Project Report

Environment 1: Cartpole

We have tried two algorithms. Carpal problem has a threshold at 500 as a standard and has been evaluated for that. We used the MLP_Policy in both.

1. Algorithm:- DQN

DQN incorporates several key techniques for effective training. It employs experience replay, enabling it to learn from a diverse range of past policies and experiences, enhancing the stability of learning. To counteract the moving target issue, a target Q-network is utilized, periodically frozen to align with the current policy. Additionally, the algorithm applies reward clipping or normalization to ensure that the neural network operates within a sensible and stable range, facilitating more efficient and reliable training. These techniques collectively contribute to the robust and effective learning of optimal policies.

The initial DQN algorithm gave us mediocre results. So we tried to change a few parameters. The parameters we changed were:

Learning Rate (learning_rate): Adjust the step size for optimization. Higher values might speed up training but can lead to instability. Lower values can make learning more stable but slower. We tried $2.3e^{-3}$ instead of the default 0.0001 as this decreases the mean value.

Batch Size (batch_size): Control the number of samples used in each training iteration. Larger batches provide more stable updates but require more resources. Smaller batches may be more memory-efficient but noisier. We tried 64 instead of 32

Replay Buffer Size (buffer_size): Determine the number of past experiences stored and sampled during training. Larger buffers provide diverse experiences, but require more memory. Smaller buffers may lead to quicker forgetting of older experiences. We tried 100000 instead of 1000000

Target Update Interval (target_update_interval): Control how often the target network is updated in DQN. Smaller intervals stabilize learning but slow down training. Larger intervals speed up training but may introduce instability. We tried 10 instead of 10000

Gradient Steps (gradient_steps): Determine the number of optimization steps in each update. More steps can lead to better policies but slower training. Fewer steps can speed up training but may result in less effective learning. We tried 128 instead of 1.

Exploration Fraction (exploration_fraction): Set the fraction of time during training for exploration. Higher values promote more exploration, potentially finding better policies. We tried 0.16 instead of 0.1.

Final Exploration Rate (exploration_final_eps): Define the final exploration rate for epsilon-greedy policies. It controls the balance between exploration and exploitation. We tried 0.04 instead of 0.05

This gave us much better results with each episode reaching the maximum of 500.

2. Algorithm:- PPO

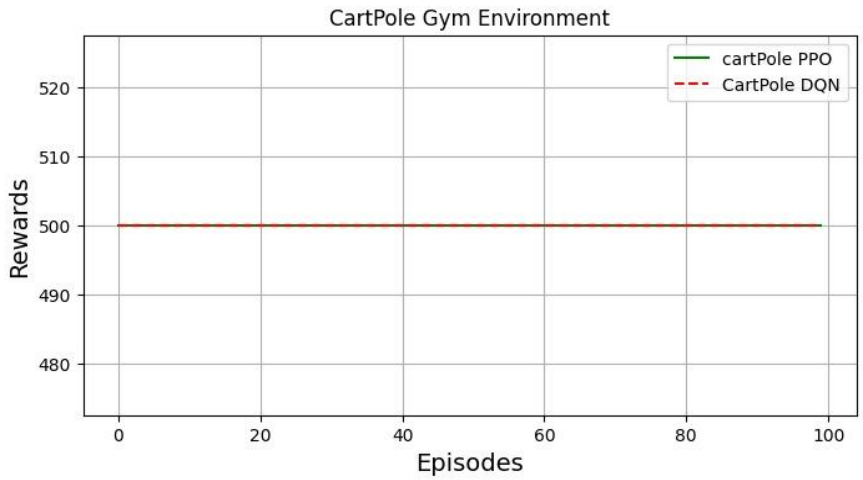
In the PPO algorithm, two networks are maintained simultaneously. The first network represents the current policy that we aim to improve, while the second is utilized for sample collection. To stabilize the learning process and prevent drastic policy changes, an important step involves clipping the objective function, ensuring that policy updates are controlled. This is achieved by calculating the ratio between the probabilities of actions under the new policy and the old policy. This ratio helps in constraining the policy updates, striking a balance between exploration and exploitation and ultimately enhancing the training stability and effectiveness of the algorithm.

This gave us a max score of 500 with the initial parameters itself.

Initial parameters are:

class **stable_baselines3.ppo.PPO**(*policy, env, learning_rate=0.0003, n_steps=2048, batch_size=64, n_epochs=10, gamma=0.99, gae_lambda=0.95, clip_range=0.2, clip_range_vf=None, normalize_advantage=True, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5, use_sde=False, sde_sample_freq=-1, target_kl=None, stats_window_size=100, tensorboard_log=None, policy_kwargs=None, verbose=0, seed=None, device='auto', _init_setup_model=True*) [source]

PPO vs DQN:



Even though they both show maximum rewards, PPO does not require any parameter tuning while DQN requires it. However after the tuning DQN shows good performance after the tuning. DQN is more sample efficient than PPO.

Environment 2: Humanoid

We have tried two algorithms, SAC and TD3

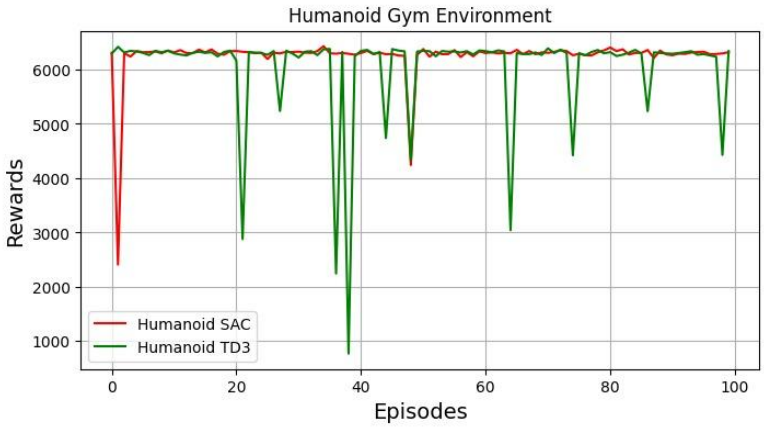
1. Algorithm:- SAC

SAC is a state-of-the-art reinforcement learning algorithm that excels in solving a wide range of continuous control tasks. SAC combines key elements from actor-critic methods and entropy regularization. It encourages both effective policy learning and exploration by optimizing for the expected cumulative rewards while simultaneously maximizing the policy's entropy. This striking feature enables SAC to balance exploration and exploitation effectively. Moreover, SAC employs automatic entropy tuning, which removes the need for manual tuning of hyperparameters and provides a powerful, yet easy-to-implement framework for continuous action space reinforcement learning. Its versatility and efficiency have made SAC a popular choice for various applications in robotics and control.

2. Algorithm:- TD3

TD3 is a reinforcement learning algorithm designed for continuous action spaces. It builds upon the DDPG (Deep Deterministic Policy Gradient) algorithm by introducing a twin Q-network architecture and target policy smoothing. TD3 enhances the stability of learning by addressing overestimation issues common in Q-learning. By maintaining two Q-networks and utilizing a delayed target policy update mechanism, TD3 minimizes the variance in Q-value estimates and delivers more robust training results. This makes it particularly effective in tasks with continuous action spaces and has been successfully applied in various real-world scenarios.

SAC vs TD3:



As we can see SAC is converging faster than TD3. TD3 requires more episodes to converge than SAC.

References:

1. DQN documentation
2. PPO documentation
3. SAC documentation
4. TD3 documentation
5. Hugging Face