UDACITY

‹ Back to Deep Reinforcement Learning Nanodegree

Continuous Control

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on successfully completing this project! This was not an easy project to accomplish, and your success in solving this complex environment shows that you are grasping the key reinforcement learning concepts in this Nanodegree.

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.

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, 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 Udacians are well-rounded and capable of working in many different environments. I personally think it's probably a combination of all 3 of these, but I like the last one best :o)

Pro Tip: 💰 TensorFlow vs. PyTorch – This blog post (written by a fellow Udacity student!) gives a well-balanced comparison of these two popular deep learning frameworks: https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4 Some of the key points include:

- TensorFlow was developed by Google and uses a **static computation graph** that must be fully constructed before it can be used. This is similar to the approach traditional compiled programming languages use: you fully develop your code and then compile and run it.
- PyTorch was developed by Facebook and uses a **dynamic computation graph** that can be executed as each part of it is being defined. This approach is more in keeping with the python philosophy of running parts of your program in steps, as each ipython notebook cell or line of code is written.
- The TensorFlow **learning curve** is much higher than for PyTorch, but the user community is also much bigger and you may find it easier to find answers to questions with TensorFlow than PyTorch.
- PyTorch lacks the equivalent of the **TensorBoard visualization** utilities. I would guess something similar will appear on the PyTorch scene before too long (TorchBoard, perhaps?).

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

Feedback: 👍 Good job! You created and submitted the required checkpoint files containing your actor and critic state dictionaries.

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 -- even though your agent solved this particular environment in not many more than that) or whenever you find an improved agent that beats the previous-best agent's score.

README

The GitHub submission includes a `README.md` file in the root of the repository.

Feedback: ⚠️ You included a README.md file, but it appears to be a direct (or, nearly a direct) copy of the one provided by Udacity as instructions to you. The markdown file you include with your submission should be an edited version of this that describes your particular implementation. Your README file should be a set of instructions to a potential user or reviewer of your project as you implemented it, not instructions to a student to implement this project "from scratch". Much of the information provided to you by Udacity will be relevant, but only include those sections that are relevant and revise the text to document what your project actually does, not what Udacity is asking its students to do. And, add information that gives some extra insight into how your particular project is structured. Please tailor submissions for future projects to meet this guideline.

Pro Tip: 💰 It's industry-standard practice to include README.MD markdown files with any git repository. The README file is the first thing a potential user of your project will see and should impart a feeling of confidence that your code is worth reviewing and potentially using or expanding upon. The README file should communicate the goals of your project and provide basic instructions the user would need to install and run your project. I commend Udacity for requiring students to provide this top-level documentation for all DRLND projects as it will help to develop good habits in their graduates and I commend you for producing a great README that provides this information!

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

Feedback: 👍 You included the required 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.

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 how to load and execute the Continuous_Control.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 (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: 👍 You described 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 the agent actor and critic models to solve this environment.

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: 👍 You included the required plot of rewards per episode, and your report also indicated the number of episodes needed to solve this environment.

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

Feedback: 👍 You included several excellent ideas for improving your agent's ability to solve this environment.

DOWNLOAD PROJECT

RETURN TO PATH