UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

Collaboration and Competition

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear Udacian,

Great job getting acquainted with the Multi Agent Deep Deterministic Policy Gradients algorithm and successfully implementing it to solve the multi-agent Tennis environment. The implementation is pretty good and the environment is solved in just 731 episodes. The architectures used for the actor and critic network are decent in size with two hidden layers each. Good work using `Relu` activation. The report is extremely informative and covers all the important aspects of the implementation. 😄

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. ✨

Training Code

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

Awesome

- Good work implementing MADDPG algorithm to solve the multi-agent Tennis environment.
- Implementation of the Actor and Critic networks is correct.
- Good work using the target networks for Actor and Critic networks
- 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 code is written in PyTorch and Python 3.

Awesome

The code is written in PyTorch and Python 3.

Lately, PyTorch and TensorFlow happen to be most extensively used frameworks in deep learning. It would be good to get some insight by comparing them, please see the following resources:

- Sebastian Thrun on TensorFlow
- PyTorch vs TensorFlow—spotting the difference
- Tensorflow or PyTorch : The Force is Strong with which One?

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

Awesome

- Saved model weights of the successful agent have been submitted.
- The following files are present in the submission.
  - `checkpoint_actor_0.pth`
  - `checkpoint_actor_1.pth`
  - `checkpoint_critic_0.pth`
  - `checkpoint_critic_1.pth`

README

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

Awesome

- Great work documenting the project details and submitting the README file.

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

Awesome

- Great work providing the details of the project environment in the `Task Goals and Details` section of the README.
- The section describes the project environment by specifying the state space, action space, and the desired results.

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

Awesome

- Great work providing all the necessary instructions in the `Environment Set Up` section.
- `Step 1 Clone the DRLND Repository` section to install the dependencies.
- `Step 2: Download the Unity Environment` section to download the 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.

Awesome

- Great work providing necessary instructions to run the code in the `Instructions` section.
- All the cells in `Tennis.ipynb` file should be executed to train the agent.

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.

Awesome

- Report for the project with all the details of the implementation has been provided in the submission.

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

Awesome

Great work providing the details of the implemented agent. Details of the learning algorithm used, hyper-parameters, and architectural information of the deep learning model have been provided.

- Good decision to choose MADDPG algorithm for the continuous action space problem.
- Good work including model architecture in the report.

The **Actor Neural Networks** use the following architecture.

```
Input nodes (8)
  -> Fully Connected Layer (in: 8*2 nodes,out: 256 + Relu activation)
      -> Fully Connected Layer (in: 256 nodes,out: 128 Relu + activation)
         -> Ouput nodes (2 nodes, tanh activation)
```

The **Critic Neural Networks** use the following architecture :

```
Input nodes (8)
  -> Fully Connected Layer (in: 8*2, out: 256 nodes + Relu)
      -> Fully Connected Layer (in: 256+ 8*2, out: 128 nodes + Relu)
         -> Ouput nodes (1 node )
```

- Hyperparameters you have used seem to be good.

  - `BATCH_SIZE = 128`: neural network mini-batch size
  - `LR_ACTOR = 1e-3`: the actor neural network learning rate use in gradient descent
  - `LR_CRITIC = 1e-3`: the critic neural network learning rate use in gradient descent
  - `WEIGHT_DECAY = 0.0`: L2 weight decay
  - `LEARN_EVERY` = 5: steps to wait until learning. Strategy used in the last ddpg multi-agent environment to stabilize training
  - `LEARN_NUM` = 5: number of learning passes
  - `GAMMA = 0.99`: the discount factor used in the discounted sum of rewards
  - `TAU = 1e-3`: the τ parameter used to soft update of target parameters
  - `OU_SIGMA` = 0.2: Ornstein-Uhlenbeck dispersion parameter
  - `OU_THETA` = 0.12: Ornstein-Uhlenbeck location parameter
  - `EPS_START` = 5.5: initial value for epsilon in noise decay process in Agent.act()
  - `EPS_EP_END` = 250: episode to end the noise decay process
  - `EPS_FINAL` = 0: final value for epsilon after decay

Suggestions

You should definitely try using batch normalization.

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

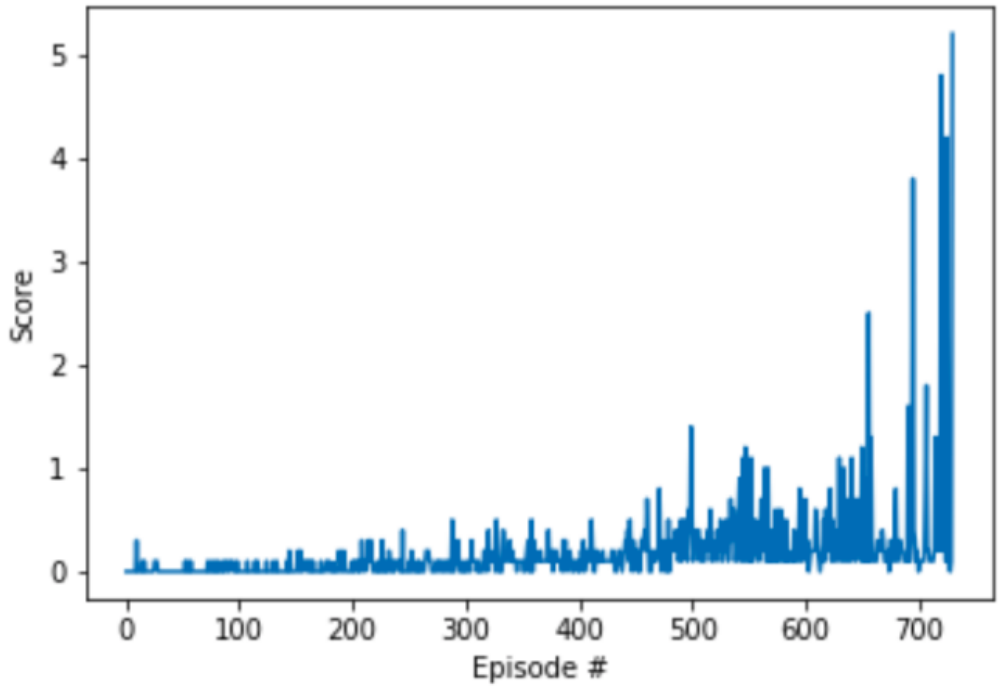
- Deep Deterministic Policy Gradients in TensorFlow
- Continuous control with Deep Reinforcement Learning

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.

Awesome

- Discussion for the rewards is provided in the report.
- The rewards plot seems to be good and average score of +0.52 is achieved in 731 episodes.



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.

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

Awesome

- Thanks for providing the following concrete ideas for improvement.
    - Trying to make the algorithm more stable
    - Trying batch normalization
    - Trying adaptive policies
    - Trying Minimax MADDPG

Suggestions

- Please check the following resources that are specifically for multi-agent environments:
    - Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments
    - Simple Reinforcement Learning: Asynchronous Actor-Critic Agents (A3C)
    - RL — Proximal Policy Optimization (PPO)
    - Mean Field Multi-Agent Reinforcement Learning

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START