Build a Balancing Bot with OpenAI Gym, Pt I: Setting up

When I started working with OpenAI Gym, one of the things I was looking forward to was writing my own environment and have one of the available algorithms derive a model for it. Creating an environment is not obvious, so I had to go through some experimentation till I got it right. I decided to write a tutorial series for those that would like to create their own environments in the future, by taking an example of a task that is both fun and simple, but also extendable: A balancing bot.

This post assumes that you have some understanding of Reinforcement Learning principles. Specific understanding of Deep-RL algorithms is not necessary, although it wouldn't hurt. Also, it is good if you have some familiarity with Python and tools such as pip, Miniconda and setuptools.

This is part I of the series, covering setting up the OpenAI Gym environment. The second part can be found here.

Introduction

We will be using OpenAI Gym to implement the Balancing Bot task. In addition we will be using Baselines and pyBullet. Baselines is an OpenAI project that includes implementations of several state of the art reinforcement learning algorithms. Using Baselines will allow us to focus on creating the environment and not worry about training the agent. In addition, since our environment is defined by physics, we will be using pyBullet to perform the necessary computations and visualize experiment progress. pyBullet is easy to setup and use and includes a viewport module that has lots of visualization and interaction features built in.

At the end of this two-part tutorial we will be able to train a balancing bot controller with just a few lines of code. As an example, below is the code that we'll use:

```
import gym
from baselines import deepq
import balance_bot

def callback(lcl, glb):
    # stop training if reward exceeds 199
    is_solved = lcl['t'] > 100 and sum(lcl['episode_rewards'][-101:-1])
    return is_solved

def main():
    # create the environment
    env = gym.make("balancebot-v0") # <-- this we need to create
    # create the learning agent
    model = deepq.models.mlp([16, 16])

    # train the agent on the environment
    act = deepq.learn(
        env, q_func=model, lr=1e-3,
        max_timesteps=200000, buffer_size=50000, exploration_fraction=0
        exploration_final_eps=0.02, print_freq=10, callback=callback
    )

    # save trained model
    act.save("balance.pkl")

if __name__ == '__main__':
    main()
```

If the above code doesn't make much sense right now, hang on. It will become clear at the end of this post.

Setting up a Python Environment

It goes without saying that the first step is to make our project dir and switch to it:

```
mkdir balancebot-project
cd balancebot-project
```

I also recommend setting up a Miniconda environment prior to working with projects such as this one. Once you have Miniconda installed, you can create a new Python environment and switch to it as follows:

```
conda create -n balance_bot python
source activate balance_bot
```

Once inside the Miniconda environment you created you can install packages using `pip` as usual. You can go ahead and install some basic packages that will be used in this tutorial:

```
pip install gym baselines pybullet
```

Structuring the balancing bot project

Next step is the creation of the Gym environment structure. OpenAI have included an informative guide to creating a folder structure, which we will be following in this tutorial. Our folder structure will look like below:

Read also: Microcontroller Servo Control and Compliant Motion

```
balance-bot/
  README.md
  setup.py
  balance_bot/
    __init__.py
    envs/
      __init__.py
      balancebot_env.py
```

Go ahead and create this structure in your project folder. Once you are finished, it's time to start creating the individual files.

`balance-bot/setup.py` includes the following:

```
from setuptools import setup

setup(name='balance_bot',
      version='0.0.1',
      install_requires=['gym',
                        'pybullet']
)
```

Mostly standard `setuptools` stuff. Note the `baselines` package is missing from the requirements. This is because the environment itself is independent from the agents. Thus one may choose to use a different agent to solve the environment task.

Next is `balance-bot/balance_bot/__init__.py`:

```
from gym.envs.registration import register

register(
    id='balancebot-v0',
    entry_point='balance_bot.envs:BalancebotEnv',
)
```

The use of `register()` needs clarification. In our first code block above, we used the `gym.make()` function to instantiate our environment, and later on pass it to the training function. `gym.make()` accepts an id (a string) and looks for environments registered with OpenAI Gym that have this id. If it finds one, it performs instantiation and returns a handle to the environment. All further interaction with the environment is done through that handle. Without going in too many details, the `register()` function is what tells Gym where to find the environment class and what name it should have.

The `entry_point` parameter specifies the subclass of `Env` that describes our environment. Implementation of the class will be the subject of the second part of this tutorial series. The `entry_point` parameter value has a specific format: `{base module name}.envs:{Env subclass name}`. Even though `BalancebotEnv` resides within another module, Gym doesn't need to know about it because the class is imported at `balance_bot/envs/__init__.py`.

Finally, Gym expects the id of our environment to follow the pattern `{name}-v{version}` so we'll just stick to it.

Next comes `balance-bot/balance_bot/envs/__init__.py`:

```
from balance_bot.envs.balancebot_env import BalancebotEnv
```

As mentioned earlier, this file simply imports `BalancebotEnv` from the corresponding Python module, so that it is available to the `register()`

function above.

Finally, the `balance-bot/balance_bot/envs/balancebot_env.py` file is the one that will contain the actual environment class. Below is a stripped-down version of the file contents. We will be fleshing these out in the second part of this tutorial series:

```
import gym
from gym import error, spaces, utils
from gym.utils import seeding

class BalancebotEnv(gym.Env):
  metadata = {'render.modes': ['human']}

  def __init__(self):
    pass

  def _step(self, action):
    pass

  def _reset(self):
    pass

  def _render(self, mode='human', close=False):
    pass
```

Installing with pip

At this point we are done with preliminary environment setting up, and we can start fleshing out our `Env` subclass. Just before doing that let's try and do a pip install in our new environment. From the root of your project:

```
cd balance-bot
pip install -e .
```

This should install the environment in editable mode, which means that changes you make to your files inside balance-bot will affect the installed package as well.

Conclusion

This was the first in a tutorial series on creating a custom environment for reinforcement learning using OpenAI Gym, Baselines and pyBullet. We discussed structuring of the project environment and files. Our environment is not yet functional but take a look at the video below for a glimpse on what we'll be building in the end of the series:

Go ahead and visit the second part of this tutorial.

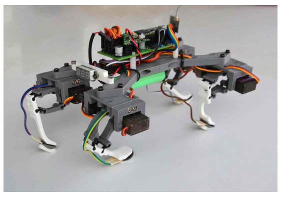
The code for this tutorial is now available on Github.

Have any questions or comments? Ask and share in the comments below.

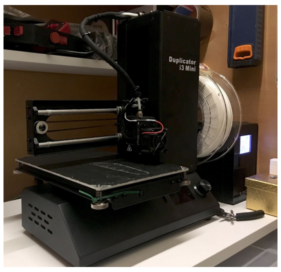
Email address

Sign me up!

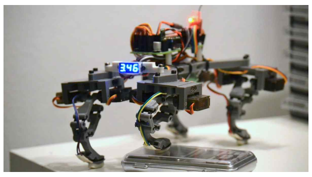
You may also like



Quadruped Robot Part III: Movement, Balance and Testing



Wanhao i3 Mini: Essential Mods & Printing More Materials



Quadruped Robot Part II: Assembly, Electronics

2 replies on " Build a Balancing Bot with OpenAI Gym, Pt I: Setting up "

Rokas
April 9, 2018 at 1:18 pm

Wow! If only I had found this earlier, was a struggle trying to understand everything and you've made this so clear. Trying to create an environment of my own and currently stumped on the environment file. Looking forward to going over your second tutorial now.

REPLY

yconst
April 9, 2018 at 7:29 pm

Hi, glad you found it useful. Setting up an environment is quite the learning curve at first but then you get the hang of it. I'm wondering is the OpenAI folks would care to simplify stuff a bit.

REPLY

Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam.Learn how your comment data is processed

Share this: 4

Next Post
Build a Balancing Bot with OpenAI Gym, Pt II: The Robot and Environmen

Previous Post
A new identity!

backyard robotics

About BYR

BYR on Facebook

BYR on Twitter

BYR on Youtube

BYR on Thingiverse

BYR on Pinterest

Privacy Policy

Subscribe to our email:

Email Address

Sign me up!