PROJECT 1: NAVIGATION UDACITY DEEP REINFORCEMENT LEARNING

IMPLEMENTATION REPORT

Andrew Sperazza

IMPLEMENTATION DESCRIPTION

The Navigation Project is an environment that has both yellow and blue bananas in a square 3-dimensional environment. The agent is required to collect only yellow bananas and avoid collecting the blue bananas.

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

A reward of +1 is provided for collecting a yellow banana, and a reward of −1 is provided for collecting a blue banana.

A Deep-Q-Network is utilized to learn the proper actions for each state presented. The agent is trained in episodic format, for a maximum of 2000 episodes. The project is considered solved if the score is at least 13, averaged over 100 episodes.

This implementation solved the environment in 900 episodes, using only an increasing gamma, and exponentially decreasing epsilon value. Adjusting the gamma growth factor and alpha decay factor can improve this outcome.

LEARNING ALGORITHM

The learning algorithm used was a Deep Q-Network. This implementation has two primary differences from standard Temporal-Difference methods, such as Q-Learning and SARSA, by utilizing a Replay Buffer and a Neural Network.

The neural network replaces the Q-Table and allows for function approximation in a continuous state space.

The Replay Buffer keeps track of recent state-action-reward-nextstate steps in a tuple, allowing for a reduction in negative effects induced by the correlation of serial events in the sequence of order runs. The replay buffer is a fixed and adjustable size, whose contents are maintained by a deque data structure. After a batch of runs are added to the Replay buffer, the agent then uses experience replay and trains the Q-Network by sampling from the Replay buffer tuples.

TARGET AND LOCAL NEURAL NETWORKS

This implementation includes 2 neural networks, one which is trained using samples from the replay buffer, and another which is used to estimate the target values of the current learned environment.

The local Q-network is a simple multi-layer perceptron, consisting of 4 layers with 37*3, 60, 20, and 4 nodes in each layer, each of the first 3 layers is activated using the Relu activation function. The Adam optimizer is used to train the network using a Minimum Squared Error loss function, comparing the target network and the local network outputs.

The target network is not trained, but a copy of the trained local network, whose weights are then updated using a soft update. The formula used to soft update the target network is:

$$\theta_{target} = \tau \times \theta_{local} + (1 - \tau) \times \theta_{target}$$

where τ is an adjustable interpolation parameter.

EPSILON AND GAMMA PARAMETERS

Epsilon (ϵ) is used to determine how much the agent utilizes exploration or exploitation. Generally, in the beginning of training, the agent needs to stochastically explore actions based on its current state, because it has no knowledge of optimal policies. As training progresses, the agent learns some actions that are beneficial, and needs to reduce its exploration, see figure 1, this is done using the following formula (1) for the ϵ parameter:

$$\epsilon = \max(\epsilon_{\min,} e^{-0.015 \times i}) \qquad (1)$$

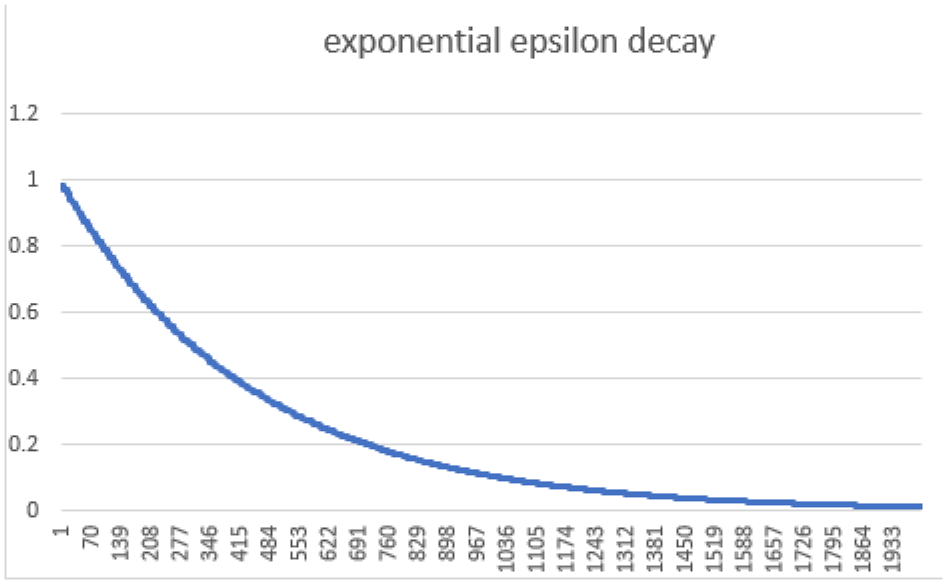

Figure 1: This implementation decreases the epsilon parameter exponentially

Reducing the epsilong parameter exponentially, decreases the required number of training episodes.

Gamma (γ) is the discount factor used to adjust the future rewards considered in the calculation of the current actions reward. A γ of 1 indicates only consider the current reward at t0, and a smaller γ incorporates future rewards. The research paper "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies", from the Univerity of Liege, Jan 2016.[https://arxiv.org/pdf/1512.02011.pdf], shows the best results are obtained if γ is increased and ϵ is decreased simultaneously. The formula for increasing γ, as shown in figure 2, is:

$$\gamma_{k+1} = 1 - 0.98(1 - \gamma_k) \quad \textbf{(2)}$$

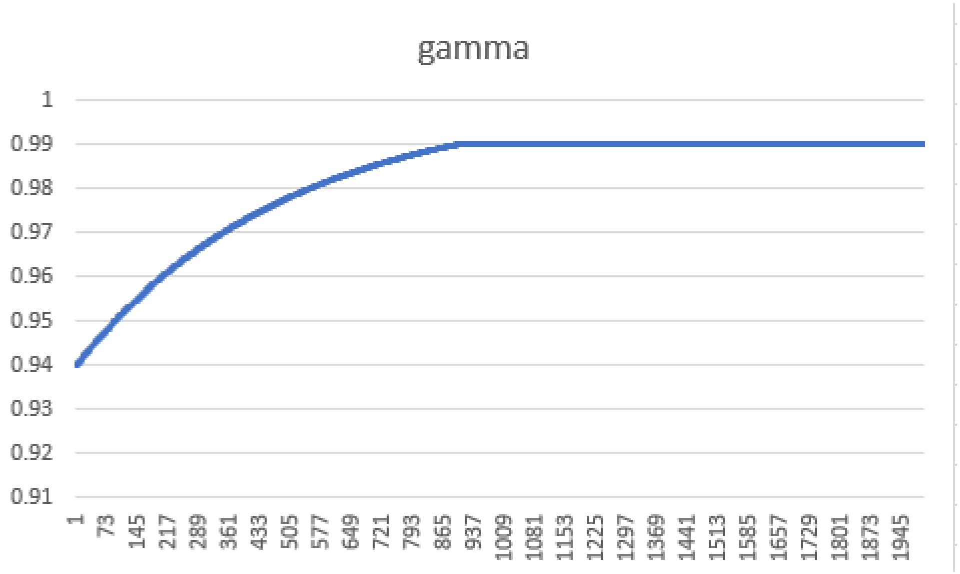

Figure 2: values used for gamma by episode number (max value applied at .99)

PLOT OF REWARDS

The agent was trained for a total of 2000 episodes, and was solved at episode 900.

```
Episode 100     Average Score: 0.51     Epsilon: 0.80     Gamma: 0.95
Episode 200     Average Score: 1.92     Epsilon: 0.64     Gamma: 0.96
Episode 300     Average Score: 4.86     Epsilon: 0.51     Gamma: 0.97
Episode 400     Average Score: 6.46     Epsilon: 0.41     Gamma: 0.97
Episode 500     Average Score: 8.07     Epsilon: 0.32     Gamma: 0.98
Episode 600     Average Score: 9.46     Epsilon: 0.26     Gamma: 0.98
Episode 700     Average Score: 11.21    Epsilon: 0.21     Gamma: 0.99
Episode 800     Average Score: 11.56    Epsilon: 0.17     Gamma: 0.99
Episode 900     Average Score: 13.01    Epsilon: 0.13     Gamma: 0.99
Episode 1000    Average Score: 13.71    Epsilon: 0.11     Gamma: 0.99
Episode 1100    Average Score: 13.50    Epsilon: 0.08     Gamma: 0.99
Episode 1200    Average Score: 14.89    Epsilon: 0.07     Gamma: 0.99
Episode 1300    Average Score: 14.47    Epsilon: 0.05     Gamma: 0.99
Episode 1400    Average Score: 14.29    Epsilon: 0.04     Gamma: 0.99
Episode 1500    Average Score: 14.62    Epsilon: 0.03     Gamma: 0.99
Episode 1600    Average Score: 14.41    Epsilon: 0.03     Gamma: 0.99
Episode 1700    Average Score: 15.84    Epsilon: 0.02     Gamma: 0.99

Environment solved in 1600 episodes!    Average Score: 15.84
```

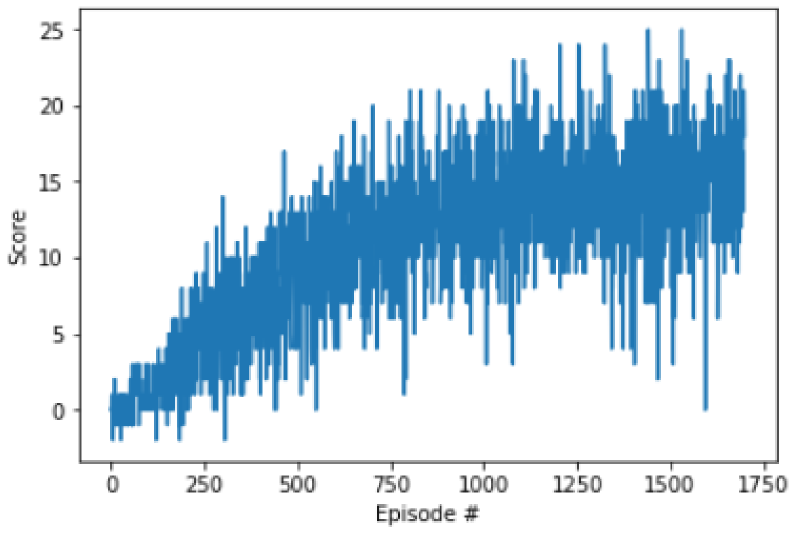

Figure 3) Plot of rewards for 2000 episodes

The agent was trained past 900 episodes and trained until it reached a higher score of 15, which occurred at episode 1700.

IDEAS FOR FUTURE WORK

There are quite a few areas that can further improve the results, these are:

1) Adjusting epsilon and gamma, possibly through automatic hyperparameter tuning
2) Adjusting the neural network architecture and possibly include Recurrent neural networks
3) Implement Double Q Learning which will help to reduce overestimation of action values
4) Prioritize Experience Replay which will allow the agent to learn more effectively from more important transitions
5) Utilize Dueling Deep Q-Networks allowing the agent to access the value of each state without having to learn the effect of each action.
6) Implement a combination of the above, or a Rainbow network