Report - project 1: Navigation, Banana Simulator

I. Problem statement:

The goal of this project was to create Deep Reinforcement Learning (DQN) algorithm to play banana simulator. The agent's goal is to collect as many yellow bananas (reward +1) as possible in given time, and avoid blue bananas (reward -1). The environment is considered to be solved when agent collects average score of +13 over 100 consecutive episodes.

To achieve the goal, agent can undertake one of four actions: move forward, backward, left and right.

The environment state space observed by agent has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction.

Fig 1. Banan simulator screenshot

II. Algorithm Overview:

To solve the problem I've decided to implement deep Q-learning algorithm with 3 layer neural network. To reduce oscillation in learning I utilise fixed Q-target. In addition I apply replay memory to improve learning rate.

Using this algorithm, agent is able to solve environment (average score of +13 over 100 episodes) after ~410 episodes. After further training the agent gets close to learning asymptote which is about +16.5 average score points over 100 episodes.

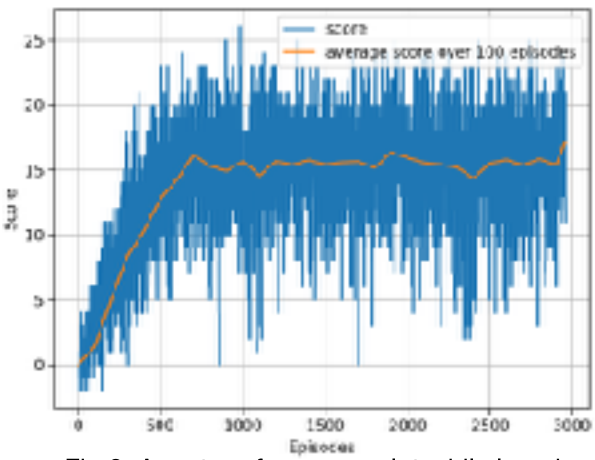

Fig 2. Agent performance plot while learning

III. Neural-Network:

The neural network I've chosen is a simple 3 linear layers network, with:
- Input layer with 37 inputs, and 64 outputs
- Hidden layer with 64 inputs, and 64 outputs
- Output layer with 64 inputs, and 4 outputs

IV. Replay memory:

To overcome problem with temporal correlation between state and actions I decided to use Memory Replay (MR). MR is a technique where agent's experience is stored in buffer and in each time stamp experience is randomly selected to update parameters.

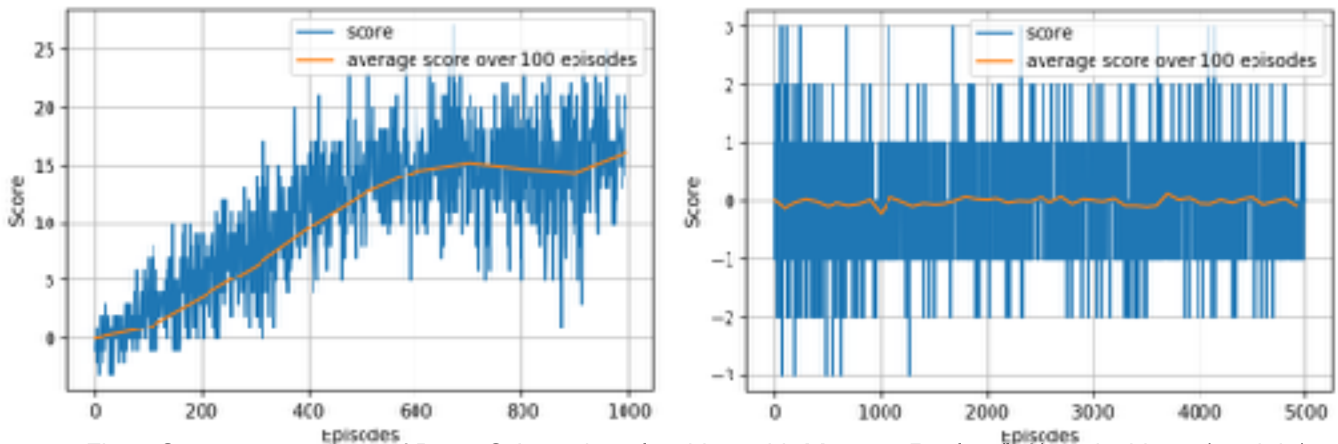

Fig 2. Score comparison of Deep Q-Learning algorithm with Memory Replay (left) and without (on right).

V. Hyperparameters tuning:

The solution has multiple hyperparameters including:
- number of neural network layers,
- number of neutrons in each layer,
- memory replay buffer size
- memory replay batch size
- discount factor: gamma
- learning rate
- epsilon.

All those parameters influence learning rate and algorithm performance. Because of number of those parameters I decided to use some parameters which worked in other projects and just tune: number of neurons in layers, memory replay buffer and batch size.

a) Number of hidden neurons: I tested 4 different neural network architectures, with different number of neurons in input and hidden layers.

Number of input neurons	Number of hidden neurons	Number of output neurons	Time to solve environment	Time Δ to chosen NN	Learning asymptote
64	64	4	409 episodes	0%	+16.5
32	32	4	492 episodes	20%	+14.0
128	64	4	399 episodes	-3%	+17.0
128	128	4	427 episodes	5%	+17.0

I've decided to use 64 neurons for input and hidden layer. Although 128 neurons on input layer increase a little bit learning time and final score it makes NN much more complicated and because of that I reject it.

b) Memory replay buffer size: I've tested number of memory replay buffer size (in range 10^2 to 10^7) and this parameter has almost no effect on learning rate nor learning asymptote.

c) Memory replay batch size: I've investigated batch in size of 16, 32, 64 and 128. The biggest batch size, the less episodes agent needed to solve environment. There is a huge difference between 16, 32 and 64. On the other hand 128 batch size has reduced learning time just by few episodes. As a result I decided that 64 memories will be the best choice taking into account learning time and computational cost.

VI. Area for improvement:

To achieve better performance and decrease learning time I see following are for improvement:

- To catalyse learning rate I can apply Prioritised Memory Replay instead of MR
- To get over oscillation in region where algorithm gets close to it's asymptote I can implement Double DQN or Dueling DQL.

Another possible solution is to use image instead of provided state space.