Continuous Control

In this project, we train an agent to move a double-jointed arm 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 our 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 should be a number between -1 and 1.

Agent implementation

DDPG: Deep Deterministic Policy Gradient, Continuous Action-space

It's an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. The DQN architecture is composed of a couple of convolutional layers and no fully-connected layers, which takes the state as input, and returns the corresponding predicted action values for each possible game action.

It is not possible to straightforwardly apply Q-learning to continuous action spaces, because in continuous spaces finding the greedy policy requires an optimization of at every timestep; this optimization is too slow to be practical with large, unconstrained function approximators and non-trivial action spaces. Instead, here we used an actor-critic approach based on the DPG algorithm.

The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. The critic $Q(s,a)$ is learned using the Bellman equation as in Q-learning. The actor is updated by following the applying the chain rule to the expected return from the start distribution J with respect to the actor parameters:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)}\nabla_{\theta_\mu}\mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned}$$

Replay Buffer

One challenge when using neural networks for reinforcement learning is that most optimization algorithms assume that the samples are independently and identically distributed. DDPG uses a replay buffer to address these issues. The replay buffer is a finite sized cache R. Transitions were sampled from the environment according to the exploration policy and the tuple (st, at, rt, s_{t+1}) was stored in the replay buffer. When the replay buffer was full the oldest samples were discarded.

Target Network

Q learning with neural networks proved to be unstable in many environments. Since the network $Q(s,a|\theta^Q)$ being updated is also used in calculating the target value, the Q update is prone to divergence. DDPG uses something similar to the target network but modified for actor-critic and using “soft” target updates, rather than directly copying the weights. We create a copy of the

actor and critic networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ respectively, that are used for calculating the target values.

Batch Normalization

When learning from low dimensional feature vector observations, the different components of the observation may have different physical units (for example, positions versus velocities) and the ranges may vary across environments. This can make it difficult for the network to learn effectively and may make it difficult to find hyper-parameters which generalise across environments with different scales of state values. DDPG addresses this issue by adapting a recent technique from deep learning called batch normalization . This technique normalizes each dimension across the samples in a mini batch to have unit mean and variance. In addition, it maintains a running average of the mean and variance to use for normalization during testing

Model

I used single DDPG agent with one Replay buffer.

Actor Model is a neural network with 3 hidden layers with size 400, 300, and 200 with a batch normalization. Tanh is the final layer that maps states to actions.

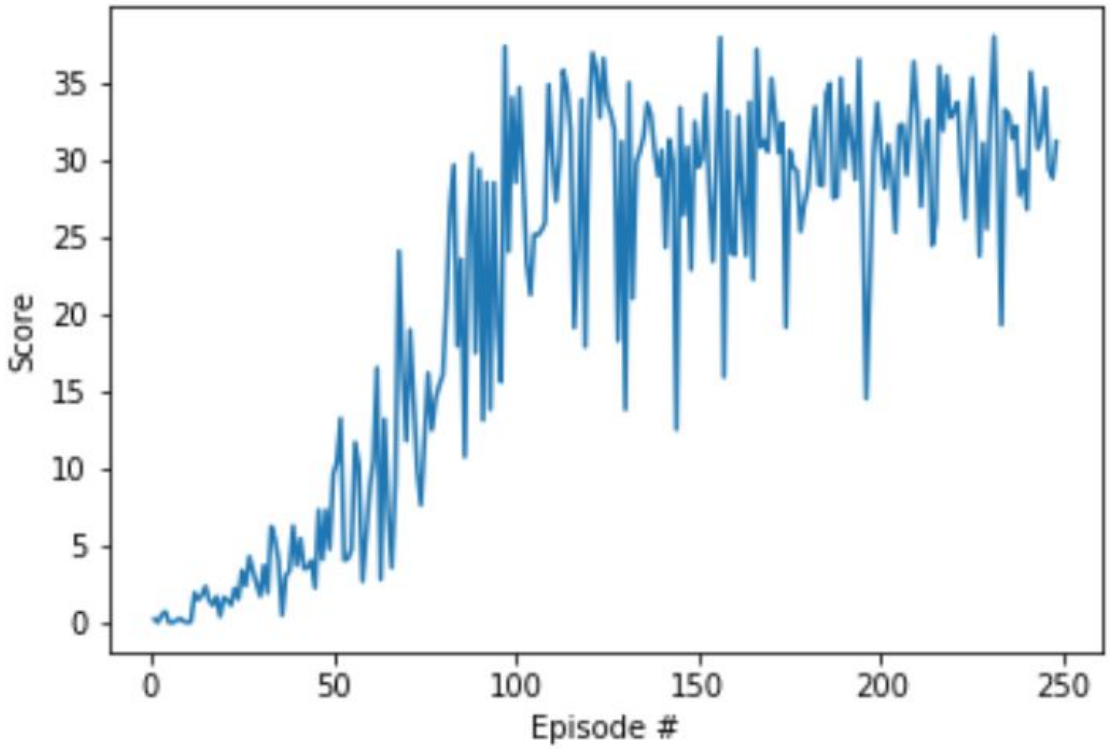
Critic Model has similar architecture to Actor model but the final layer is fully connected layer that maps states and actions to Q-values

Hyperparameter

```
Actor Leraning rate = 1e-3
Critic Leraning rate = 1e-3
Gamma discount = 0.99
Replay Buffer size = 1e6
weight decay = 0
Batch size = 1024
soft update of target = 1e-3
```

Training Results

DDPG agent takes 248 episodes to achieve an average reward of 30. Agent starts of learning linearly, agent obtains higher scores after 60 episodes, progress seems to flatten after 150 episodes.



Future Improvements

1. We could improve DDPG algorithm by applying Prioritized Experience Replay - which prioritizes important experiences and samples them more by using the index to reflect the importance of each transition.
2. Grid search could be used to find optimal hyperparameters (learning rates, gamma discount, batch size, soft update ratio and so on).
3. Try adding additional layers or hidden layers to Actor and Critic Neural networks to see if we can learn faster.