Continuous Control Report

This report includes the description of the implementation of the Udacity Reinforcement Learning Continuous Control project.

The 1-agent problem

1. This project contains a single agent.
2. Actions: The agent is a double-jointed arm. Each action has 4 numbers in the range between -1 and 1 corresponding to torque applicable to two joints.
3. States: The state space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm.
4. Reward: The task is episodic. A reward of +0.01 is provided for each step and the component of the agent's hand in the goal location. If all components are in the goal location, the reward for that step is +0.04. There are at maximum 1000 steps for each episode.
5. Goal: The agent must get an average score of +30 over 100 consecutive episodes to solve the environment.

The 20-agent problem

1. This project contains 20 agents.
2. Actions: Each agent is a double-jointed arm, the same as 1-agent problem.
3. States: The state space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm.
4. Reward: The task is episodic. For 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).T
5. Goal: The agents must get an average score of +30 over 100 consecutive episodes to solve the environment.

Learning Algorithm, hyperparameters, model architectures

1. This project uses Deep Deterministic Policy Gradients (DDPG) algorithm, an example of Actor-Critic methods. The Actor-Critic method combines the advantages of actor-only (Policy-based) and critic-only (Value-based) methods. In this method, the critic learns the value function and uses it to determine how the actor's policy parameters should be changed. In this case, the actor brings the advantage of computing continuous actions without the need for optimization procedures on a value function, while the critic supplies the actor with knowledge of the performance. Actor-critic methods usually have good convergence properties, in contrast to critic-only methods.
2. An agent is created with both Actor and Critic networks built with different deep learning (DL) models.
3. The learning algorithm is to train the weights for all layers of the Actor network and the Critic network to find a policy to optimize the action taken at each step to maximize the score of an episode. The model is trained with a mini batch containing a given number of samples.

4. For each step in an episode, from the current states, the agent decides the action. After an action is taken, the environment generates the next state, reward and whether the episode is done or not.
5. The Critic loss function is the mean squared error (MSE) of two Q-value (action-value function) results. After the next actions are generated from the Actor model with next states, one of Q-value results is the predicted Q-value generated by the target Critic DL model using next states and next actions as the inputs and then adjusted by discounting factor Gamma and added by rewards. The other one of the Q-value results is the expected Q-value generated by the local Critic DL model using the current states as the inputs. The local Critic model is trained and updated by using back propagation on the loss function.
6. For the Actor model here, the predicted actions are generated with the Actor model and the current states. Then the Actor loss function is the negative mean of the value generated by the Critic model with the current states and predicted actions, which represents the . Actor model parameters are trained by using this Actor loss function.
7. Critic (Value-based) formulae:

$$L(\theta) = E\{(reward + \gamma * \max_{a'} Q(s', a'; \bar{\theta}) - Q(s, a; \theta))^2\}$$

$$\Delta\theta = \alpha * \left(reward + \gamma * \max_{a'} Q(s', a'; \bar{\theta}) - Q(s, a; \theta)\right) * \nabla_\theta Q(s, a; \theta)$$

$$\bar{\theta} = \tau * \theta + (1 - \tau) * \bar{\theta}$$

8. Actor (Policy-based) formulae:
   Policy is $\pi_\theta(a|s)$ that maps all states to a probability distribution over all actions. U is the expected return, τ is the trajectory based on the policy, and $R(\tau)$ is the return of the trajectory:

$$U = \sum_\tau R(\tau) * P(\tau)$$

   Gradient:

$$\nabla_\theta U = \sum_\tau R(\tau) * \nabla_\theta P(\tau) = \sum_\tau R(\tau) * \frac{P(\tau)}{P(\tau)} \nabla_\theta P(\tau) = \sum_\tau R(\tau) * P(\tau) * \nabla_\theta \log(P(\tau))$$

$$= \sum_\tau R(\tau) * P(\tau) * \nabla_\theta \log(\pi_\theta(a|s)) = \frac{1}{m} \sum_i R(i) * \nabla_\theta \log\big(\pi_\theta(a|s)\big) =$$

$$= \frac{1}{m} \sum_i R(i) * \nabla_\theta \frac{\pi_\theta(a|s)}{\pi_{\theta\_old}(a|s)}$$

   Update the parameters θ. This is gradient ascent:

$$\theta = \theta + \alpha * \nabla_\theta U$$

   The action is chosen to be the one corresponding to the maximum of the expected Q-value. In order to avoid the situation with local maximum, it needs to randomly choose action from time to time and eps is used to control how often to use random sample.
9. DDPG algorithm formulae:
   Loss function for the Critic Model:

$$L(\theta) = E\{(r_i + \gamma * Q'(s_{i+1}, \mu'(s_{i+1}; \overline{w}); \bar{\theta}) - Q(s_i, a_i; \theta))^2\}$$

$$\theta = \theta + \alpha * \nabla_\theta L$$

Update the Actor policy using the sampled policy gradient:

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

$$w = w + \alpha * \nabla_w J$$

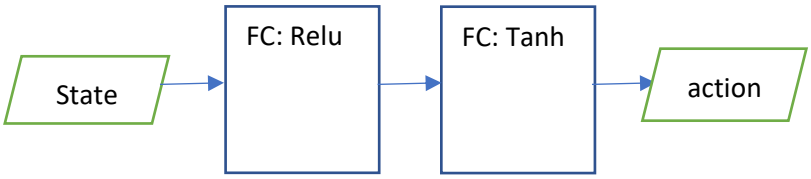
Update the target networks:

$$\bar{\theta} = \tau * \theta + (1-\tau) * \bar{\theta}$$
$$\bar{w} = \tau * w + (1-\tau) * \bar{w}$$

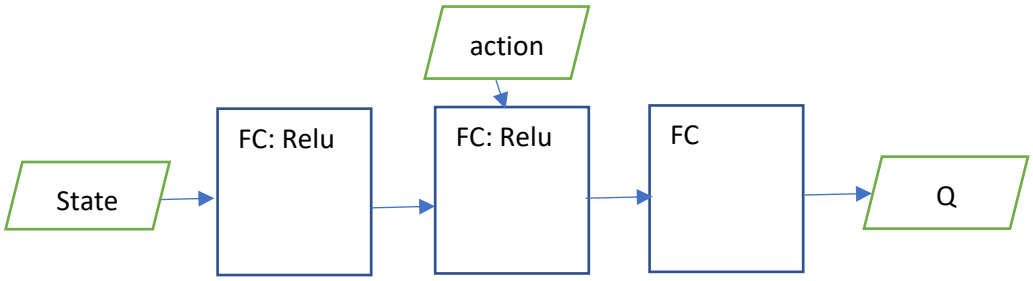
10. The hyperparameters for one agent are:

BUFFER_SIZE = int(1e6)	replay buffer size
BATCH_SIZE = 100	minibatch size
GAMMA = 0.97	discount factor
TAU = 1e-3	for soft update of target parameters
LR_ACTOR = 1e-4	learning rate of the actor
LR_CRITIC = 1e-3	learning rate of the critic
WEIGHT_DECAY = 0	L2 weight decay
UPDATE_EVERY = 20	
UPDATE_TIMES = 10	
Actor fc1_units=64	Actor network hidden layer dimension
Critic fc1_units=64, fc2_units=64	Critic network hidden layer dimension

11. The DL model architectures for the Actor (Policy-based) neural networks are 2 fully connected (FC) layers shown in the chart below. The weights are stored in the checkpoint_actor.pth file.



12. The DL model architectures for the Critic (Value-based) neural networks are 3 fully connected (FC) layers shown in the chart below. The weights are stored in the checkpoint_critic.pth file.



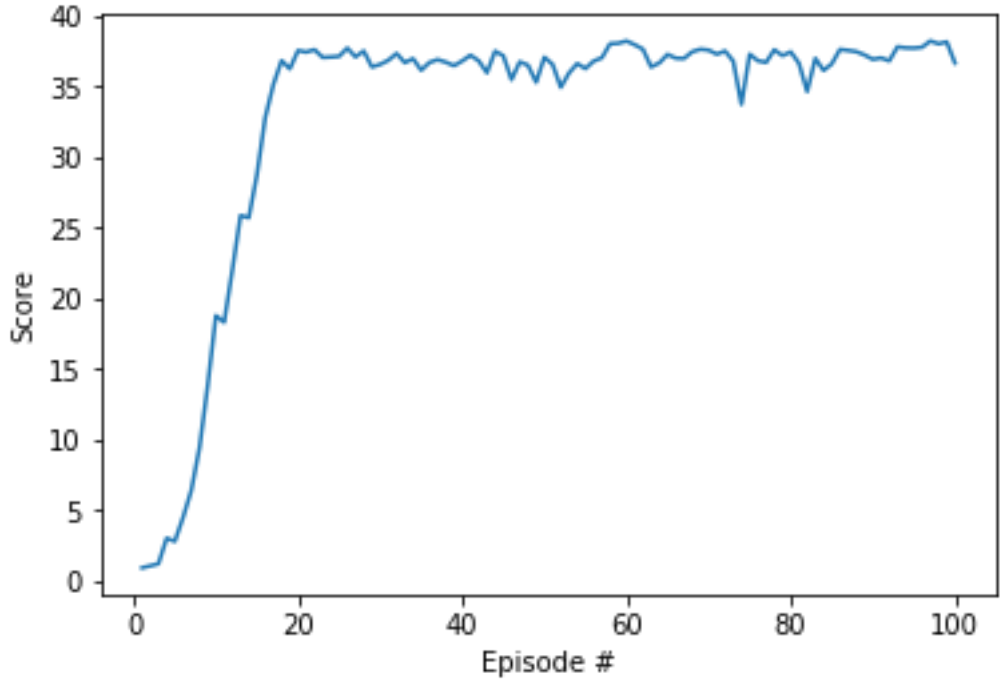
13. For 20 agents, each agent uses an action with noise added individually, and then is trained individually.
14. The hyperparameters for 20 agents are:

BUFFER_SIZE = int(1e6)	replay buffer size
BATCH_SIZE = 100	minibatch size
GAMMA = 0.97	discount factor
TAU = 1e-3	for soft update of target parameters
LR_ACTOR = 1e-4	learning rate of the actor
LR_CRITIC = 1e-3	learning rate of the critic
WEIGHT_DECAY = 0	L2 weight decay
UPDATE_EVERY = 20	
UPDATE_TIMES = 10	
Actor fc1_units=256	Actor network hidden layer dimension
Critic fc1_units=256, fc2_units=128	Critic network hidden layer dimension

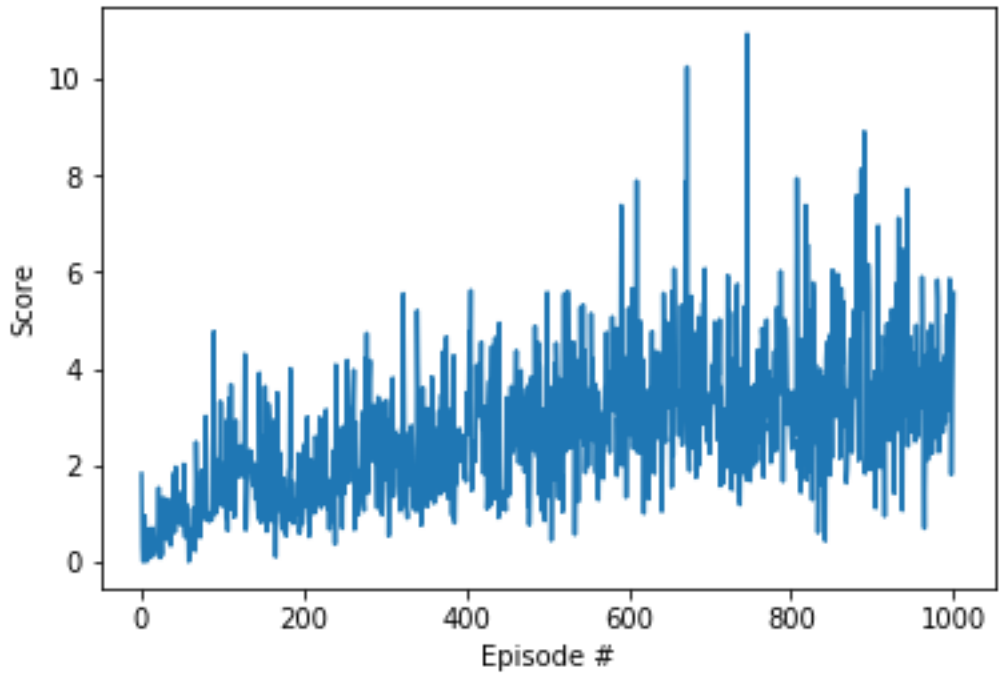
Plot of Rewards

The plot of rewards per episode illustrates that the agent is able to receive an average reward (over 100 episodes) of at least 30.

For 20 agents, The number of episodes needed to solve the environment is 100 with average reward 33.18. This took about 8 hours to solve the environment. The scores are much more stable because each step, the neural networks are trained 20 times and the scores are the average scores of the 20 agents. In addition, there are more units of hidden layers and BatchNorm1d is used too.



For one agent, 1000 episodes have been processed.



Ideas for Future Work for improving the agent's performance

1. Use Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG).
2. Use Proximal Policy Optimization (PPO).
3. Use Distributed Distributional Deterministic Policy Gradients (D4PG) algorithm.

References

1. Udacity Deep Reinforcement Learning Nanodegree Program: 3. Policy-based Methods https://www.udacity.com/
2. https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum
3. Continuous control with deep reinforcement learning https://arxiv.org/abs/1509.02971
4. https://github.com/ShangtongZhang/DeepRL