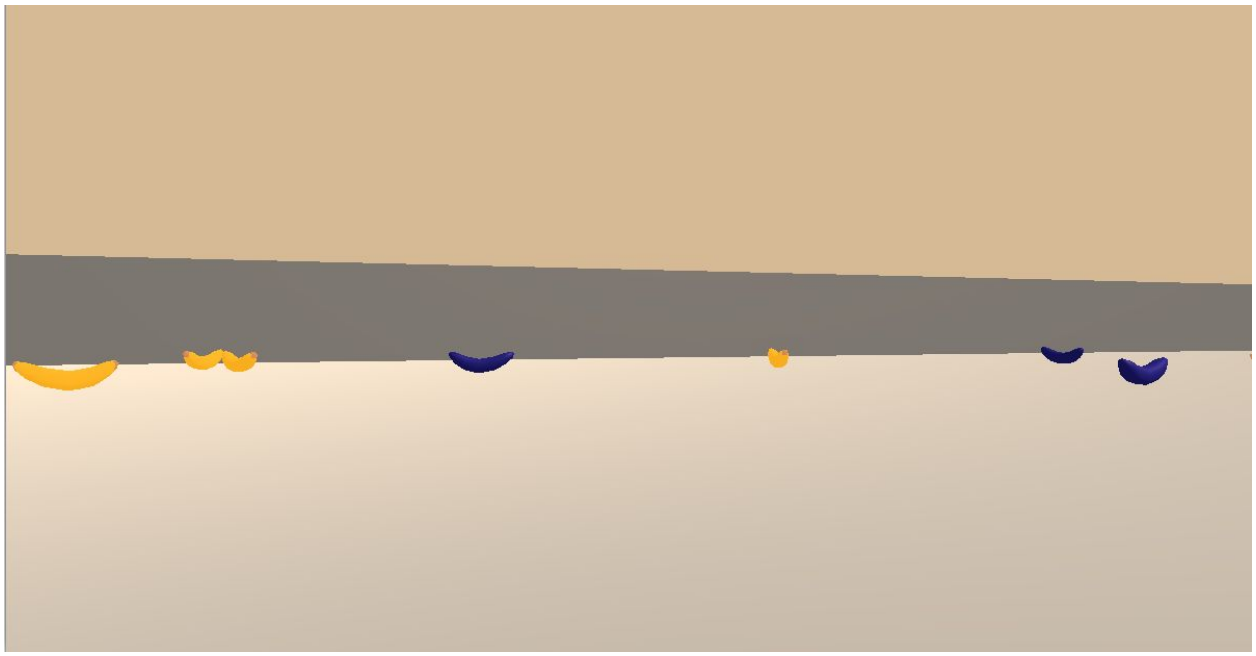


Project Report

Navigation with Double Q-Learning

REDZHEP MEHMEDOV REDZHEBOV

23/04/2020

Introduction

In this project, the Double Deep Q-Learning algorithm is used to solve the Navigation environment. Double Q-Learning is an advanced variation of Q-Learning to overcome overestimation problems. Using the local q network to determine the best actions for the next state. The algorithm requires two neural networks(q_local and q_target) and those networks are identical. In the algorithm, the local network is responsible for all the interactions with the environment. The target network is responsible for determining the expected next state-action values. Thus, it allows us to train the local network using the difference between expected next state action(q_target) values and given next state action values(q_local). After several training steps, the target network needs to be updated using the local network's trained weights because if the delay is too long, the target network can remain out-dated and it can directly influence the training of the local network.

Pseudo-Code :

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

Implementation Details

Target models are updated once in every “4” step. During model initialization, the initialized local model’s weights are copied into the target model’s weights to prevent initialization based on high model difference problems.

Neural Networks(Q_Local & Q_Target) : State_Size -> 64 -> 64 -> action_size
Output Gate of Actor Network : Linear f(x) = x
Learning Rate : 5e-4

Batch Size : 128
Buffer Type : Classic Experience Replay Buffer
Buffer Size : 1e5

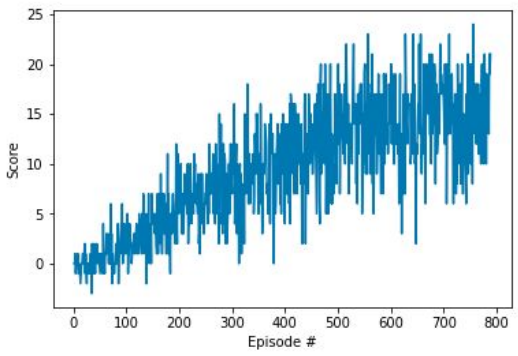
Model Update Rate (Tau) : 1e-3
Weighted Decay : 0
Discount Rate : 0.99

SEED : 1337(LEET FTW!)

Results

The environment is solved in 789 episodes - 689 epochs and the targeted average score was 15.0.

```
Episode 100     Average Score: 0.65
Episode 200     Average Score: 3.51
Episode 300     Average Score: 6.95
Episode 400     Average Score: 9.09
Episode 500     Average Score: 11.79
Episode 600     Average Score: 14.10
Episode 700     Average Score: 14.53
Episode 789     Average Score: 15.03
Environment solved in 689 episodes!     Average Score: 15.03
```



Further Work

- Distributed Training to collect different observations faster. It will allow the agent to learn faster.
- Hyper-parameter Optimization: Better initial hyper-parameters can lead the agent to learn faster.
- Prioritized Replay: Instead of sampling a chunk of observations randomly, we can sample them based on the observation's quality. Observation's quality can force the agent to focus on the cases that it failed very badly. It is kinda adding extra intuition into the model
- Dueling Network: Instead of calculating the Q values directly, we can compute the advantage A(s, a) values by estimating state value and state action vectors. It can force the agent to learn faster by understanding the importance of possible actions in any observed but badly performed state.