Solving Continuous Control environment using Deep Deterministic Policy Gradient (DDPG) agent

Author: Henry Chan

Learning Algorithm

The learning algorithm I chose is based on the paper *"Continuous control with deep reinforcement learning"* with my own modification based on my experiment and intuition in solving the "Research multiple ML Agents" environment. My DDPG implementation is modified from the vanilla DDPG agent in solving single agent pendulum environment.

This project is an extension of the previous project in applying Deep Q-Network (DQN) to solve single agent navigation environment. The difference is this project has a more complex environment with continuous action spaces and multiple agents. To learn more about the basic of DQN, please feel to refer to my previous project .

DDPG is a model-free policy based learning algorithm in which the agent will learn directly from the un-processed observation spaces without knowing the domain dynamic information. That means the same algorithm can be applied across domains which is a huge step forward comparing with the traditional planning algorithm.

In contrast with DQN that learn indirectly through Q-values tables, DDPG learns directly from the observation spaces through policy gradient method which estimates the weights of an optimal policy through gradient ascent which is similar to gradient descent used in neural network. Also, policy based method is better suited in solving continuous action space environment.

DDPG also employs Actor-Critic model in which the Critic model learns the value function like DQN and uses it to determine how the Actor's policy based model should change. The Actor brings the advantage of learning in continuous actions space without the need for extra layer of optimization procedures required in a value based function while the Critic supplies the Actor with knowledge of the performance.

To mitigate the challenge of unstable learning, a number of techniques are applied like Gradient Clipping, Soft Target Update through twin local / target network and Replay Buffer. The most important one is Replay Buffer where it allows the DDPG agent to learn offline by gathering experiences collected from environment agents and sampling experiences from large Replay Memory Buffer across a set of unrelated experiences. This enables a very effective and quicker training process. Also, Batch Normalization plays an important role to ensure training can happen in mini batch and is GPU hardware optimization friendly.

Model Architecture

At the beginning of training, I used 20 individual DDPG agents corresponding to 20 agents in the environment and a single Replay Buffer which is shared by all DDPG agents. I then switched to a single DDPG agent with one Replay Buffer that has experiences collected from all 20 environment agents. I believed it’s a more intuitive way to train as that means I only need to train one single “brain” instead of 20 individual “brains”. Training 1 brain will definitely be quicker than 20 brains. I think one “super intelligent brain” can perform better overtime. My assumption seems to have paid off after I made the switch as is shown in the graph below in training result. The learning process is more stable and encouraging. As a retrospective, 20 individuals DDPG agents might be more favourable in a distributed training environment but I haven’t tried.

The Actor model is a neural network with 2 hidden layers with size of 400 and 300, Tanh is used in the final layer that maps states to actions. Batch normalization is used for mini batch training.

The Critic model is similar to Actor model except the final layer is a fully connected layer that maps states and actions to Q-values.

Hyperparameter

The learning rate for both Actor and Critic is 1e-3 with soft update of target set to the same 1e-3. Gamma discount is 0.99 with weight decay set to 0. Replay Buffer has size of 1e6. Batch size is 1024 with max time step of 1000 in each episode. A large Replay Buffer is crucial in the success of the learning. Number of time steps plays an important role as the agent needs to have enough time steps to have a good balance between exploitation and exploration.

Training Result

The DDPG agent takes 102 episodes to achieve an average rewards of 30 scores in about 1 hour 30 mins trained in GPU instance. The score becomes stable when it reaches 37 at about 40th episode. A sharp increase of score is observed between 20th and 30th episode.

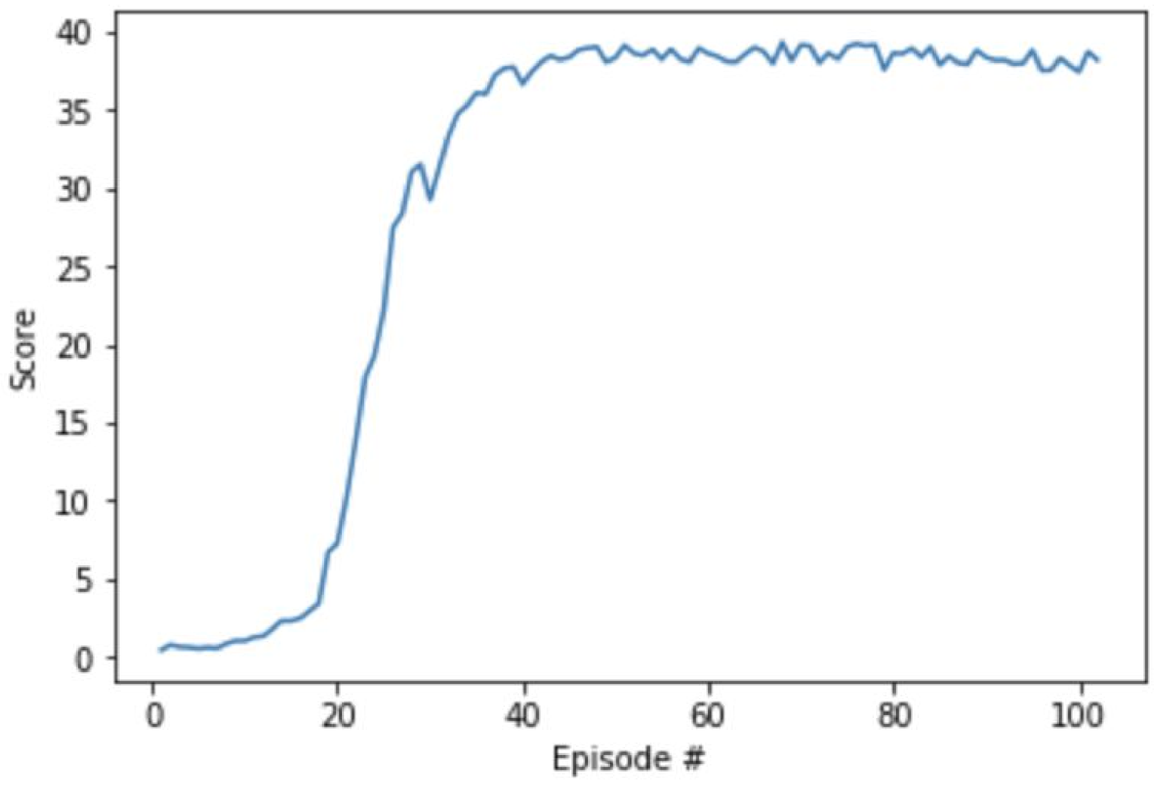

Figure 1 - No. of episodes / Scores

The trained DDPG agent performs pretty well in tracing the moving target location for all 20 arms.

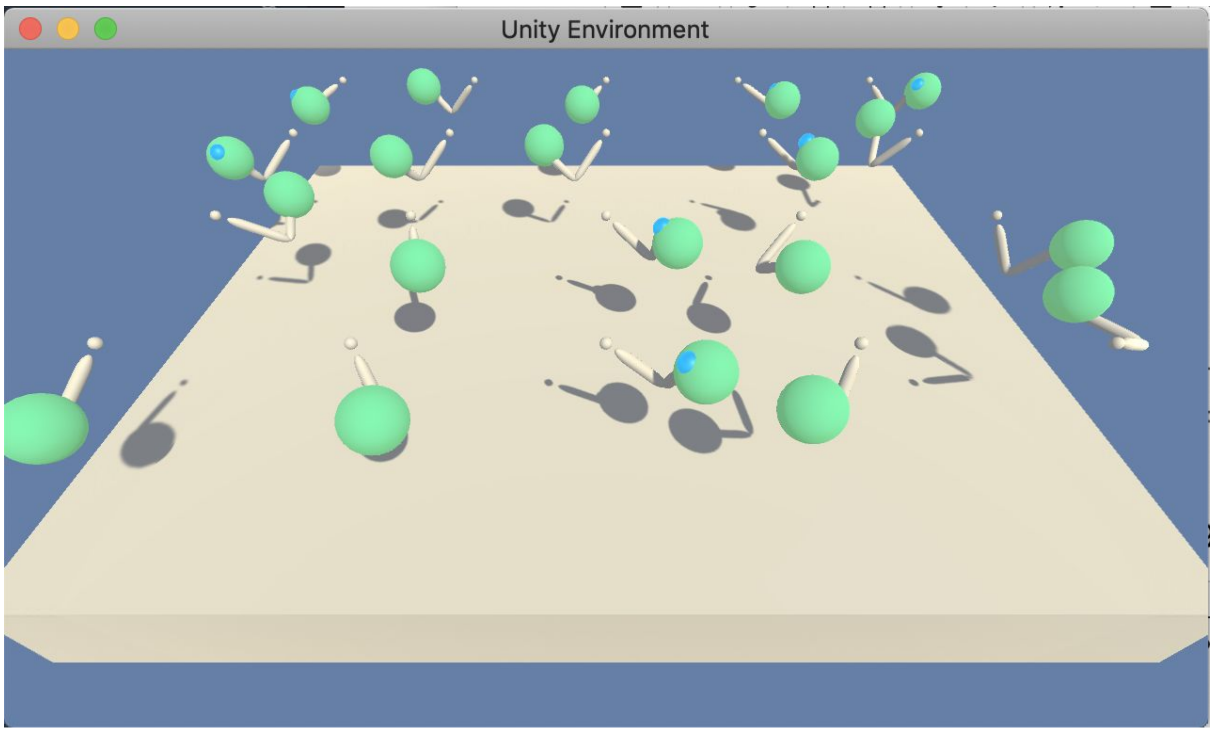

Figure 2 - Running the DDPG agent with saved weights

Future Improvements

We could improve the novel DDPG algorithm by applying Priority Experienced Replay in which important experience will be sampled more often. Based on the paper "A novel DDPG method with prioritized experience replay", it can reduce the training time, improve the stability of the training process and is less sensitive to the change in hyperparameters.

Credits

- Continuous control with deep reinforcement learning https://arxiv.org/abs/1509.02971
- Deep Deterministic Policy Gradient Actor-Critic Model https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum
- A novel DDPG method with prioritized experience replay https://www.semanticscholar.org/paper/A-novel-DDPG-method-with-prioritized-experience-Hou-Liu/027d002d205e49989d734603ff0c2f7cbfa6b6dd