Meets Specifications

Congratulations on successfully completing this project! This was not an easy project to accomplish, and I can tell you put a lot of effort into it. Your implementation was very good and the obvious effort you put into the report was evident. Overall, this was an excellent project! It was a joy to review your work.

Insight: The same type of Reinforcement Learning agent you created in this project is used in many real-world applications. In particular, industrial control applications would benefit greatly from the continuous control aspects that DDPG is able to provide. This Medium blog post (https://medium.com/@BonsaiAI/industrial-control-systems-is-reinforcement-learning-the-answer-6380ab2eddeb) describes several potential applications of this technology, including:

- **Robotic Control Systems**: Very similar to the Reacher environment you just solved!
- **HVAC Control Systems:** Heating and Ventilation systems are everywhere, and improvements in these systems can lead to great energy savings.
- **Automotive Control Systems:** Deep learning can not only help bring driverless cars to the masses, but DDPG-like algorithms in particular could help to fine-tune air-to-fuel ratios, ignition control, etc. - improving fuel economy and reducing emissions.
- **Automatic Calibration:** calibrating industrial equipment can help to optimize performance, and DDPG networks have the potential to detect anomalies and issue alerts to avoid disasters.

In fact, Google recently reported that they have turned over complete control of their data center cooling systems to an AI system, reducing cooling costs by 40%. And, a Jul 2018 paper by Li et al addresses a similar issue for the National Super Computing Center (NSCC). Their algorithm uses an actor-critic model that is an off-line version of the same DDPG algorithm you used in this project, which reduced NSCC electricity costs by 15%. Here's a link to the paper if you're interested: https://arxiv.org/pdf/1709.05077.pdf

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.
- 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.
- Your agent's learn() method correctly updates the target network using soft updates, controlled by Tau.
- Your agent's act() method makes good use of the Ornstein-Uhlenbeck noise process and you have chosen a reasonable value of 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. It is also possible 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.
- 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 (although it appears you may have experimented with this). Batch normalization is something you might consider in the future, as I have found it 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. Finally, 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: 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, I was happy to see that you saved your actor and critic state_dicts every episode (although once every 100 or so episodes would probably suffice). Nice!

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 providing this type of information in the README for your project.

✓ **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, including the Unity ML-Agents library and the customized Reacher environment.

Feedback: 👍 You 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 Reacher environment.

✓ **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 the main files of your implementation. However, although it's pretty obvious, you should mention that continuous_control.ipynb is the starting point for your implementation of this project and is the file the user should load and run if he (or, she) wants to run your program. Please include this information in future projects of this Nanodegree.

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.md file, in markdown format. The report provides the required description of your implementation of the Continuous Control project.

✓ **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 a good report. You listed key features of the actor-critic DDPG learning algorithm, the actor and critic networks, and the hyperparameters you chose. Your report also describes the model architectures for the neural network you used in training your agent's actor and critic models to solve this environment. Your description of the key features of the DDPG algorithm was particularly good.

Bonus Pts: 💥 You not only included the pseudo code for the DDPG algorithm, but also described in your own words the key features of this algorithm, thus demonstrating your understanding of how this algorithm works. Great job, here!

✓ **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.

Suggestion: 💡 For future projects, you might consider adding two things to your plots:

1. A target/goal line showing the score that is required to be met for solving the environment.
2. The 100-episode average line. Where these two cross indicates the point the environment was solved.

These are just suggestions for future projects (which will also require you to produce similar plots of the rewards achieved) -- it's certainly not required by the rubric, but would provide some additional information to make the plots more complete.

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

Feedback: ❓ You provided the required list of potential improvements, so you meet the minimum requirements for this rubric item. However, I would like to have seen a short description of what these improvements might entail and how they differ from and improve on the basic DDPG you used in this project. Future projects in this course follow the same basic rubric structure and requirements (including this request to comment on potential improvements). Please consider adding this kind of detail in future projects of this course - if you do so, you will not only meet the rubric requirements but also learn some valuable insights in the process.