Report for the Navigation Project

1. Introduction

This report introduces the algorithm used in this project to solve the navigation problem. Results with different neural network structures and hyperparameters are shown. Future work ideas are also discussed at the last part of the report.

2. Approach

The algorithm used in this project is based on the algorithm introduced in the Nature paper from Google DeepMind[1]. Also, two further improvements are tried in this project: the double Q-Learning[2] and dueling network structures[3]. In [1], two key ideas to train the agent are experience replay and keeping two sets of neural network weights during the training.

The method used in this project is a value-based method which means the policy: $\pi: S \to A$ is implicit. The optimal state-action function $Q^*(S,A)$ is estimated. After the $Q^*(S,A)$ is estimated, the optimal policy can be derived as:

$$\pi^*(S) = argmax_{A \in A_{finite}} Q^*(S,A)$$

To estimate the optimal state-action function, Bellman optimality equation is used:

$$Q^*(S,A) = E_{S'}[R + \gamma max_{A' \in A_{finite}} Q^*(S',A') | S, A]$$

A neural network is used to approximate the state-action function which we refer as $Q(S,A;\theta)$. The neural network is then called a Q-network. The parameter in the Q-network is adjusted by minimizing the mean squared error from the Bellman optimality equation iteratively from the experience of the agent interacting with the environment. The algorithm is shown below, which basically follows the algorithm in [1] with the double Q-learning idea in [2].

```
-----------------------------------------------------------------------------------------------------
Initialize the online action-value function Q-network with random parameters $\theta$
Initialize the target action-value function Q-network as $\theta^- = \theta$     (online Q-network and target Q-network)
Initialize the replay memory D to capacity N                                  (Experience Replay)
Initialize $\varepsilon = 1.0$

For episode = 1, M do
    For t = 1, T do
        With probability $\varepsilon$, randomly select an action $A_t$
        Otherwise select $A_t = argmax_{A \in A_{finite}} Q(S_t, A; \theta)$.
        Execute action $A_t$ on the game environment and observe next state $S_{t+1}$ and reward $R_t$
        Store the transition $(S_t, A_t, R_t, S_{t+1})$ in D.
        Sample random minibatch of transitions $(S_j, A_j, R_j, S_{j+1})$ from D.
        Set $y_j = \begin{cases} R_j & if\ trip\ terminates\ at\ step\ j+1 \\ R_j + \gamma Q(S_{j+1}, argmax_{A' \in A_{finite}} Q(S_{j+1}, A; \theta); \theta^-) & otherwise \end{cases}$     (Double Q-learning)
        Perform a gradient descent step on $[y_j - Q(S_j, A_j; \theta)]^2$ with respect to $\theta$
        Perform soft update of the target network $\theta^-$
    End For
    $\varepsilon = \max(0.01, 0.995 \times \varepsilon)$
End For
-----------------------------------------------------------------------------------------------------
```

3. Results

I do informal search to determine the structure of the neural network with two hidden layers. The results are shown in fig. and table II. The hyperparameters are fixed for these training which is shown in table I. It can be shown that the performance does not vary a lot. The structure of 64-64 means a neural network with two hidden layers and the number of hidden units in the two layers are 64 and 64. This structure is used for further hyperparameter tuning as it achieves a highest average score in the last 100 episodes. Notice that the following results for each neural network structure only comes from one run with 1000 episodes and the stochastic nature of the game environment, the chosen structure may not be the best one.

The dueling network structure is tried and does not perform well so that the performance is not reported. The implementation is included in the NN_model.py file.

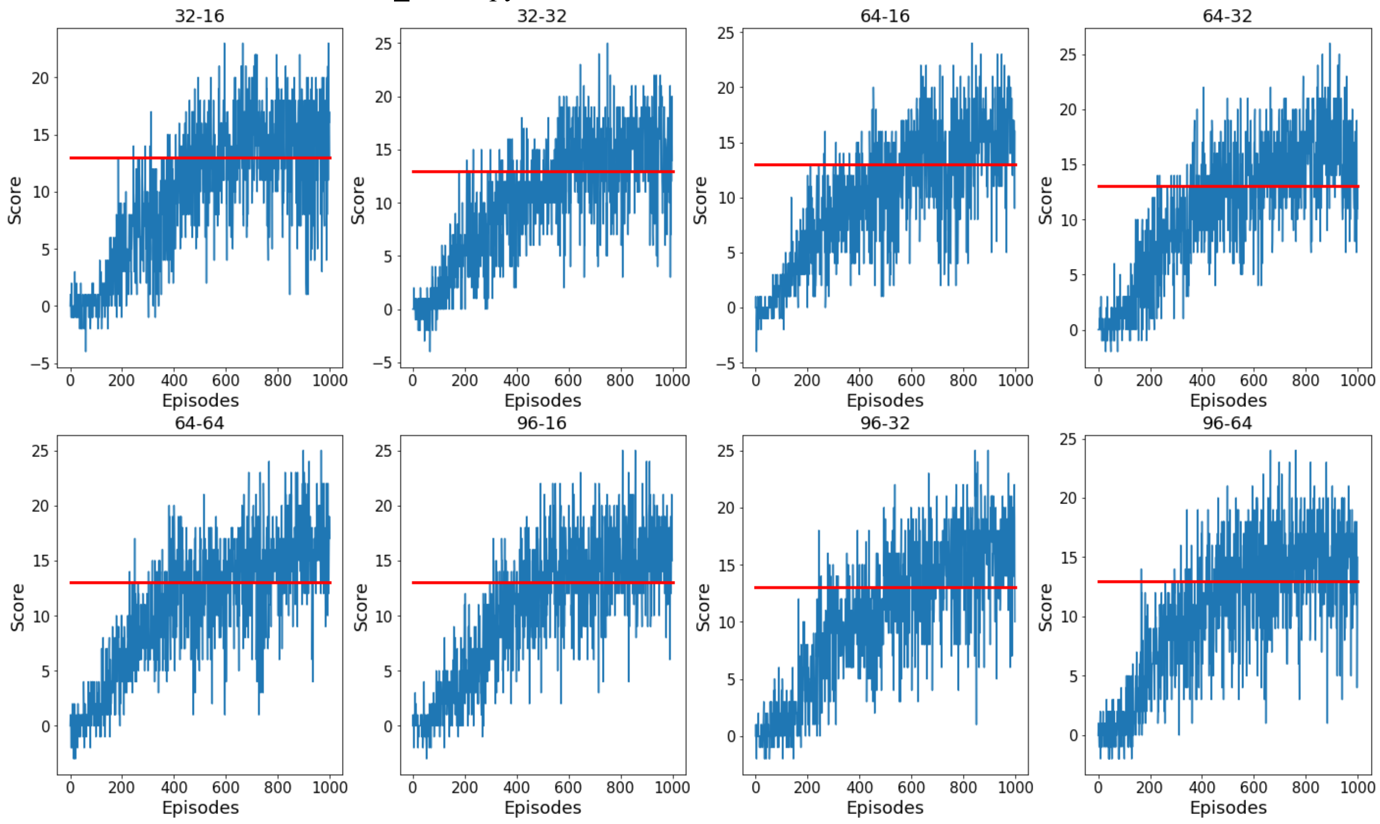

Figure 1

Table I

Hyperparameter	Value
Learning rate	0.00025
Batch size	16
Gamma	0.99
TAU	0.001
Update every	4

Table II

NN structure	Average score for the last 100 episodes	Episodes needed to achieve an average of +13 score for consecutive 100 episodes
32-16	13.24	~600
32-32	14.44	~620
64-16	15.28	~650
64-32	15.72	~630
64-64	**16.17**	**~660**
96-16	15.57	~550
96-32	16.03	~660
96-64	14.23	~660

The hyperparameter tuning is also informal. Methods like grid search, random search or Bayesian hyperparameter search can be used to find better hyperparameters. In this part, more episodes (1500) are used. Learning rate, batch size and "update every" will be roughly tuned according to the average score of the last 100 episodes. In figure 2, the tuning results are shown. For the learning rate, I fix update every as 4 and batch size as 16 and find the learning rate should be 0.00025. For update every, I use learning rate 0.00025 and batch size 16 to find the update every can be 4. At last, I fix the learning rate as 0.00025 and update every as 4 and find batch size 32 performs the best.

Consequently, the found hyperparameter is **0.00025**, **32** and **4** for learning rate, batch size and update every respectively.

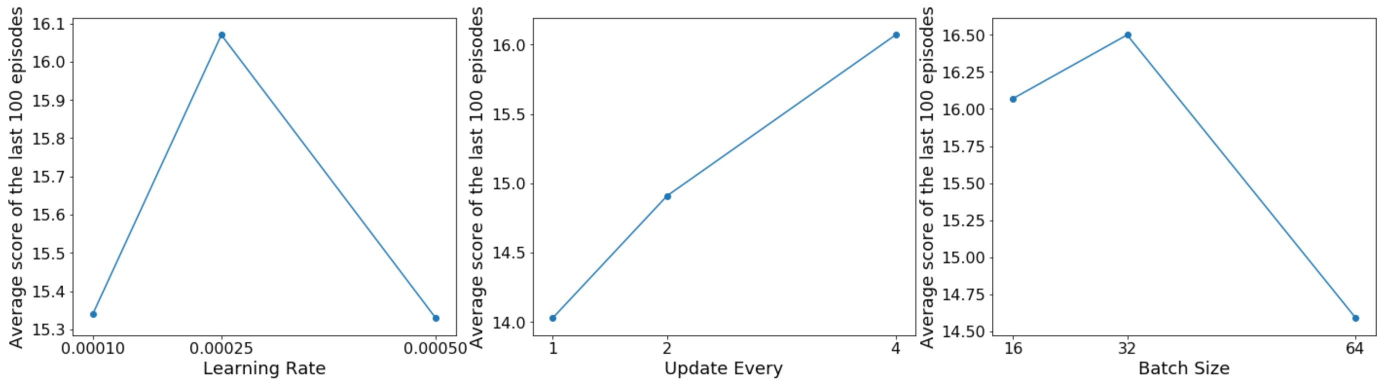

Figure 2

Using the found hyperparameters, the score is shown in figure 3. The average score for episode 495 to 595 is **13.07** so that the environment is solved within **595** episodes. The average score for the last 100 episodes is **16.5**.

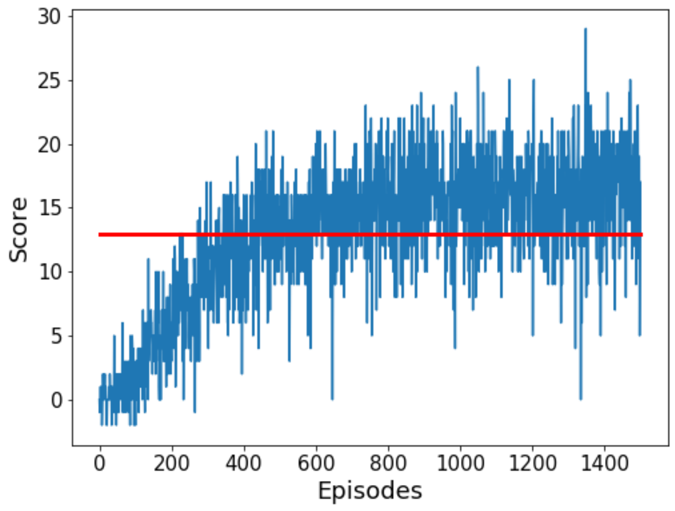

Figure 3

4. Future work

First, a more formal and systematic approach can be done to determine the structure of the neural network (number of hidden layers and number of hidden units in each layer) and the hyperparameters. Second, in this project, every time the neural network is updated, the data used is sampled uniformly from the replay buffer. It is shown that faster training speed and better performance can be achieved by prioritized sampling[5] and prioritized experience replay[6] in the corresponding games. These two sampling methods can be tried. Third, distributional RL[7], noisy nets[8] and asynchronous methods[9] are other techniques that can be explored in the future. In [10], many techniques are combined with the original work in [1], showing a new state of the art performance in the Atari games.

Reference

[1] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529.
[2] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." *AAAI*. Vol. 2. 2016.
[3] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." *arXiv preprint arXiv:1511.06581*(2015).
[4] Schaul, Tom, et al. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952* (2015).
[5] Narasimhan, Karthik, Tejas Kulkarni, and Regina Barzilay. "Language understanding for text-based games using deep reinforcement learning." *arXiv preprint arXiv:1506.08941*(2015).
[6] Schaul, Tom, et al. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952* (2015).

[7] Bellemare, Marc G., Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." *arXiv preprint arXiv:1707.06887* (2017).

[8] Fortunato, Meire, et al. "Noisy networks for exploration." *arXiv preprint arXiv:1706.10295* (2017).

[9] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." *International conference on machine learning*. 2016.

[10] Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning." *arXiv preprint arXiv:1710.02298*(2017).