STAT 421 Final Project:

Bipedal Walker Problem using Reinforcement Learning

By: Andy Callahan and Jaden Oca
https://github.com/lol-andy1/STAT421Project

Introduction:

The OpenAI Bipedal Walker Problem is a machine learning activity where one is tasked with training a "bipedal walker" to walk as efficiently as possible in a virtual environment. The walker consists of a hull-shaped body, two legs, and four controllable joints. By leveraging reinforcement learning algorithms, we aim to develop a self-balancing agent that can mimic human locomotion that converges efficiently.
To tackle the OpenAI Bipedal Walker Problem and train the bipedal walker to navigate its environment effectively, we utilize the **Deep Deterministic Policy Gradient** (DDPG) algorithm, and the **Ornstein-Uhlenbeck** (OU) process. The combination of these two techniques offers unique advantages and methodologies for addressing the challenges inherent in training a bipedal agent.
Reinforcement learning is a branch of machine learning that enables an "agent" to "learn" through interactions with its environment. Similar to the concept of neural networks that we covered in class, it involves trial and error over successive iterations, slowly tweaking model parameters, and reevaluating performance. Reinforcement learning differs from neural networks, however, through its use of specialized loss/reward functions to encourage or discourage certain behaviors, mincing the concept of operant conditioning from psychology.
A key challenge that reinforcement learning poses is generalization. This pertains to the ability of an agent to apply learned policies to situations it hasn't directly encountered, a process known as transferring knowledge. Achieving robust generalization is essential for reinforcement learning algorithms to perform effectively across diverse environments.In this project, we hope to leverage reinforcement learning techniques such as DDPG and the OU process to train a bipedal walker to navigate its environment and achieve robust locomotion capabilities.

Methods:

We used the OpenAI gymnasium library to simulate the walker and return observational data from the environment. The bipedal walker environment provides 24 different data points such as velocity of the agent and joint positions. The observational data is fed into our model which outputs torque values for how the agent should rotate its legs.

Deep Deterministic Policy Gradients (DDPG):
We trained the agent using the Deep Deterministic Policy Gradient algorithm. DDPG consists of two deep neural networks called Actor and Critic. The Actor network outputs actions while the Critic outputs a Q-value. The Q-value is a scalar that measures how good an action is in a given state. The goal of the Critic is to evaluate actions chosen by the Actor and estimate the future rewards an action can bring. Both the Actor and Critic are differentiable and have a corresponding objective function. The equation below is the loss function for the Critic.

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \right) \right)^2 \right]$$

Source: OpenAI Spinning Up

Inside the parenthesis, the term on the left is the Q-value which is the output of the Critic. The term on the right is the target Q-value which is computed using the Bellman equation. The loss is equal to the mean squared error of these two terms. The equation below is the objective function for the Actor.

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}} \left[Q_\phi(s, \mu_\theta(s)) \right].$$

Source: OpenAI Spinning Up

Rather than using gradient descent to minimize a loss value, the Actor uses gradient accent to maximize a Q-value. This is achieved by performing a forward pass through the Actor and Critic, backpropagating through both networks, then only updating the parameters of the Actor.
DDPG is known to be unstable and training with purely an Actor and Critic network will lead to suboptimal results. We will include two supplemental techniques which are experience replay and target networks. Normally, a training loop involves taking an action in the environment, observing the state, then updating the model using that single experience. Experience replay improves the training loop by storing all observations in a buffer. During each iteration, the model is trained on a batch of experiences sampled from the buffer. The experience buffer is similar to a data loader for image classification and greatly increases training efficiency.
A problem with Critic loss function is that the predicted and target Q-values are estimated using the same network. Training the Critic will inadvertently alter the target value and cause the predictions to chase a moving target. The problem is solved by introducing target networks which are a copy of the Actor and Critic. The target networks provide a stable target value and their parameters are soft updated.
Lastly, we need to incorporate an exploration strategy so that the agent thoroughly explores the environment. The most straightforward method is adding noise sampled from a normal distribution.

Ornstein-Uhlenbeck (OU) Processes:
OU processes are stochastic processes commonly used to model temporally correlated noise, which can aid in exploration during training. By adding OU noise to the actions chosen by the walker, we introduce variability that encourages exploration of the action space while still maintaining smooth and continuous movements. This noise injection helps prevent the walker from becoming overly deterministic in its actions, allowing it to discover novel strategies and adapt to changing environments more effectively. The OU process is defined by the equation below:

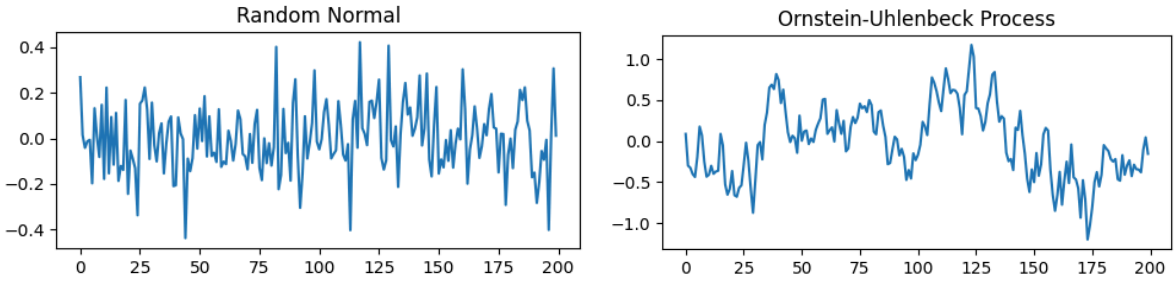
$$dx_t = \theta(\mu - x_t)\,dt + \sigma\,dW_t$$

Source: Wikipedia

The OU process involves four hyperparameters and they can be defined as follows:

- **θ (Rate of mean reversion):** The rate at which the agent returns to its mean. Higher values of theta would result in faster reversion, while lower values would result in slower reversion. In context, the walker is more willing to "explore" with slower reversion, while higher reversion means that the walker will be inclined to follow what it already knows.
- σ **(volatility):** The intensity of randomness that the OU process adds into the model. Smaller values mean less noise and less random deviation from the mean from the walker, while larger values introduce more noise into the model, resulting in more frequent deviations from the mean.

- Δ**t (step increment):** The step increment is a representation of how frequently the agent's knowledge is "updated." Smaller values of step increment make for more frequent updates and slower convergence while larger values mean less frequent updates and faster convergence.
- μ **(mean):** Center of the OU process. For the following analysis, zero (0) will be the only mean used.

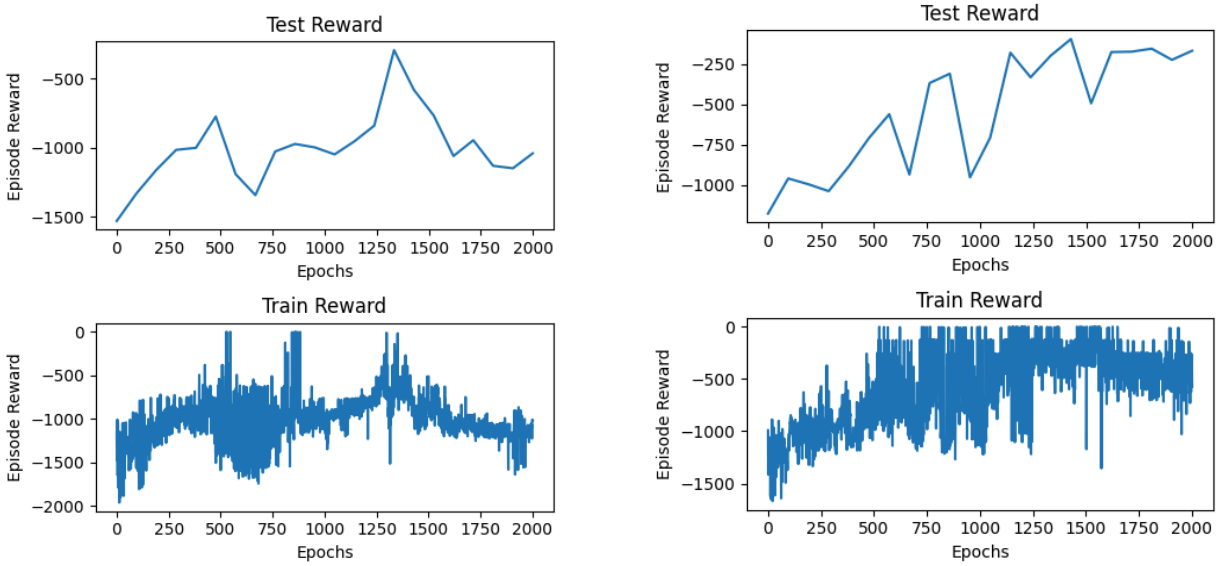


The figures above illustrate the difference between noise generated by a normal distribution and noise generated by the OU Process. The random normal data is sporadic and contains no visible patterns. An agent acting according to random noise is unstable and may not commit to the correct trajectory. On the other hand, OU noise is autocorrelated and drifts towards a certain direction. OU noise causes consecutive actions to have similar values, so training is more smooth.

Experiments:

Pendulum Problem

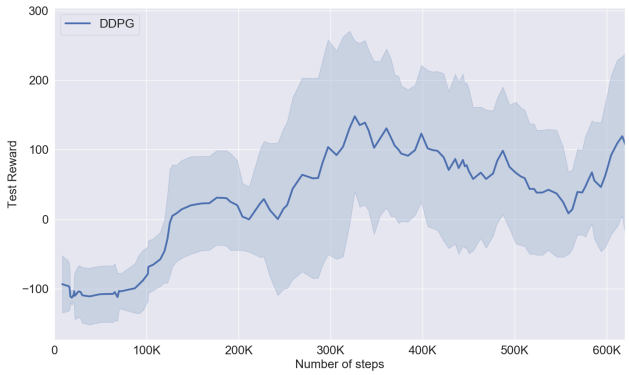
We will first demonstrate our DDPG algorithm on the pendulum environment, which is also from the Gym library. The pendulum problem involves swinging a pendulum to an upright position by applying torque to a fixed joint. The agent is penalized for how far the pendulum is from the target position and the amount of torque used. The agent was trained twice for the same amount of epochs, first using ordinary noise from a normal distribution, then with OU noise.



The figure on the left is the reward trace when training with ordinary noise. The figure on the right is the reward trace when training with OU noise. Training the agent using OU noise yielded a final reward of around -250. The agent converged to a solution by rotating the pendulum to a slanted position close to upright, then applying constant torque to maintain the position. On the other hand, training the agent with ordinary noise yielded a much lower reward of around -1000. According to the training reward, the agent displayed signs of learning and was able to achieve the target position. However, the training cycle was unstable and the agent was unable to maintain optimal performance. The reward trace shows no sign of convergence so the agent requires much more training before it can match the performance of the OU model. This experiment demonstrated that using OU noise will accelerate and stabilize training.

Bipedal Walker Problem

The bipedal walker environment rewards forward movement for a maximum score of 300, and penalizes falling over with a score of -100. Below is the reward trace of an existing solution to the bipedal walker problem which does not implement OU noise. The reward peaks at around 150 after 300k steps

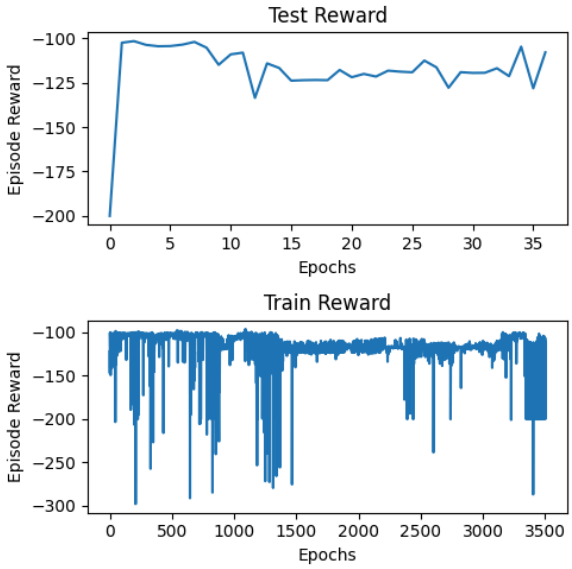


Source: Lonza, Andrea

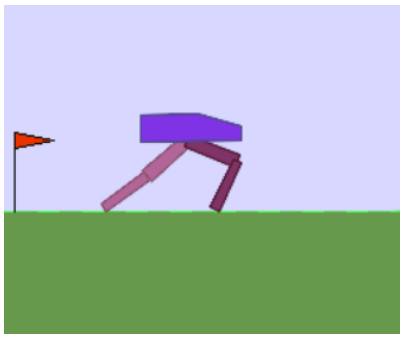
There are two main reasons for implementing the OU process in our modeling. Firstly, we wanted to see if it would affect the score of our walker. For our analysis, we experimented with 100 different combinations of hyperparameters in order to obtain the best performing combination in model performance. Our code tested every combination of the values shown below in order to obtain the best performing solution. Secondly, we wish to accelerate the training of the walker and reach convergence faster than 300k steps.

```
# Define parameter ranges to explore
thetas = [0.1, 0.3, 0.5, 0.7, 0.9]
sigmas = [0.5, 1, 1.5, 2]
dts = [0.1, 0.3, 0.5, 0.7, 0.9]
```

The figure below on the left is the reward trace when training with ordinary noise and the figure below on the right is the reward trace when training with OU noise. Unfortunately, neither model reached expected reward levels or learned to walk successfully. The notable difference between the two models is that the OU model has a smoother test trace. However, the effect of OU noise is inconclusive and further research is required to determine which model will converge faster.



The OU model converged to a suboptimal solution of remaining in a standing position. Further training or hyperparameter tuning is required to achieve more progress.



Conclusion and Future Work:

Although we were unable to make our model converge with a positive score, we were able to explore the effects of Ornstein-Uhlenbeck noise on DDPG algorithms. Our algorithm failed to converge due to the unstable nature of DDPG. The reward trace for the bipedal walker problem has high variance and fluctuates rapidly. The agent is unlikely to find high-value actions and capitalize on them.
Two solutions that may improve convergence are modifying the reward algorithm and implementing a prioritized buffer.
Firstly, the environment heavily punishes the agent for hitting the ground. The agent prioritizes balance and forgets about the main objective of walking forward. The reward system needs to be fine-tuned to encourage forward movement. The model can overcome local minima by letting the agent act more risky.
Secondly, the agent spends most of the time experimenting with different actions. The experience buffer is filled with actions that do not have much learning value. It is difficult to reinforce good behavior since effective actions rarely get sampled with the batch. The experience buffer needs to include a priority level to every entry. Experiences that encourage walking will be sampled more often and the model will be weighted towards the desired behavior.

Works Cited:

"Ornstein–Uhlenbeck Process." Wikipedia, Wikimedia Foundation, 26 Apr. 2024, en.wikipedia.org/wiki/Ornstein%E2%80%93Uhlenbeck_process.

Lonza, Andrea. "Reinforcement Learning Algorithms with Python." O'Reilly Online Learning, Packt Publishing, www.oreilly.com/library/view/reinforcement-learning-algorithms/9781789131116/da3ddd78-a1ce-4474-888b-9e100cadd997.xhtml. Accessed 1 May 2024.

"Deep Deterministic Policy Gradient." *Deep Deterministic Policy Gradient - Spinning Up Documentation*, spinningup.openai.com/en/latest/algorithms/ddpg.html. Accessed 1 May 2024.