A UNIVERSAL FRAMEWORK FOR ENSEMBLED DEEP FEDERATED REINFORCEMENT LEARNING AND APPLICATIONS TO MICRO UAVS

Kevin Han[1], *James Anderson*[2], *Brian Plancher*[3]

[1]Department of Physics and Mathematics, University of Texas at Austin
[2]Department of Computer Science, Barnard College, Columbia University
[3]Department of Electrical Engineering, Columbia University

1. INTRODUCTION AND BACKGROUND

Federated Reinforcement Learning (FRL) enables multiple agents with identical state and action spaces in independent and varied environments to collaboratively learn an optimal policy. This approach is beneficial in scenarios where agent privacy is crucial, such as in energy grids or medicine. Using PyTorch, I develop a deep FRL framework capable of supporting and ensembling any RL algorithm, and enhance two novel momentum-based algorithms, FEDSVRPG-M and FEDHAPG-M[1], alongside other state-of-the-art (SotA) RL algorithms, to train crazyflie drones. The novelty in these momentum-based algorithms are shown with their guaranteed convergence to a stationary point of the average performance function, despite environment heterogeneity. However, this is under the assumption that they are the only local algorithms.

For the other local algorithms, I use Proximal Policy Optimization (PPO) [2], Soft Actor-Critic (SAC) [3], and Twin-Delayed Deep Deterministic Policy Gradient (TD3) [4]. Each of these algorithms have their independent strengths. PPO is more of an on-policy algorithm that is stable and provides reliable performance but can be sample inefficient and sensitive to hyperparameters. SAC and TD3 are off-policy algorithms. SAC offers excellent exploration and sample efficiency but is computationally demanding. TD3 is robust against overestimation bias and sample efficient but can be slower to converge. Combined with deep learning, my ensemble method aims to speed up learning by leveraging the strengths of each other SotA sub-algorithm without losing too much of the convergence benefits of momentum-based FRL. I also explore the benefits of aggregation in value function estimation to determine if critics benefit from FRL too.

Furthermore, on the simulation side, I modify the gym-pybullet-drones[5] platform to include domain randomizations for wind and mass conditions, enhancing sim2real transfer. Using this FRL platform, I train crazyflie drones for various tasks and plan to incorporate layer freezing and LQR-based supervised learning for subtasks like hovering to advance robot learning..

[1]https://github.com/KevinHan1209/FederatedRL.git

2. METHODS

2.1. Software

To implement FRL, I create a universal server class that iterates through each agent's local environment and training process, using the process outlined in Section 2.2. I implement FEDSVRPG-M as a separate class, and utilize gradient norm clipping and adaptive learning rates to address problems exploding gradients. For other local algorithms, I adapted stable_baselines3[6], a heavily used package for baselining RL algorithms in the literature which employs SotA techniques for optimization. To ensure universal compatibility, I process policy parameters to PyTorch tensors during global optimization steps, and then back to their respective policy classes after.

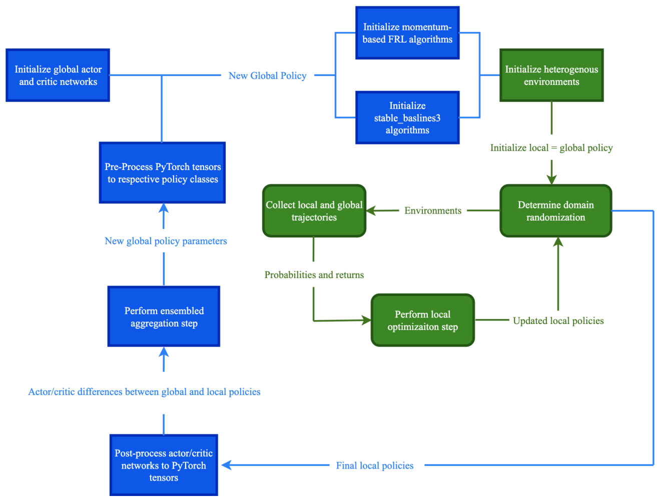
Initialize global actor and critic networks
New Global Policy
Initialize momentum-based FRL algorithms
Initialize stable_baslines3 algorithms
Initialize heterogenous environments
Initialize local = global policy
Pre-Process PyTorch tensors to respective policy classes
Collect local and global trajectories
Environments
Determine domain randomization
New global policy parameters
Probabilities and returns
Perform ensembled aggregation step
Perform local optimizaiton step
Updated local policies
Actor/critic differences between global and local policies
Post-process actor/critic networks to PyTorch tensors
Final local policies

Fig. 1. High level workflow of FRL platform.

2.2. Algorithm

Due to time constraints, I could only implement FEDSVRPG-M. For each agent, at each local iteration k for each global round r, this algorithm uses a momentum-based variancereduced stochastic policy-gradient estimator:

$$u_{r,k} = \beta g(\tau_{r,k}|\theta_{r,k}) + (1-\beta)[u_r + g(\tau_{r,k}|\theta_{r,k}) - w(\tau_{r,k}|\theta_{r-1},\theta_{r,k})g(\tau_{r,k}|\theta_{r-1})], \tag{1}$$

where $\beta \in [0,1]$ is the momentum parameter, θ are the policy parameters, τ represents the agent trajectory through the state and action spaces, $g(\tau|\theta)$ is the standard policy gradient estimator used in classic policy gradient methods such as REINFORCE[7], and w is the importance sampling weight defined by

$$w(\tau_{r,k}|\theta_{r-1},\theta_{r,k}) = \frac{p(\tau_{r,k}|\theta_{r-1})}{p(\tau_{r,k}|\theta_{r,k})}, \tag{2}$$

where p is the probability of the trajectory assuming states and actions are independently distributed. The objectives and updates for PPO, SAC, and TD3 are not mentioned here for brevity, but the specific implementations can be found in my github link[1] and in their respective documentations within stable_baselines3 [6].

After each local iteration in FEDSVRPG-M and all other algorithms, each latest local policy parameters θ_r, K is sent to the server and after post-processing, the policy displacement $\Delta_r = \theta_r, K - \theta_r$ is calculated for the actor networks and, if applicable, the critic networks. The latest local policies are also evaluated for all agents and mean episode rewards ρ are determined. The global ensembled gradient is then calculated as follows:

$$u_{r+1} = \frac{1}{\eta N K \sum_{i=1}^{N} \rho_i} \sum_{i=1}^{N} \rho_i \Delta_r^{(i)}, \tag{3}$$

where η is the local step size and N is the total number of agents. This gradient update is calculated for both the actor and available critic networks. If the critic network represents a state-action value function rather than a state action, such as in SAC and TD3, the new action parameters are just calculated as a weighted average rather than a gradient update. The gradient updates are then implemented as

$$\theta_{r+1} = \theta_r + \lambda u_{r+1}, \tag{4}$$

where λ is the global step size. All algorithm models then have θ_{r+1} initialized as their policy before starting the next global iteration.

Algorithms which use Polyak averaging between two actor or critic networks where one is serves as the target network has the other considered as the main network this process, and the target networks are initialized to be identical as the new global network at the start of each new global round.

2.3. Drone Simulation

Quadrotor drones operate in a continuous environment, with state spaces $\mathcal{S}$ of dimension 27 comprising of positions $\mathbf{x} = [x, y, z]$, quaternions $\mathbf{q}$, rolls r, pitches p, yaws y, linear velocities $\dot{\mathbf{x}}$, angular velocities ω, and motor speeds $\mathbf{P}$:

$$\mathcal{S} \ni \{\mathbf{x}, \mathbf{q}, r, p, y, \dot{\mathbf{x}}, \omega, \mathbf{P}\}. \tag{5}$$

The action spaces $\mathcal{A}$ are 4-dimensional, comprising of the 4 RPM inputs to the motors $\mathbf{P}$.

Inspired by TornadoDrone[8], I enhance the gym-pybullet-drones simulation with domain randomizations in a probabilistic setting. For each episode, there is a 50% chance that domain randomization will be applied where the mass is randomized up to 15 grams more than the default of 27 grams. In each domain randomization episode, there is also a 30% chance that a wind force up to 0.005 Newtons in any direction would be applied. I also incorporate nonlinear aerodynamic effects such as drag and ground effects. These modifications improve the robustness of sim-to-real transfer by enabling the model to learn chaotic effects more effectively, and, combined with deep learning, serve to prepare the model for more complex tasks.

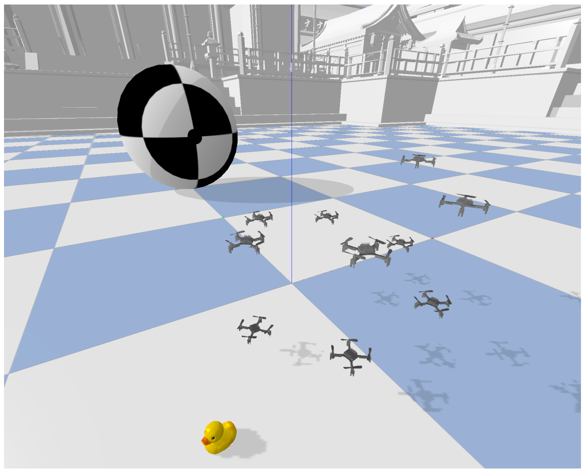

Fig. 2. The rendering of gym-pybullet-drones simulation[5].

2.4. Deep Learning

I develop a diagonal Gaussian policy network with hyperbolic tangent activation, featuring 4 hidden linear layers (512, 512, 256, 128 units) and standard deviation layers, shared among all agents.

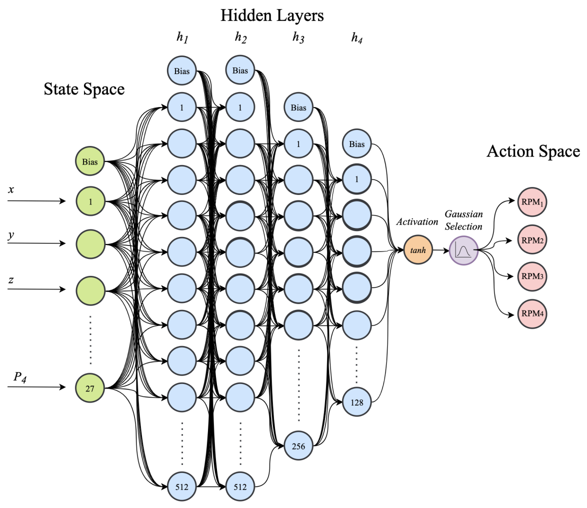

Fig. 3. Actor network architecture. Note that activation functions are utilized between every hidden layer as well.

Actor-critic algorithms utilize a separate critic network with ReLU activation and 2 hidden layers of 32 units each.

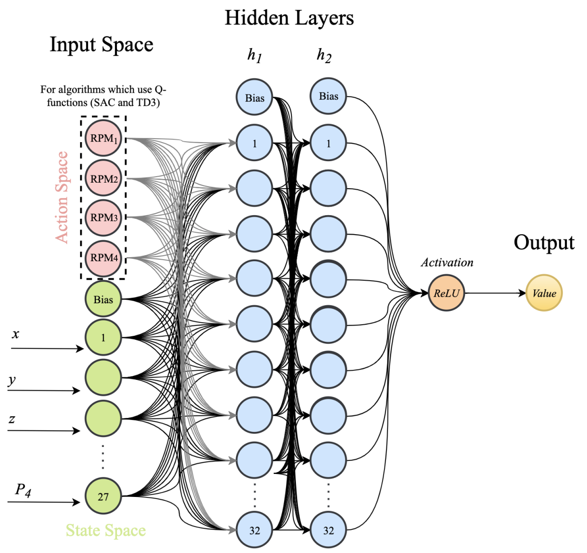

Fig. 4. Critic network architecture. Note that activation functions are utilized between every hidden layer as well.

3. RESULTS

Due to time constraints and switching labs 4 weeks into the program, I unfortunately am not able to complete training and obtain results before the deadline of this abstract. However, I have established all of the necessary software, which was one of my target accomplishments. I have developed a universal platform to conduct ensembled and/or federated reinforcement learning with custom algorithms or algorithms from stable_baselines3 [6], and are compatible with environments from OpenAI gymnasium [9].

I will have results and comparisons of all of the aforementioned algorithms and their federated and ensembled versions by the time of the poster presentation.

4. ACKNOWLEDGEMENTS

I would like to thank my primary supervisor, Dr. Brian Plancher, for his thorough guidance and inspiration throughout this project. I would also like to thank Dr. James Anderson for his guidance on Federated Reinforcement Learning.

5. REFERENCES

[1] Han Wang, Sihong He, Zhili Zhang, Fei Miao, and James Anderson, “Momentum for the win: Collaborative federated reinforcement learning across heterogeneous environments,” 2024.

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

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” 2018.

[4] Scott Fujimoto, Herke van Hoof, and David Meger, “Addressing function approximation error in actor-critic methods,” 2018.

[5] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig, “Learning to fly – a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control,” 2021.

[6] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[7] Junzi Zhang, Jongho Kim, Brendan O’Donoghue, and Stephen Boyd, “Sample efficient reinforcement learning with reinforce,” 2020.

[8] Robinroy Peter, Lavanya Ratnabala, Demetros Aschu, Aleksey Fedoseev, and Dzmitry Tsetserukou, “Tornadodrone: Bio-inspired drl-based drone landing on 6d platform with wind force disturbances,” 2024.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, “Openai gym,” 2016.