Project Report – Continuous Control

In the previous project, we used Deep Q-Learning to solve a high dimensional observation space (a room with yellow and purple bananas) with a low-dimensional action space (pick up yellow bananas, avoid those that are purple). While the DQN approach can achieve human-level performance and beyond for such tasks, it may not perform as well on tasks requiring continuous control. DQN tries to fund the action that maximizes the action-value function, while continuous control requires iterative optimization at every step.

The Deep Deterministic Policy Gradient (DDPG) approach, first summarized in the 2016 `Continous Control with Deep Reinforcement Learning' paper by Google Deepmind authors, involves an off-policy actor-critic algorithm using deep function approximators able to learn policies in hi-dimensional, continuous action spaces.

Since Q-learning requires an optimization of an action at each timestep, the approach can be slow with non-trivial action spaces. DDPG incorporates the actor-critic approach based on Deep Deterministic Gradient (DGP) algorithm (Silver et al., 2014) with neural network function approximators to learn from large action spaces.

Algorithm

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

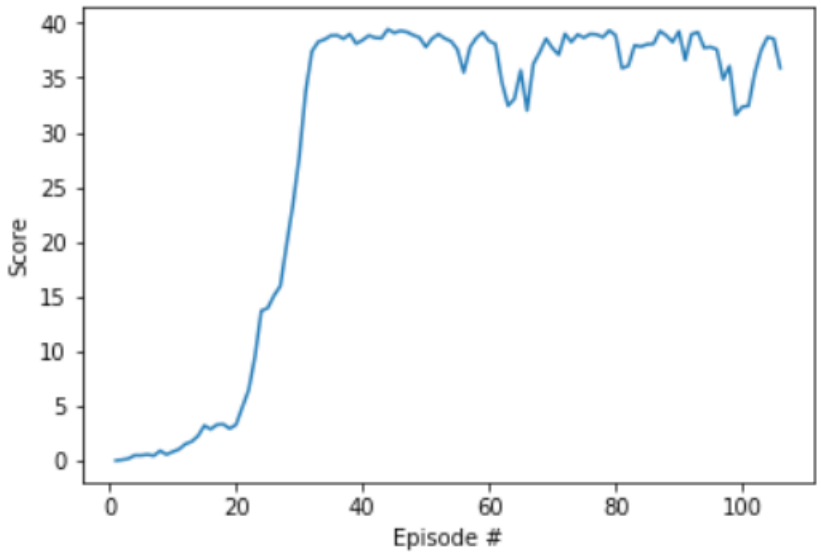
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

The continuous control project follows this architecture. I experimented with numerous adjustments to the neural network including batch normalization, dropout, additional layers, different activation functions and learning rates. Training on GPU and in the provided workspace with GPU enabled began as very slow and ineffective: after 1000+ episodes, the average score tended to cease advancing at around 14. After much experimentation with additional hidden layers and batch normalization, boosting the actor and critic learning rates, and reducing the noise function were key improvements. The model achieves an average score of 30 in just over 100 episodes.



Further work

I would like to tackle the problem using the Proximal Policy Optimization approach. I also hope to attempt the crawler project.