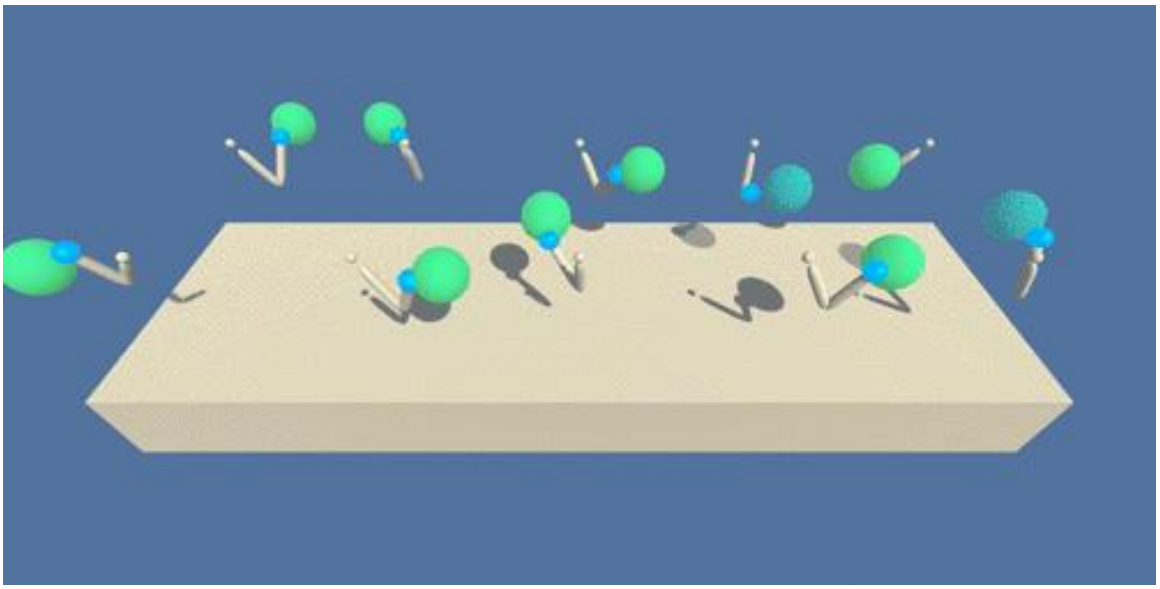
Deep Reinforcement Learning

Project 2: Continuous Control



Simon FERRAND

School : Udacity

Course name : [Deep Reinforcement Learning Nanodegree]

15/02/24

UDACITY

Abstract

This report delineates the process of designing and refining a deep reinforcement learning agent capable of continuous control tasks within the Unity ML-Agents environment. Utilizing the Deep Deterministic Policy Gradient (DDPG) algorithm, the agent was trained to navigate and maintain a double-jointed arm at target locations. The evolutionary approach to agent design involved iterative code refinement and testing, resulting in advancements in performance metrics, but addressing one of the limitations of reinforcement learning, which is the instability of learning that can manifest as huge drops in performance, particularly pronounced with DDPG.

Introduction

The realm of reinforcement learning has witnessed a paradigm shift with the advent of continuous control tasks, presenting novel challenges in training agents with a high degree of precision and adaptability. This project, set in the Unity ML-Agents environment, entails the development of an agent tasked with the operation of a double-jointed arm to reach and sustain position within target locations. The sophistication of this environment lies in its thirty-three-dimensional state space and four-dimensional action space, requiring a nuanced policy for effective torque application on the arm's joints.

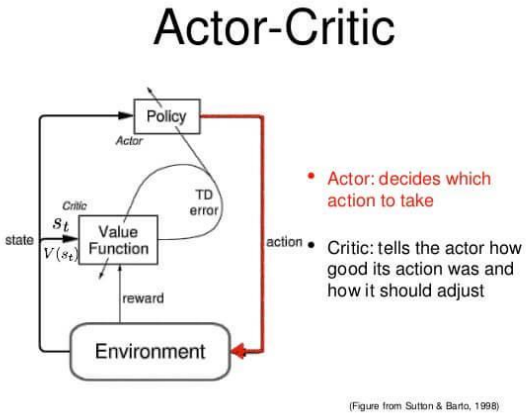
Given the episodic nature of the task and the benchmark of an average score of +30 over 100 consecutive episodes, the project's goal transcends mere arm navigation, aiming for sustained and precise control. The initial model adopted a classical structure, later undergoing significant refinements to enhance its environmental interaction capabilities.

The subsequent sections will detail the methodological framework, code evolution, testing paradigms, and a critical analysis of performance indicators, culminating in a reflective discussion on the project's outcomes and future potential.

I) Methodology

1.1) Deep Deterministic Policy Gradient (DDPG) algorithm

The DDPG algorithm stands as a cornerstone in our methodology, merging ideas from DQN (Deep Q-Network) and actor-critic methods. Its foundation lies in the utilization of deep neural networks to approximate both the policy (actor) and the value (critic) functions, enhancing the agent's ability to tackle continuous action spaces with high-dimensional state inputs. DDPG is essentially a precursor to TD3, with both sharing identical policies and implementations, highlighting TD3 as an advanced iteration of DDPG.

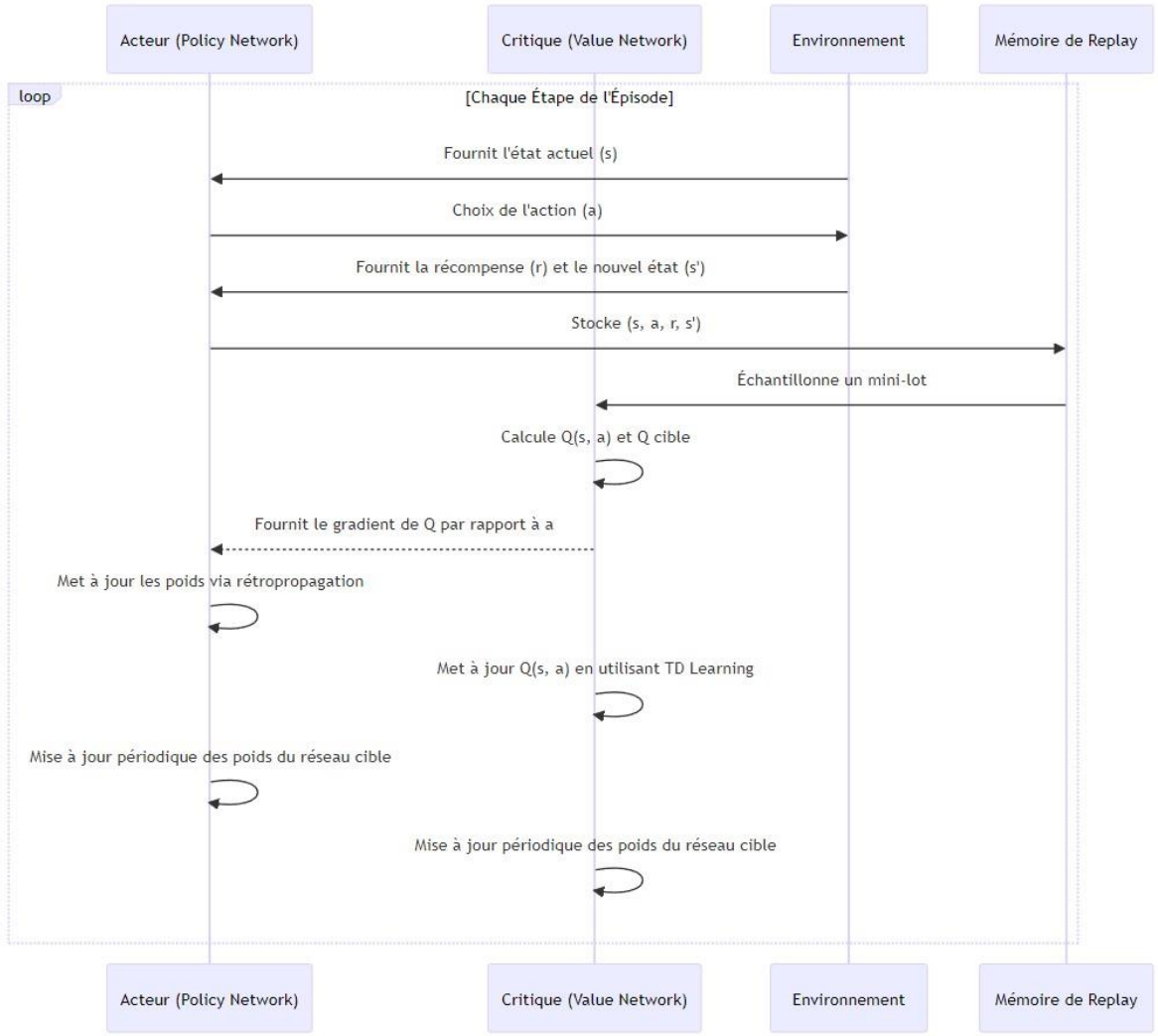


Actor (Policy Network): The actor network directly maps states to actions, providing the agent with a deterministic policy. It takes the current state as input and outputs a specific action that maximizes performance. The learning process involves adjusting the actor's parameters to maximize the expected reward, which is estimated by the critic network.

Critic (Value Network): The critic network evaluates the potential of the current state-action pair by estimating the Q-value. It takes both the state and the action as inputs and outputs a scalar value representing the expected return. The critic's role is to critique the actions taken by the actor by providing a gradient of the Q-value with respect to the action, which is used to update the actor's policy.

Environment: This is the setting within which the agent operates. It provides the agent with states and evaluates the actions that the agent takes. In return, it offers rewards that guide the learning process and new states that result from the actions of the agent.

Replay Memory: A vital component of DDPG, replay memory stores the agent's experiences. Each entry in the memory is a tuple containing the state, the action taken, the reward received, and the next state. This data is sampled randomly to break the correlation between consecutive learning steps, stabilizing the training process.



DDPG Diagram

The DDPG algorithm proceeds in a loop over each episode, as depicted in the attached diagram. At each timestep within an episode, the actor selects an action based on the current policy and the state provided by the environment. The environment responds with a reward and a state. The tuple of the current state, action, reward, and new state is stored in the replay memory. Periodically, a minibatch is sampled from the replay memory to train the network.

The critic network computes the Q-value for the current state-action pair and the target Q-value for the next state-action pair. These values are used to compute the temporal difference (TD) error, which measures the difference between the predicted and the actual returns. The TD error is then utilized to update the critic network's parameters through backpropagation.

Simultaneously, the actor network's parameters are updated using the policy gradient, which is derived from the critic's assessment. The gradient of the Q-value with respect to the action is used to guide the policy updates, aiming to increase the probability of actions that lead to higher returns.

Both the actor and critic networks have corresponding target networks, which are slowly updated with the learned networks' weights to provide stable targets during temporal difference learning. This gradual update helps to mitigate the risk of the learning process diverging.

In essence, DDPG combines the strength of policy-based methods in continuous action spaces with the stability and robustness of value-based methods, encapsulating a powerful approach for reinforcement learning tasks that require sophisticated policies and value estimations.

1.2) Code Evolution and Feature Integration

Baseline to Advanced: Initiated with a baseline model inspired by a simplified financial example from Udacity, which later incorporated mechanisms to balance exploration with exploitation.

Noise Decay Implementation: Implemented linear noise decay, followed by exponential decay to regulate exploration over time, enhancing the agent's ability to focus on promising strategies as learning progressed [OpenAI; 2024].

Learning Iteration Enhancement: Explored the impact of increasing learning iterations, hypothesizing accelerated learning.

Prioritized Experience Replay (PER): Prioritized Experience Replay (PER) refines the conventional experience replay by selecting experiences for retraining based on their significance, as determined by the size of their temporal difference (TD) error. This method ensures that critical experiences are revisited more often, thereby potentially hastening and enhancing the efficiency of the learning process.

Performance Indicators Monitoring: Key indicators were meticulously tracked, providing granular insight into the learning process and the overall health of the agent's decision-making mechanisms.

- **Actor and Critic Loss Monitoring**: After enhancing the model with the ability to measure the losses of both the actor and critic, we gained a more nuanced understanding of the learning efficacy. The actor loss measurements shed light on how well the policy network was performing in selecting actions, while the critic loss offered visibility into the value network's accuracy in estimating potential returns.
- **Weights and Gradients Tracking**: Close on the heels of implementing loss monitoring, we augmented the code to track the weights and gradients within the actor and critic networks.

By capturing the mean and standard deviation of the weights and biases, as well as their gradients, the model's learning stability and parameter convergence were under constant review.

- **Importance Sampling Weights (PER)**: A key addition to the performance metrics was the monitoring of importance sampling weights within the PER mechanism. This advanced feature provides a window into the prioritization process of the experience replay, guiding the agent to learn more from significant experiences.
- **TD Error Analysis**: The TD errors' trajectory offered a granular view into the discrepancy between predicted and actual outcomes, serving as a beacon for policy refinement and adjustment.

1.3) Presentation of the best algorithm

The detailed presentation of the model architecture and hyperparameters of the algorithm that achieved the objective most rapidly will be provided.

a) Model Architecture

Actor Network: Designed to determine the optimal action given the current state, featuring a fully connected neural network with two hidden layers. The first layer contains 164 units, while the second encompasses 128 units, both utilizing the Leaky ReLU activation function for non-linearity. The output layer consists of 4 nodes, applying the tanh activation function to ensure the output action values are within the required range of -1 to 1.

Critic Network: Evaluates the potential of the chosen actions by estimating their Q-values. Similar to the actor, it employs a fully connected neural network with two hidden layers of 164 and 128 units, respectively. The critic's output layer is designed to produce a single value, indicating the expected return of the state-action pair, without applying any activation function to the final output.

b) Training Configuration and Hyperparameters

Maximum Episodes and Timesteps: The training regimen is structured to run for a maximum of 300 episodes, with each episode allowing up to 1000 timesteps. This setup ensures ample opportunity for the agent to interact with the environment, learn from a wide range of situations, and refine its policy over time.

Experience Replay: The model utilizes an experience replay buffer with the capacity to store 1,000,000 tuples, facilitating efficient learning by breaking the correlation of sequential experiences.

Batch Size: 1024 - At each learning step, 1024 tuples are sampled from the experience replay buffer to update the network weights.

Learning Rate: Both the actor and critic networks are trained with a learning rate of 0.001, enabling gradual but steady learning.

Soft Updates: The model employs soft updates for the target networks with a tau (τ) of 0.001, ensuring the target networks are gradually adjusted to the learned networks, promoting stability.

Discount Factor (Gamma): Set at 0.99, this factor discounts future rewards, balancing immediate and future gains.

Noise: To encourage exploration, a noise process with an initial sigma value of 0.2 is incorporated into policy execution. This exploration is exponentially tapered with a decay rate of 0.978, progressively shifting the emphasis towards exploitation over time, while maintaining a baseline level of exploration by setting a minimum sigma threshold of 0.01.

Prioritized Experience Replay (PER): The dynamic Beta parameter in the PER mechanism, starting at 0.4 and increasing to 1.0 over 110 episodes, strategically modifies the focus from predominantly learning from experiences with high TD errors to a more balanced approach over time. Initially, this ensures that experiences which are likely to provide the most learning value are prioritized. However, as Beta increases, the selection process becomes less biased towards these high-priority experiences, encouraging a broader exploration of the experience space and a more uniform learning from all stored experiences. This gradual adjustment aims to optimize the learning process by initially exploiting the most informative experiences and then expanding the scope of learning to include a wider range of experiences.

Learning Frequency: The agent initiates the learning process every 20 timesteps (from 20 agents in parallel). This is achieved by accumulating experiences in the environment and storing them in a prioritized experience replay buffer. Once there are enough samples in the buffer (at least equal to the batch size of 1024), the agent begins the learning process. During this phase, it randomly samples a batch of experiences from the buffer to update both the actor and critic networks.

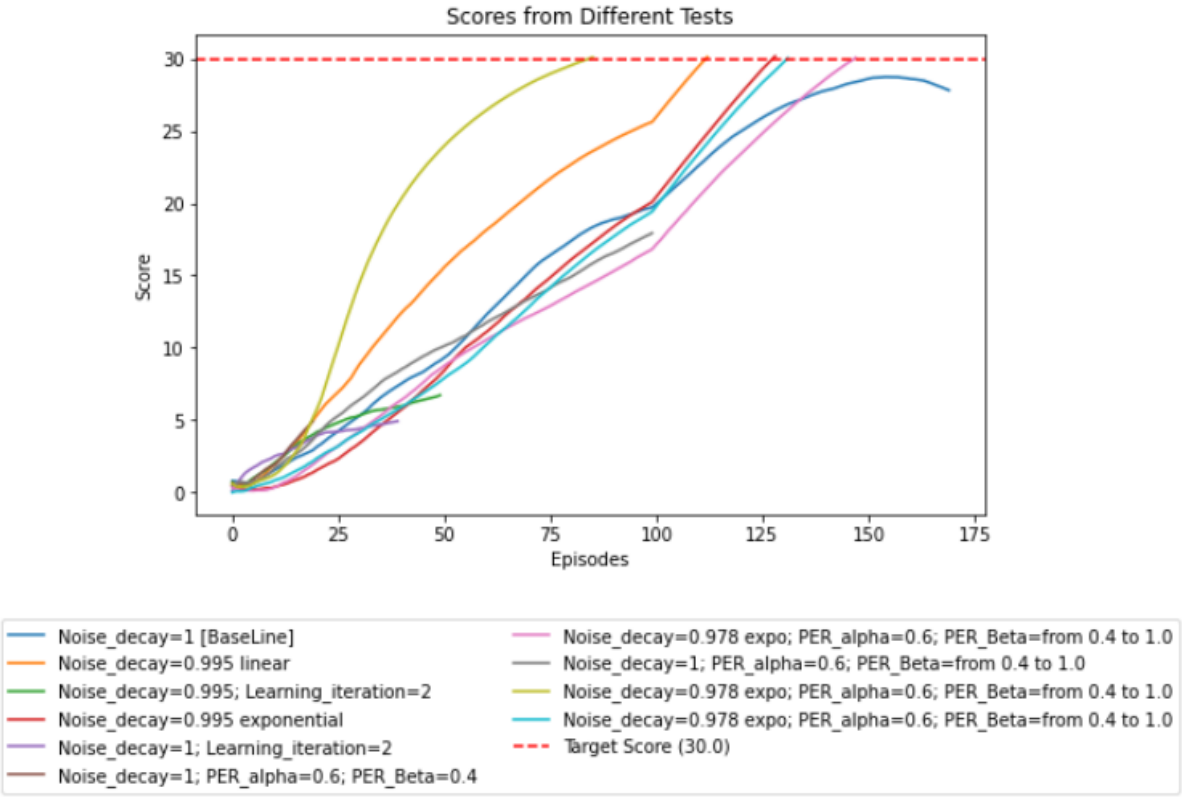
This structured approach allows the agent to learn from a diverse set of experiences, reducing the correlation between sequential learning steps and improving the stability and efficiency of the learning process. The prioritization of experiences based on their TD-error ensures that the agent focuses more on learning from experiences where it has the most to gain in terms of improving its policy and value function estimations.

Weight Decay: For both the actor and critic networks, a weight decay parameter is set to 0, indicating that no additional regularization is applied through weight decay. This choice focuses the learning updates purely on the optimization of the loss functions, without the influence of weight size penalties.

These additions provide a comprehensive picture of the training strategy, including the duration of the learning process and the decision to forego weight decay in the optimization process, further elaborating on the model's configuration for achieving effective learning outcomes.

II) Results

2.1) Experimentation and Testing



BaseLine hypermarameters: max_t=1000, buffer_size=1e6, batch_size=1.024, gamma=0.99, tau=1e-3, LR_actor=1e-3, LR_critic=1e-3; weight_decay=0, learing_iteration=1 , Noise_dacay=1

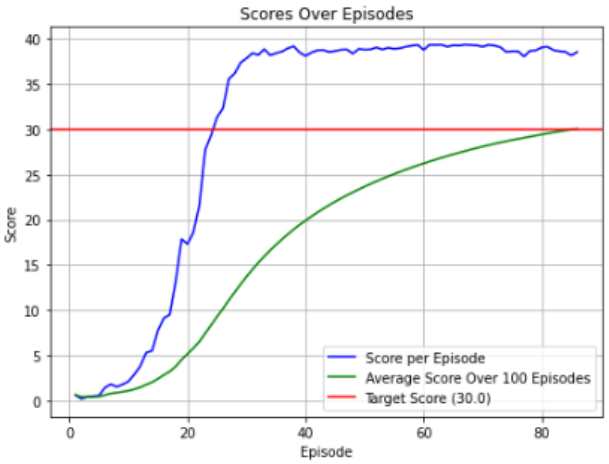
During the project, a series of experimental trials were conducted with each iteration of the agent's code, recording the performance impacts (see figure above). The baseline adjustment did not achieve convergence over 300 episodes. The previously presented algorithm tuning "'Noise_decay=0.978 expo; PER_alpha=0.6; PER_Beta=from 0.4 to 1.0'" was performed 3 times with different seeds, resulting in average scores of 30.0 attained in 148, 86, and 132 episodes (mean 122). Thus, significant instability in learning is observed with differences on the order of a factor of 2 between results. The second-best performance was achieved with a linear noise decay of 0.995 added to the baseline, reaching the goal in 113 episodes. These tests have enabled me to understand the importance of fine-tuning exploration strategies within the learning process. **Further studies are necessary to evaluate and compare all developed algorithms.**

As described in the literature, one limitation in reinforcement learning is the instability observed during training. Specifically, there can be considerable fluctuations in performance, including sudden drops. This issue is especially pronounced with DDPG, prompting the development of its extension, TD3, which aims to address this instability. Other methods, such as TRPO or PPO, employ a trust region approach to mitigate this issue by preventing excessively large updates [Stable-baselines3; 2024].

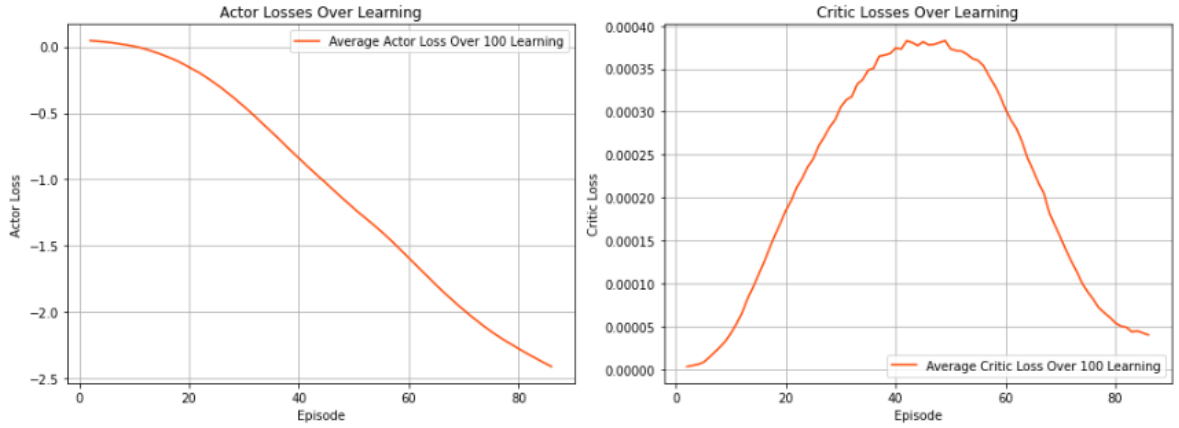
2.2) Detailed presentation of the best training

This section will proceed with the detailed results from the best training performed in 86 episodes.

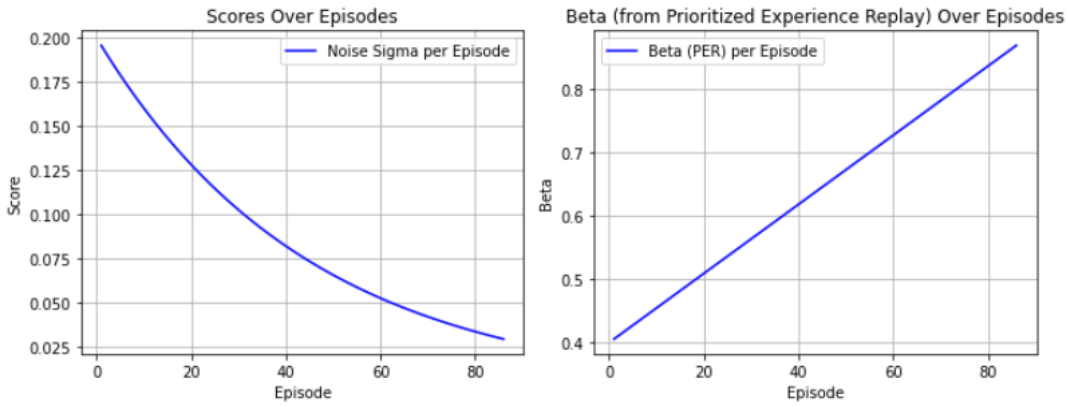
Scores Over Episodes: The agent's performance improved steadily, with average scores incrementally increasing each episode. A pivotal moment was observed when the scores consistently surpassed the target, reflecting the agent's mastery over the task.



Actor and Critic Losses: Analysis of the actor losses revealed a trend of decreasing magnitude, suggesting a refinement of policy over time. The critic losses exhibited a peak before stabilizing, indicating the agent's value function was adjusting to a more accurate representation of the state-action space.



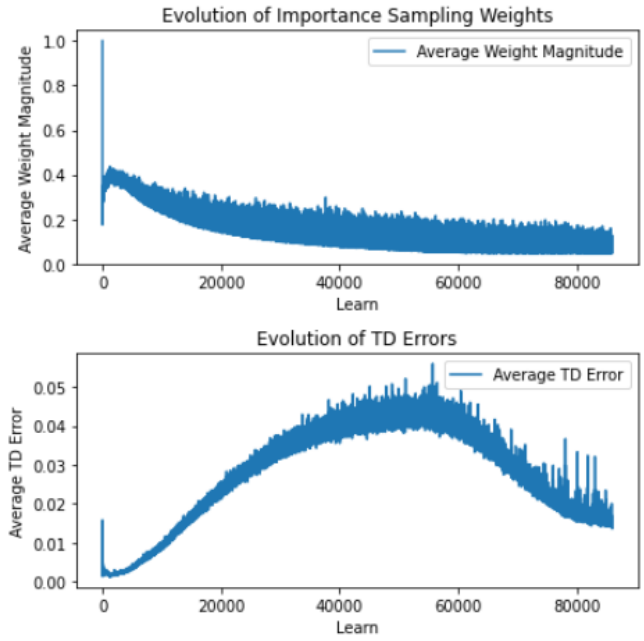
Noise Sigma and Beta (PER) Analysis: The noise sigma decay was integral to the agent's shift from exploration to exploitation. The Beta parameter's dynamic scaling, a feature of PER, helped in fine-tuning the agent's sampling process from the replay buffer, emphasizing more on significant experiences as learning progressed.



TD Errors and Importance Sampling Weights (PER):

The observed increase in TD errors during the intermediate learning phase indicated an active policy adjustment by the agent to better understand the environment's complexities. As the learning continued, the decrease in TD errors suggested a refinement of policy. However, the variability in later stages pointed to unexplored areas that could benefit from further learning.

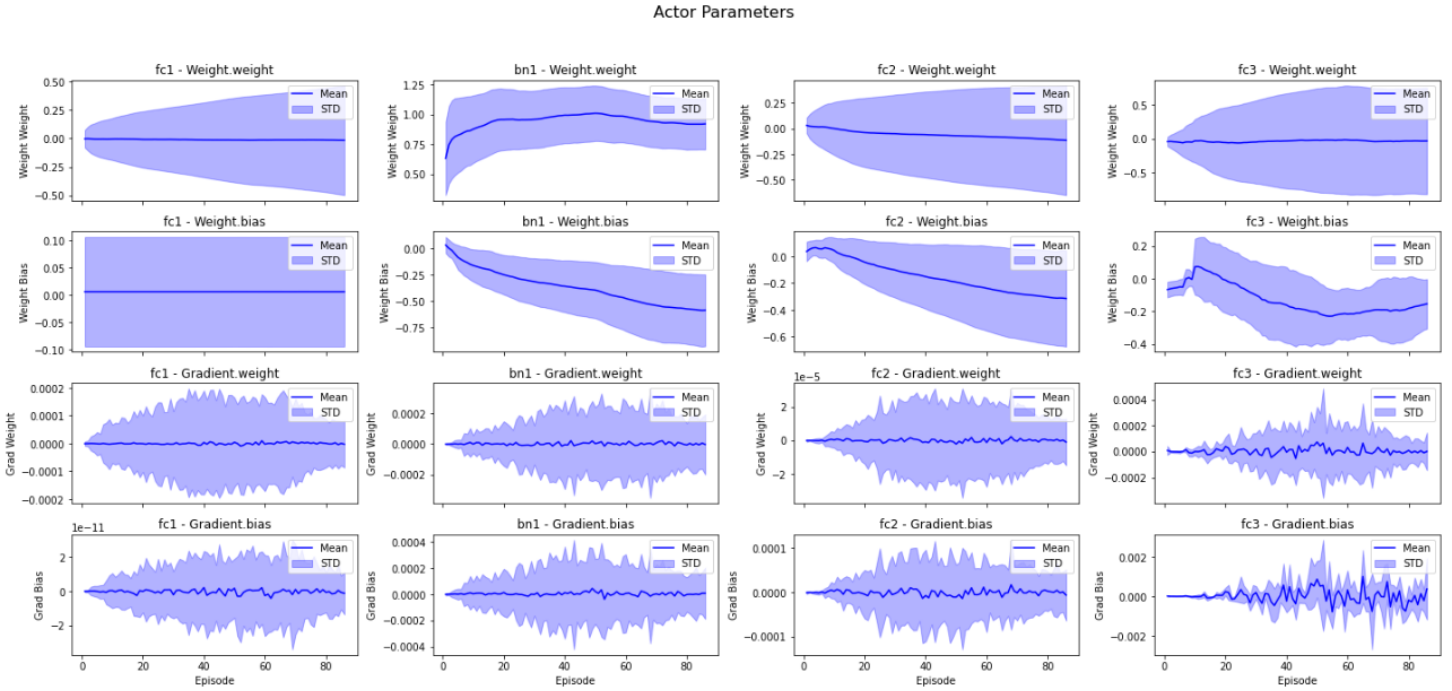
The importance sampling weights within PER played a pivotal role here, as they adjusted the sampling probability of experiences based on their TD error, allowing the agent to focus on more surprising or informative experiences. This mechanism remedies the bias introduced by prioritized replay, ensuring a more balanced and efficient learning process.

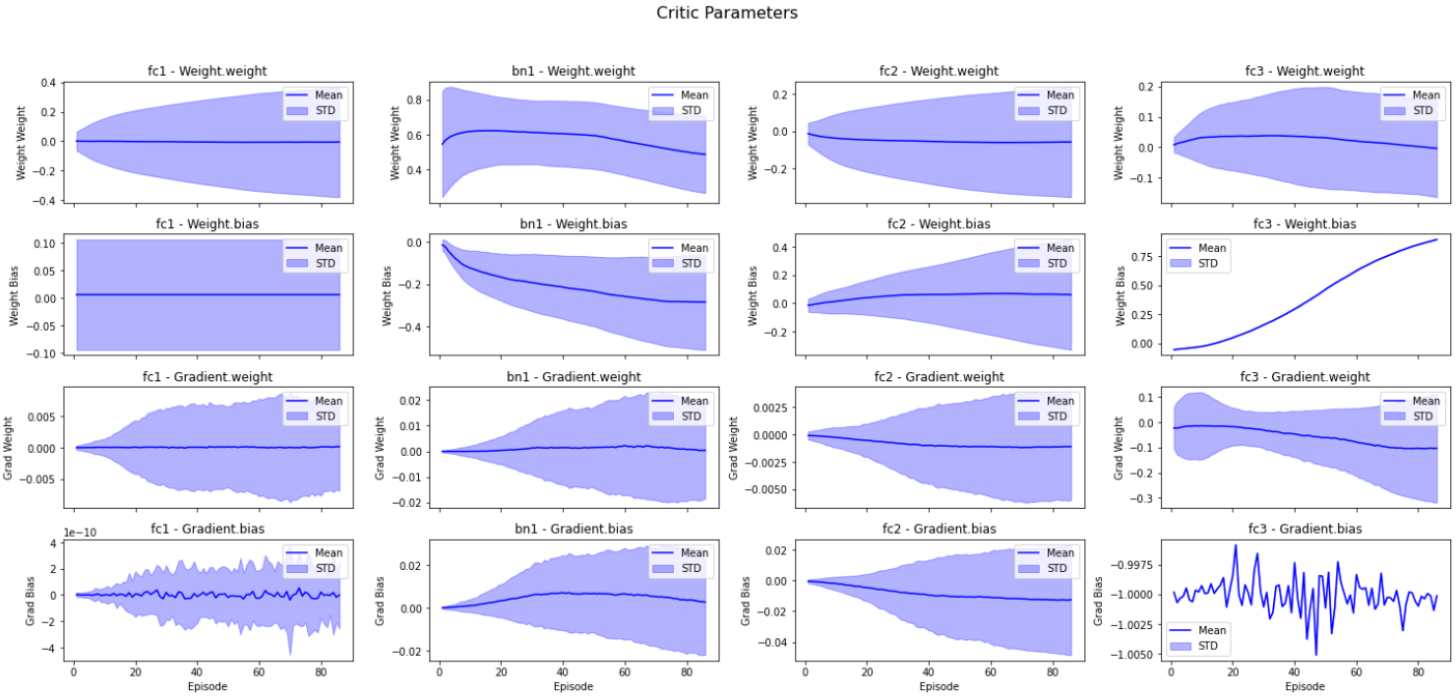


Weights and Gradients:

- **The Actor Parameters:** The weights of the actor display a stabilization of means, indicating that the policy of the neural network is becoming more consistent over episodes. Biases are also stable, suggesting that the network's threshold adjustments are adequate for the current policy. The gradients have relatively low noise but show occasional spikes, which could indicate an ongoing adaptation to new situations or a fine adjustment needed to optimize policy.

- **Critic Parameters:** For the critic, the trend of the weights is similar to that of the actor, with means that stabilize, suggesting that the network is evaluating the value of actions more accurately over time. Biases increase over time, particularly for the last layers, which may indicate that the network is adjusting its value predictions to account for more complex aspects of the environment. The gradients of the critic are more volatile than those of the actor, with notable variability, suggesting that the critic network continues to learn and adjust its value function throughout the training.





These observations show that the neural network continues to learn and refine its policies and value functions, with signs of convergence towards a stable strategy. However, the persistent variability in the gradients also indicates potential for additional learning or adjustments in the network's architecture or training.

Conclusion

Navigating through the project's various phases, from initial code development to the incorporation of sophisticated performance indicators, and integrating features such as noise decay and Prioritized Experience Replay (PER), has illuminated the nuanced equilibrium between exploration and exploitation essential in reinforcement learning. This endeavor has surfaced the pivotal challenge of training instability within the field. Despite this, our most efficacious training episode impressively achieved an average score of 30.0 in merely 86 episodes. The performance metrics revealed a consistent learning pattern: starting with an exploration phase to amass diverse experiences, transitioning to exploitation, and finally achieving convergence. This progression accentuates the vital need to balance exploration with exploitation to realize efficient learning outcomes. Through this project, we've delved into the intricate dance between various hyperparameters and feature integrations, enriching our understanding of their impact on the learning journey.

Future Work

This project has laid a robust foundation, yet the horizon of possibilities in refining and expanding its scope remains vast. To elevate the agent's performance and adaptability to complex environments, the following strategies are proposed:

Exploration of Advanced Estimation and Sampling Techniques: To further refine the agent's learning efficiency and policy effectiveness, an integrated exploration of advanced estimation and sampling methodologies is proposed. This includes:

- **Generalized Advantage Estimation (GAE):** Employing GAE to achieve a more nuanced balance between bias and variance in advantage estimation, potentially leading to more precise and stable policy updates.
- **TD Error vs. Monte Carlo Approaches:** Evaluating the trade-offs between TD error and Monte Carlo methods for reward estimation, aiming to identify the most suitable approach for varying environment dynamics and reward structures.
- **Q-Prop for Enhanced Sample Efficiency:** Investigating the potential of Q-Prop as an alternative to Prioritized Experience Replay (PER) or traditional Replay Buffer mechanisms within the DDPG framework, with a focus on optimizing sample usage for gradient calculation and potentially accelerating the learning process.

Systematic Hyperparameter Optimization: Leveraging automated tools like Hyperopt, Optuna, Ray Tune, ou GridSearchCV, for a more exhaustive search across the hyperparameter space can unearth configurations that significantly boost performance. This process could be enriched by incorporating Bayesian optimization techniques to intelligently navigate the hyperparameter landscape, potentially uncovering optimal settings faster.

Repeatability and Robustness Checks: Addressing the repeatability of results by conducting runs with varied random seeds is crucial for assessing the robustness of the learning algorithm. This step will help to understand the variance in performance due to initialization and stochastic elements of the training process, ensuring that the model's success is not a product of fortuitous seed selection.

Cloud Computing Resources: Transitioning to cloud-based platforms (e.g., AWS, Google Cloud, or Azure) equipped with powerful GPUs will drastically reduce training times. This enables the exploration of larger models and more extensive hyperparameter grids, pushing the boundaries of the agent's capabilities.

Parallel Experimentation: To expedite the exploration of hyperparameter spaces, running experiments in parallel across computational clusters or cloud resources is a valuable strategy. Distributed computing frameworks like Dask or Ray can facilitate the parallel execution of training sessions, reducing the time required to evaluate different configurations.

Alternative Reinforcement Learning Algorithms: Diving into newer or less conventional algorithms such as Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), or Twin Delayed DDPG (TD3) could offer insights into different learning dynamics and performance benchmarks in continuous control tasks.

Exploration of Model-Based RL: Integrating model-based elements into the predominantly model-free DDPG framework could offer new avenues for efficiency gains. By learning or approximating the dynamics of the environment, the agent could potentially achieve faster learning rates and better generalization.

Ensemble Learning and Policy Distillation: Implementing an ensemble of models to aggregate learning across different seeds or algorithmic strategies could stabilize learning outcomes and enhance performance. Further, policy distillation techniques can be explored to consolidate knowledge from multiple agents into a single, more efficient model.

Exploration of Multi-Agent Learning Dynamics: Investigating how multiple agents learning simultaneously in shared or competitive environments influence the learning process and outcomes could unveil strategies for cooperative or competitive behaviors, enriching the agent's applicability to a broader range of scenarios.

Continuous Integration (CI): Implementing CI workflows to automate model training and evaluation across various hyperparameter configurations can significantly streamline the development process. Tools like Jenkins, CircleCI, or GitHub Actions can be configured to trigger automated training runs, ensuring that changes in the codebase or hyperparameters are systematically tested.

Automated Model Selection: Beyond hyperparameter tuning, automating the process of selecting the most effective model architecture from a set of candidates can further optimize performance. Libraries such as AutoML, H2O.ai, or TPOT offer capabilities for automatic model selection, simplifying the search for the optimal architecture.

Experiment Logging and Tracking: Maintaining detailed logs of experiments, including the hyperparameters used, model performance metrics, and other relevant data, is essential for analyzing trends and making informed adjustments. Tools like MLflow and TensorBoard provide structured environments for tracking experiments, visualizing training progress, and comparing different runs.

References

DDPG Deep Deterministic Policy Gradient - Continuous Action-space

Q-Prop: Sample-Efficient Policy Gradient

GAE_Generalized Advantage Estimation

Proximal Policy Optimization Algorithms

NAF

OpenAI. (2024, February 14). Deep Deterministic Policy Gradient. Spinning Up in Deep RL. https://spinningup.openai.com/en/latest/algorithms/ddpg.html

Stable-baselines3. (2024, February 14). Reinforcement Learning Tips and Tricks. Retrieved from https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html

Annexe

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for