UDACITY

Logout

Return to Classroom

DISCUSS ON STUDENT HUB

Collaboration and Competition

REVIEW | CODE REVIEW 2 | HISTORY

Meets Specifications

Congratulations on passing your final project 🎉🎁

Assuming you passed the first two projects, THIS IS GRADUAT

Rate this review

START

You made a great submission that solved this quite unstable environment

episodes. It's common for this environment to give variant results every run, but your model is stable. Give yourself proper credit.

To answer your student note, Batch Normalization was introduced in a 2015 paper to address the vanishing/exploding gradients problems. This operation simply zero-centers and normalizes each input, then scales and shifts the result using two new parameter vectors per layer: one for scaling, one for shifting. Basically, Batch Normalization lets the model learn the optimal scale and mean of each of the layer's inputs. Thus speeding the learning process.

THIS IS A GREAT QUESTION. I SUGGEST YOU READ THE PAPER CAREFULLY AS THIS IS A VERY COMMON AND BENEFICIAL TECHNIQUE.

I hope this nanodegree gives you a good kick towards your dreams. These resources will give you a different point of view for Deep Reinforcement Learning:

- MIT 6.S094: Deep Reinforcement Learning video.
- Deep Reinforcement Learning by David Silver.

Training Code

The repository includes functional, well-documented, and organized code for training the agent.

Your code functions correctly. Here's the result I got w[...] I ran it:

```
Episode 100     Average Score: 0.010    Moving Avg: 0.010
Episode 200     Average Score: 0.011    Moving Avg: 0.011
Episode 300     Average Score: 0.014    Moving Avg: 0.014
Episode 400     Average Score: 0.020    Moving Avg: 0.020
Episode 500     Average Score: 0.039    Moving Avg: 0.029
Episode 600     Average Score: 0.048    Moving Avg: 0.038
Episode 700     Average Score: 0.064    Moving Avg: 0.064
Episode 800     Average Score: 0.108    Moving Avg: 0.098
Episode 900     Average Score: 0.191    Moving Avg: 0.191
Episode 1000    Average Score: 0.318    Moving Avg: 0.308
Episode 1028    Average Score: 0.326    Moving Avg: 0.326
```

PS: I didn't have enough time to wait for the training to finish.

As you see, the resuts are similar to yours:

```
Episode 100     Average Score: 0.000    Moving Avg: 0.000
Episode 200     Average Score: 0.001    Moving Avg: 0.001
Episode 300     Average Score: 0.003    Moving Avg: 0.003
Episode 400     Average Score: 0.002    Moving Avg: 0.002
Episode 500     Average Score: 0.003    Moving Avg: 0.003
Episode 600     Average Score: 0.016    Moving Avg: 0.006
Episode 700     Average Score: 0.013    Moving Avg: 0.013
Episode 800     Average Score: 0.029    Moving Avg: 0.019
Episode 900     Average Score: 0.023    Moving Avg: 0.023
Episode 1000    Average Score: 0.034    Moving Avg: 0.034
Episode 1100    Average Score: 0.076    Moving Avg: 0.066
Episode 1200    Average Score: 0.119    Moving Avg: 0.109
Episode 1300    Average Score: 0.278    Moving Avg: 0.268
Episode 1400    Average Score: 0.439    Moving Avg: 0.429
Episode 1467    Average Score: 0.500    Moving Avg: 0.500
Environment solved in 1467 episodes!    Average Score: 0.50
```

The code is written in PyTorch and Python 3.

Rate this review

START

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

The model weights of the successful agent are saved in `checkpoint_final.pth`.

README

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

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

The `Introduction` of the README **correctly** describes all the project environment details.

The README has instructions for installing dependencies or downloading needed files.

The `Getting Started` section of the README gives correct instructions for:

✅Downloading the environment.
✅Installing dependencies.

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.

The `Instructions` section of the README **correctly** describes how to run your code.

Rate this review

START

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.

The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.

All requirements are explained.

✅Hyperparameters in the `Implementation` section.
✅Learning Algorithm in the `MADDPG - Agent` section.
⬜ Model Architecture in the `Implementation` section. Please mention using `tanh` on one of the layers.

YOU ALREADY PASSED THIS POINT. YET, IT'S MISSING THIS ONE DETAIL. PART OF OUR JOB IS TO GUIDE YOU IN BUILDING A GREAT PORTFOLIO. I HOPE THIS HELPS.

A plot of rewards per episode is included to illustrate that the agents get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

The submission reports the number of episodes needed to solve the environment.

Great job 👏Your agents train in a reasonable number of episodes as demonstrated in the `Implementation` section.

✅The Plot for the trained agents is included.
✅The Report mentions the number of episodes needed to solve the environment.

Rate this review

START

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

- Yes, you can almost always fine-tune the hyperparameters and the model. Grid Search is a very good suggestion.
- Have fun with the football version using shared communications.

You can also try using different algorithms. Here are resources to help:

- A3C
- PPO and a PPO implementation.
- D4PG

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›

Rate this review

START

RETURN TO PATH

Rate this review

START