

[Return to Classroom](#)

Navigation

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulation on completing all project requirements successfully 🎉🎉

You did a great job 🎉🎉

- You have completed your Training code module successfully.
- You have completed your Readme file module successfully.
- You have completed your Report section successfully.

If you have any doubts or queries over any comment, feel free to post them on [Knowledge section](#) and our whole community along with our mentors are waiting to help you resolve all your queries and doubts.

Stay Udacious 😊

Training Code



The repository (or zip file) includes functional, well-documented, and organized code for training the agent.

🟢 **Feedback:** All the required files were submitted. The submission included well-documented and organized code for training the agent. You made an excellent choice in implementing the DQN algorithm for this project as it is an effective reinforcement learning algorithm for relatively simple, discrete action-space environments such as the one found in this project.

You should definitely check the following resources to progress with the learning further:

- [Check out Using Keras and Deep Q-Network to Play FlappyBird to train an agent on an environment just by taking the screen pixels as the input](#)
- [Speeding up DQN on PyTorch: how to solve Pong in 30 minutes](#)
- [Advanced DQNs: Playing Pac-man with Deep Reinforcement Learning by mapping pixel images to Q values](#)



The code is written in PyTorch and Python 3.

● **Feedback:** You used the Python 3 and the PyTorch framework, as required.

💡 **Insight:** You may be wondering why Udacity has standardized on PyTorch for this DRLND program. I don't know the definitive answer, but I would guess that:

- PyTorch is easier to understand and more straight-forward to use than other alternatives (compared to TensorFlow in particular).
- TensorFlow and Keras are used in other Udacity AI NanoDegrees and it's important to be familiar with many different frameworks, so picking a different one for the DRLND program is a way to ensure Udacity's graduates are well-rounded and capable of working in many different environments.
- The most probable reason is that many baseline implementations of Deep RL environments and agents are written using the PyTorch framework so students of this Nanodegree will best be able to understand and build on those implementations by coding in this commonly used framework.



The submission includes the saved model weights of the successful agent.

● **Feedback:** You have done great job of submitting model learned weights for your trained model.

💡 **Bonus Pts:** It's good practice to save model state and weights in any deep learning project you work on, as you have done with this project. You will often find it necessary to revisit a project and perform various analyses on your deep learning models. And, perhaps just as important, deep learning training can take a long time and if something goes wrong during training, you want to be able to go back to the "last good" training point and pick up from there. So, it's nice to see that you have developed the habit of automatically creating checkpoints at defined intervals (every episode, in this case) and not just at the end of training. Kudos!

README



The GitHub (or zip file) submission includes a `README.md` file in the root of the repository.

● **Feedback:** You included the required README.md file.

💡 **Pro Tip:** [Github provides some excellent guidance on creating README files:](#)

Here's a summary of their key points:

A README is often the first item a visitor will see when visiting your repository. It tells other people why your project is useful, what they can do with your project, and how they can use it. Your README file helps you to communicate expectations for and manage contributions to your project. README files typically include information on:


- What the project does
- Why the project is useful
- How users can get started with the project
- Where users can get help with your project
- Who maintains and contributes to the project

Take a look at the following links to improve how your README looks

- [How to write Meaningful Readme.md file](#)
- [How to write a kick ass readme](#)



The README describes the the project environment details (i.e., the state and action spaces, and when the environment is considered solved).

 **Feedback:** Your README file meets the requirements of the rubric by describing the project environment details, including the success criteria.


```
A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana.
Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.


The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction.
Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:
* 0 - move forward.
* 1 - move backward.
* 2 - turn left.
* 3 - turn right.

The task is episodic, and in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.
```



The README has instructions for installing dependencies or downloading needed files.

 **Feedback:** Your README provided all the information needed for a new user to create an environment in which your code will run, including the Unity ML-Agents library and the customized Banana environment.

 **Pro Tip:** You can produce a list of libraries that your script is dependent upon, along with specific versions of each required library with a simple pip command: "pip freeze > requirements.txt". This will produce a requirements.txt file that your users will appreciate because it will allow them to install the correct version of every python library your project requires with another simple, one-line pip command: "pip requirements.txt".

- [How to install python packages with pip and requirements.txt](#)



The README describes how to run the code in the repository, to train the agent. For additional resources on creating READMEs or using Markdown, see [here](#) and [here](#).

 **Feedback:** Your README file describes how to load and execute the files that are the starting point for

🟢 **Feedback:** Your README file describes how to load and execute the files that are the starting point for your implementation of this project.

Report



The submission includes a file in the root of the GitHub repository or zip file (one of `Report.md`, `Report.ipynb`, or `Report.pdf`) that provides a description of the implementation.

🟢 **Feedback:** You included the required Report file.



The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.

🟢 **Feedback:** Description of all learning algorithm, hyperparameters and network architecture have been provided in great detail in the project report.

```
## DQN Parameters and Training Details:

### Parameters:

```shell

BUFFER_SIZE = int(1e5) # replay buffer size

class Config(BaseModel):
 unity_env_path: str
 batch_size: int = 64 # minibatch size
 gamma: float = 0.90 # discount factor
 eps: float = 1.0 # Greedy action (1 - eps)
 eps_end: float = 0.01
 eps_decay: float = 0.995
 tau: float = 1e-3 # for soft update of target parameters
 lr: float = 1e-3 # learning rate
 update_every: int = 4 # how often to update the network
 graphics: bool = True # Weather to show unity simulator during model training
 seed: int = 42 # Random seed

...

```



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.

🟢 **Feedback:** Thank you for including the plot of rewards per episode, and also indicating the number of episodes your agent needed to solve this environment. (13.06 is a very descent average score)

💡 **Bonus Pts:** Your graphic not only showed the episode-by-episode results, but you also added additional features to the graphic to show the 100- episode average score. Very nice!

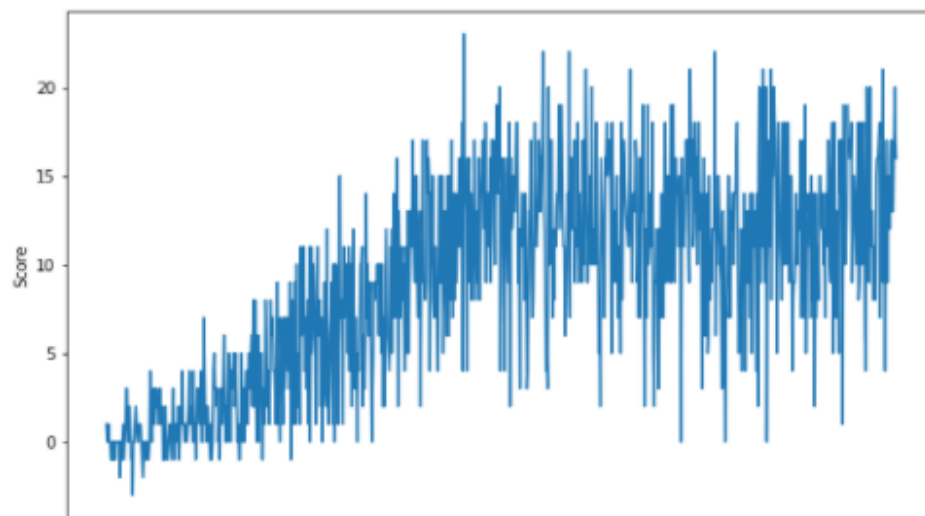
```
Episode 500 Average Score: 12.02
Episode 600 Average Score: 12.44
Episode 700 Average Score: 12.08
Episode 800 Average Score: 11.94
Episode 900 Average Score: 11.94
Episode 989/2000 Score: 13.06
Environment solved in 888 episodes! Average Score: 13.06
```

```
In []:
```

```
In [19]: from matplotlib import pyplot as plt
```

```
In [25]: plt.figure(figsize=(10,6))
plt.plot(scores)
plt.xlabel("Episode #")
plt.ylabel("Score")
```

```
Out[25]: Text(0, 0.5, 'Score')
```



The submission has concrete future ideas for improving the agent's performance.

● You have done great job of suggesting ideas such as,

- Implementing advanced RL algorithms such as Double DQN, Dueling DQN and Prioritized Action Replay

I would like you take a look on the below resources.

- [Let's make a DQN: Double Learning and Prioritized Experience Replay](#)
- [Rainbow paper which examines six extensions to the DQN algorithm and empirically studies their combination](#)
- [Conquering OpenAI Retro Contest 2: Demystifying Rainbow Baseline](#)

#### ## Future Work:

1. Using CNN Head:
  - > As discussed in the Udacity Course, further improvement could be to train this DQN using agent to observe raw pixels instead of using the environment internal state (37) dimensions.
  - > To train Convolutional Neural Network, a raw pixel image need to go for minimal preprocessing like (rescaling of an input image, converting from RGB to gray scale, ...).
2. [Double DQN](<https://arxiv.org/abs/1509.06461>):
  - > The popular Q-learning algorithm is known to overestimate action values under certain conditions.
  - > It was not previously known whether, in practice, such overestimations are common, whether they harm performance, and whether they can generally be prevented. In this paper, we answer all these questions affirmatively.
  - > In particular, we first show that the recent DQN algorithm, which combines Q-learning with a deep neural network, suffers from substantial overestimations in some games in the Atari 2600 domain.
  - > We then show that the idea behind the Double Q-learning algorithm, which was introduced in a tabular setting, can be generalized to work with large-scale function approximation. We propose a specific adaptation to the DQN algorithm and show that the resulting algorithm not only reduces the observed overestimations, as hypothesized, but that this also leads to much better performance on several games.
3. [Prioritized Experience Replay](<https://arxiv.org/abs/1511.05952>):
  - > Experience replay lets online reinforcement learning agents remember and reuse experiences from the past.
  - > In prior work, experience transitions were uniformly sampled from a replay memory. However, this approach simply replays transitions at the same frequency that they were originally experienced, regardless of their significance.
  - > In this paper we develop a framework for prioritizing experience, so as to replay important transitions more frequently, and therefore learn more efficiently. We use prioritized experience replay in Deep Q-Networks (DQN), a reinforcement learning algorithm that achieved human-level performance across many Atari games. DQN with prioritized experience replay achieves a new state-of-the-art, outperforming DQN with uniform replay on 41 out of 49 games.
4. [Dueling DQN](<https://arxiv.org/abs/1511.06581>):
  - > In recent years there have been many successes of using deep representations in reinforcement learning. Still, many of these applications use conventional architectures, such as convolutional networks, LSTMs, or auto-encoders.
  - > In this paper, we present a new neural network architecture for model-free reinforcement learning. Our dueling network represents two separate estimators: one for the state value function and one for the state-dependent action advantage function. The main benefit of this factoring is to generalize learning across actions without imposing any change to the underlying reinforcement learning algorithm. Our results show that this architecture leads to better policy evaluation in the presence of many similar-valued actions. Moreover, the dueling architecture enables our RL agent to outperform the state-of-the-art on the Atari 2600 domain.

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)

Rate this review

[START](#)