DyLam: A Dynamic Reward Weighting Framework for Reinforcement Learning Algorithms

Extended Abstract

Mateus Machado
Centro de Informática - UFPE
Recife, Brazil
mgm4@cin.ufpe.br

Hanseclever Bassani
Centro de Informática - UFPE
Recife, Brazil
hfb@cin.ufpe.br

ABSTRACT

Creating a Reinforcement Learning (RL) training environment is a known difficulty in the field. **When the reward is a composition of different signals**, defining the weights for each signal represents the **learning** curricula the agent will follow during training. The process of trying new weights endures until the agent reaches the objective of the environment. We present DyLam, a robust automated **self-curriculum learning framework for RL algorithms**. **By requiring only an estimate** of the theoretical maximum and minimum of each **reward component**, DyLam can **adjust** its weights **dynamically** during training, depending on which signal needs to be optimized at each training stage. We **show experimentally the robustness** of this method compared to **the state-of-the-art in the Lunar Lander discrete control benchmark context**.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**.

KEYWORDS

Reinforcement Learning, Curriculum Learning, Multi-Objective Optimization

ACM Reference Format:
Mateus Machado and Hanseclever Bassani. 2025. DyLam: A Dynamic Reward Weighting Framework for Reinforcement Learning Algorithms: Extended Abstract. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 3 pages.

1 INTRODUCTION

The reward shaping technique, introduced to address the challenge of sparse rewards, involves adding supplementary reward signals aligned with **sub-tasks** or **intermediate objectives** within a Reinforcement Learning (RL) environment [6]. Each reward component contributes a weighted value to the final reward signal, represented by the weights λ. These weights influence the agent's focus, dictating which aspects of the task should receive more attention during the whole training process, effectively shaping a static curriculum the agent follows [8].

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Consider a soccer environment where the agent's final goal is to score. Before achieving this, the agent must learn sub-skills, such as moving towards the ball, carrying it, and shooting accurately. A sensible curriculum would prioritize simpler tasks like movement and ball control, before advancing to more complex behaviors like shooting and scoring. If these tasks are incorrectly weighted, an agent could become proficient at carrying the ball but fail to develop the capability to score.

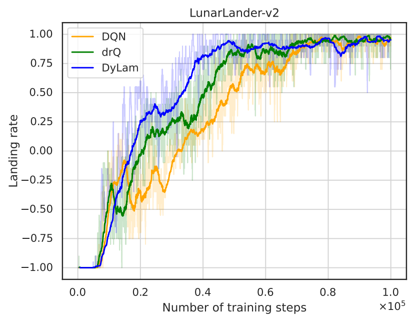
In this paper, we propose a novel RL framework called the Dynamic Lambda Reward Weighting Function (***DyLam***), which dynamically adjusts the weights of stratified value functions throughout training. DyLam is designed to address the challenge of balancing multiple reward components by learning optimal value functions for each distinct component and adapting policies to reflect changing priorities. By continuously adjusting the λ weights, DyLam shifts focus towards under-optimized components, promoting a more efficient learning process. Positioned at the intersection of single-objective and multi-objective problems, DyLam offers a flexible approach that integrates automated curriculum learning with dynamic reward balancing.

2 DYNAMIC LAMBDA REWARD WEIGHTING FUNCTION

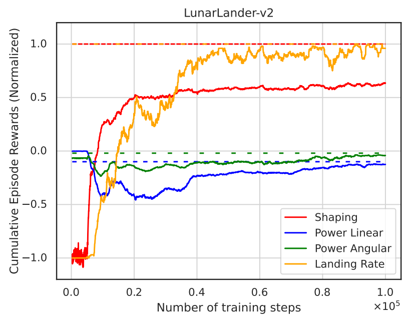
In this section, we formalize our approach called Dynamic Lambda Reward Weighting Function (***DyLam***). We denote $\lambda \in \mathbb{R}^n$ as the vector of weights associated with the vector $r : S \times A \times S - > \mathbb{R}^n$. In essence, DyLam stands as an automated self-curriculum learning framework, dynamically adapting weights according to environmental rewards. Our approach steers the agent's attention towards the most crucial reward component in the current context throughout the training process. As the easiest components are optimized, the method gradually reduces their weights and increases the weights of other components yet to be optimized.

Firstly, **the reward function is decomposed into components that should vary in similar ranges**. For example, we can decompose the reward function of a grid world environment into two functions of x and y distances to the objective [1], each ranging from zero to one. It is recommended to have all reward components varying in similar scales so that the method applies similar levels of importance for them, differing only in their learning difficulty, thus prioritizing learning the easiest ones first.

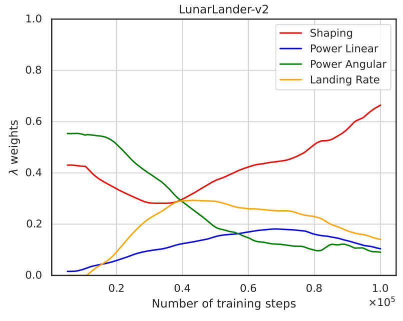
The second step is to **decompose the value function into a list of Q-values, each associated with one reward component**. [7] proof of decomposition of value functions is crucial to this part of our work, as it ensures it is a sound optimization objective for



(a)



(b)



(c)

Figure 1: (a): Training results for the Landing Rate using DQN, DQN+drQ, and DQN+DyLam methods. The shaded areas indicate the range of observed values, while the thick curves represent the moving average over 100 time steps. (b): Evolution of rewards (moving average of 100 Episodes). Traced lines represent the set practical maximum of each reward. (c): λ weights fit.

every MDP. Our value functions then turn to regression models that fit each expected reward component accumulated without applying weights.

The last step varies when using the local Q-learning or Actor-Critic frameworks. The value-based approach will use the weights λ only in the policy step with a dot product in the Q-values: $Q(s,a) = \sum_{i=0}^{n} \lambda_i Q_i(s,a)$. While in the Policy Gradient, we reorganize the policy training process to multiply the value function by the weights λ found using a weight adjustment method.

Using the static weighting function, we proved that the Critic becomes a regression model 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: $\overline{R_t^i} = R_t^i + \tau_\lambda(R_{t-1}^i - R_t^i)$; $\zeta_i(\overline{R_t^i}) = \frac{R_{max}^i - \overline{R_t^i}}{R_{max}^i - R_{min}^i}$:

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

where, R_t^i is the moving average accumulated return for the i-th component, τ_λ 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 Deep Deterministic Policy Gradient (DDPG) [4]. 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.

3 EXPERIMENTS

We evaluate our framework in the LunarLander-v2 scenario, modifying only the reward ranges, normalizing each component by its theoretical **step (not cumulative)** maximum and minimum, and applied the Multi-Objective Reinforcement Learning (MORL) learning framework from MO-Gymnasium [2].

A key aspect of our experimental setup is that we compare the proposed method against the environment's baseline only[1]. We evaluate Deep Q-Networks (DQN) [5], drQ (an adapted form of the decomposed Q-Learning of [7] using neural networks), and DQN-DyLam. As we explain in Section 2, **we scale all reward components to have similar ranges**. Due to this adjustment, we could not compare cumulative episode rewards directly, as the reward scales differed from the original benchmarks [3]. In the baseline and drQ methods, all reward components are treated with equal importance, i.e. $\lambda_i = 1 \div \#rewards$. Importantly, this changes and setup did not affect the baselines' original performances when analyzing the proposed metrics.

As shown in Fig. 1a, DyLam's agent outperforms the baselines in early training, achieving a 90% landing success rate by halfway through the training period, while, particularly, DQN reaches this rate only after 75% of training. Upon analyzing Fig. 1b, we observe that DyLam's agent incurs more penalties during early training. This behavior arises because the agent prioritizes learning how to land first, as reflected in Fig. 1c, before optimizing for landing efficiency—an intuitive approach akin to "first focus on landing, then on landing efficiently."

4 CONCLUSION

This extended abstract introduced the Dynamic Lambda Reward Weighting Function (DyLam) algorithm, an automated curriculum learning approach for reinforcement learning methods. By decomposing the reward function into components of comparable scales, DyLam dynamically adjusts weights based on the agent's performance for each reward component, effectively prioritizing what the agent needs to learn at each stage of the training. The experiments conducted in the LunarLander-v2 environment substantiate the effectiveness of DyLam's automated self-curricula learning approach. The results not only reveal its superior performance compared to the baseline but also highlight its potential to provide insight into the learning process of the model over time.

[1]Code available in: https://github.com/goncamateus/dylam

REFERENCES

[1] Tim Brys, Anna Harutyunyan, Peter Vrancx, Matthew E Taylor, Daniel Kudenko, and Ann Nowé. 2014. Multi-objectivization of reinforcement learning problems by reward shaping. In *2014 international joint conference on neural networks (IJCNN)*. IEEE, 2315–2322.

[2] Florian Felten, Lucas N. Alegre, Ann Nowé, Ana L. C. Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno C. da Silva. 2023. A Toolkit for Reliable Benchmarking and Research in Multi-Objective Reinforcement Learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*. NeurIPS, Cham.

[3] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. 2022. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research* 23, 274 (2022), 1–18. http://jmlr.org/papers/v23/21-1342.html

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs.LG]

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs.LG]

[6] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, Vol. 99. 278–287.

[7] Stuart J Russell and Andrew Zimdars. 2003. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 656–663.

[8] Xin Wang, Yudong Chen, and Wenwu Zhu. 2020. A Survey on Curriculum Learning. https://doi.org/10.48550/ARXIV.2010.13166