Learning to Reach
Project Submission for Udacity's Deep Reinforcement Learning Nanodegree
Submitted by Patrick Nalepka

Background

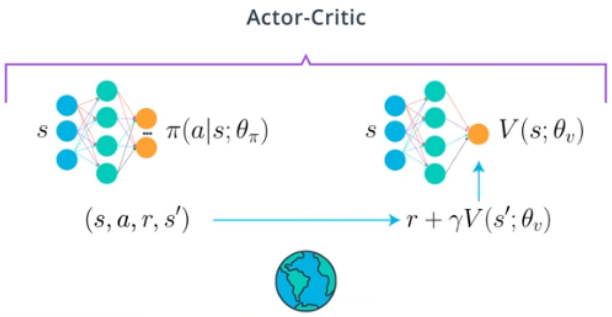
1. Introduction to Policy-Based Methods

Policy-based methods are approaches that directly searches for an optimal policy (i.e., actions in response to a given state). This is in contrast to value-based methods (such as DQN), which involves the maintenance of action-value estimates that are then used to select the desired action. The benefits of policy-based methods are three-fold (as cited by Udacity). First, policy-based methods are simpler than value-based methods as no intermediate step of calculating action-value estimates via a lookup table or neural network. Instead, the agent is learning the policy directly from input. Second, policy-based methods are capable of learning a true stochastic policy. Value-based methods solve for deterministic or near-deterministic policies which may be detrimental in problems where there is uncertainty in the task environment (e.g., an agent being unsure which direction to turn if they start in similar states such as identical rooms). Finally, policy-based methods are better suited for continuous problems, which does not require the estimate of a "global or local maxima" given a current state as is the case with value-based methods.

2. Introduction to Actor-Critic Methods

Actor-Critic methods offers the "best of both worlds" by incorporating the strengths of both value-based and policy-based methods. In these methods, value-based methods are utilized as "baselines" to reduce the variance of policy-based approaches. Here, the cumulative reward received through interacting with the environment is compared to current value estimates to determine how much *better* (or *worse*) an action is compared to others. Actor-Critic methods are more stable than value-based methods alone (which are biased based on previous experiences) and require fewer training examples (which is a negative to policy-based only methods).

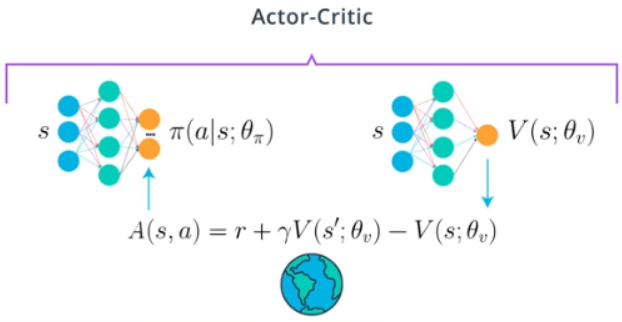
The algorithmic approach to a basic Actor-Critic method is shown in the following images, taken from the Udacity Deep Reinforcement Learning Nanodegree Course. First, given a state, you select an action from the actor's network and obtain the reward and next state:



Screen grab taken from Udacity's Deep Reinforcement Learning Course (https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893)

Given the (state, action, reward, next state) tuple, you train the critic's network by comparing the critic's current value estimate with that derived from the reward received by the actor and the value estimate of the next state.

Then, you compute the advantage of the actor's current action by the following function:



Screen grab taken from Udacity's Deep Reinforcement Learning Course (https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893)

3. Deep Deterministic Policy Gradient (DDPG)

An Actor-Critic-like algorithm which was used to solve the Reacher environment in this repository is DDPG. DDPG is suitable for continuous control problems and is closely related to value-based DQN architectures. Here, the Critic's network is updated to provide accurate action-value estimate for a given action (as opposed to only a value estimate). The Critic then informs the Actor as to whether its action is valued compared to the Critic's action-value estimate. The result is that high action-value estimates will reward actions more than those with smaller (or negative) values. The original DDPG paper that discusses the algorithm can be found here: https://arxiv.org/abs/1509.02971. A snippet of their algorithm is found below:

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

Learning to Reach
Project Submission for Udacity's Deep Reinforcement Learning Nanodegree
Submitted by Patrick Nalepka

Learning Algorithm

The algorithm in this repository implements the DDPG algorithm, adapted from the ddpg-pendulum example from Udacity: https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum

1. Neural Network Architecture

The Actor and Critic were represented via a neural network. The Actor had an input layer size of 33 (dimensions of the environment), which fed to a fully connected layer of 256 nodes. This layer was then fed to a second hidden layer of 128 nodes, which outputted to 4 output nodes (representing the action dimensions of the Reacher agent). ReLU activation functions were used for the hidden layers, whereas a tanh function was used for the output layer. The Critic had a similar network architecture with the following exceptions. First, an additional 4 nodes were added to the first hidden layer which were the actions of the Actor. Second, the output layer was a simple linear operation to one output node.

2. Replay Buffer

During reinforcement learning, each (S_0,A_0,R_1,S_1) experience tuple is added to a queue (with maximum length **buffer size** [see below]). A mini-batch of experiences are sampled at random and used to train the Actor and Critic neural network.

3. Soft Network Updating

To provide more stable learning, a "local" and "target" network are created. Fast time-scale updates are applied to the "local" networks of the Actor and Critic and are gradually updated to the "target" network so as to have a stable reference point for learning.

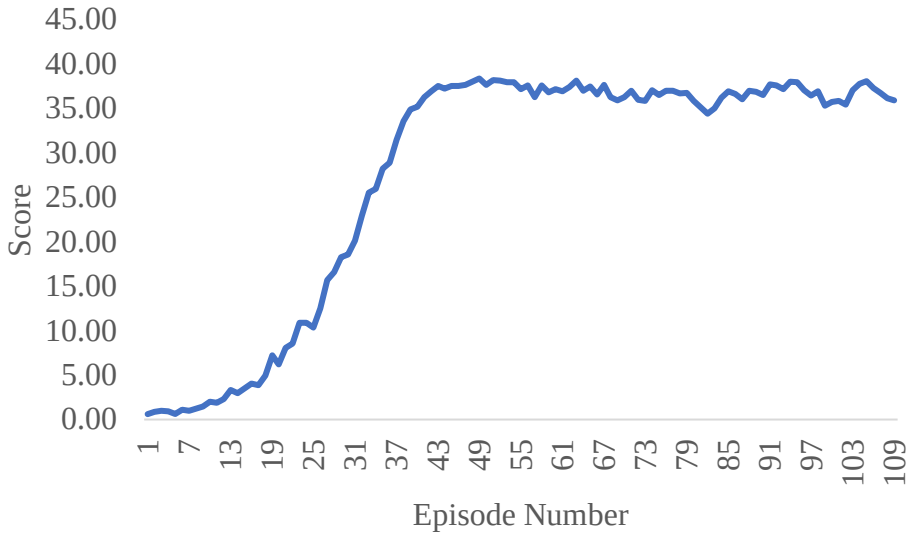
4. DDPG Hyperparameters:

- **Replay Memory Buffer Size:** 1×10^5
- **Mini-batch Size:** 128
- **τ (tau/network interpolation):** 1×10^{-3}
- **γ (gamma/discount factor):** 0.99
- **α (alpha/learning rate):** 1×10^{-4}
- **Maximum number of episodes:** 200 (with the potential to quit early if required score reached.

Plot of Rewards

Given the architecture and settings above, the Reacher task was solved (with the required average score being +30 over 100 consecutive trials) after 109 episodes. The mean score across 20 Reacher agents at this episode was +30.06, with early stopping applied. A plot of the learning trajectory is seen below:

Learning to Reach
Project Submission for Udacity's Deep Reinforcement Learning Nanodegree
Submitted by Patrick Nalepka



Ideas for Future Work

Other actor-critic methods can be utilized in place of DDPG, such as the A2C or A3C algorithm. Additionally, policy-gradient based methods such as Proximal Policy Optimization (PPO), adapted for continuous action spaces, will be explored in the future to determine performance differences between Actor-Critic and pure policy-based methods. Further, curriculum-based approaches can be explored to investigate developmental changes from arm reaching to more complex tasks, such as throwing a ball.