Project 2 – Continuous Control

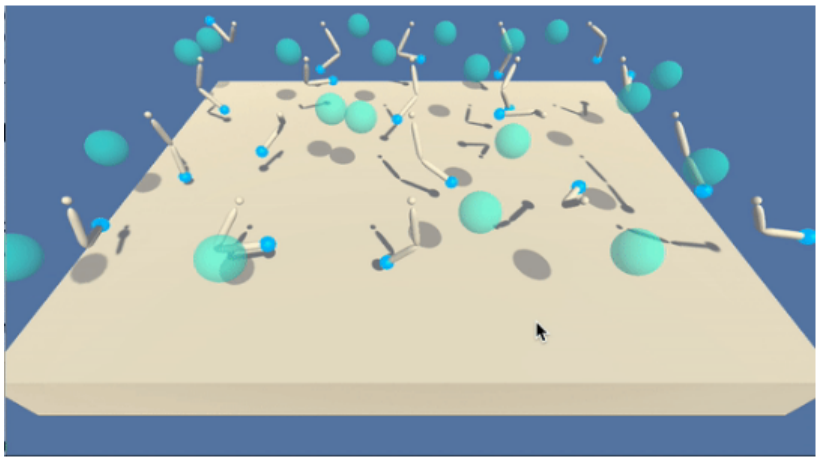
September 27, 2019

https://medium.com/@jasdeepsidhu13/project-2-continuous-control-of-udacity-s-deep-reinforcement-learning-c16fef28f24e

Introduction

The project demonstrates how policy-based methods can be used to learn the optimal policy in a model-free Reinforcement Learning setting using a Unity environment, in which 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 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 is a number between -1 and 1. An agent choosing actions randomly can be seen in motion below:



Learning Algorithm

Model

Actor-critic method

We use Deep Deterministic Policy Gradients (DDPG) to solve the problem. which is a different kind of actor-critic method. Firstly, let us give ourselves a quick refresher of the Actor-critic method. One may recall that the agent can learn the policy either directly from the states using Policy-based methods or via the action valued function as in the case of Value-based methods. The policy based methods tend to have high variance and low bias and use Monte-Carlo eistimate whereas the value based methods have low variance but high bias as they use TD estimates. Now, the actor-critic methods were introduced to solve the bias-variance problem by combining the two methods. In Actor-Critic, the actor is a neural network which updates the policy and the critic is another neural network which evaluates the policy being learned which is, in turn, used to train the actor.In vanilla policy gradients, the rewards accumulated over the episode is used to compute the average reward and then, calculate the gradient to perform gradient ascent. Now, instead of the reward given by the environment, the actor uses the value provided by the critic to make the new policy update.

Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradient (DDPG) lies under the class of Actor Critic Methods but is a bit different than the vanilla Actor-Critic algorithm. The actor produces a deterministic policy instead of the usual stochastic policy and the critic evaluates the deterministic policy. The critic is updated using the TD-error and the actor is trained using the deterministic policy gradient algorithm. Infact, it could be seen as approximate Deep-Q Network(DQN) method. The reason for this is the critic in DDPG is used to approximate the maximizer over the Q-values of the next state, and not as a learned baseline, as have seen so far. One of the limitation of the DQN agent is that it is not straightforward to use in continuous action spaces. For example, how do you get the value of a continuous action with DQN architecture? This is the problem DDPG sovles.

Network Architecture The network achitecture is comprised of two fully connected hidden layers of 128 units each with ReLU activations. Inorder to help speed up learning and avoid getting stuck in local minimum, batch normalization was introduced to each hidden layer. The hyperbolic tan activation was used on the output layer for the actor network as it ensures that every entry in the action vector is a number between -1 and 1. Adam was used as an optimizer for both actor and critic networks.

Two sets of Target and Local Networks Recall, that the concept of fixed targets was first introduced in the Deep Q Networks. We use the same concept in DDPG. In total we use four deep neural networks –local and target networks for both actor and the critic. Now the actor here is used to approximate the optimal policy deterministically. That means we want to always output the best believed action for any given state. This is unlike a stochastic policy in which we want to learn a probability distribution over the action. In DDPG we want the best believed action every time we query the actor network. That is a deterministic policy. The actor is basically learning the argmax of Q(s,a) which is the best action. The critic

learns to evaluate the optimal action value function by using the actors best believed action. Again we use this action which is an approximate maximizer to calculate a new target value for training the action value function much in the way DQN does.

Replay Buffer In reinforcement learning, the agent interacts with the environment and learns from sequence of state(S), actions(A) and rewards(R) and next state(S') which is stored in the form of an experienced tuple. These sequence of experience tuples can be highly correlated, and the DDQN algorithm like the Q-learning algorithm that learns from each of these experienced tuples in sequential order and can be swayed by the effects of this correlation. This can lead to oscillation or divergence in the action values, which can be prevented by having a replay buffer, from which experience replay tuples are used to randomly sample from the buffer. The replay buffer contains a collection of experience tuples (SS, AA, RR, S'S ´). The tuples are gradually added to the buffer as we are interacting with the environment. The act of sampling a small batch of tuples from the replay buffer in order to learn is known as experience replay. In addition to breaking harmful correlations, experience replay allows us to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of our experience.

Ornstein-Uhlenbeck(O-U) Noise In Reinforcement learning for discrete action spaces, exploration is done via probabilistically selecting a random action such as epsilon-greedy or Boltzmann exploration. For continuous action spaces, exploration is done via adding noise to the action itself. For details on the O-U noise please see the DDPG paper: https://arxiv.org/abs/1509.02971.

Soft Updates In DDPG, target networks are updated using a soft update strategy. In DQN you have two copies of your network weights the regular and the target network. In DDPG, you have two copies of your network

weights for each network, a regular for the actor, a regular for the critic, and a target for the actor, and a target for the critic.. A soft update strategy consits of slowly blending in your regular network weights with your target network weights. So, every timestep you make your target network be 99.99 percent of your target network weights and only 0.01 percent of your regular network weights.You are are slowly mixing your regular network weights into your target network weights. Recall, the regular network is the most upto date network because it is the one we are training while the target network is the one we use for prediction to stabilize strain. In practive, you will get faster convergence by using the update strategy , and in fact this way for updating the target network weights can be used with other algorithms that use target networks including DQN.

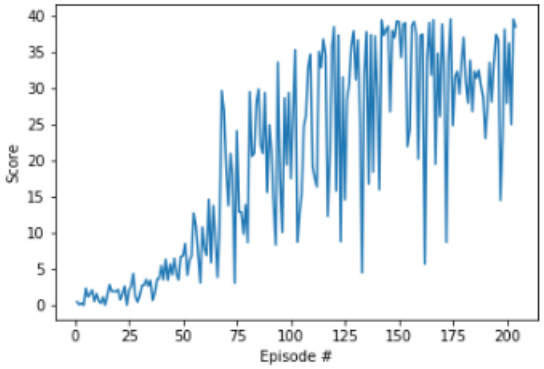
Hyperparameters

Hyperparameters	Values
Batch size	128
Replay buffer size	1e5
Discount factor(GAMMA)	0.99
update interval	4
tau	1e-3
learning rate(Actor,Critic)	2e-4, 2e-4
Number of episodes	1000
Number of time-steps per episode	5000
Weight decay	0

Results

The problem was solved using DDPG algorithm where the average reward of +30 over atleast 100 episodes was achieved in 204 episodes. The result depends significantly on the fine tuning of the hyperparameters. For example, if the number of time steps are too low, learning rate or the seed is too

large or small, the system can fall into a local minima where the score may start to decrease after a certain number of episodes.Batch normalization layer was added to the network architecture to help improve training. The plot of the rewards across episodes is shown below:



Future Improvements

I have only solved the single agent problem. Future work should including solving the multi agent continuous control problem with DDPG. Algorithms such as TRPO, PPO, A3C AND A2C might also be usefull specially in working with multi agent problem particularly when parallelization may be very useful. The Q-prop algorithm, which combines both off-policy and on-policy learning may also be used. For DDPG which uses replay buffer, one can also try implementing it with prioritized replay which may yield better results.