

Navigation Project Report

Udacity Nano Degree Program

Student:

Nikhil Masinete

Mentor:

Alessandro R

Course:

Deep Reinforcement Learning

Introduction

Reinforcement Learning is a branch of machine learning where a software agent is taught to how to take actions in an environment in order to maximize its cumulative reward. This theory is fundamentally based on Markov Decision Process. This concept resembles the way living beings live. The following are the terms which are frequently used in this report.

Agent: An agent is the smart being which is trained to achieve tasks given to it.

Environment: An environment is a situation where the agent lives.

State: A state is a quantified description of a situation.

Action: An action is the way how the agent should respond in each state.

Reward: A reward is the outcome given to the agent for taking a particular action in each state. A reward is positive or negative based on the action taken was desirable or undesirable.





State-Action value function (Q function): A state action value function is a matrix that stores the expected reward when a particular action is taken in each state.

Policy: The policy is the strategy that the agent employs to determine the next action based on the current state.

Episode: All states that come in between an initial-state and a terminal state.

Challenge

Banana collector is a game where environment consists yellow and blue bananas scattered all over the environment. A gamer can move in the environment in 2 directions.

-  Right
-  Left
-  Forward
-  Backward

When the gamer pass over a yellow banana he gets +1 reward and -1 if he passes over a blue banana. So, the challenge is to train an agent which can collect 13 bananas.

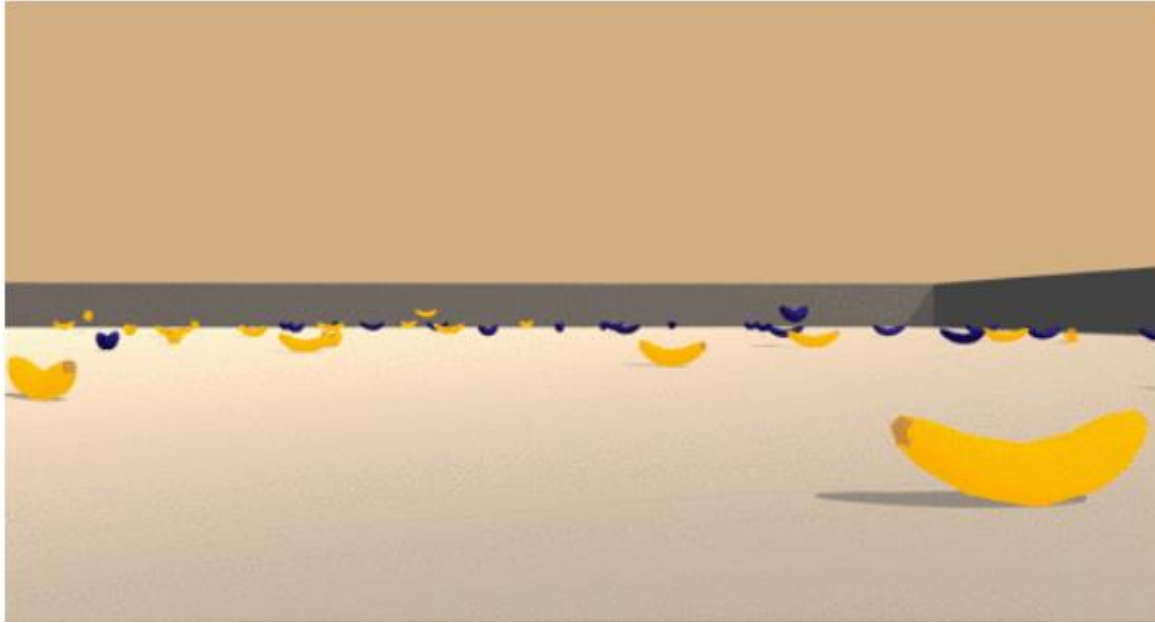


Figure 1: Banana Collector Environment

Algorithm

In deep reinforcement learning, neural networks are used as Q functions. Based on the values of these Q-functions the agent chooses an action at every state.

In 2015, Double Deep Q learning algorithm was implemented by Google DeepMind to make an agent play the classic Atari 2600 games. The trained agent was able to achieve a level comparable to that of a professional human games tester across a set of 49 games.

The same algorithm was implemented in this case to solve the given problem. In this algorithm, replay memory is initiated with a capacity of N . This stores the state, action reward and next state in the form of tuples. The neural network is trained on the batches with random samples drawn from this memory set randomly. A neural network model is created, and it called twice. One is the current action-value function and the other is the target action value function. The algorithm runs until it completes M number of episodes and each episode is run T number of times. At each and every state an action is taken randomly or according to the current policy. The reward and the next state will be obtained on passing the selected action into the step function. After this pre-processing the next state is done. A batch of random samples are drawn from the memory and the neural nets are trained on these tuples. The current action value function is given the current state as input whereas the target action value function is given next state as input. The difference of the action values is used to train the current network. Later these weights are copied to the target action values.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

With probability ε select a random action a_t

otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

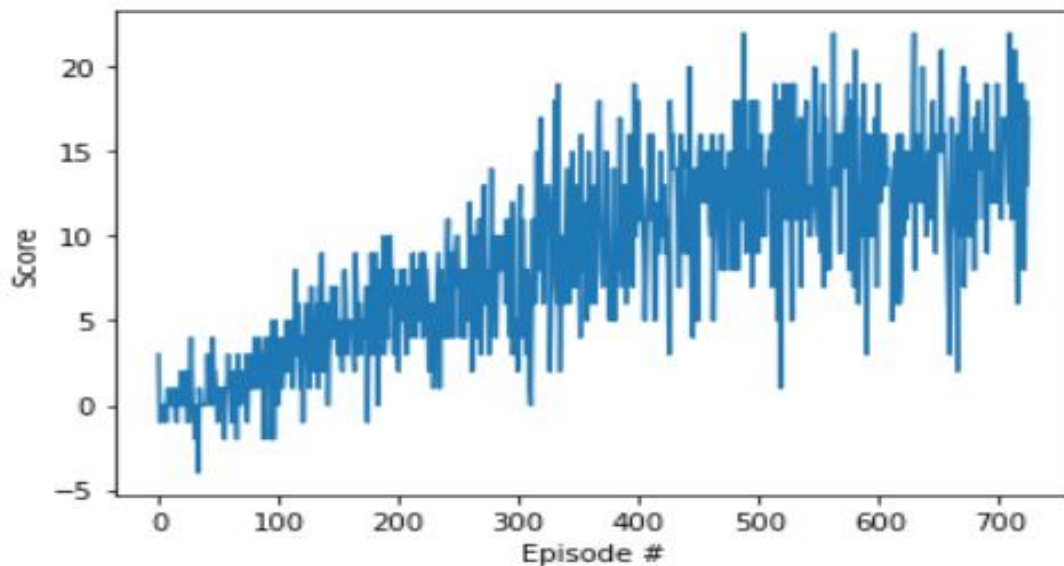
End For

End For

Figure 2 Algorithm of Deep Q Learning [1]

A neural network model of 3 hidden layers is used. The number of units are 64, 32, 16.

Result



The target is to score a 13 for 100 consecutive episodes. This model achieved an average score of 14 in 625 episodes.

Figure 3 Plot showing the score for episode.

Future Improvements

The performance can be improved by implementing Prioritized experience replay technique and Duel DQN. Currently, efforts are being taken to implement Prioritized experience replay.

References

1. Human-level control through deep reinforcement learning, Google DeepMind.