**Report: Deep Deterministic Policy Gradient (DDPG) Implementation**

**Introduction**

The Walker2d-v4 environment is a continuous control task from OpenAI's Gymnasium, where an agent controls a two-legged walker robot. The goal is to learn a policy that maximizes the robot's movement while avoiding falls. This environment has a high-dimensional state space and continuous action space, making it a challenging benchmark for reinforcement learning algorithms.

Deep Deterministic Policy Gradient (DDPG) is chosen as the algorithm because it is well-suited for continuous action spaces. DDPG is an off-policy actor-critic algorithm that leverages deep function approximators to learn deterministic policies. Its ability to use a replay buffer ensures sample efficiency, and the use of target networks helps stabilize learning.

**Methodology**

**Algorithm Design**

The DDPG algorithm consists of two neural networks:

1. **Actor Network:** Outputs deterministic actions based on the current state.
2. **Critic Network:** Estimates the Q-value for a given state-action pair.

The algorithm maintains target networks for both the actor and critic to reduce the variance of target value estimates. These target networks are updated slowly using a soft update mechanism.

A replay buffer stores experience tuples (state, action, reward, next\_state, done) for training, allowing the algorithm to reuse past experiences and break the correlation between consecutive samples.

**Pseudocode**

1. Initialize actor and critic networks, along with their target counterparts.
2. Initialize the replay buffer and environment.
3. For each episode:
   * Reset the environment and initialize total\_reward.
   * For each step:
     + Select an action using the actor network, adding exploration noise.
     + Execute the action and store the resulting experience in the replay buffer.
     + Sample a minibatch from the replay buffer if it contains enough data.
     + Update the critic by minimizing the loss between the predicted and target Q-values.
     + Update the actor by maximizing the Q-values for the actions it selects.
     + Soft-update the target networks.
     + Terminate the episode if the environment indicates the task is done.
   * Record the total reward and save the models periodically.
4. Plot the learning curve after training.

**Training Pipeline**

1. **Actor-Critic Networks:**
   * **Actor:** A three-layer fully connected network with ReLU activations and a final layer using the tanh activation function scaled to the action range.
   * **Critic:** A three-layer fully connected network that takes both state and action as input and outputs the Q-value.
2. **Replay Buffer:** Stores up to 1,000,000 experiences. Samples minibatches of size 64 for training.
3. **Exploration Strategy:** Adds Gaussian noise to the actions during training to encourage exploration.
4. **Target Network Updates:** Use a soft-update mechanism controlled by a parameter τ = 0.005.
5. **Model Saving:** The actor and critic models are saved every 100 episodes to ensure training checkpoints.

**Hyperparameter Choices**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Parameter** | **Value** | | --- | --- | | Episodes | 5000 | | Max Steps per Episode | 1000 | | Learning Rate (Actor) | 0.0001 | | Learning Rate (Critic) | 0.001 | | Discount Factor (γ) | 0.99 | | Replay Buffer Size | 1,000,000 | | Batch Size | 64 | | Soft Update Parameter (τ) | 0.005 | | Exploration Noise (σ) | 0.1 | |

This structure ensures that the DDPG implementation balances exploration and exploitation, stabilizes learning through target networks, and uses efficient sample utilization with a replay buffer. The saved models provide a way to monitor progress and restore training if needed.