MS Thesis Report

Christo Aluckal[1]
May 15, 2024

***Abstract*— Deep Reinforcement Learning (DRL) can be used to solve traditional classical control problems with high generalizability which can be extended to harder problems. While simple problems like Inverted Pendulum pose minimal training challenges, complex tasks such as locomotion exponentially increase training difficulty and duration. Leveraging Policy Distillation, we observed promising outcomes in sharing knowledge among concurrent agents through Curriculum Learning. In this paradigm, an agent learning on an easier curriculum serves as a sub-optimal teacher for another tackling a more challenging curriculum. This study focuses on training agents to race within a simulated racing environment, inspired by the F1tenth Gym. Discussions on possible improvements and the potential for policy reuse is also discussed.**

I. INTRODUCTION

Deep Reinforcement Learning is a powerful tool for robotics. Many tasks such as locomotion, gripping, navigation etc. which were traditionally solved with classical control can be abstracted and generalized using DRL. This knowledge can be further built upon and improved by retraining the learnt policy to imitate a new set of directives. Due to the high sample inefficiency of DRL methods, policies are trained in simulation environments before being deployed on real robots. Problems such as Inverted Pendulum and Mountain Car are simple toy problems that do not require a long training time. However, for complicated tasks such as locomotion and navigation, training is exponentially harder and can take longer. The DRL framework described below provides insight on two methodologies to share knowledge between concurrent agents using the Curriculum Learning paradigm. The idea is that an agent learning on an easy curricula can be considered as a sub-optimal teacher to another agent learning on a harder curricula. The agent is tasked with learning to race in a racing environment based on the F1tenth environment (F1tenth Gym).

II. MOTIVATION

The motivation for this work is to show the efficacy of utilizing concurrent agents to teach an agent to race. For the baseline, when an agent based on the Unity Simulation environment is tasked with racing on a set of waypoints, even a simple waypoint configuration of a loop can result in upto 36 hours of training time before convergence. It is not feasible to train this agent on multiple tracks in multiple environments with multiple adversaries in a sequential manner. Thus, a method to parallelize and speed up the convergence is necessary.

*This work was not supported by any organization

[1]Christo Aluckal is with School of Computer Science, University at Buffalo christoa@buffalo.edu

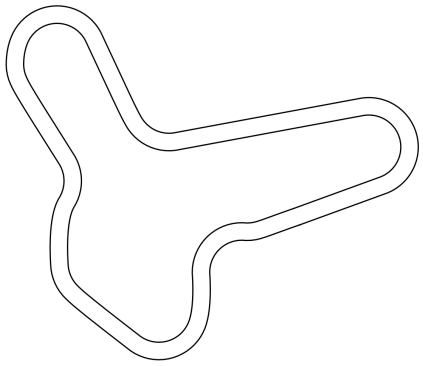

Fig. 1. Evaluation Environment

III. RELATED WORKS

Policy Reuse and Policy Distillation (Knowledge Transfer) are the two major fields where this methodology can be applied. The collective methodologies is called as Transfer Learning. [11] provides an up to date and comprehensive list detailing many methodologies for Transfer Learning, including Imitation Learning, Hierarchical RL, Reward Shaping, etc. [5] provide insight into massive parallelization of robots in the same simulator using on-policy DRL. They show a massive improvement in the average reward attained for stair climbing scenarios. [1] provide a generalized approach for reusing learned network weights and data to accelerate heterogeneous agent tasks. They use a QDAGGER approach to minimize the student loss compared to the teacher loss. Knowledge transfer is also used in other disciplines such as Computer Vision for Model Compression as well as an RL policy selector. [10, 2] use Knowledge Transfer to train a smaller Resnet architecture for classification tasks by teaching the smaller network to approximate the same function as a more complex network. [8] train an agent to learn to select the Teacher rather than relying on heuristic weights.

IV. Methodology

A. Environment

The environment is a modified version of the F1Tenth Gym environment. The environment simulates a race car based on its current orientation and the applied speed and steering angle using an RK4 integration of an approximation of Ackermann Dynamics. The environment consists of a track with a centerline and walls on both sides with equal widths. A simulated LiDAR is also present but is not utilized. 1 shows a sample figure of a map.

B. MDP

The MDP consists of a (S, A, R, P, γ) where $\gamma = 0.99$ and transition probability P is set to the RK4 Ackermann update. Each track consists of a csv containing (X, Y) coordinates sequentially. The environment uses the X,Y values and interpolates a Cubic Spline with the independent variable being the Track Length (s). Thus $x_i = X_{spline}(s_i)$ and $y_i = Y_{spline}(s_i)$. The state space S is computed using a vector with 62 elements. The first two values are the current yaw ψ and the current velocity v of the car. The remaining 60 values are in the form of $[x_0, y_0, x_1, y_1, ..., x_{59}, y_{59}]$ with the closest x, y values on the spline upto a distance of 3m in intervals of 0.1. Thus 30 x and 30 y values. The action space A consists of (v, δ) with v being the input velocity and δ being the steering angle of the car.

1) Reward: The reward formulation is $\Delta s * v$ which is the progress along the spline multiplied by the scalar current velocity v, thus encouraging faster runs.

The termination criteria is the completion of the whole track or collision with the walls which automatically stops the current episode.

C. Transfer Learning Preliminaries

Each methodology uses a common setup. There are 4 environments available during the runs. This consists of an Evaluation environment, an Easy Configuraion, a Medium Configuration and a Hard Configuration. Each agent writes the policy weights on the disk using an identifier that is known to all concurrent agents.

D. Policy Reuse

For policy reuse, the SAC implementation of Stable-baselines3 is utilized. In policy reuse the policy weights are directly updated. For a set of weights $(\theta_1, \theta_2, \theta_3, ...)$, the new policy weights θ_p are computed using

$$\theta_p = w_1 * \theta_1 + w_2 * \theta_2 + w_3 * \theta_3 + ... \quad (1)$$

where $(w_1, w_2, w_3, ...)$ are probabilities of the weights and $\sum_{i=1}^{n} w_i = 1$

During each agents' run, during an evaluation loop, the training is paused and each policy stored on the disk is evaluated for 10 episodes. The average rewards for each policy is stored. For policy p, (θ_p) is computed in 2 ways;

1) Baseline Retention: Here, w_p is set to a constant value r. The probability distribution of the remaining weights are computed as a softmax probability summing upto $(1 - r)$ based on the mean reward obtained by the remaining weights. These weights are then used in eq 1. For eg, a $r = 0.9$ for w_1 with 3 concurrent policies having softmax probabilities $[0.06, 0.04]$ respectively would be

$$\theta_1 = 0.9 * \theta_1 + 0.06 * \theta_2 + 0.04 * \theta_3 \quad (2)$$

2) Dynamic Retention: Similar to the baseline retention but there is no concept of a retention ratio. The probability distribution is strictly based on the mean rewards obtained by each policies. For policy 1 if the softmax probabilities are $[0.1, 0.6, 0.3]$ then the update would be

$$\theta_1 = 0.1 * \theta_1 + 0.6 * \theta_2 + 0.3 * \theta_3 \quad (3)$$

E. Policy Distillation

A similar approach is employed for distillation process [6]. However, each agent now has access to a set of evaluation states S_{eval}. This set of states does not exist in the respective Easy, Medium, Hard configurations and thus would not result in memorization. Each agent now computes the Expected Advantage over the S_{eval} and determines which policy results in the maximum EA. The policy with the maximum EA is considered as the current teacher and the SAC policy loss is modified with

$$loss_{policy} = loss_{actor} + \beta_s * KL(\pi_p(S_{eval})||\pi_t(S_{eval})) \quad (4)$$

where π_p is the current policy distribution, π_t is the best teacher distribution and KL is the KL-divergence between these distributions. This augmented loss would result in the student policy π_p to imitate or learn the teacher policy π_t and β_s is a scaling factor. This method is employed by [9] for RL purposes.

V. Results

1) Policy Reuse: Figures 2,3,4 show the result of directly modifying the network weights. As is evident from the figures, there is no definite improvement when modifying the weights of either the actor, the critic or both. The easy and medium configs do show an improvement upon the baseline but the baseline is far superior in the hard configuration. This lack of consistency can be attributed to simply breaking the chain of Stochastic Gradient Descent by directly changing the weights in a non-differentiable manner. Since the weights are updated as a weighted sum, the convergence is disrupted.

2) Policy Distillation: Figures 5,6,7 show the result of adding a regularization to the policy loss in the form of the KL divergence between the current policy distribution and the current best teacher distribution results in more consistent growth. There is no marked improvement for the easy policy as this policy is considered as the easiest to solve and thus would be it's own teacher. For harder configs, this gap is increased substantially. A higher scaling factor

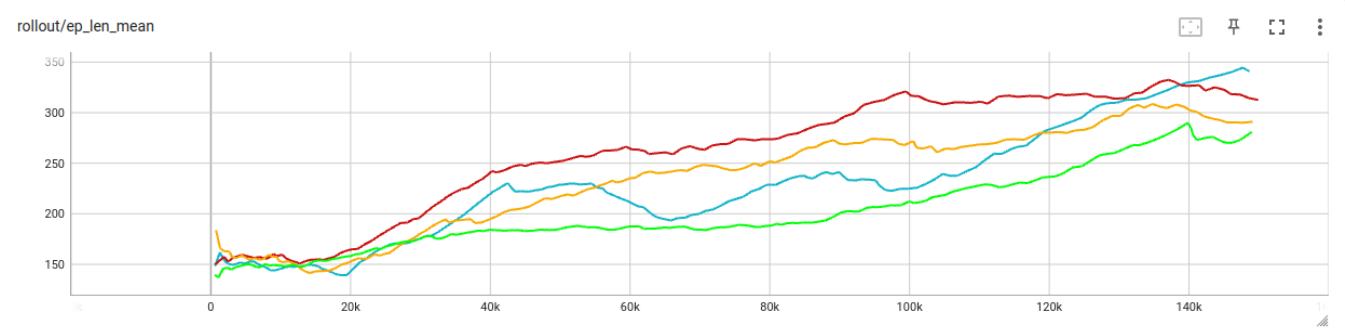

Fig. 2. Policy Reuse on Easy Config with 90% retention. Red 90% critic, Yellow 90% actor, Blue baseline, Green 90% actor and critic

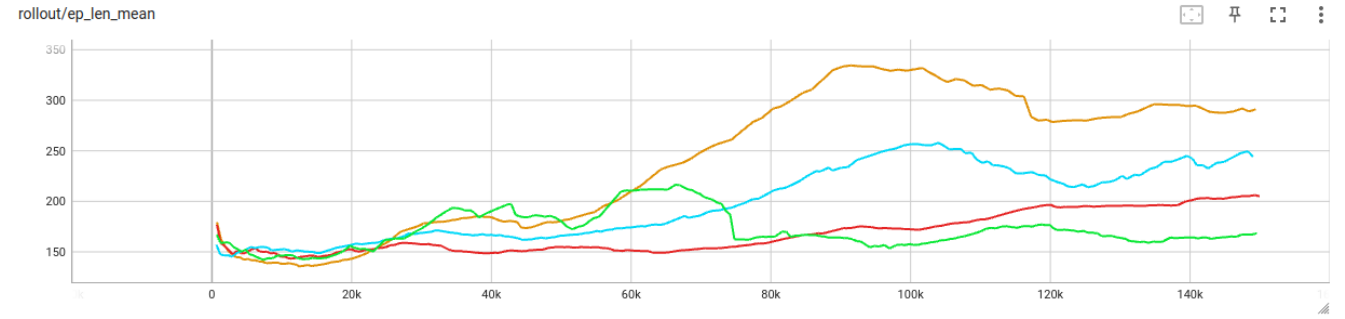

Fig. 3. Policy Reuse on Medium Config with 90% retention. Red 90% critic, Yellow 90% actor, Blue baseline, Green 90% actor and critic

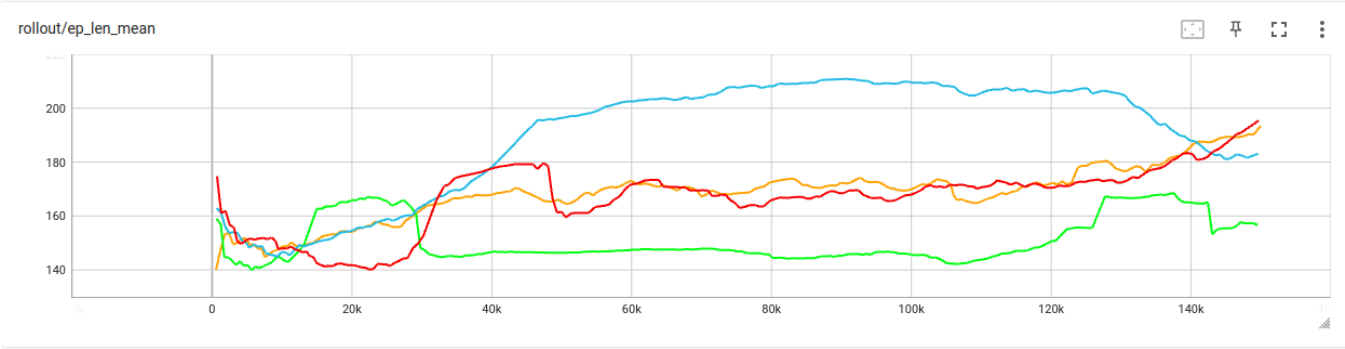

Fig. 4. Policy Reuse on Hard Config with 90% retention. Red 90% critic, Yellow 90% actor, Blue baseline, Green 90% actor and critic

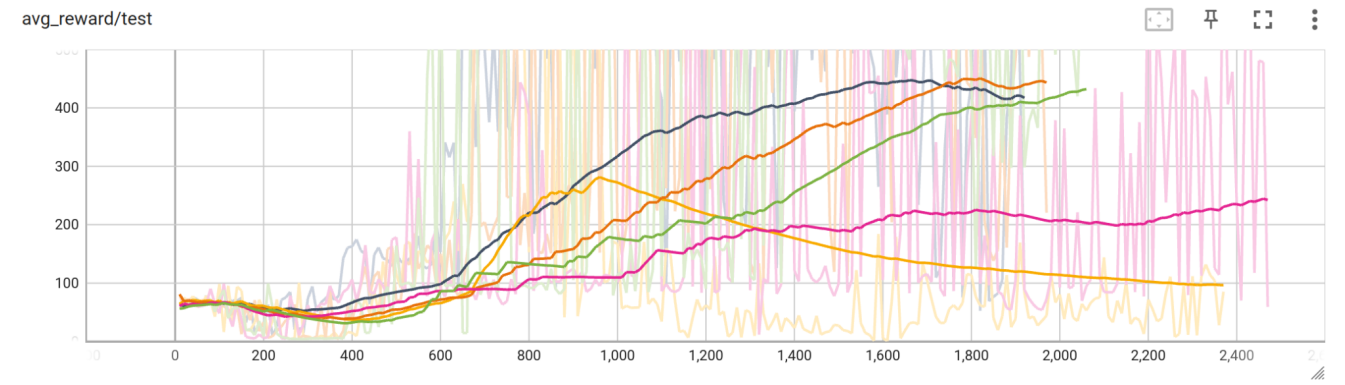

Fig. 5. Policy Distillation on Easy config with different β_s. Black baseline, Yellow 10, Orange 30, Red 50, Green 100

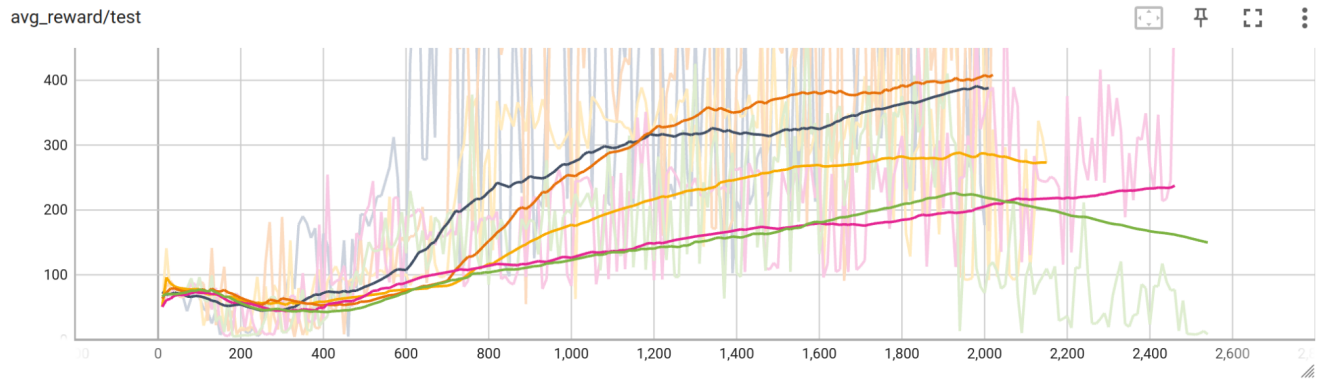

Fig. 6. Policy Distillation on Medium config with different β_s. Black baseline, Yellow 10, Orange 30, Red 50, Green 100

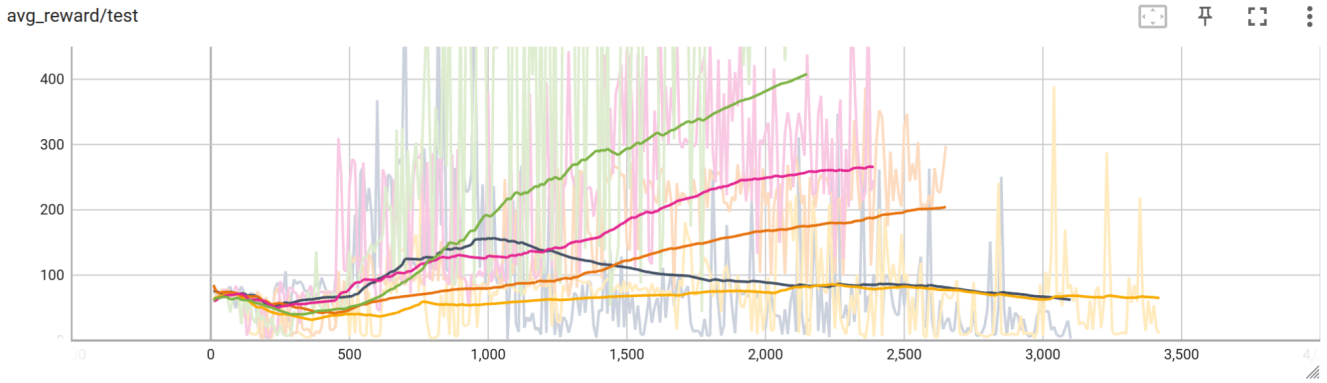

Fig. 7. Policy Distillation on Hard config with different β_s. Black baseline, Yellow 10, Orange 30, Red 50, Green 100

indicates a stronger regularization which helps harder configs to converge more efficiently. This can be seen from 6 which shows the best result for a factor of 30 and 7 for a factor of 100.

VI. Discussions and Conclusion

From the results it can be shown that Policy Distillation provides the most consistent improvement. This methodology can be scaled and used in any environment as long as a curricula or order of difficulty can be established. If a baseline shows a clear order of difficulty then each policy (except for the easiest) can learn from slightly better policies. Other scaling methods such as Teacher Confidence, Variable β_s scaling can be incorporated rather than heuristically selecting it. Certain improvements to the selection of S_{eval} to use rarity of states can be incorporated to use rare states more strongly. Another possible improvement is the combination of reuse and distillation.

VII. Acknowledgements

The base environment is a modified version of the F1tenth Gym [3] Policy reuse was implemented on the stabe-baselines3 platform [4] Policy distillation using SAC was implemented using [7].

I would also like to thank Prof. Souma Chowdhury, Prof. Karthik Dantu and Dr. Charuvahan Adhivarahan for their continued guidance.

References

[1] Rishabh Agarwal et al. “Reincarnating reinforcement learning: Reusing prior computation to accelerate progress”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 28955–28971.

[2] Lucas Beyer et al. “Knowledge distillation: A good teacher is patient and consistent”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10925–10934.

[3] Matthew O’Kelly et al. “F1TENTH: An Open-source Evaluation Environment for Continuous Control and Reinforcement Learning”. In: *NeurIPS 2019 Competition and Demonstration Track*. PMLR. 2020, pp. 77–89.

[4] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[5] Nikita Rudin et al. “Learning to walk in minutes using massively parallel deep reinforcement learning”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

[6] Andrei A Rusu et al. “Policy distillation”. In: *arXiv preprint arXiv:1511.06295* (2015).

[7] Pranjal Tandon. *Soft Actor Critic in Pytorch*. https://github.com/pranz24/pytorch-soft-actor-critic. [Online; accessed 15-May-2024]. 2021.

[8] Fei Yuan et al. “Reinforced Multi-Teacher Selection for Knowledge Distillation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.16 (May 2021), pp. 14284–14291. DOI: 10.1609/aaai.v35i16.17680. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17680.

[9] Jin Zhang, Siyuan Li, and Chongjie Zhang. *CUP: Critic-Guided Policy Reuse*. 2022. arXiv: 2210.08153 [cs.AI].

[10] Linfeng Zhang et al. “Be your own teacher: Improve the performance of convolutional neural networks via self distillation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 3713–3722.

[11] Zhuangdi Zhu et al. “Transfer Learning in Deep Reinforcement Learning: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.11 (2023), pp. 13344–13362. DOI: 10.1109/TPAMI.2023.3292075.