Report

The problem was solved using a policy gradient algorithm, more specifically simple "Deep Deterministic Policy Gradient". This consists of have two neural networks, called Actor and Critic. The former is used to predict the optimal action given the state input, while the former network approximates the Q-value for a state-action pair. The architectures of both networks are very similar (128 neurons in the hidden layer and 128 neurons in the second one, ReLU activation function after each hidden layer), with just three differences:

- The input and the output sizes
- In the hidden layer of the Critic layer, the actions are concatenated, thus a number of neurons equal to the action size is added to 128
- The TANH activation function is used after the output layer of the actor (as the action space is between -1 and 1)

Both neural networks (the Actor and the Critic) have an exact copy, called target network, which is updated at every step. The update is done slowly and progressively, by keeping 99% of the current weights and adding 1% of the weights of the respective local network. The general methodology is used to train the local networks: a loss is calculated given a criterion, then this is used to calculate the gradient which is then back-propagated through the network and the weights are updated using the gradients.

One of the hyperparameters chosen is the discount factor, set to 0.99. This allows for future rewards to be highly considered. Another hyperparameter is the batch size. Using 128 allows the model to better generalize.

The pipeline begins by using the local actor to get the action for the current state. Before the action is taken, noise is added to it. This has an important role in the exploration property, as the action space is continuous. Then, the new action is taken and the rewards and the next state are observed. This transition is stored in a memory buffer, from which batches are then sampled. The role of the memory buffer is use the transitions in the learning algorithms in a way that they are uncorrelated, so the network don't become unstable. Moreover, past transitions can be reused without the need to experience the state again.

The loss for the critic network is computed as the mean squared error between the current Q-value (obtained from the local critic network using current state) and a predicted Q-value (obtained using the Bellman equation). The predicted Q-value is calculated adding the rewards with the discounted state value of the next state (this is obtained by getting the action for the next state using the target actor network, then the Q-value using the target critic).

The loss of the actor is computed as the negative of the mean of the Q-value resulted from the local critical network given as input the current state and the actions from the local actor. At the end of each step in the pipeline, the soft-update is done on both networks.

The scores obtained throughout the training algorithm is presented in figure 1. As it can be seen, the score is increasing continuously until it is stopped when the desired value was obtained.

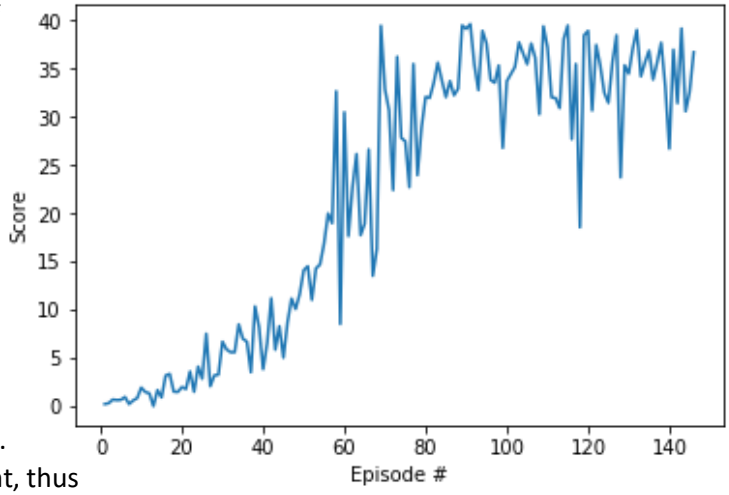

Figure 1. The scores over the episodes

Another result can be observed in the figure bellow. Over time, the agent starts to learn the environment, thus it accumulates more positive rewards and the scores

increases. It took 146 episodes in order to solve the environment.

One method to improve the algorithm is to train the algorithm progressively. This means that the training loop should start with a small maximum length for an episode (around 300) and then increase it. This would result in the robot arm to firstly focus on learning the first movements, and then focus on the full motion. Another improvement is increase the size of bootstrapping. This will give a better estimation of the Q-value. Finally, the capacity of the network can be increased, as well as add features such as batch normalisation for an increased stability and better convergence.