P1_Navigation: Banana Collector Environment with Deep Q-Learning

Ashwati Das

Introduction *(From Udacity Course Website)*

The goal of this task is to train an agent to collect as many yellow bananas and avoid the blue bananas in a square world. The following reward structure is thus employed:

- +1 when collecting every yellow banana
- -1 when collecting every blue banana

The state space consists of 37 states including the agent's velocity and ray-based perception vectors in the forward direction. The action space consists of the following four discrete actions:

- 0 – move forward
- 1 – move backward
- 2 – turn left
- 3 – turn right

In this episodic task, the agent is considered successful if an average score of +13 over 100 consecutive episodes is achieved and maintained.

Learning Algorithm

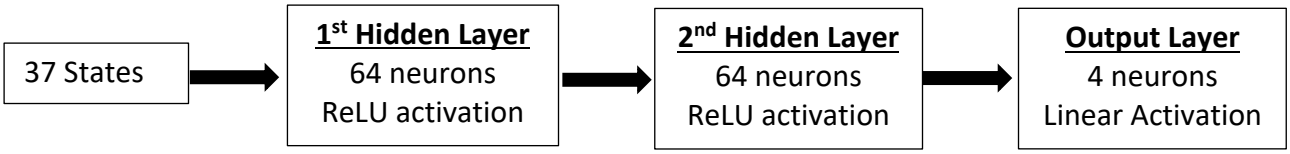
This task is approached by blending traditional reinforcement learning methods with Artificial Neural Networks, leading to a Deep Reinforcement Learning regime. In this approach, the aim is to maximize the cumulative discounted rewards obtained by the agent over time. The Deep Q-Learning strategy initializes two identical ANNs – the local and target networks. In an attempt to reduce the instability associated with nonlinear function approximation, a replay buffer is employed to store an agent's experience-related data and recall random batches from this set as training progresses. Another effort to reduce instability is to adjust the target network parameters only periodically, and the outputs from this network are used to compute the Q values of the local network. Training is improved by aiming to minimize the loss function - the error between the local and target network Q-values. Volodymyr et al. [1] describes these ideas and the Deep Q-Learning algorithm in detail.

Implementation

The implementation of the algorithm consists of design choices associated with the layer type, neurons and algorithms in the neural networks, and the hyperparameters for tuning these networks and for the training process.

Neural Network Architecture

The neural network consists of fully connected layers, specifically 2 hidden layers and one output layer with the following neuron count and activation functions.



Hyperparameters

The training outcomes are very sensitive to the hyperparameters – thus, appropriate tuning is key to the agent being successful in its task. The following are the parameters used for the training process:

Parameter	**Value**
Number of episodes	1000
Number of steps within an episode	1000
Epsilon Start	1.0
Epsilon Decay	0.995
Epsilon End	0.01
Random Seed	0

The following are the hyperparameters used to tune the reinforcement learning process:

Hyperparameter	**Value**
Replay Buffer Size	1e5
Batch Size	64
Discount Factor (γ)	0.99
Soft Update Parameter (τ)	1e-3
Learning Rate	5e-4
How often to update the network	Every 4 steps

The Adam algorithm is employed for the gradient updates associated with the neural network computations.

Files

Navigation.ipynb – Main file consisting of the environment set-up and also the training parameters for user modification.
dqn_agent.py – File consisting of the definitions for the 'agent' and 'replay buffer' classes, and also the reinforcement learning hyperparameters for user modification.
model.py – File consisting of the neural network architecture and choices

Plot of Rewards

Using the parameters and hyperparameter choices listed earlier, the environment is solved (average reward of 13 over 100 consecutive episodes) in 572 episodes, as displayed in Fig. 1.

```
Episode 568     Average Score: 12.89
Episode 569     Average Score: 12.88
Episode 570     Average Score: 12.81
Episode 571     Average Score: 12.83
Episode 572     Average Score: 13.00

Environment solved in 572 episodes!     Average Score: 13.00
```

In [7]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores)+1), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```

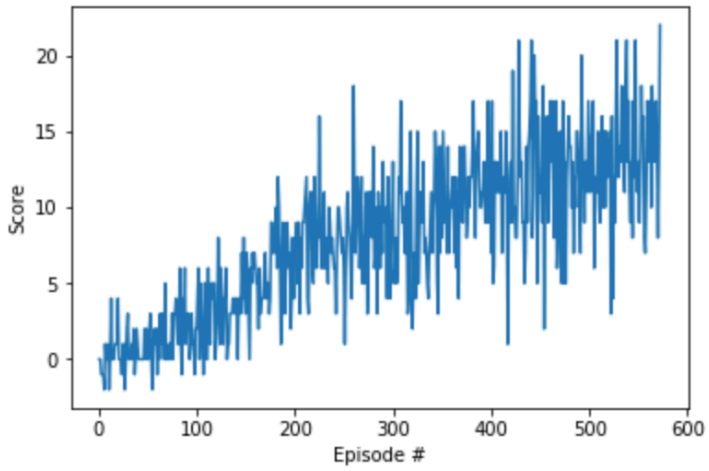


Figure 1: Plot of scores over episodes

Ideas for Future Work

- Further tuning of the hyperparameters would be attempted to investigate the potential impact on the results
- Other reinforcement learning algorithms such as Dueling Q-networks could be implemented to isolate the significance of a state to the training outcome independently of the associated actions from this state [2]. Double DQN and Prioritized Experience Replay strategies may also be potentially useful.
- Modification of the neural network architecture choices could also be investigated.

References

1. M. Volodymyr et al., *Human Level Control Through Deep Reinforcement Learning, Nature*, Vol. 518, Page 530, 2015
2. Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Feitas, *Dueling Network Architectures for Deep Reinforcement Learning*, Proceedings of the 33 rd International Conference on Machine Learning, New York, NY, USA, 2016.