Navigation

Meets Specifications

Congratulations on completing this project! I can tell you put a lot of effort into the project and went the extra mile to make a good report to go along with your implementation.

Insight: 📓 The same type of Reinforcement Learning agent you created in this project is used in many real-world applications. Reference, in particular, the famous paper by Mnih et al with Google's DeepMind group ("Human-Level Control Through Deep Reinforcement Learning") that you may already be quite familiar with. That seminal paper used a DQN network very similar to the one you just programmed to play 49 different classic Atari video games, many at human and even super-human performance levels. This was the breakthrough that gave deep reinforcement learning respect in the AI industry and has led to a major increase in new research using, characterizing, and improving on these methods. Note also the recent announcement of the OpenAI group that developed a Reinforcement Learning bot to play Dota2 at a professional level: https://blog.openai.com/dota-2/

Good luck to you on the future projects of this Nanodegree!

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.

Comments on your implementation:

- You have implemented the Deep Q Network (DQN) and have integrated this general algorithm into the Unity ML-Agents environment so that you can control actions and get rewards from the Bananas environment.
- You have correctly decoupled parameters being updated for learning from the ones being used to produce target values. Your implementation of the soft_update() method helps to smooth out individual learning batches by preventing large fluctuations in rewards from being generated for the various actions encountered during learning. You have correctly applied the Tau hyper parameter to control your soft-update process in your agent's soft_update() method.
- You are correctly using an experience replay buffer, storing (and later, sampling) experience tuples consisting of (state, action, reward, next_state, done) fields. Your agent's step() function is correctly saving new experience tuples as they are encountered and then randomly selecting experience tuples to learn from.
- Although your agent's step() method stores new experiences in the replay buffer every time step, you have added an UPDATE_EVERY check to only perform learning every 4th step of gaining experience. This is a good technique for letting new data trajectories develop before changing things too much in the network that determines new estimates of q-values.
- Your agent's learn() method evaluates an experience sample (using DQN's target network, appropriately discounted by gamma) to estimate a Q-value. This same method also obtains a 2nd Q-value estimate from DQN's local network, uses the MSE difference between these values as a loss function to perform a back-prop learning step, and finally - nudges the target network a bit in the direction of the local network under control of the Tau hyper-parameter.
- Your agent's act() function includes an epsilon-greedy action selection mechanism to encourage exploratory behavior in the agent, particularly during the early episodes. You are correctly using the Epsilon hyper parameter to control this process in your agent's act() method.
- Your model does not include any normalization (such as batchnorm1d). Although not required for this project, you might find this a useful addition in future projects as it helps to normalize inputs to a NN layer from batch to batch and typically helps improve learning.

✓ **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:

1. PyTorch is easier to understand and more straight-forward to use than other alternatives (compared to TensorFlow in particular), or
2. 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. Or, finally,
3. 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: 🌟 Good! You created and submitted a checkpoint file containing your model's state_dict.

Pro Tip: 💰 It's good to get into the habit of saving model state and weights in any deep learning project you work on. 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, don't just checkpoint your work at the very end, but get into the habit of automatically creating checkpoints at defined intervals (say, every 100 episodes in typical RL work) or (preferably) whenever the agent improves on its previous best score. This was not a big deal with this relatively simple environment that could be solved in a few hundred episodes, but will be more important in other projects that may require several thousand episodes - or real-world problems that may require hundreds of thousands of episodes.

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: https://help.github.com/articles/about-readmes/ 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

Thank you for meeting many of these goals with your README. Although you have met the rubric requirements for this course, you might consider expanding on these points if you choose to use this project in a portfolio to show future employers.

✓ **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.

✓ **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. Although this is information that, for the most part, applied to you as a student, it is equally relevant to anyone who might want to recreate what you have done or to simply execute your implementation of this project.

✓ **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.md markdown describes not only how to set up the environment, but also how to load and execute the Navigation.ipynb file that is 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.md file, in markdown format.

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

Feedback: 🌟 This is an excellent report. You described the DQN learning algorithm, along with the hyperparameters you chose. Your report also describes the model architectures for the neural network you used in training the agent to solve this environment.

✓ **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.

Bonus Pts: 🌟 Your graphics 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!

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

Feedback: 🌟 You have provided the required concrete ideas for improving your Deep-RL system.