Unity Reacher Arm Control

Austin Silveria

We examine the PPO algorithm applied to the Unity Reacher environment. We then analyze the performance of our agent and propose ideas for future improvements.

1 Learning Algorithm

Proximal Policy Optimization is a policy gradient method that interacts with the environment for a fixed number of steps, n, then optimizes with a gradient ascent step for a specified number of optimization epochs. For each epoch clips the original update function to filter large updates and keep the agent's learning stable. We implemented this algorithm using an Actor-Critic method that trained two different networks, an actor and a critic. From these two networks, the action advantages were computed using the state values and actions, then the policy loss combined with the value loss was back propagated through the networks.

Gamma	.99
Tau	.95
LR	3e-4
Rollout	1024
Optimization Epochs	10
Batch Size	32
PPO Clip	.2
Gradient Clip	5
EPS	1e-5

TABLE 1. The chosen hyperparameters for our implementation.

Table 1 shows the eight chosen hyperparameters for our implementation. A standard value of .99 was chosen for gamma to discount future rewards when computing the expected state-value. A tau value of .001 corresponds to the "softness" of an update to the target model with respect to the local model. A smaller value pushes the weights of the target model's network in the direction of the local model a smaller amount than a larger value. The rollout was chosen to be

1024 states because a full episode was just around 1000 for this environment. Optimization epochs is the number of times the network is optimized with the current rollout of states. The batch size corresponds to the number of experiences sampled at a time and can be adjusted based on the GPU memory of the running system to maximize utilization if running in parallel. The PPO clip was set based on the hyperparameters used in the paper, and gradient clip ensured even better control of update size to keep the learning stable. EPS was specified to be 1e-5 for the Adam Optimizer, a value greater than the default of 1e-8. A larger EPS causes the weight changes to be smaller in updates, leading to more stabilized training for our agent.

2 Performance Analysis

With a state dimension of 33 and an action dimension of 4, two networks of two fully connected hidden layers were used to approximate the optimal policy for choosing actions and valuing states. Both layers consisted of 64 neurons and used the ReLU activation function. The small networks allowed the network to train with a small amount of compute and avoid get stuck in patches while training. Figure 1 illustrates the best learning we recorded of an agent using the PPO algorithm. The agent's learning was very stable and maintained a strong positive slope of learning without crashing or getting stuck.

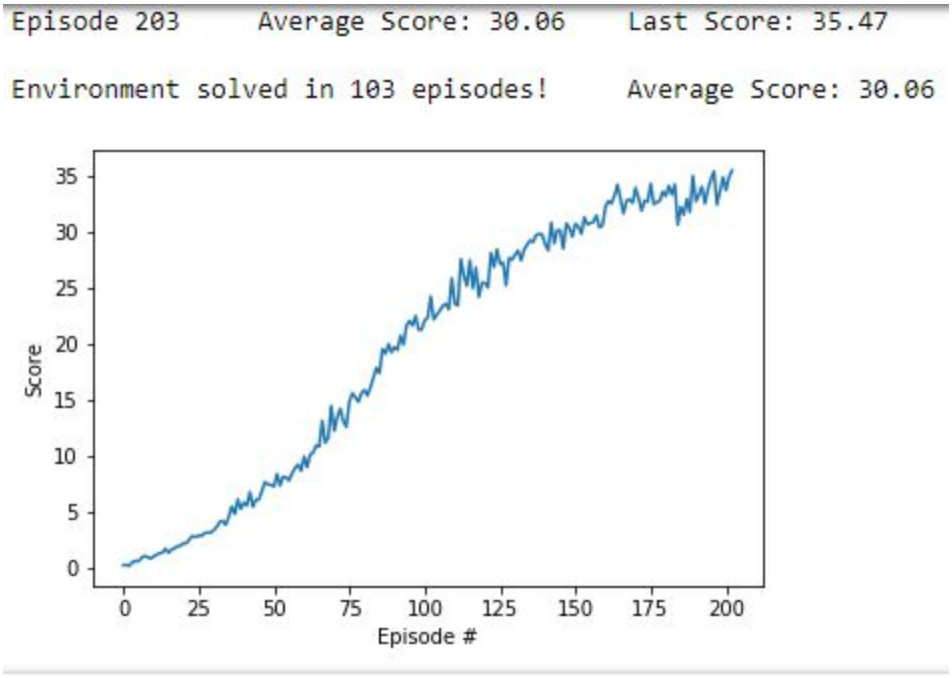

Figure 1: Best PPO performance on the Unity Reacher Arm Control environment using the hyperparameters shown in Table 1.

3 Future Work

Stability for Neural Networks: Our implementation suffered from high variance in different test runs of training the agent. The number of episodes taken to converge on the optimal policy differed by a large amount for each test run, possibly pointing to an issue of the neural networks responding differently to longer sequences of states, which can vary highly for each test run. Exploring hybrids of on and off policy methods may allow networks better control over what states they see in order to stabilize their learning.