REINFORCEMENT LEARNING DRIVEN WALKER VIA PPO (PROXIMAL POLICY OPTIMIZATION)

IGNACIO VILLASMIL

Please refer to the code while following this write up.

In this PPO Reinforcement Learning algorithm designed for a walker, Deepmind's Control Suite is used, which includes the DM Control software package and a Mujoco simuator. A similar structure to the Spinning Up code (which uses OpenAI Gym) is used in this case, but again, using DM Control instead of the OpenAI Gym, which have very different frameworks. Neural networks are used for the actor and critic in the PPO model. A Gaussian actor is used in this case, with the forward pass on the actor/controller returning mean and standard deviation values for a Normal distribution; an action can then be sampled from this distribution. Similar to as done in Spinning Up, a buffer is used to keep track of the history of the values of importance for updating the actor and critic in PPO. The buffer keeps track of the history of actions taken, observations/states, advantages, rewards, returns (i.e. rewards-to-go), log probabilities, and values from the value function (critic) throughout all trajectories taken. It is important to note that the advantages are normalized, which is a crucial step in ensuring that the neural networks for the controller (actor) and value function (critic) are properly trained and updated.

The policy loss is computed through a function that, first, computes a new log probability via a forward pass of the policy/actor/controller for a given state observation and action. The old log probability, along with this newly computed one, are used for the importance ratio. It is important to note that in PPO this importance ratio should be close to 1; this helps ensure that the controllers have very similar log probabilities for the actions taken by the old controller (this results in the new controller having a very similar state visitation frequency as the old controller). A set clip ratio variable is used to do this. The value loss is computed by a simple mean squared error between the returns (rewards-to-go) and the forward pass of the value function (critic) on a given state observation input.

Using an Adam optimizer for both the actor and the critic, an update function is used, over several iterations, to compute the policy and value losses, compute their gradients for back-propagation, and perform optimization steps for the Adam optimizers. This effectively updates/trains the actor and critic neural networks in an iterative fashion each time this update is called.

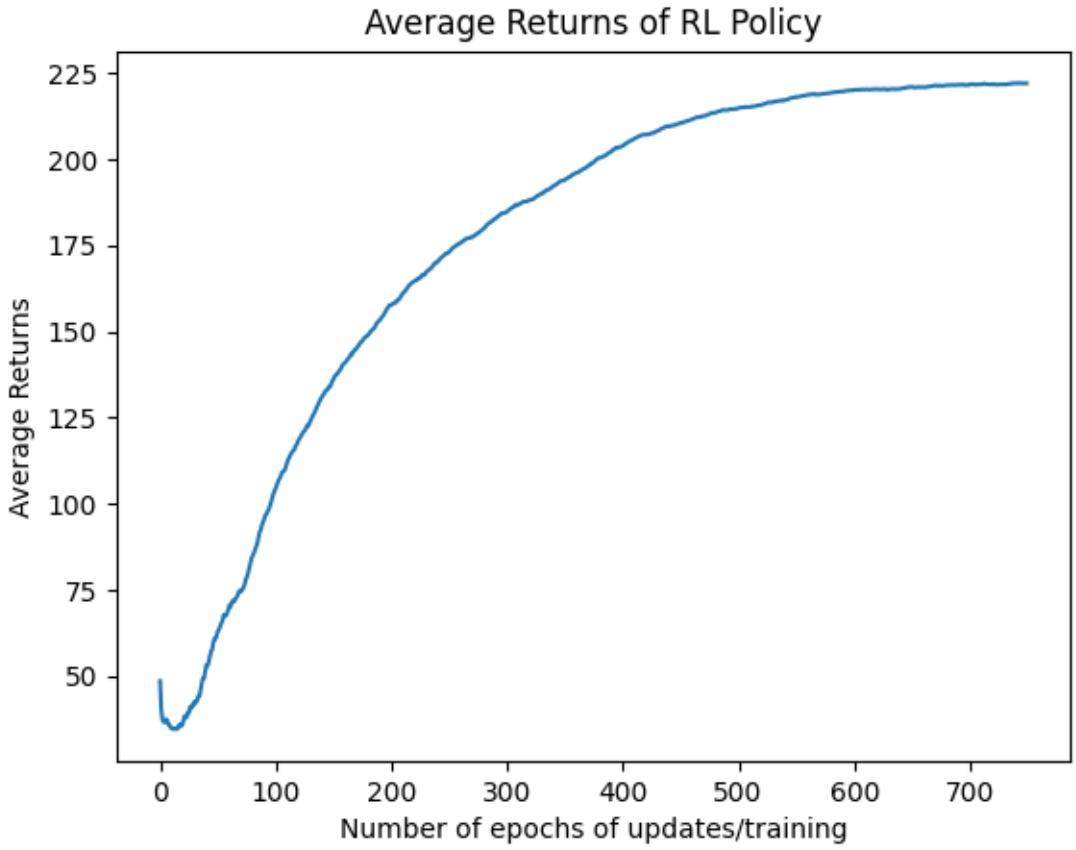
These neural networks, and the PPO overall, were trained over 750 epochs/trajectories, each for a length of 1000 time steps. In each time step, a step was taken in the environment by the actor and critic, resulting in a new set of rewards, state observations, actions, and log probabilities for actions. This would continue until the trajectory finished. The returns/rewards-to-go would also be saved and

tracked over time with the intent of plotting and analyzing how the returns changed over time as the PPO and neural networks were trained and updated.

For the calculation of returns/rewards-to-go, I used a discount factor of 0.99. Instead of a ReLU activation function, a hyperbolic tangent (TanH) was used as the activation function; when testing the model, this activation function seemed to lead to better results.

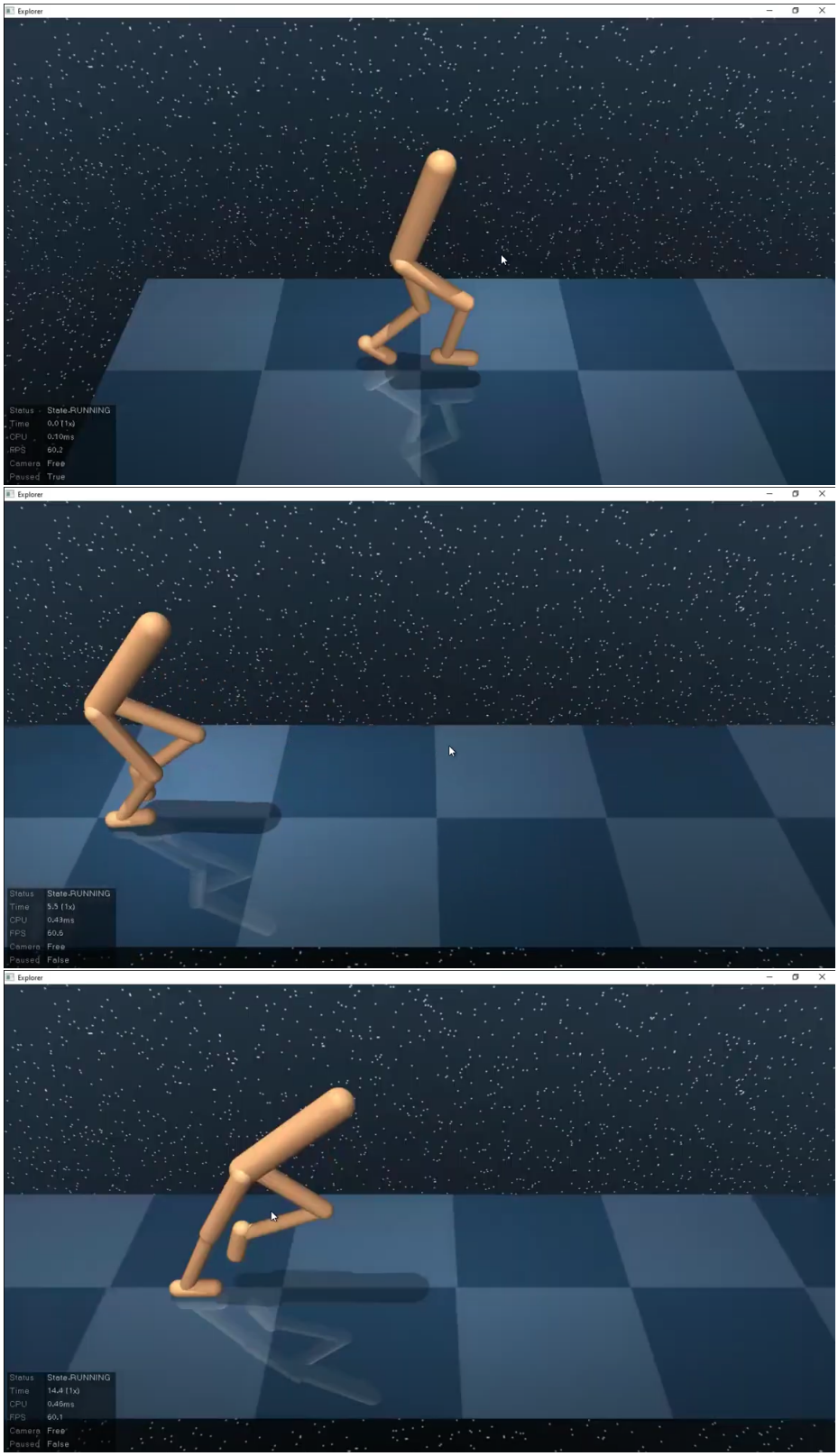
Results

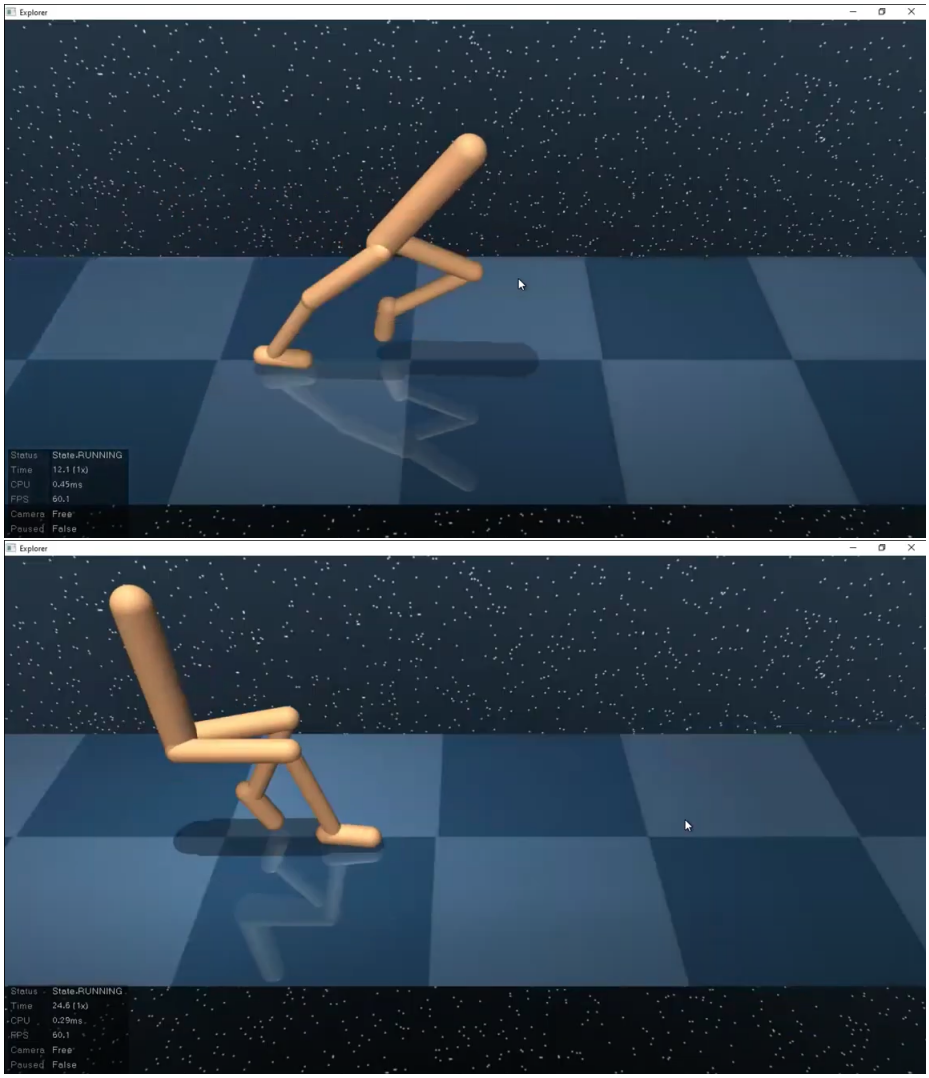
After keeping track of the returns (rewards-to-go) over time, as the PPO is trained and updated more as additional trajectories are sampled, the plot below demonstrates how the returns change as the actor and critic are both further updated, trained, and improved on more trajectories. Overall, as seen on the plot, the returns increase as more epochs of training are done on more trajectories. The x-axis represents the number of epochs of training (also equal to the number of trajectories sampled). The y-axis represents the average return.



In addition, a link to a video of the trained actor neural network for the PPO being used to control the action of a walking humanoid is included below. This video is able to show that the humanoid can move forward effectively as it was trained. Screenshots throughout the video are also included below.

Link: `https://drive.google.com/file/d/1KkJk8IJGeEhpcdXJrmyeuF7hKjzVXiei/view?usp=sharing`


Explorer
Status State-RUNNING
Time 0.0 (1x)
CPU 0.10ms
FPS 60.2
Camera Free
Paused True
Explorer
Status State-RUNNING
Time 5.5 (1x)
CPU 0.43ms
FPS 60.6
Camera Free
Paused False
Explorer
Status State-RUNNING
Time 14.4 (1x)
CPU 0.46ms
FPS 60.1
Camera Free
Paused False


Explorer
Status State:RUNNING
Time 12.1 (1x)
CPU 0.45ms
FPS 60.1
Camera Free
Paused False
Explorer
Status State:RUNNING
Time 24.6 (1x)
CPU 0.29ms
FPS 60.1
Camera Free
Paused False