Report: Project #3 Collaboration and Competition

5/11/2019

Soyoung Park

Jupyter notebook can be viewed here:
https://nbviewer.jupyter.org/github/parksoy/Soyoung_Udacity_ND_DeepReinforcementLearning/blob/master/p3_collab-compet/Tennis.ipynb

This project is built with the Tennis environment where two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play. The observation space consists of total 24 states, i.e., 8 variables(States) corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping. The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically, we add up the rewards that each agent received (without discounting), to get a score for each agent after each episode. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores. This yields a single **score** for each episode. The environment is considered solved, when the average (over 100 episodes) of those **scores** is at least +0.5.

1. Learning Algorithm

- **Learning algorithm**

Multi Agent Deep Deterministic Policy Gradient algorithm (MADDPG paper) was used and its template code from the previous project #2 Continuous Control (Single agent based DDPG code) was modified to adapted to run with Tennis environment. To adapt it to train multiple agents, **each agent receives its own, local observation**. Thus, both agents are simultaneously trained through self-play. Each agent used the same actor neural network to select actions, and the experience was added to a shared replay buffer.

The various hyperparameters and settings were tweaked and attempted.

- **The chosen hyperparameters**
    - max_steps
      : In one episode, how many maximum number of steps to be explored to move onto the next step
    - num_episodes
      : How many times to iterate of training of max_steps

- batch_size
  : How many Replay buffer memory to be sequentially to learn through the neural networks in a batch processing manner

- buffer_size
  : Replay buffer size that consists of namedtuple("Experience", field_names=["state", "action", "reward", "next_state", "done"])

- actor_learn_rate
  : How fast Actor neural network learns

- critic_learn_rate
  : How critic Actor neural network learns

- update_every
  : Update occurs only when certain number of steps is reached to learn more stably.

- num_updates
  : For every certain step that is defined by update_every, only limited number of update is made.

- tau
  : Control soft update. Only portion of local network is updated, while the majority of update comes from the stable target.

- gamma
  : Discount factor. Future reward is less valuable than immediate reward.

- noise_sigma
  : Amount of noise user wants to introduce.

- layer_sizes
  : Unit size of the first and second hidden layer

- noise_factor_decay
  : Controls how fast the noise decays over episodes as agents learn the process.

- **Model architectures for neural networks**

Each agent has actor network and critic network.

Actor network

```
bn0 = nn.BatchNorm1d(state_size)
fc1 = nn.Linear(state_size, fc1_units)
bn1 = nn.BatchNorm1d(fc1_units)
fc2 = nn.Linear(fc1_units, fc2_units)
bn2 = nn.BatchNorm1d(fc2_units)
fc3 = nn.Linear(fc2_units, action_size)
```

Critic network

```
bn0 = nn.BatchNorm1d(state_size)
fcs1 = nn.Linear(state_size, fcs1_units)
fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
fc3 = nn.Linear(fc2_units, 1)
```

where `state_size=24, action_size=2, fc1_units=512,fc2_units=512`

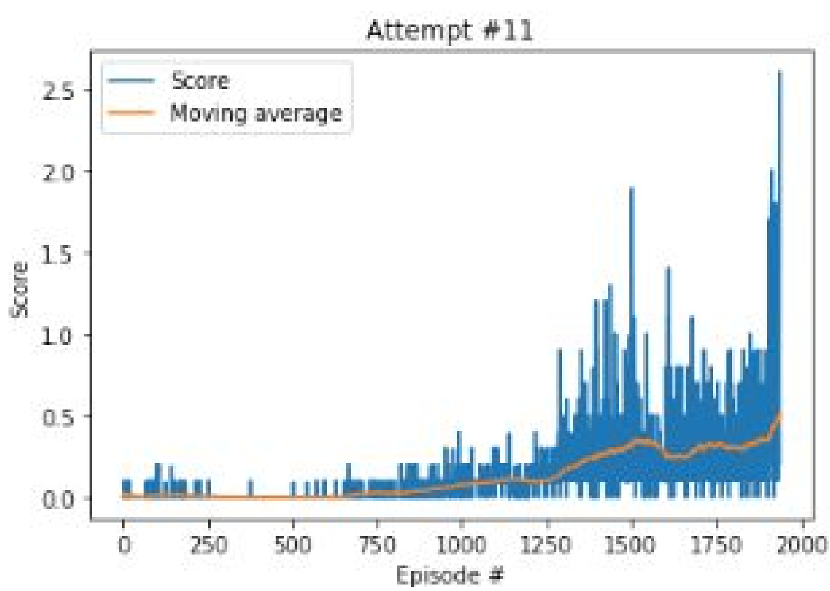
2. Plot of Rewards: Optimize the hyperparameters

Many attempts were made with different hyperparameters, different tricks to sample/learn to solve the environment as shown in the following table.

Attempt	batch_size	actor_learn_rate	critic_learn_rate	update_every	num_updates	noise_factor_decay	layer_sizes	buffer size	batch norm	score@N episode	what's changed
1	1024	1.00E-05	1.00E-04	1	1	NA	[128,128,128]	1.00E+06	yes	0 @1000	POR
2	1024	1.00E-03	1.00E-03	20	10	NA	[128,128]	3.00E+06	yes	0.025 at 2000	make light, faster learning
3	1024	1.00E-03	1.00E-03	20	10	1.00E-06	[128,128]	3.00E+06	yes	0 at 3000	POR
4	1024	1.00E-04	1.00E-03	20	10	1.00E-06	[128,128]	3.00E+06	yes	0 at 1000	MADDPG fix
5	1024	1.00E-04	1.00E-03	20	10	1.00E-06	[128,128]	3.00E+06	yes	0.032 at 2000	3000 episodes at cpu
6	1024	1.00E-03	1.00E-03	20	10	1.00E-06	[512,512]	3.00E+06	no	0 at 2000	move onto gpu
7	256	1.00E-03	1.00E-03	10	10	1.00E-06	[512,512]	3.00E+06	no	0 at 2000	update more frequently
8	256	1.00E-03	1.00E-03	5	5	1.00E-06	[512,512]	1.00E+06	no	0 at 1000	update even more frequently
9	1024	1.00E-04	1.00E-03	20	10	1.00E-06	[128,128]	3.00E+06	no	0 at 3000	gpu-visit of attempt 4
10	1024	1.00E-04	1.00E-03	20	10	1.00E-06	[512,512]	3.00E+06	yes	0.5 at 2400	gpu-visit of attempt 4 with bn
11	512	1.00E-04	1.00E-03	5	5	1.00E-06	[256,256]	3.00E+06	yes	0.5 at 1938	make nn smaller to make training faster, learn more often

For example, Attempt 5 shows actor net loss of each agent decreases successfully as shown in the following tensorboard screen capture, but the rewards started to very slowly accumulate after 2400 episodes. Critic net loss shows stable over the whole training.

After many different attempts, the final attempt #11 with more frequent learning and smaller network as shown in the following figure demonstrated that 1938 episodes were needed to solve the environment (i.e. to receive an average reward 0.5 points) over 2 agents. Note that due to the multi-agent nature of this problem, instability is seen during training. The blue line shows the maximum score over both agents, for each episode, and the orange line shows the average score (after taking the maximum over both agents) over the next 100 episodes.



3. Ideas for Future Work

DDPG is very sensitive to hyperparameters. It would be critical to systematically understand the hyperparameter space by ranking out the sensitivity of each parameter, then tweaking the parameters within those high ranked parameters only to reduce number of trials, also to achieve the best learning faster.

Additionally, the various deep RL algorithms on continuous control tasks, such as, REINFORCE, TNPG, RWR, REPS, TRPO, CEM, CMA-ES and DDPG, may be pursuited based on suggestion in the review paper to improve the agent's performance further.