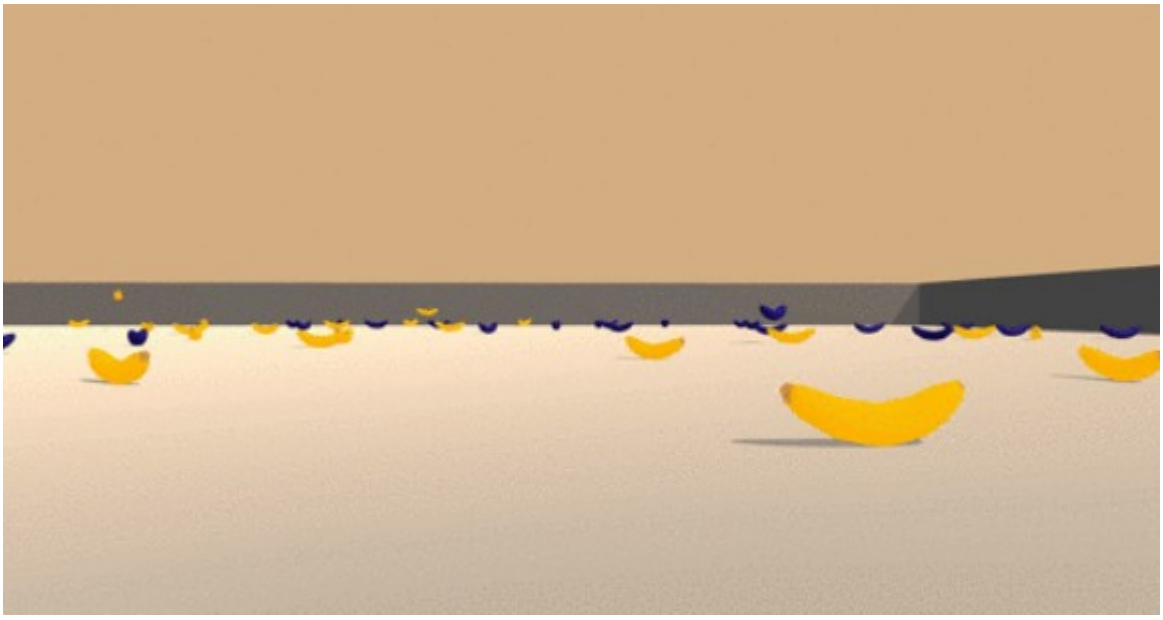
Deep Reinforcement Learning – Navigation

Arpit Savarkar

1. Overview

The project consists in training an agent that learns by himself; that is by trials and errors, to collect special type of bananas (yellow ones) while avoiding blue ones in a restricted environment. which consists of a continuous state space of 37 dimensions, with the goal to navigate around and collect yellow bananas (reward: +1) while avoiding blue bananas (reward: -1). There are 4 actions to choose from: move left, move right, move forward and move backward.



2. Method of Approach

Deep Q-Learning alone has been found to be notoriously unstable when neural networks are used to represent the action values. In particular, two key features addresses these instabilities **:**

• Experience Replay

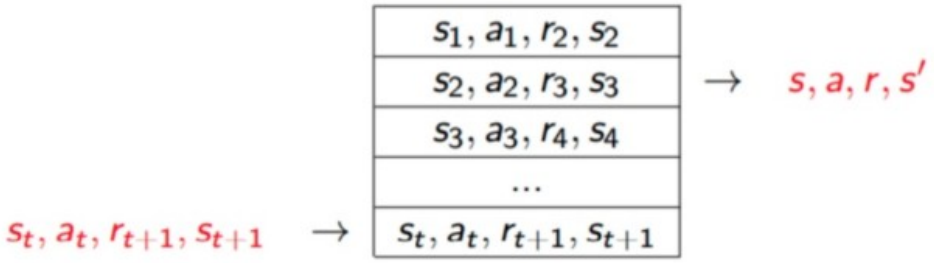
• Fixed Q-Targets

To step back for a bit, the idea of Q-learning is to learn the action-value function, often denoted as Q(s, a) , where S represents the current state and a represents the action being evaluated. Q-learning is a form of Temporal-Difference learning (TD-learning), where unlike Monte-Carlo methods, we can learn from each step rather than waiting for an episode to complete. The idea is that once we take an action and are thrust into a new state, we use the current Q-value of that state as the estimate for future rewards.

The replay buffer contains a collection of experience tuples (S, A, R, S 0). 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 reaking 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.

A Buffer size of 45000-50000 is used to store the experience. These values are high enough to prevent correlations between the frames and learn from a more vivid background.

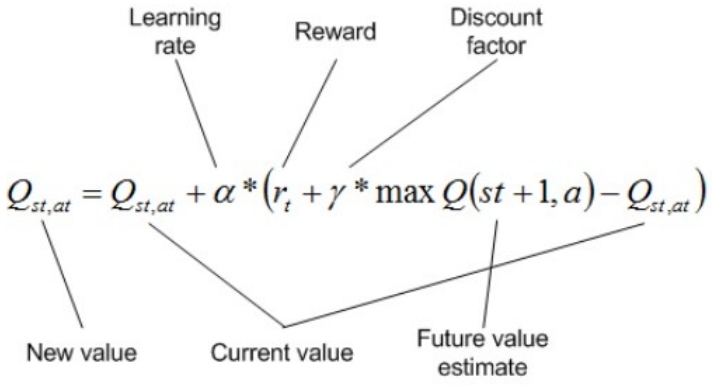


Replay Buffer – fixed size

There's one specific problem here. Since our space is continuous, we can't use a tabular representation. Hence, we use a Function Approximator . The idea behind a function approximator is to introduce a new parameter the Q(s, a) , approximation that helps us to obtain an approximation of. So, this becomes a supervised learning problem where the represents the expected value and becomes the target. We then use mean-square error as the loss function and update the weights accordingly using gradient descent. Now, the choice remains to choose the function approximator. Enter Deep Learning! We use a neural network as function approximator here. More specifically, we choose a 2-hidden layer network with both the layers having 64 hidden units with relu activation applied after each fully-connected layer. Adam was used as the optimizer for finding the optimal weights.

2.1 Fixed Q – targets

The target during training itself is dependent on w , the parameter being updated. This leads to constantly moving targets and hurts training. The idea behind fixed q-targets is to fix the parameter w used in the calculation of the target, . This is achieved by having two separate networks, one is the online network being learned and the other being the target network. The weights of the target network are taken from the online network itself by freezing the model parameters for a few iterations and updating it



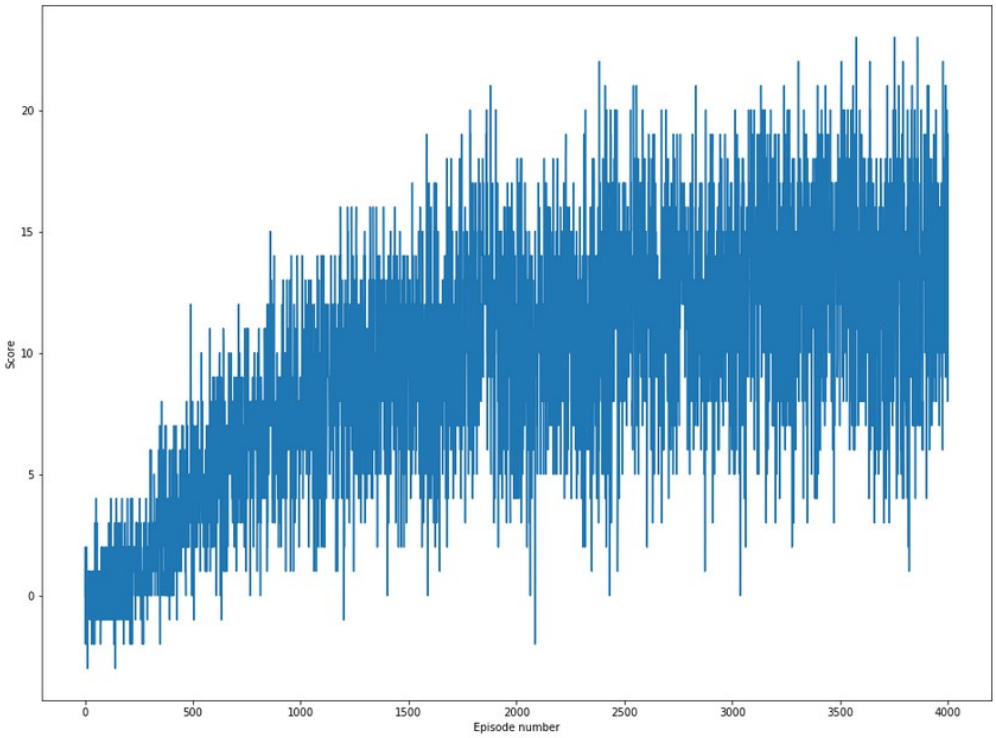
periodically after a few steps. By freezing the parameters this way, it ensures that the target network parameters are significantly different from the online network parameters.

However, the problem is that we are using the same parameters (weights) for estimating the target and the Q-value. As a consequence, there is a big correlation between the TD target and the parameters we are changing. Therefore, it means that at every step of training, our Q-values shift but also the target value shifts. So we are getting closer to our target but the target is also moving. It is like chasing a moving target and thus results to divergence or oscillations in the training phase. To remove these correlations with the target, we use a separate target network that has the same architecture as the Deep Q-Network. The change lies in the fact that we are updating the target Q Network's weights less often than the primary Q-Network.

To remove these correlations with the target, we use a separate target network that has the same architecture as the Deep Q-Network. The change lies in the fact that we are updating the target Q-Network's weights less often than the primary Q-Network. with : w − ← w at every C steps In the implementation, we update the target Q-Network's weights every 4 time steps. As for the other parameters involved in the update, α or the learning rate is initialized at 5e − 4 and is learned thanks to the usage of Adam optimizer. The discount rate γ is set to 0.99 which is very close to 1, meaning that the return objective takes future rewards into account more strongly and the agent becomes more farsighted.

3. Policy & Result

The learning policy is Greedy in the Limit with Infinite Exploration (GLIE). This strategy initially favors exploration versus exploitation, and then gradually prefers exploitation over ex- ploration. As such, we train using an #-greedy policy with # annealed exponentially from 1.0 to 0.005 with a decay of 0.999 every episode. After 2491 episodes, the parameter # is fixed at 0.005, but in this project we do not train the agent this long.



Improvement Scope

Using Prioritized Replay (paper) showed a massive improvement over Double DQNs for Atari games. It is expected that it'll lead to an improved performance here too. Other improvements to the original DQN algorithms that were briefly mentioned in the course could be potentially beneficial too: learning from multi-step bootstrap targets , Distributional DQN, Noisy DQN Hyperparameter search for both Double DQNs and Dueling Double DQNs should lead to better performance too.