights from the environment and adding it to train only the policy in Policy Gradient methods.
- We prove the equivalency of this method using the Policy Gradient theorem.
- We proposed and evaluated an automated method to adjust the weights for each reward component during the training.
- During the development of this work, we also created the rSoccer framework Martins et al. (2022), which was used to train the champion soccer agents of the IronCup and Latin America Robot Competition (LUI, 2021; PINTO, 2021).

This work had its primary studies on the rSoccer environment of this framework. We later used the Lunar Lander environment to validate the method using other PG methods. We also published other works related to the rSoccer using different frameworks. In Bassani et al. (2020), we describe a framework to study sim-to-real approach in VSS. Pena et al. (2020) describe a coach agent that defines each player's role, and the player has a deterministic policy controlling it. These two last works encouraged us to create the rSoccer framework, which is complete and follows the OpenAI Gym's pattern. We want to expand this thesis and apply it as a conference paper or journal. The following list describes the published works:

- BASSANI, H. F.; DELGADO, R. A.; JUNIOR, J. N. d. O. L.; MEDEIROS, H. R.; BRAGA, P. H.; **MACHADO, M. G.**;SANTOS,L. H.;TAPP,A. A framework for studying reinforcement learning and sim-to-real in robot soccer.arXiv preprint arXiv:2008.12624, 2020.

- PENA, C. H.; **MACHADO, M. G.**; BARROS, M. S.; SILVA, J. D.; MACIEL, L. D.; REN, T. I.; BARROS, E. N.; BRAGA, P. H.; BASSANI, H. F. An analysis of reinforcement learning applied to coach task in ieee very small size soccer. In: IEEE. 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE). [S.l.], 2020. p. 1–6.

- MARTINS, F. B.; **MACHADO, M. G.**; BASSANI, H. F.; BRAGA, P. H. M.; BARROS, E. S. rsoccer: A framework for studying reinforcement learning in small and very small size robot soccer. In: ALAMI, R.; BISWAS, J.; CAKMAK, M.; OBST, O. (Ed.). RoboCup 2021: Robot World Cup XXIV. Cham: Springer International Publishing, 2022. p. 165–176. ISBN 978-3-030-98682-7.

6.3 LIMITATIONS OF THE PROPOSED METHODS

We describe the limitations of this work in the following list:

- Our method works only for Policy Gradient methods.
- The agent designer has to know the problem well enough to understand each reward component range.
- We found no environment adapted to our method, so we had to adapt the environments of our experiments.
- Some of the OpenAI Gym environments have no documentation so we do not know what are the components of the reward shaping (BROCKMAN et al., 2016). Therefore it can be difficult to apply the proposed method.
- The number of enviroments evaluated so far is small, what casts doubts in the generality of the proposed method.
- We executed just three experiments with the same setup.

6.4 POSSIBLE APPLICATIONS

We can apply our method to multiobjective problems by training a value function in which each component corresponds to a specific subtask or objective. After training, we use the specific value function to train or fine-tune an agent to focus on one or some components of the tasks. In soccer environments, this is a great addition. For example, even with a great goalkeeper, it is crucial to train the best penalty shootout defender for real competitions.

6.5 FUTURE WORKS

We want to solve the sparse reward problem with DyLam. We expect that decomposing as much as possible a general agent, and using DyLam to train each sub-agent as a component, we can slowly generalize the model as we did in rSoccer and Lunar Lander. We also want to mitigate our limitation of knowing the environment well, investigating weight updates using functions similar to those proposed in (PERNY; WENG, 2010). A significant expansion of this work is to adapt our dynamic method for general RL and to validate it in environments used in (MNIH et al., 2013), (LILLICRAP et al., 2019), and (SCHULMAN et al., 2017).

REFERENCES

ANDRYCHOWICZ, M.; WOLSKI, F.; RAY, A.; SCHNEIDER, J.; FONG, R.; WELINDER, P.; MCGREW, B.; TOBIN, J.; ABBEEL, P.; ZAREMBA, W. *Hindsight Experience Replay*. 2018.

BASSANI, H. F.; DELGADO, R. A.; JUNIOR, J. N. d. O. L.; MEDEIROS, H. R.; BRAGA, P. H.; MACHADO, M. G.; SANTOS, L. H.; TAPP, A. A framework for studying reinforcement learning and sim-to-real in robot soccer. *arXiv preprint arXiv:2008.12624*, 2020.

BRITZ, D. *reinforcement-learning*. [S.l.]: GitHub, 2019. <https://github.com/dennybritz/reinforcement-learning>.

BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

BRYS, T.; HARUTYUNYAN, A.; VRANCX, P.; TAYLOR, M. E.; KUDENKO, D.; NOWÉ, A. Multi-objectivization of reinforcement learning problems by reward shaping. In: IEEE. *2014 international joint conference on neural networks (IJCNN)*. [S.l.], 2014. p. 2315–2322.

BRYS, T.; HARUTYUNYAN, A.; VRANCX, P.; NOWé, A.; TAYLOR, M. E. Multi-objectivization and ensembles of shapings in reinforcement learning. *Neurocomputing*, v. 263, p. 48–59, 2017. ISSN 0925-2312. Multiobjective Reinforcement Learning: Theory and Applications. Available at: <https://www.sciencedirect.com/science/article/pii/S0925231217310962>.

DEGRIS, T.; PILARSKI, P. M.; SUTTON, R. S. Model-free reinforcement learning with continuous action in practice. In: IEEE. *2012 American Control Conference (ACC)*. [S.l.], 2012. p. 2177–2182.

FANG, M.; ZHOU, C.; SHI, B.; GONG, B.; XU, J.; ZHANG, T. Dher: Hindsight experience replay for dynamic goals. In: *International Conference on Learning Representations*. [S.l.: s.n.], 2018.

FANG, M.; ZHOU, T.; DU, Y.; HAN, L.; ZHANG, Z. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems*, v. 32, 2019.

GAMBLE, C.; GAO, J. *Safety-first AI for autonomous data centre cooling and industrial control*. 2018. Urlhttps://www.deepmind.com/blog/safety-first-ai-for-autonomous-data-centre-cooling-and-industrial-control.

GOODFELLOW, I. J.; BULATOV, Y.; IBARZ, J.; ARNOUD, S.; SHET, V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

HENDERSON, P.; ISLAM, R.; BACHMAN, P.; PINEAU, J.; PRECUP, D.; MEGER, D. Deep reinforcement learning that matters. In: *Proceedings of the AAAI conference on artificial intelligence*. [S.l.: s.n.], 2018. v. 32, n. 1.

HU, Y.; WANG, W.; JIA, H.; WANG, Y.; CHEN, Y.; HAO, J.; WU, F.; FAN, C. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, v. 33, p. 15931–15941, 2020.

HUANG, S.; DOSSA, R. F. J.; YE, C.; BRAGA, J. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. 2021.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285, 1996.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. *Continuous control with deep reinforcement learning*. 2019.

LUI, C. *Futebol de Robôs: Robôcin (UFPE) É campeã da iron cup*. 2021. Available at: <https://www.leiaja.com/tecnologia/2021/03/02/futebol-de-robos-robocin-ufpe-e-campea-da-iron-cup/>.

MANNION, P.; DEVLIN, S.; MASON, K.; DUGGAN, J.; HOWLEY, E. Policy invariance under reward transformations for multi-objective reinforcement learning. *Neurocomputing*, v. 263, p. 60–73, 2017. ISSN 0925-2312. Multiobjective Reinforcement Learning: Theory and Applications. Available at: <https://www.sciencedirect.com/science/article/pii/S0925231217311037>.

MARTINS, F. B.; MACHADO, M. G.; BASSANI, H. F.; BRAGA, P. H. M.; BARROS, E. S. rsoccer: A framework for studying reinforcement learning in small and very small size robot soccer. In: ALAMI, R.; BISWAS, J.; CAKMAK, M.; OBST, O. (Ed.). *RoboCup 2021: Robot World Cup XXIV*. Cham: Springer International Publishing, 2022. p. 165–176. ISBN 978-3-030-98682-7.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. *Playing Atari with Deep Reinforcement Learning*. 2013.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G. et al. Human-level control through deep reinforcement learning. *nature*, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

NG, A. Y.; HARADA, D.; RUSSELL, S. Policy invariance under reward transformations: Theory and application to reward shaping. In: *Icml*. [S.l.: s.n.], 1999. v. 99, p. 278–287.

OPENAI. *OpenAI Five*. 2018. <https://blog.openai.com/openai-five/>.

PENA, C. H.; MACHADO, M. G.; BARROS, M. S.; SILVA, J. D.; MACIEL, L. D.; REN, T. I.; BARROS, E. N.; BRAGA, P. H.; BASSANI, H. F. An analysis of reinforcement learning applied to coach task in ieee very small size soccer. In: IEEE. *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. [S.l.], 2020. p. 1–6.

PERNY, P.; WENG, P. On finding compromise solutions in multiobjective markov decision processes. In: *ECAI 2010*. [S.l.]: IOS Press, 2010. p. 969–970.

PINTO, A. H. M. *RoboCup Brasil Wiki*. 2021. Available at: <https://robocup.org.br/wiki/doku.php?id=very_small>.

POPOV, I.; HEESS, N.; LILLICRAP, T.; HAFNER, R.; BARTH-MARON, G.; VECERIK, M.; LAMPE, T.; TASSA, Y.; EREZ, T.; RIEDMILLER, M. *Data-efficient Deep Reinforcement Learning for Dexterous Manipulation*. 2017.

RUSSELL, S. J.; ZIMDARS, A. Q-decomposition for reinforcement learning agents. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. [S.l.: s.n.], 2003. p. 656–663.

SCHULMAN, J.; LEVINE, S.; ABBEEL, P.; JORDAN, M.; MORITZ, P. Trust region policy optimization. In: PMLR. *International conference on machine learning*. [S.l.], 2015. p. 1889–1897.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

SHELTON, C. R. Importance sampling for reinforcement learning with multiple objectives. 2001.

SILVER, D.; LEVER, G.; HEESS, N.; DEGRIS, T.; WIERSTRA, D.; RIEDMILLER, M. Deterministic policy gradient algorithms. In: PMLR. *International conference on machine learning*. [S.l.], 2014. p. 387–395.

SINGH, S.; BARTO, A. G.; CHENTANEZ, N. *Intrinsically motivated reinforcement learning*. [S.l.], 2005.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

SUTTON, R. S.; PRECUP, D.; SINGH, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, Elsevier, v. 112, n. 1-2, p. 181–211, 1999.

TESAURO, G. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , v. 6, n. 2, p. 215–219, 1994.

UHLENBECK, G. E.; ORNSTEIN, L. S. On the theory of the brownian motion. *Physical review*, APS, v. 36, n. 5, p. 823, 1930.

WANG, X.; CHEN, Y.; ZHU, W. *A Survey on Curriculum Learning*. arXiv, 2020. Available at: <https://arxiv.org/abs/2010.13166>.