Part 3 - Proximal Policy Optimization

Runar, Lars and Årne

1. Take an implementation of your actor-critic algorithm and transform it by changing its objective to the equivalent of the clipping objective of Equation (7).

   We've previously implemented the actor-critic algorithm to solve a reinforcement learning problem. Now we are tasked with implementing the Proximal Policy Optimization algorithm, PPO. As PPO is an actor-critic method, it will be reasonably simple to modify an existing actor-critic algorithm to become PPO. There are two main modifications which need to be done.

   Firstly, the loss function of the actor network needs to be changed to Equation (7) (or possibly Equation (9)) in the Proximal Policy Optimization paper, $L_t^{\text{CLIP+VF+S}}(\theta)$. However, in our implementation we are using separate networks for the actor and critic, as well as omitting the entropy bonus term, reducing the equation down to Equation (7), $L_t^{\text{CLIP}}(\theta)$. Secondly, we need to modify the general training function to fulfill the requirements of PPO. First we run several parallelized actors which gather different trajectories from the environment. These are then used to compute `td_error` and `advantage` for each timestep, which are values used later to optimize the critic and actor respectively. Afterwards we subsample the collected values using mini-batching and perform several epochs optimization for both actor and critic.

2. Run your algorithm on the `InvertedPendulum` environment in gymnasium. Explore the results for different configurations of relevant parameters.

   Proximal Policy Optimization is a heavy algorithm, and there are several inner loops consisting of gather trajectories, computing values and optimizing loss functions. Therefore training takes a substantial amount of time. There are also a great amount of hyperparameters to tune in this algorithm. As we do not have the capability of running grid-search over all different hyperparameters we have to compromise and choose which hyperparameters we consider "most important". Building upon the conclusion from the paper we decided to tune the hyperparameters learning rate `lr` $\in \{5 \cdot 10^{-6}, 1 \cdot 10^{-5}, 3 \cdot 10^{-5}\}$, number of epochs `n_epochs` $\in \{10, 25\}$ and number of parallel actors run `episodes` $\in \{5, 10\}$. The other settings are pulled from the paper, and are $\epsilon = 0.2$, $\gamma = 0.9$ and $\lambda = 0.95$. We ourselves decided on using a mini-batch size of 256, and as we do not have to possibility to run our models for 1 million iterations (like they do in the paper), we instead only run for 5000 iterations.

3. Compare the results across different hyperparameter settings and against the results presented in the paper.

   Results from hyperparameter tuning are plotted under figure 1. From these, we can conclude that almost every model seem to learn reasonably well! However, different values do help decide how fast a model learns, and its probability of "crashing" and hitting a early local optima. The green and brown agents both have a low learning rate, but as brown runs several more actors and run for more epochs every iteration it in general learns faster. We also see that red and blue have the same (higher) learning rate, but here blue benefits from a low number of parallel actors and low epoch count. Red seem to take too large gradient steps every iteration which results in stagnation of learning after just 1000 iterations. This does not happen case for the blue agent.

   Directly comparing our results to the results in the paper brings with it some challenges, as they run their models for 1 million iterations. However, as our models do show the capability of learning a policy well, one could extrapolate and imagine that our models reach similar results given enough iterations. This is substantiated by the paper reaching around average 400-600 timesteps per episode within the first 100000 (one-hundred thousand) iterations, which would not be outrageous to hypothesize our models could do as well given their learning trajectory.

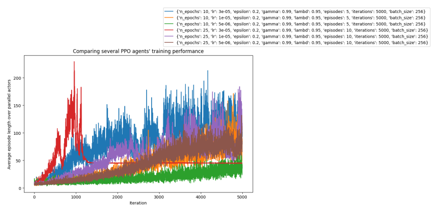

Figure 1: Results of hyperparameter tuning of Proximal Policy Optimization