Deep Reinforcement Learning Nanodegree

Project 2: Continuous Control

Report

The game is episodic with state space of 33 variables and action space of size 4. The solution was implemented as a DDPG actor-critic agent architecture.

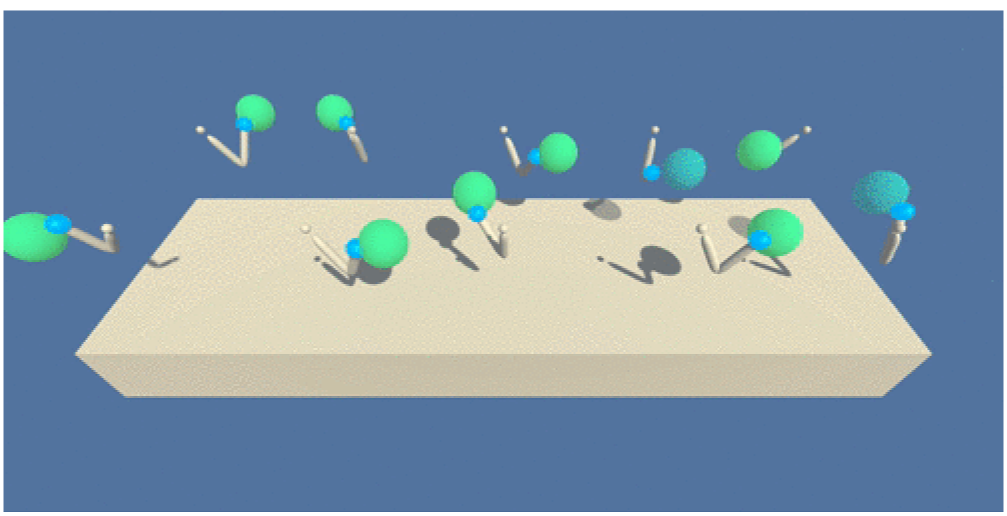
A 100 thousand length buffer was defined to collect the agent experience while learning. This buffer is then periodically sampled to get a 128 long batch which is then used to update the agent nn model through learning. Every 20 experiences 5 learning steps are performed.

Agent structure:

The agent actor is a simple 2 fully connected (linear+relu) layers of size 256 in addition to the output layer with a size equal to the action space size (4).

The critic is identical to the structure of the actor except that the actor uses tanh for the output layer to get actions within the +1/-1 range.

Both actor & critic learning rates were set to 2e-4



Learning:

Given an experience tuple the agent goes through the following steps to update its actor & critic:

Local networks update:

Critic update:

- Compute the **Q-expected** passing the states and actions to the critic local network
- Compute the next actions for the next states using the actor target network without noise addition (deterministic policy). Then compute the **Q-target** passing the next states and previously computed next actions to the critic target network
- Compute the loss function from the **Q-expected/Q-target**, set the gradient clip normalization and train the critic.

Actor update:

- Compute the actions for the experience states. Here we dont use the actions from the experience tuple because they were noise-added to ensure stochastic exploring interaction policy and we need to learn a deterministic policy.
- Compute the actor loss function from the states and the previously computed deterministic actions and train the actor

Target networks update:

Actor & Critic target networks are updated through soft update where the local network parameters are tau-weighted and then added to (1-tau)-weighted target parameters to obtain the new updated parameters. Tau was set to 1e-3

The agent has the following capabilities:

1. Act: given the current state of the environment, it runs the state through the available agent nn model and selects the action according to the epsilon-greedy algorithm.
2. Step: given a complete experience, it saves it in the replay memory and if its the fifth experience since the last learning it calls the learning routine.
3. Learn: given the a sampled batch of size 64 experiences and the learning discount factor, it computes the error of the current agent model (which is computed as the mmean_square_error between the agent current q-values of the given states to the

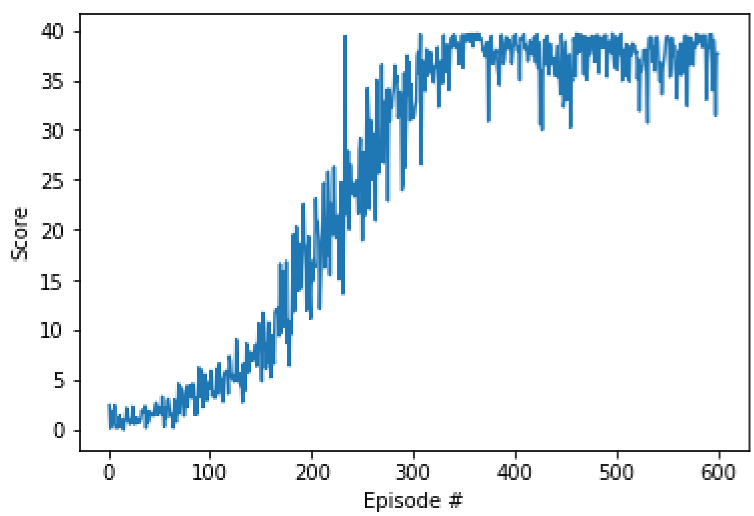
target q-values of the same states computed from the target nn) and propagates it back to update the agent model.

4. Soft_update: given the target-nn model update factor and the current agent-nn model, it updates the target-nn model in a way to ensure smooth slower change in the target-nn model so as to improve the whole DQN stability.

Performance:

The agent was set to play for (600) episodes and learn throughout. Finally the agent learnt model is saved periodically every 100 episodes.

Below is the plot showing the learning performance of the agent. The average score exceeds +30 after 233 episodes and the performance went steady afterwards above the target score of +30.



Future Ideas for improving theAgent performance:

The solution here implemented is a DDPG structure. Indeed other structures can be used, however with this agent we were able to solve the environment with few training episodes and the results were stable.

One of the things that might have a relatively strong impact on improving the performance is maybe reducing the networks to 128 or 64 size, and add batch normalization to the first layers of both the actor and critic.