UDACITY

Back to Machine Learning Engineer Nanodegree

Teach a Quadcopter How to Fly

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on finishing this project! You have done lot of work in this project and you should be proud of yourself for having this worked out. I hope that you had fun in this project. Overall it is very good submission, describing the experience of building this project quite extensively. Well done and good luck with your future endeavors. :)

With so much of extensive description of the implementation, I'd suggest you to write a blog post on it like on Medium https://medium.com/ which is easier to write on. It will not take you much longer as you can just copy paste your write-up as it is. :D This is a great way to contribute to the AI community by sharing the knowledge.

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

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

The environment is still not that large as state_size and action_size is bounded. But you will definitely going to have issues when you are working on the image based simulator where the states are pixel based. In those cases, deep learning based RL agents are going to be helpful as then you can use the function approximation technique to approximate the continuous state space when the discretization may fail to work.

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

These python visualizations are great way to visualize the movement of the Quadcopter in the simulated environment.

Plot the Rewards

A plot of rewards per episode is used 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.

It is well documented answer. You have described your intuition behind designing of the reward function.

Getting a reinforcement learning agent to learn what you actually want it to learn can be hard, and very time consuming. It is quite hard to address all aspects of the behavior desired. But you have managed to teach to the Quadcopter to performs actions in the x, y, z axes with stable learning. That's really awesome!

Nice job in clipping the rewards to the standard range of -10.0 to +10.0 to avoid instability in training due to exploding gradients.

Good work in including the `self.sim.v[2]` (which is the vertical velocity) into the reward function. So the agent will be penalized if not flying vertical upwards.

Check out these talks to watch some more tips for designing the reward function. https://www.youtube.com/watch?v=GOsUHlr4DKE&feature=youtu.be and https://www.youtube.com/watch?v=0R3PnJEisqk

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

Another well documented answer in which you have described the intuition behind tuning of the hyper-parameters used. Using neural networks would have increased the number of hyper-parameters, and it would have been more work to tune those.

You should've some discussion on the the value-based methods specifically on expected sarsa algorithm. Check out some of these details given in the classroom lesson videos. Here's another udacity course lesson video of the overview of Q-table method is also given out where Q-learning is explained though;

https://www.youtube.com/watch?time_continue=94&v=WQgdnzzhSLM

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.

I realized I should give the quadcopter a head start by setting the z-axis value to some initial value rather than 0.

I believe it is because if the value of z-axis goes below 0 then `done=True` will be triggered, resetting the simulator. Check out the codes given in the physics_sim as follows;

```
env_bounds = 300.0  # 300 m / 300 m / 300 m
self.lower_bounds = np.array([-env_bounds / 2, -env_bounds / 2, 0])
self.upper_bounds = np.array([env_bounds / 2, env_bounds / 2, env_bounds])
```

```
    new_positions = []
    for ii in range(3):
        if position[ii] <= self.lower_bounds[ii]:
            new_positions.append(self.lower_bounds[ii])
            self.done = True
```

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

It is an excellent submission. I've actually learned a lot from your submission in using the Expected Sarsa algorithm. This is the only submission review which managed to get the splendid results without using neural networks.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review