Continuous Control

REVIEW | CODE REVIEW | HISTORY

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

You have a really good start on this project! The hard part (getting the agent to actually learn this difficult task!) is complete. Your DDPG algorithm, agent, and model code all look very good and your results are also good. So, there is no need to re-run your code or produce any additional graphics.

However, you need to include just a few additional items in the final report to satisfy the rubric requirements. I know that after spending so much time getting the code to actually work it seems like you should be done, but Udacity places a lot of emphasis on the report and README as a way to encourage you to reflect on what you have accomplished, review some of the reasons various choices were made (in algorithm selection, hyperparmameter choices, NN architecture decisions, etc.). Everything but the NN architecture looks good, but please see my in-line comments for that NN element to determine what I think you need to add to meet the rubric requirements.

Training Code

✓ **The repository includes functional, well-documented, and organized code for training the agent.**

Feedback: 🌟 All the required files were submitted. The repository includes functional, organized, and documented code for training the agent for this environment. Implementing the DDPG algorithm was a good choice for this project as it has been found to work very well with continuous action and state spaces.

Some comments on your implementation:

- You have correctly implemented the Actor and Critic networks. Using two hidden layers with (128x128) units is similar to, but a bit smaller than the (400x300) architecture used by Lillicrap et al in the original DDPG implementation, but the reduced capacity has worked well for you in this environment.
- You have used a memory replay buffer to store and recall experience tuples. Your agent's step() method saves each new experience tuple and appropriately samples from this buffer when it determines that it's time to learn. The random sampling of experience from this buffer helps to prevent getting stuck in oscillatory trajectories and improves stability of the algorithm. Performing 10 steps of learning every 20 steps of processing is an excellent technique and I think is key to your agent's ability to quickly learn this complex environment.
- Your agent's learn() method correctly updates the target network using soft updates, controlled by Tau.
- You have added gradient clipping to your agent's learn() method. The authors of the original DDPG algorithm (Lillicrap et al) also employed this technique to improve stability of the algorithm. Nice work here!
- Your agent's act() method makes good use of the Ornstein-Uhlenbeck noise process and you have chosen reasonable values for Theta and Sigma to control this process, thus expanding on your agent's ability to explore the action space. Note: It is also possible to achieve good (even better?) results using simple Gaussian noise for this purpose. I see that you have chosen to scale the noise with an Epsilon hyper-parameter, and even reduce this parameter over time to implement a version of the epsilon-greedy algorithm within the DDPG framework. Nice work!
- Your actor neural network ensures all actions are within the [-1, +1] range by using the tanh() activation function. You are also explicitly clipping your data to this same range in the agent's act() function, which is reasonable considering the fact that your are adding OU noise to the actor-specified action and it's certainly possible that the resulting noise could cause the desired [-1, +1] range to be exceeded (on either end). So, good work here.
- You have not included batch normalization in your actor and critic NN models. This is something you might consider in the future, as I have found batch normalization to be very advantageous to efficiently learning policies and value functions.

✓ **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. The course developers were most comfortable with this framework, or
2. PyTorch is easier to understand and more straight-forward to use than other alternatives (compared to TensorFlow in particular), or
3. 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, and finally
4. 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 the required checkpoint files containing your actor and critic state dictionaries.

Bonus Pts: 🌟 You not only included the code to produce a checkpoint file, but also verified that the checkpoint is valid and could be used to restore your model to the defined state in the inference.ipynb notebook. Most excellent!! Note, however, that although loading just the local copy of the actor's weights is sufficient for viewing a trained agent, you should load both the local and target copies of both actor and critic if you wanted to continue training from the saved state dictionary.

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 RL work) or whenever you find an improved agent that beats the previous-best agent's score. If you choose to use this technique, don't forget to load both the local and target weights upon reading the saved values while also saving and restoring the state of the experience replay buffer or training will be negatively impacted.

README

✓ **The GitHub 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: 🌟 Thank you for including an excellent description of the project's environment details, including the success criteria.

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

Feedback: 🌟 You provided all the information needed for a new user to create an environment in which your code will run.

Bonus Pts: 🌟 Including the standard dependencies in a requirements.txt file is excellent practice as this enables users of your repository to install all dependent libraries with a single pip command. Very nice!

✓ **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 Continuous_Control.ipynb file that is the starting point for your implementation of this project.

Bonus Pts: 🌟 Your description in this section was very clear. You summarized not only the required "main" program, but also all the other important files in your project. Well Done!

Report

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

Feedback: 🌟 You included the required report, in PDF format. The report provides the required description of your implementation. Thank you for taking the time to create a professionally-formatted and well-organized report.

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

Feedback: 🚧 Overall, this is a good report. You listed key features of the actor-critic DDPG learning algorithm and the hyperparameters you chose.

However, the rubric also requires your report to describe the NN architectures used (both actor and critic, here). The graphic showing the interactions between actor and critic were excellent, but your neural network architecture description should also describe the number of layers, the number of units in each layer (along with reasons for your choices, particularly for the input and output layers), the types of activation functions used, and any normalization that you chose to apply. Can you please add this detail to your report?

✓ **A plot of rewards per episode is included to illustrate that either:**

- ***[version 1]* the agent receives an average reward (over 100 episodes) of at least +30, or**
- ***[version 2]* the agent is able to receive an average reward (over 100 episodes, and over all 20 agents) of at least +30.**

The submission reports the number of episodes needed to solve the environment.

Feedback: 🌟 Very nice. You included the required plot of rewards per episode demonstrating that your agent has successfully solved this environment, and your report also indicated the number of episodes needed to solve the environment. Your plot looks very professional and includes a line indicating the goal for this agent. Excellent work here!

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

Feedback: 🌟 You included some excellent ideas for improving your agent's ability to solve this environment. It was good to get experience using DDPG as that algorithm is the key early implementation of the actor-critic methodology, but any real-world problems will likely use one of the more advanced versions of the actor-critic process -- like PPO, A3C, or D4PG as you have recommended here.

RESUBMIT PROJECT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review