Deep Reinforcement Learning Nanodegree
Project 1: Navigation

Submission by
Avinash Kaur
(12th May 2019)

1. Objective

This projects intends to translate our understanding of different RL algorithms to solve an environment. Understanding the nature of the state-space and therefore choosing the appropriate algorithm is inherent to the solution.

2. Problem Statement

For this project, we will train an agent to navigate (and collect bananas!) in a large, square world.

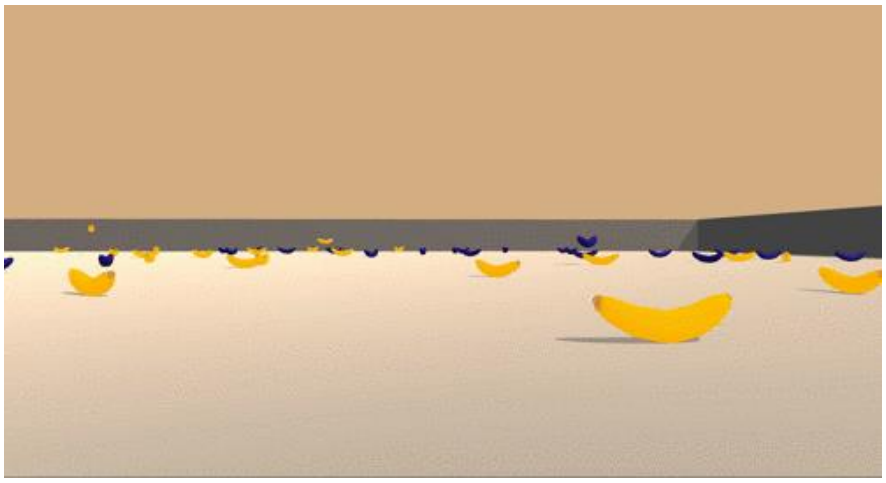
A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

- **0** - move forward.
- **1** - move backward.
- **2** - turn left.
- **3** - turn right.

The task is episodic, and in order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

3. Introduction



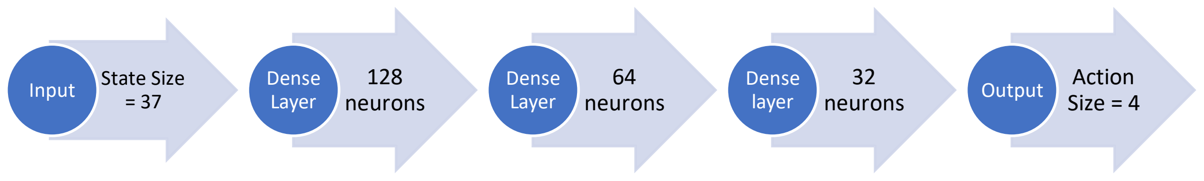
There are several ways that RL problems are posed. A finite state Markov Decision Process that can be solved through Monte Carlo Methods generally requires the agent to go through the entire episode before making any decision, and is therefore suitable to discrete, finite number of states. Whereas, TD control methods update the Q table much more frequently and therefore, the agent learns on the go. However, in Q learning as well, we do construct a Q table that can only accommodate discrete, finite number of states.
In real world applications, the environments usually consist of a large number of continuous states, and a discrete number of actions. Even amongst the states, there are non-linear relationships among different basis functions of our state-space, which cannot be modelled by simple linear function approximators. This brings us to parameterize the given state space to accommodate these non-linearities and hence poses this RL Problem, the solution for which can be effectively solved using Deep Neural Networks.

We parameterize our state-value function (as well as our action-value function) to introduce the vector ***W*** that essentially form the 'weights' in our regular deep-conv nets. We train this Convolutional Neural Network with our input being the different (S, A, R, S') tuples in a batch. To decouple correlations amongst consecutive episodes, we use the Experience Replay Strategy to uniformly sample 'batch_number' number of episodes from our created replay buffer. This allows us to exploit the experience that the agent gains over time. We also use fixed Q targets to decouple correlation in a moving target while accommodating mathematical coherence in the solution.

In this project, we are using PyTorch to train the neural network. The Banana Collector environment has been created using Unity ML Agents. For the purpose of this project, we use the environment provided as is and do not create our own.

4. Technical Description of the Solution

We trained a fully connected Neural Network with 3 dense layers. The Configuration of the Neural network is as follows:



The saved model has been trained using the following hyperparameters:

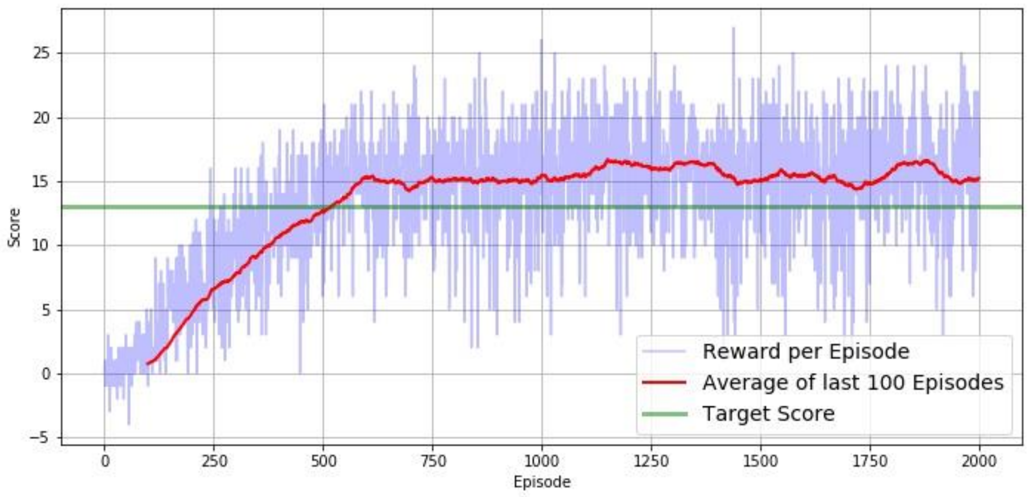
Optimizer	Type	Adam
	Beta Values	[0.9, 0.999]
	Learning Rate	5e-4
	Weight Decay	0.0001
GLIE params	Eps_start	1.0
	Eps_end	0.005

	Eps_decay	0.995
DQN	Gamma	0.99
	Tau	1e-3
	Update_every	4
	Buffer_size	5e4
Training	Batch Size	64

- The learning policy is Greedy in the limit with Infinite exploration. The epsilon value starts at1.0 and keeps decreasing by 0.995 at every episode. The final epsilon value is 0.005.
- The DQN that was trained uses Experience Replay strategy to stabilize the training. The replay buffer stores 50,000 experience tuples at a given time.
- We update the Fixed Q targets at every 4 time steps.
- We used the Adam optimizer with a learning rate of 0.0005.
- Gamma or the discount rate is pretty high – 0.99. This allows the agent to look far into the future while making current decisions.

5. <u>Results</u>

The environment was solved in 421 episodes. The plot of scores per episode is as follows:



6. <u>Future Improvements</u>

Just like any other neural network, the future improvement for our current DQN definitely involves changing the model architecture and tweaking the hyperparameters. Along with this, we can experiment and observe the effects of prioritized experience replay, Double Q Networks, Dueling Q networks. We can even go beyond that and train a Rainbow network to solve this environment with a more ambitious target.