Udacity Continuous Control Project Report

by: Jon F. Hauris

Introduction

In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible. The environment used the Unity Reacher environment.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

There were 2 options for this project. This code implements option 2 which has 20 simultaneous agents running. An initial effort was undertaken to run option one (a single agent version) but training was extremely long and not promising. Because of the 20 agents providing additional information to the learner, training was achieved much faster.

The problem is solved for the second version of the environment when the agents must get an average score of +30 (over 100 consecutive episodes, and over all agents). Specifically, after each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 20 (potentially different) scores. We then take the average of these 20 scores. This yields an average score for each episode (where the average is over all 20 agents).

Learning Algorithm

For this project I used the Proximal Policy Optimization (PPO) algorithm with Actor-Critic networks to train an agent to remain in close contact with a designated coordinate. This was modified for continuous action space and was based on code by Shangtong Zhang and Jeremi Kaczmarczyk.

The outline of the algorithm is:

1. Collect a group of trajectories based on PI(theta)
    a. Initialize theta' = theta
2. Calculate the gradient via: g = DEL(theta') {L(Clip-Sur)(theta', theta)
    a. where L(Clip-Sur) is the Surrogate function which is clipped and a function of the present policy distribution (theta) and the previous policy distribution(theta')
3. Update: theta' = theta' + alpha*g
4. Repeat steps 2.)-3.) a few times (~2-4-8)
5. Set: theta = theta'
    a. Repeat from step 1.)

The code to implement this consists of three basic parts:

- MainCode.ipynb: which performs initialization and training
- ppo_agent.py: a PPOAgent which creates the step and learning abilities
- network_model.py: a PPONet which instaniates an actor and critic network

The input to the system is the observation vector consisting of the 33 state variables. These observations are fed into two separate neural networks: an actor net and a critic net. These nets are instantiated in the PPONet class of the network_model.py code. Each actor and critic net is constructed of a three layer fully connected neural network. The 33-element state vector is fed into each of these nets and the actor outputs 4 control signals which are bound between -1 and +1 via a tanh function. The critic net outputs a single real number state value. This output is directly from the linear network with no activation function required. This state value is used to calculate the Advantage which is used in the PPO algorithm to reduce signal variance. The action outputs from the actor net are used to create action probabilities. The action values are used as a mean value and then samples drawn from a normal distribution with variance = 1.

<u>Defined Classes and Functions</u>

The following is an output description of the actor and critic model from the PPONet class.

```
PPONet(
  (actor_body): FullyConnectedNetwork(
    (linear1): Linear(in_features=33, out_features=512, bias=True)
    (linear2): Linear(in_features=512, out_features=512, bias=True)
    (linear3): Linear(in_features=512, out_features=4, bias=True)
  )
  (critic_body): FullyConnectedNetwork(
    (linear1): Linear(in_features=33, out_features=512, bias=True)
    (linear2): Linear(in_features=512, out_features=512, bias=True)
    (linear3): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

The size of the hidden layer was chosen after trying various values. 512 gave good results.
Main hyperparameters
- Discount rate - `0.99`
- Tau - `0.95`
- Rollout length - `2048`
- Optimization epochs - `10`
- Gradient clip - `5`
- Learning rate - `3e-4`
- ppo clip – `0.2`

The ppo_agent.py code implemented the PPOAgent class. This class instantiated the initialization and step (=learn) capability. Basically this agent calculates the Advantage over a number of time steps and uses the GAE formulation to progressively combine later quantities of the reward. Also, the PPO method uses a long sequence of samples obtained from the environment and then the advantage is estimated from this sequence. During this period no training updates occur. After this calculation the Advantage is used in training for several epochs.

Also, instead of using the gradient of the scaled log probabilities of the actions, the PPO uses the ratio of the present action probability distribution to previous action probability distribution.

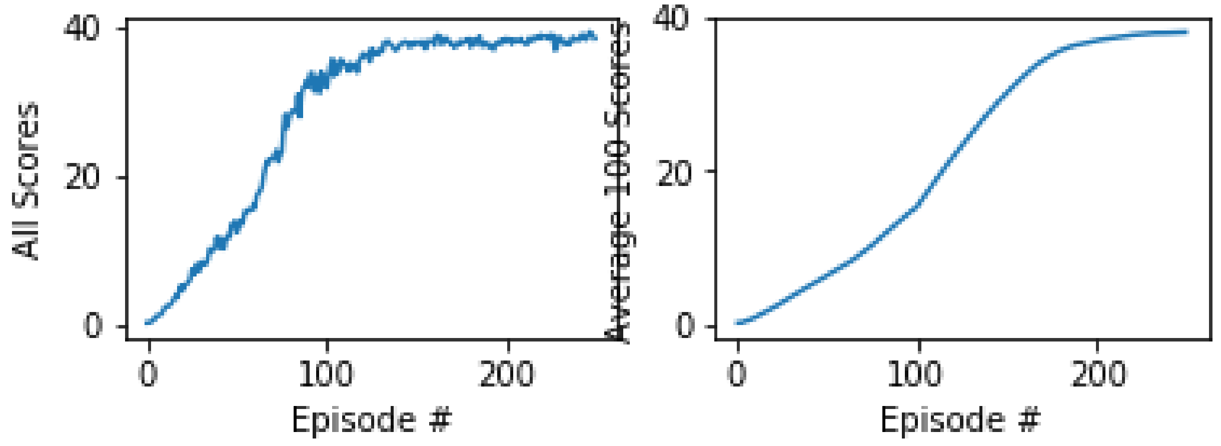
Running the MainCode.ipynb will run the training and inference engines of the code. This code keeps track of the mean reward for each epoch and the average of the last 100 epochs. Once the reward attains 30.0 or greater the network considers it a success. At this point the code is saved (as BestModel.pth). The process continues for the designated number of epochs and if the model is further improved, this result replaces the previously saved model.

Results

The network achieved a 100 epoch average >= 30.0 at around epoch 206. Upon further running (til epoch 250) further improvement was attained of 38.05. A number of runs were made, and this number varied from around 33 to 40. However, each run was successful im meeting the requirement. The following shows the output at epoch 250:

```
100%|██████████| 250/250 [1:27:32<00:00, 21.61s/it]
Episode: 250 Total score this episode: 38.4069 Last 100 average: 38.0482
```

Additionally, the following plot depicts the history of the networks progress through training, again sho wing a successful completion.



Suggestions for Future Work

- The primary hyperparameters explored were the gradient clip and size of the hidden layer. Further exploration of this process could be undertaken.
- Normalization of states, rewards, and state values could provide significant improvement
- Different values of lambda for GAE processing can be explored.
- Exploring how to use the PPO on the single agent case would be interesting. It is interesting that the 20 agents helped so much, and I wonder if this method would even work on the single agent case.
- Compare this result to TRPO, A2C, DDPG, and D4PG. PPO is supposed to be the best of these but it would be good to bench mark these.