UDACITY

< Back to Machine Learning Engineer Nanodegree

Teach a Quadcopter How to Fly

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear Udacian,

I have to say that the project is a fairly complicated one and you have done a great work. The task of coming up with the task and choosing the appropriate algorithm to implement the agent is in itself a challenge. Then, formulating a suitable reward function is a very innovative challenge and requires a lot of experimentation.
I have to say that it's not uncommon to see the agent not learning properly even after a lot of experimentation. Congratulations on completing the project successfully.

I would suggest you to go through Deep Reinforcement Learning for Self Driving Car by MIT. You'd get to know more about reinforcement learning algorithms in broader and real-world perspective and, more importantly, how to apply these techniques to real-world problems.

All the best for future endeavors. ✨

Define the Task, Define the Agent, and Train Your Agent!

The `agent.py` file contains a functional implementation of a reinforcement learning algorithm.

Awesome

- **DDPG algorithm** has been implemented. It is found to work very well in case of **continuous action space and state space.**
- **Implementation** of the **Actor and Critic networks** is correct.
- Good work using the **target networks** for Actor and Critic networks. The **original DDPG paper** suggests it as well.
- Good work using **soft updates** for the **target network.**
- Good choice to use **tau** to perform **soft update.**
- Correct usage of **replay memory** to store and recall **experience tuples.**
- The implementation is **easy to debug and easily extensible,** good work keeping it **highly modular.**

The `Quadcopter_Project.ipynb` notebook includes code to train the agent.

Awesome

Good work implementing the code to train the agent for the task **take off**. It's been very neatly presented in the notebook **Quadcopter_Project.ipynb**.

Plot the Rewards

A plot of rewards per episode is used to illustrate how the agent learns over time.

Awesome

Thanks for providing the plot of rewards for the task **take off** to illustrate how the agent learns over time.

Reflections

The submission describes the task and reward function, and the description lines up with the implementation in `task.py`. It is clear how the reward function can be used to guide the agent to accomplish the task.

Awesome

Good work implementing the task **take off**. The notebook contains detailed description of the task and the formulated reward function.
Good work experimenting with the reward function.

- Good decision choosing to **clip the overall reward** between -1 and 1 using **np.tanh** to avoid the problem of **exploding gradients.** A **good read** on **exploding gradients:** A Gentle Introduction to Exploding Gradients in Neural Networks

Suggestions

Some suggestions are given below that you should try for the reward function for the **take off** task.

- **z-axis** could be **prioritized** as the agent needs only vertical force to **take off.**
- **self.sim.v[2] (which is the vertical velocity)** can be used in the reward function so the agent will be rewarded to fly **vertically upwards.**
- When the agent crosses the **target height,** it could be given **bonus reward** so that the agent is prompted to learn the **optimal policy.**

```
if self.sim.pose[2] >= self.target_pose[2]:  # agent has crossed t
he target height
    reward += 10.0  # bonus reward
    done = True
```

- **Penalizing** the agent with the **negative rewards** can be considered when the agent crashes.

```
elif timestamp > self.max_duration:  # agent has run out of time
    reward -= 10.0  # extra penalty
    done = True
```

The submission provides a detailed description of the agent in `agent.py`.

Awesome

Great work providing the details of the implemented agent. The answer describes the learning agent by specifying the **details of the learning algorithm used, hyper-parameters and architectural information** of the deep learning model used.

- Good decision to choose **DDPG algorithm** for the **continuous state space** problem.
- Good work including **network architecture** in the notebook.

- **Hyperparameters** you have used seem to be fine.

Suggestions:

You should definitely try using **l2 regularization, dropout** and **batch normalization**.

To experiment more with the **architecture and hyperparameters,** you can check the following **resources:**

- Deep Deterministic Policy Gradients in TensorFlow
- Continuous control with Deep Reinforcement Learning

The submission discusses the rewards plot. Ideally, the plot shows that the agent has learned (with episode rewards that are gradually increasing). If not, the submission describes in detail various attempted settings (hyperparameters and architectures, etc) that were tested to teach the agent.

Awesome

- Discussion for the rewards is provided in the notebook.
- I have to say that the project is a **fairly complicated** one.
- It's **not uncommon to see the agent not learning gradually** even after a lot of experimentation.
- The **rewards plot** seems to be good.
- You should definitely try the suggestions for the reward function. They should be helpful.

Reinforcement learning algorithms are really **hard to make work.**
But it is **substantial to put efforts** in reinforcement learning as it is close to **Artificial General Intelligence.**
This article is a **must read:** Deep Reinforcement Learning Doesn't Work Yet

A brief overall summary of the experience working on the project is provided, with ideas for further improving the project.

Awesome

- Thanks for sharing the experience and providing the details.
- The project is indeed a hard one and requires the learning and combining of a lot of concepts.

- A good reward function is important to train the agent properly. Good job working on the reward function.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review