Proximal Policy Distillation in Reinforcement Learning

Andrea Gaudino
s346119@studenti.polito.it

Lorenzo Grivet Talocia
s346559@studenti.polito.it

Elena Rossi
s349342@studenti.polito.it

Andrea Spinnicchia
s344912@studenti.polito.it

Politecnico di Torino

Abstract

In this paper, we present Proximal Policy Distillation (PPD), a Reinforcement Learning method that combines the stability of Proximal Policy Optimization (PPO) with a distillation loss. Starting from a robust teacher policy trained with PPO on a distribution of randomized environments, the method performs policy distillation onto three different student network sizes: smaller, identical and larger than the teacher network. Our objective is to compare PPD with standard PPO (without the distillation loss) to assess the impact of incorporating teacher guidance through distillation. The full source code is available at this Github repository: https://github.com/andreaGaudino/ML_project

1. Introduction

Reinforcement Learning (RL) is a subfield of machine learning focused on training agents to make sequential decisions by interacting with a dynamic environment. The agent aims to learn a policy that provides a guideline on what is the optimal action to take in a certain state with the objective of maximizing a cumulative reward over time. In contrast to supervised learning, RL does not rely on labeled input-output pairs but instead learns optimal behaviours through exploration and feedback from the environment. RL is also different from unsupervised learning, which aims to find hidden structures in collections of unlabeled data [1].

In this work, we describe and compare different RL algorithms: REINFORCE, Actor-Critic and Proximal Policy Optimization. We conducted our experiments using an environment provided by the Mujoco library. In particular, we trained our models on a custom version of the Gym Hopper on which we performed Uniform Domain Randomization. Lastly, we propose a possible extension by presenting the Proximal Policy Distillation algorithm.

2. Related work

2.1. REINFORCE and Actor-Critic

REINFORCE and Actor-Critic are two algorithms based on the policy gradient, a method that targets at modelling and optimizing the policy directly in order to obtain the best reward [2].

The REINFORCE algorithm updates policy parameters based on returns estimated from complete episodes using Monte Carlo sampling, taking into account a discount factor to weigh future rewards. Its effectiveness derives from the fact that the expected value of the discounted sampled gradient equals the true policy gradient. However, this method is known to suffer from high variance in gradient estimates. In order to mitigate this phenomenon, we introduced a fixed baseline and we subtracted it from the rewards. This approach reduces variance without introducing bias, which should indirectly lead to a more stable and smoother training process.

The Actor-Critic method combine two key components: an Actor, which updates the policy, and a Critic, which estimates value functions to guide the policy update. The Critic evaluates how good the current policy is by estimating either the state-value function or the action-value function. This value estimate helps reduce the variance of the policy gradient updates and provides more stable learning compared to pure Monte Carlo methods like REINFORCE. The Actor then adjusts the policy parameters in the direction suggested by the Critic to improve performance.

2.2. Proximal Policy Optimization

Proximal Policy Optimization (PPO) refers to a set of policy optimization techniques that update policies through repeated stochastic gradient ascent over several epochs. By clipping the ratio between the old and the new policy to a specific range, PPO ensures that the new policy is not too far

from the old one. This mechanism helps maintain the balance between exploration and convergence and contributes to the robustness of the learning process. Each time the model is updated, PPO goes through the training data multiple epochs, which helps it learn more effectively. It is versatile enough to be used in a variety of contexts, including models that share parameters between the policy and value functions, and it typically yields strong empirical results [3].

2.3. Uniform Domain Randomization

Uniform Domain Randomization (UDR) is a technique that introduces controlled variability in the parameters of the environment. Specifically, we applied UDR to the link masses of the robot, excluding the torso, which are randomized at the beginning of each episode by sampling from manually designed uniform distributions. These distributions are chosen to introduce variability in the robot's dynamics, thus encouraging the policy to learn behaviours that generalize across a range of environments and increasing the policy robustness to mass perturbations [4].

2.4. Evaluation of results

As shown in Figure 1, we analyse the effectiveness of the introduction of a baseline with respect to the standard REINFORCE algorithm. With a baseline value of b = 20, we observe a clear reduction in variance, but in terms of rewards, it makes almost no difference. While a fixed baseline provides some benefits, it represents only a simple variance reduction strategy. More advanced approaches, such as using state-dependent baselines, learned value functions or running averages of returns, are typically more effective, as they adapt to the learning dynamics over time.

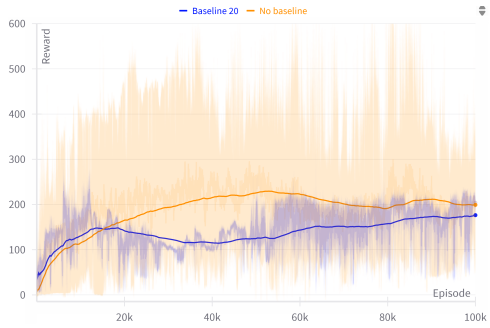

Figure 1. REINFORCE training rewards

The Actor-Critic algorithm achieves higher total rewards and improved training stability, compared to REINFORCE. This improvement is due to the use of the Critic, which reduces the variance of the policy gradient estimates, leading to more efficient and stable learning [5].

To evaluate the generalization capability of PPO, we have to distinguish between the source and target domains. The source domain offers a controlled and efficient training setup, allowing rapid policy development. In contrast, the target domain presents a more realistic scenario where the final policy must perform reliably. In our case, the source environment simulates an imprecise setting by reducing the torso mass by 30%, introducing a domain shift. We report results for three key "training→test" configurations: source→source, representing performance in the training domain; source→target, serving as a lower bound for sim-to-sim transfer; and target→target, as an upper bound reference for ideal performance in the target domain.

To analyse the effectiveness of UDR, the agent was first trained in the source environment and evaluated in the target environment. Although the policy showed some transferability, its performance was significantly lower than in the source→source and target→target cases, indicating limited robustness to unseen mass configurations. To address this problem, we randomly scaled all body masses, except the torso, within a uniform range of [0.7, 1.3] at each training episode. This approach led to improved returns, particularly in the source→target evaluation, suggesting that the policy adapted more effectively to variability in mass and generalized better to the target domain, as shown in Figure 2.

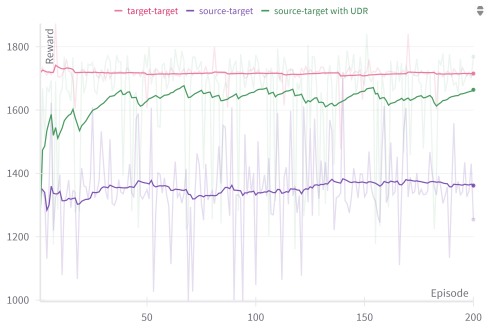

Figure 2. Comparison between training PPO with UDR (green line) and without UDR (purple line)

Table 1 reports the average training runtimes over 5 random seeds, for each of the three algorithms evaluated. PPO clearly outperforms both REINFORCE and Actor-Critic, not only in terms of rewards but also in computational efficiency.

Algorithm	Time per Episode [s/ep]
REINFORCE	$5.4 \cdot 10^{-2}$
Actor-Critic	$1.6 \cdot 10^{-1}$
PPO	$1.2 \cdot 10^{-3}$

Table 1. Average runtimes per algorithm

In conclusion, we compare the best-performing version of each algorithm: REINFORCE with a baseline

equal to 20, Actor-Critic and source→target PPO combined with UDR. Figure 3 shows the rewards obtained by agents trained with these methods, evaluated over 200 test episodes. The results are consistent with expectations, with PPO achieving the highest returns and demonstrating superior generalization capabilities.

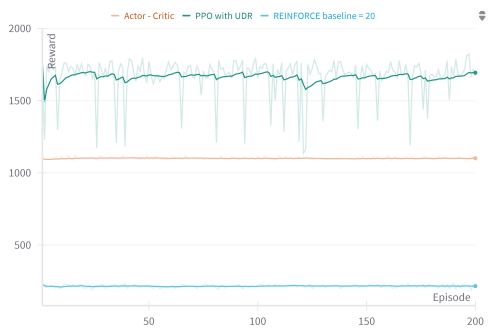

Figure 3. Test returns per algorithm

3. Project extension

3.1. Introduction

In the second part of the project, we explore Proximal Policy Distillation (PPD), an extension of PPO that integrates policy distillation to enhance learning efficiency. PPD combines the stability of PPO with a distillation loss, allowing a student agent to learn from a pre-trained teacher policy. The teacher serves as a skill prior, transferring useful sub-task behaviours that guide the student toward solving more complex tasks.

3.2. Policy Distillation

Policy distillation assumes the presence of a teacher agent trained to solve a RL task within the Markov Decision Process (MDP) framework, here obtained with PPO. A student policy is then randomly initialized and trained by both interacting with the environment and distilling knowledge from the teacher. In this context, policy distillation acts as a regularizer that biases the student toward promising behaviours without constraining it to merely imitate the teacher. The objective is to accelerate learning and ensure robustness, even in the presence of an imperfect teacher. Potentially, the student can surpass the teacher by effectively combining both imitation and exploration [6].

3.3. Implementation

The distillation loss is defined as the Kullback-Leibler (KL) divergence between the action distributions of the teacher and student policies (Equation 1). To enhance training stability, we applied a clipping mechanism to the KL divergence, enforcing a proximal constraint ϵ similar to that used in PPO-Clip. This approach helps balance the ratio between new and old policy probabilities, to ensure stable optimization.

$$L_{\text{PPD}}(s,a,\theta) = \text{KL}(\pi_{\text{teacher}}(\cdot \mid s_t) \parallel \pi_\theta(\cdot \mid s_t)) \times \max(\rho_t(\theta), 1-\epsilon) \quad (1)$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$.

During each training mini-batch $\mathcal{D}_k^m$, the student policy parameters θ_{k+1} are updated by maximizing a combination of the PPO policy loss, entropy loss and the distillation loss.

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k^m|} \sum_{(s,a)\in\mathcal{D}_k^m} \Big[L_{\text{PPO}}(s,a,\theta) + \alpha L_{\text{entropy}}(s,a,\theta) - \lambda L_{\text{PPD}}(s,a,\theta) \Big] \quad (2)$$

Where the KL-based term $L_{\text{PPD}}(s,a,\theta)$ encourages the student policy to remain close to a target distribution defined by the teacher. This loss is weighted by a configurable coefficient λ, which can be set as a constant and integrated into the PPO training loop. The PPO clipped loss term is defined as:

$$L_{\text{PPO}}(s,a,\theta) = \min\Big(\rho(\theta)\cdot\hat{A}(s,a),\ g(\epsilon,\hat{A}(s,a))\Big) \quad (3)$$

where:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A < 0 \end{cases} \quad (4)$$

and $\hat{A}(s,a)$ is the estimate of the advantage function.

This computation of the total loss allows the student to balance learning from both its own experience and the teacher's guidance.

3.4. Teacher

The teacher policy was trained using the PPO algorithm, as implemented in the *Stable-Baselines3* (SB3) library [7]. To ensure efficient and robust learning, we instantiated 18 parallelized and normalized Mujoco environments based on the Custom Hopper environment, using vectorized normalization to stabilize both observations and rewards. The teacher's neural network architecture consists of separate networks to represent the policy and value functions, using a MultiLayer Perceptron (MLP) with two hidden layers of 256 units and ReLU activation functions. Training was performed for 10 million steps and over multiple random seeds to ensure reproducibility and statistical robustness. Once completed, the trained teacher models were saved for subsequent use in the distillation process. Our goal is to transfer the knowledge from the teacher policy to the student while training the student on the same teacher's task [8].

3.5. Distillation

We applied policy distillation to train student models with three different network architectures, using multiple random seeds to ensure robustness. Distillation was performed over 2 million environment steps across all configurations. Table 2 shows the dimensions of the different networks in terms of layers and parameters. We decided to maintain two hidden layers for each network.

Size	Policy	Value function	Parameters
Smaller	[128, 128]	[64, 64]	18k (0.25x)
Same	[256, 256]	[256, 256]	70k (1x)
Larger	[512, 512]	[512, 512]	270k (4x)

Table 2. Student networks

A smaller architecture uses fewer neurons per layer, resulting in faster training, lower memory usage and potentially better generalization due to a reduced risk of overfitting. However, its limited capacity may limit performance on more complex tasks. On the other hand, a larger architecture has more parameters and greater expressive power, which can lead to higher accuracy, but at the cost of slower training and a higher risk of overfitting.

In our approach, we distilled only the policy network from the teacher, since it governs action selection and plays a key role in behaviour transfer. The value function, by contrast, was not distilled, as it is strictly linked to the direct interaction with the environment. The student must learn this component independently, guided by its own experience under a policy similar to the teacher's.

Our implementation extends the SB3 PPO training script, overriding the train method to incorporate the additional distillation loss. This loss, computed as the KL divergence between the action distributions of the teacher and student on the same observations (Equation 1), complements the standard PPO objectives. As the teacher, the student agent interacts with 18 parallelized and normalized Mujoco environments during training. To monitor progress, we periodically evaluated the student's performance in a separate validation environment. Both the student and teacher were assessed after training to measure the effectiveness of the distillation process.

3.6. Evaluation

3.6.1 Training

The evaluation of our PPD implementation demonstrates several important findings regarding the effectiveness of policy distillation in RL. We first examine the training curves during distillation to evaluate the sample efficiency of the different methods. In Figure 4 we report the comparison of the training result between the teacher model and the three different student sizes.

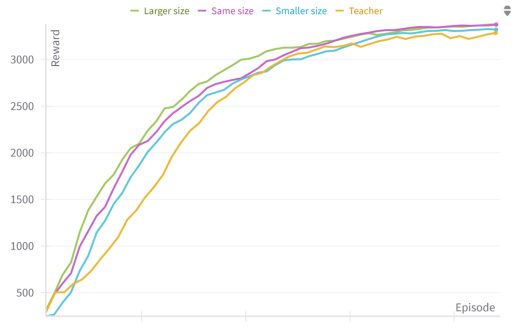

Figure 4. Training curves of the teacher model and of the three different students

The distillation algorithm clearly yields better performance, particularly in the first half of the training loop: all student models outperform the teacher in terms of sample efficiency, indicating that the distilled policies benefit from the guidance of the teacher during training. This supports the hypothesis that policy distillation can serve as a regularizer that biases the student toward effective behaviours without strictly constraining it to imitation, thus allowing for exploration and optimization beyond the teacher's solution [9].

Furthermore, the difference among the three student models is noticeable: among the students, the larger model achieved the highest reward trajectory in the early and middle stages of training, suggesting that model capacity plays a role in how well the student can absorb and generalize the teacher's behaviour. However, the performance of the other two models is still competitive. As the number of episodes increases, the difference between the models tends to narrow, reaching the maximum reward at the end of the training.

Our experiments also highlight the impact of the distillation coefficient λ in shaping the learning dynamics of PPD. As shown in Figure 5, increasing λ accelerates learning by assigning more weight to the distillation loss, thereby encouraging the student to stay closer to the teacher's policy. In particular, students trained with higher values of the coefficient ($\lambda = 5$) consistently learn more effective policies than those with lower values ($\lambda = 0.5$). This behaviour aligns with the expectations: a higher λ prioritizes the distillation loss against the PPO loss, reducing the student's deviation from the teacher's actions and stabilizing training. On the other hand, lower λ values lead to slower and less stable convergence, ultimately resulting in lower final rewards. These results confirm that the distillation coefficient contributes significantly to the student's performance, improving standard PPO and demonstrating the utility of policy distillation in learning.

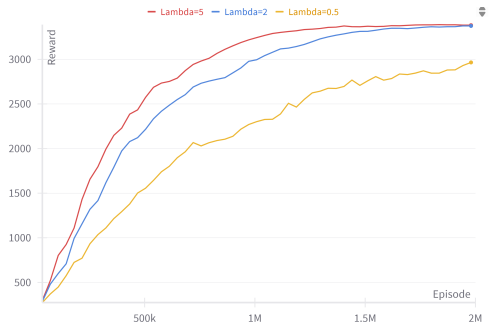

Figure 5. Training curves of same-size model for each value of λ

3.6.2 Test

The evaluation of the test results reveals a different trend compared to the training phase. In the test environment, as shown in Table 3, the teacher model achieves slightly higher mean rewards than all three student models, and the differences among the student models themselves are minimal, with their average performances nearly overlapping. This suggests that, despite the advantages observed during training, particularly in terms of sample efficiency and early learning, the benefits of distillation do not translate into a significant performance gap in the final test outcomes for this specific task.

Model	Mean Reward
Teacher	3399.49
Smaller	3374.59
Same	3391.71
Larger	3397.43

Table 3. Mean model reward of the models over 200 steps test

A likely explanation for this result is the simplicity of the chosen environment: the Mujoco custom Hopper features a relatively low-dimensional action space and limited behavioural complexity, which may restrict the potential for the student models to surpass the teacher or to exhibit substantial differences based on model capacity. In such a setting, the teacher's policy already captures most of the relevant behaviours, leaving little room for the students to further improve or differentiate themselves through distillation. It is reasonable to hypothesize that in more complex absolute reward values environments, with higher-dimensional state and action spaces or more challenging dynamics, the advantages of policy distillation, such as improved generalization and the ability to surpass the teacher, would become more pronounced, and differences among student architectures would be more evident.

Finally, we compare the performance of the best performing student model, the larger PPD architecture, with that of a standard PPO model. As shown in Figure 6, the PPD-based student consistently outperforms the baseline PPO model, in terms of absolute reward values and stability, across evaluation episodes, despite both models being evaluated under identical conditions.

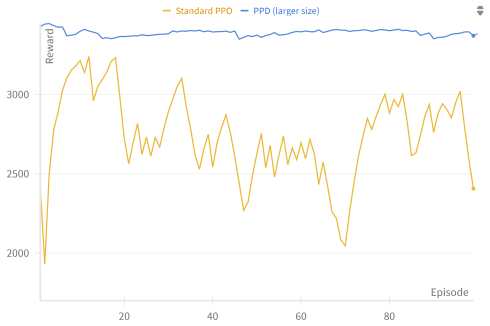

Figure 6. Comparison between PPD and standard PPO

3.7. Conclusion

This report has explored different RL algorithms, demonstrating that simple baselines and Actor-Critic methods improve stability, while UDR enhances robustness to domain shifts. Building on these insights, we introduced Proximal Policy Distillation (PPD), which combines the robustness of PPO with a KL-based distillation loss to transfer knowledge from a pre-trained teacher to student agents of different sizes. Our results highlight the benefits of incorporating teacher guidance during training, which enables the student to converge to strong policies more efficiently. While in the test setting the student does not surpass the teacher for this particular environment, it clearly demonstrates superior generalization compared to a non-distilled agent. These findings suggest that PPD could offer a promising direction for improving RL efficiency, especially in settings where training from scratch is expensive or where high-performing teacher policies are available. Future research should focus on further quantitative evaluations, exploring different distillation loss weights and testing PPD in more complex and diverse environments to fully assess its potential for real-world applications.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2020. PDF. 1

[2] Weng Lilian. Policy gradient algorithms. *lilianweng.github.io*, 2018. Blog. 1

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. PDF. 2

[4] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017. PDF. 2

[5] Dhanoop Karunakaran. Advantage actor-critic rl in pytorch, 2024. Blog. 2

[6] Giacomo Spigler. Proximal policy distillation, 2025. PDF. 3

[7] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021. PDF. 3

[8] Pablo Samuel Castro Aaron Courville Marc G. Bellemare Rishabh Agarwal, Max Schwarzer. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress, 2022. PDF. 3

[9] Çağlar Gülçehre Guillaume Desjardins James Kirkpatrick Razvan Pascanu Andrei A. Rusu, Sergio Gómez Colmenarejo and Raia Hadsell Volodymyr Mnih, Koray Kavukcuoglu. Policy distillation, 2016. PDF. 4