

## PROJECT SPECIFICATION

### Navigation

#### Training Code

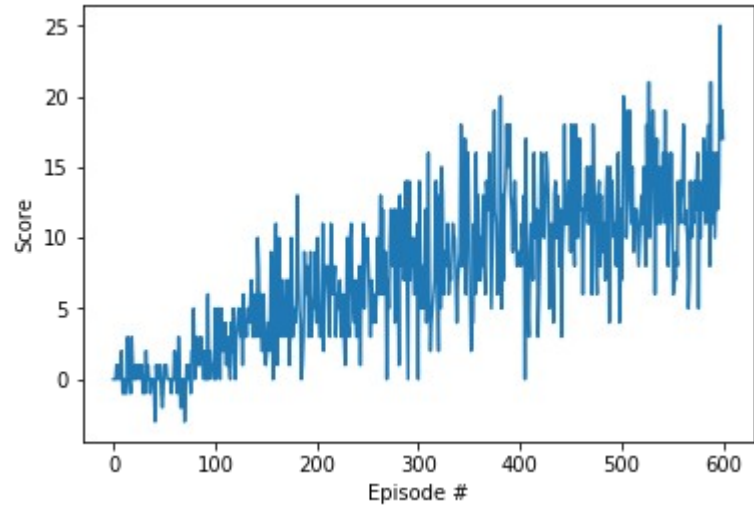
CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Training Code	The repository (or zip file) includes functional, well-documented, and organized code for training the agent.	<p>The code is mainly in 3 files.</p> <ol style="list-style-type: none"><li>1. The ipython notebook (Navigation.ipynb)</li><li>2. dqn_agent.py</li><li>3. model.py</li></ol>
Framework	The code is written in PyTorch and Python 3.	The project is implemented in PyTorch and Python 3
Saved Model Weights	The submission includes the saved model weights of the successful agent.	The model weights are saved to 'checkpoint.pth' in the project directory

# README

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
README.md	The GitHub (or zip file) submission includes a README.md file in the root of the repository.	The README.md file is present in the main directory of the git repo.
Project Details	The README describes the the project environment details (i.e., the state and action spaces, and when the environment is considered solved).	These are documented under “Project Details” section of README.md
Getting Started	The README has instructions for installing dependencies or downloading needed files.	These are documented under “Getting Started” section of README.md
Instructions	The README describes how to run the code in the repository, to train the agent.	These are documented under “Instructiona” section of README.md

## Report

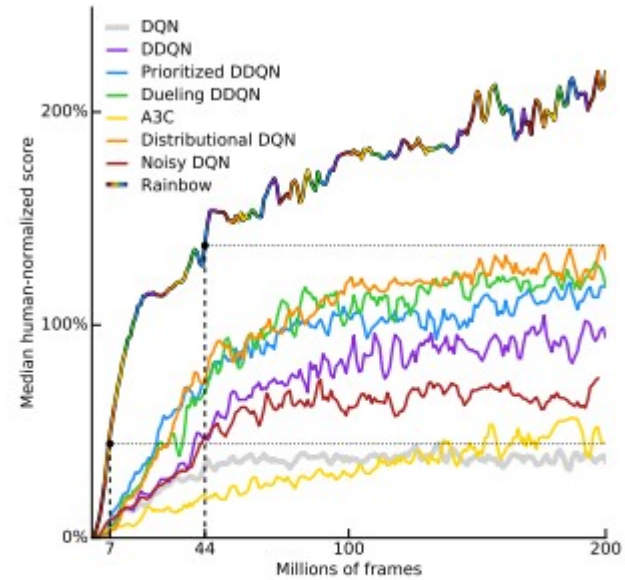
CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Report	The submission includes a file in the root of the GitHub repository or zip file (one of <code>Report.md</code> , <code>Report.ipynb</code> , or <code>Report.pdf</code> ) that provides a description of the implementation.	The Report.pdf (this file) is provide in the main directory of the repository
Learning Algorithm	The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.	<p>The learning algorithm is provided in the next page below this table. The algorithm is from the DQN paper (<a href="https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf">https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf</a>)</p> <p>The hyperparameters used are:</p> <p><code>BUFFER_SIZE = int(1e5),</code></p> <p><code>BATCH_SIZE = 256,</code></p> <p><code>GAMMA = 0.99,</code></p> <p><code>TAU = 1e-3,</code></p> <p><code>LR = 5e-4,</code></p> <p><code>UPDATE_EVERY = 4</code></p> <p>Network Architecture:</p> <p>The network has two hidden layers with 64 nodes and relu activation.</p>

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Plot of Rewards	A plot of rewards per episode is included to illustrate that the agent is able to receive an average reward (over 100 episodes) of at least +13. The submission reports the number of episodes needed to solve the environment.	<p>The rewards plot is as below:</p>  <p>The plot shows the score per episode over 600 episodes. The x-axis is labeled 'Episode #' and ranges from 0 to 600. The y-axis is labeled 'Score' and ranges from 0 to 25. The plot shows a noisy upward trend, starting near 0 and ending near 25.</p>
Ideas for Future Work	The submission has concrete future ideas for improving the agent's performance.	<p>The DQN implementation can be improved several ways for better performance.</p> <p>1. One of the most effective so far I have seen is the rainbow algorithm as mentioned in the rainbow paper (<a href="https://arxiv.org/pdf/1710.02298.pdf">https://arxiv.org/pdf/1710.02298.pdf</a>). As seen in the picture below, we can try individual algorithm. But the most effective one is the rainbow.</p>

CRITERIA

MEETS SPECIFICATIONS

STUDENT COMMENTS



**Figure 1: Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

2. Additionally, we can use systematic ways of tuning the neural network hyperparameters (such as learning rate) and their loss functions and optimizers and the network architectures themselves.

Learning Algorithm:

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

        With probability  $\varepsilon$  select a random action  $a_t$

        otherwise select  $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$

        Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the network parameters  $\theta$

        Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**