Reacher Continuous Control – Project 2

Problem description

The Reacher environment to solve is a double-jointed arm that must maintain its hand position given a moving target. The end of the arm is the equivalent of a hand and the agent controlling the arm is given a reward of +0.1 for each time step the hand is in the target location. The goal is for the agent to maintain this correct position for as long as possible. The environment is considered solved when the average score over 100 episodes is at least +30.

Observation space

The observation space is a vector of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. If using an environment with multiple agents, then, say there are 20 agents, the observation returned from the environment is a vector of size [20 x 33].

Action space

The action space for the double-jointed arm is a vector of size [1 x 4], corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

Proposed approach

Q-Learning techniques cannot solve problems with continuous observation and action spaces so policy-based methods must be considered. For this project, the chosen method was the Advantage Actor-Critic algorithm solving the multi-agent environment.

The rest of the report covers a brief overview of policy-based methods, their advantages over Q-Learning, and the pros and cons of some methods compared to others. Then an overview of the chosen network is presented followed by the results the network achieves. A conclusion with ideas about future work is presented at the end.

Policy Gradients

From [1], the agent directly learns the optimal policy, without having to maintain a separate value function estimate. This intuitively means that when a state is fed to the agent, the agent directly returns an action to perform. There is no comparison of network output values to see which action is the best as there is in Deep Q-Learning or other value-based methods.

It is common to represent the policy using a neural network where the environment state is the input. The output of the network depends on the type of action space. For discrete actions, for example the agent only has the choice of up, down, left, and right, the agent outputs the probability of taking each action for that state. Sampling from this action distribution returns the action to take at that time. For a continuous action space, for example where the agent can output how far to turn a wheel and how much to accelerate/decelerate, there is no probability of taking either action as each action is required at each timestep. Here the agent outputs the mean and variance for each action, allowing a distribution to be generated for each action and then sampled.

From [2], another advantage is not having to explicitly explore the environment. As actions returned are a probability distribution (whether discrete or continuous) the distributions start off uniform, corresponding to random behaviour. As the network gains more certainty the actions become more deterministic as the action space probability distributions have converged to their true distribution.

Advantage Actor-Critic

The basic REINFORCE algorithm from the policy based method has a few main problems [3]: it is inefficient as many trajectories are needed for convergence; the gradient estimate can be very noisy and may not be representative of the policy; and there is no clear credit assignment. A trajectory that led to an overall success may contain many bad actions. These issues have been addressed by subsequent techniques.

The high variance experienced by REINFORCE, due to using Monte-Carlo estimates for the expected return, can be removed by using the TD error for a trajectory, where an estimate of the expected return is based on the next state. A halfway solution described by [2] is to use multistep Bellman rollouts. This proposes to bootstrap n steps into the trajectory, not to the end of the task but further than the single step proposed with the TD error computation.

Further improvements to counter the high variance are to use a baseline [4], subtracting this baseline from the expected reward. This makes the reward smaller, leading to smaller gradients and therefore reducing the noise in the gradient estimates. This leads to Actor-Critic methods where the baselines are calculated by different techniques. One such technique is to calculate the Advantage function which is the Action-value function (Q value) minus the State-value function (V value) for the same state. Using this Advantage to scale the gradient leads to the Advantage Actor-Critic (A2C) algorithm.

A2C Network

To setup the network for the A2C algorithm, a shared body is used but with multiple heads. The heads output the values needed for the action distribution and value function estimates.

The chosen architecture for this solution is to have a shared body of 2 fully connected hidden layers bot of size 128 and using the ReLU activation function, referred to as the base. To output the mean of the action distribution, the output of the base is fed into a fully connected layer with 4 outputs as each action is a [1 x 4] vector. The activation function used is Tanh to squeeze the actions into the [-1, +1] range. The variance of the action distribution is found by its own network head. The output of the base is fed into a fully connected layer with 4 outputs but uses the Softplus activation function, which is a smooth version of ReLU. Finally, the third head of the network estimates the value function for the state. It is fully connected to the base and returns the value function.

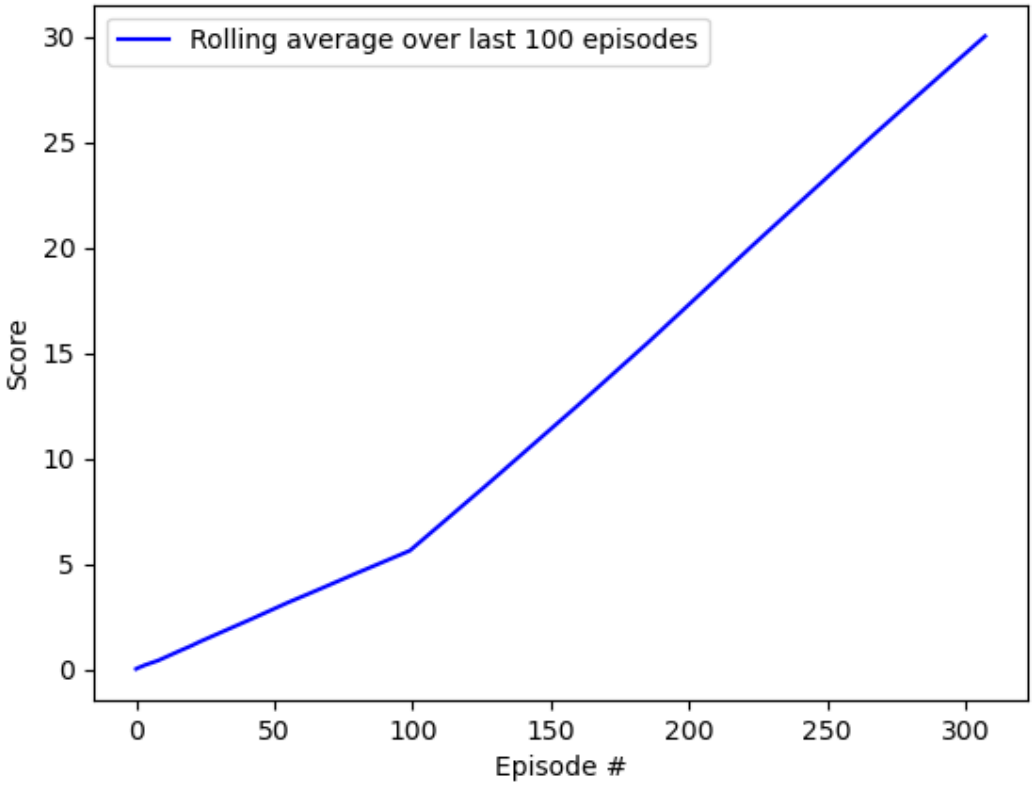
The mean and variance are used to form a Normal distribution from which actions are sampled. Actions are clipped to be in the range [-1, +1] to ensure all actions are legal for the environment. From the distributions, the entropy can be generated which is a measure of the uncertainty in a probability distribution. This is used in learning to help prevent the network from getting stuck in a local optimum [2].

Hyperparameters

The following hyperparameters were used in the model architecture. Gamma, the reward discount was set to 0.99. The learning rate for the model optimiser was set to 1e-2. The entropy weight was set to 0.01 and the critic weight in the update algorithm was set to 1.

Results

After applying the Advantage Actor-Critic method the environment was solved in 308 episodes where the average score was 30.02.



Conclusion and Future Work

The agent performed well but took longer to solve the environment than the DDPG example provided by Udacity. Future work to improve this would be to use D4PG or to consider Proximal Policy Optimisation (PPO) as implementing more state-of-the-art techniques should lead to better results. Another important aspect is to explore the effect all the hyperparameters have in training and to consider using Generalised Advantage Estimation to improve the effect from the Bellman Rollouts.

References

[1] Udacity lesson 2 – Introduction to Policy-Based methods

[2] Skow – Policy Gradients Methods Tutorial https://www.youtube.com/watch?v=0c3r5EWeBvo

[3] Udacity lesson 4 – Beyond REINFORCE

[4] Chris Yoon, Understanding Actor-Critic Methods and A2C
https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f