Bonus Lab: Deep Deterministic Policy Gradient

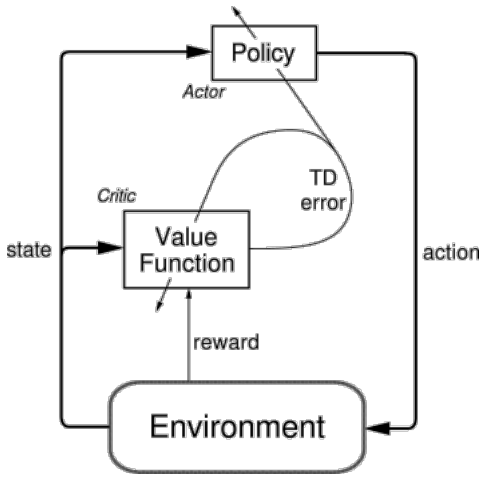
Lab Objective:

In this lab, the only thing you need to do is to **run a baseline solution** from AI Driving Olympics (AIDO). During the process, you will learn to apply deep reinforcement learning in a continuous action space for self-driving car.

The method we would use in the lab is "deep deterministic policy gradient" (DDPG) which is an off-policy, soft target update, method that involves actor and critic concept. We have used it for a simpler scene (pendulum) last week and now we extend it to a more complex task (self-driving car). The actor is the policy that generate action in each time stamp whereas the critic try to predict a good Q-value function (Q(s, a)). You will apply the DDPG in gym-duckietown, a virtual environment, and try to make your "Duckiebot" perform well in lane following task.

Lab Description:

- Learn how to combine deep learning and policy gradient.
  - Network design of actor/critic network
  - Cooperation between actor and critic network.
- Understand the "off-policy" and "soft target update" in DDPG
  - Replay Buffer
  - Behavior network and target network
- Implement DDPG to solve a lane following case
  - Try to use a different network for the actor/critic network. (ex: ResNet)
  - Try giving different reward.
- Apply your DDPG policy to LF task



Requirements:

- Understand how actor-critic algorithm, off-policy, soft target update work.
- Run your DDPG in gym-duckietown and try to do lane following.

AI Driving Olympics and Duckietown:

AI Driving Olympics (AIDO) is a competition focused on AI self-driving robots, proposed and held by Duckietown foundation. The purpose of the competition is to probe the frontier of the state of art in machine learning.

The first AIDO event is held in NeurIPS, a premier machine learning conference, last December. The second AIDO competition will be held in International Conference on Robotics and Automation (ICRA) 2019 which is the first AIDO competition with more sophisticated tasks and interesting components.
You are highly encourage to attend AIDO II even as your final project. You can join the competition simply by reading the guide lines of submission and turn in your cool algorithms. There's no necessary needs to get a real Duckiebot, but if you need one, you are welcome to come to TAs for help.

There are tons of ways to solve Lane following task of course. In AIDO I, there're some submission templates and baselines provided by Duckietown foundation. One of the baselines is the DDPG algorithm written in Pytorch. Your goal in this lab is to optimize the algorithm with possible ways as we mentioned above.

What is Duckietown and gym-duckietown?

Duckietown is a research and education platform that now spreads world-wide, being an initiative to realize a new vision in AI/Robotics, enabling researchers and educators of AI/Robotics perform their magic upon these tiny Duckies.

A Duckietown platform has two part: a Duckietown and Duckiebots.

Gym-Duckietown is a virtual environment that has simulated physical laws, roads and Duckiebots for your to trained and test your DDPG algorithm.

Environment:

Game Environment - gym-duckietown:

- Introduction: Your goal is try to make your Duckiebot stay in the middle of the right lane and travel as far as possible.
- Check out file: src/gym-duckietown/gym_duckietown/ simulation.py and duckietown_rl/wrappers.py
- State:
  - RGB-image: 160*120*3 (0~255) (after wrapping)
- Actions:
  - Left and right wheels torques: [-1, 1]
- Reward:
  - If in invalid step: -10 (after wrapping)
  - If reach maximum step: 0+4 (after wrapping)

- Others: 1*robot_speed*dot_direction + -10*distance_to_lane_center + 40*collision_penalty
  - If reward > 0: reward += 10
  - else: reward += 4
- You can customize DtWrapper to customize your reward.

Implement Detail:

Network Architecture (the one you will find in the original code):
- Actor
  - Convolution (3, 32, 8, stride=2) Leaky ReLU
  - 2D Batch Normalization
  - Convolution (32, 32, 4, stride=2) Leaky ReLU
  - 2D Batch Normalization
  - Convolution (32, 32, 4, stride=2) Leaky ReLU
  - 2D Batch Normalization
  - Convolution (32, 32, 4, stride=1) Leaky ReLU
  - 2D Batch Normalization
  - Flatten
  - Linear combination (flat_size, 512) Leaky ReLU
  - Linear combination (512, 2)

- Critic
  - Convolution (3, 32, 8, stride=2) Leaky ReLU
  - 2D Batch Normalization
  - Convolution (32, 32, 4, stride=2) Leaky ReLU
  - 2D Batch Normalization
  - Convolution (32, 32, 4, stride=2) Leaky ReLU
  - 2D Batch Normalization
  - Convolution (32, 32, 4, stride=1) Leaky ReLU
  - 2D Batch Normalization
  - Flatten
  - Linear combination (flat_size, 256) Leaky ReLU
  - Linear combination (256+action_dimension, 2) Leaky ReLU
  - Linear combination (128, 1)

Training Arguments:
- Optimizer: Adam
- Learning rate Actor: 0.0001
- Learning rate Critic: 0.001
- Tau: 0.005
- Batch size (for both Actor and Critic): 32
- Replay Buffer size: 10000
- Discount factor Gamma: 0.99
- Total training time step: 1000000
- Evaluate Frequency: every 5000 time step (You can count how many episode it will run)

Methodology:

Algorithm - DDPG algorithm that applied in this baseline

Slightly different from the one we have in DDPG lab.
In the example of pendulum, we train our network "every step"
In the example of self-driving car, we first execute for some amount of time **t**, then train of network for **t** iteration.
The total amount of training are the same, but what is the different?

Randomly initialize critic network $Q(s,a\,|\,\theta^Q)$ and actor network $\mu(s\,|\,\theta^\mu)$ with weight θ^Q, θ^μ
Initialize the target network (both actor critic) Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize the replay buffer R

while timestamp < max_timestamp **do**

  Initialize a random process N for actor to process
  Receive initial observation state S_1

  Iteration = 0
  for t=1, evaluate_frequency **do**
    if timestamp < start_timestamp
      Select action randomly
    else
      Select action according to the current policy and exploration noice $a_t = \mu(s_t\,|\,\theta^\mu) + N_t$
    end if
    Execute action a_t and observation reward r_t and new observation state s_{t+1}
    Store Transition (s_t, a_t, r_t, s_{t+1}) into R
    Iteration+=1
  end for

  for it=1, Iteration do
  Sample random mini batch of N transition from R
  Set $y_i = r_i + \gamma Q'(S_{t+1}, \mu'(s_{t+1}\,|\,\theta^{\mu'})\,|\,\theta^{Q'})$
  Update critic network by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i\,|\,\theta^Q))^2$
  Update actor network policy using the sample gradient:

$$\nabla_{\theta^\mu}\,\mu\,|\,s_i \approx \frac{1}{N}\sum_i \nabla_a\, Q(s,a\,|\,\theta^Q)\,|_{s=s_i,a=\mu(s_i)}\,\nabla_{\theta^\mu}\,\mu(s\,|\,\theta^\mu)\,|\,s_i$$

  Update the target network softly:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Tutorial:

Requirement: **Python3.7** (You might need a update), pip3, github
Machine: You may NOT use ssh to a remote machine. You could use either local machines or remote desktop app (e.x. teamviewer).

```
Laptop $ git clone git://github.com/duckietown/challenge-aido_LF-baseline-RL-sim-pytorch.git
Laptop $ cd challenge-aido1_LF1-baseline-RL-sim-pytorch

Laptop $ sudo pip3 install -e .
Laptop $ pip3 install -e git://github.com/duckietown/gym-duckietown.git#egg=gym-duckietown
```

Start training

```
Laptop $ cd duckietown_rl
Laptop $ python3 -m scripts.train_cnn.py --seed 123
```

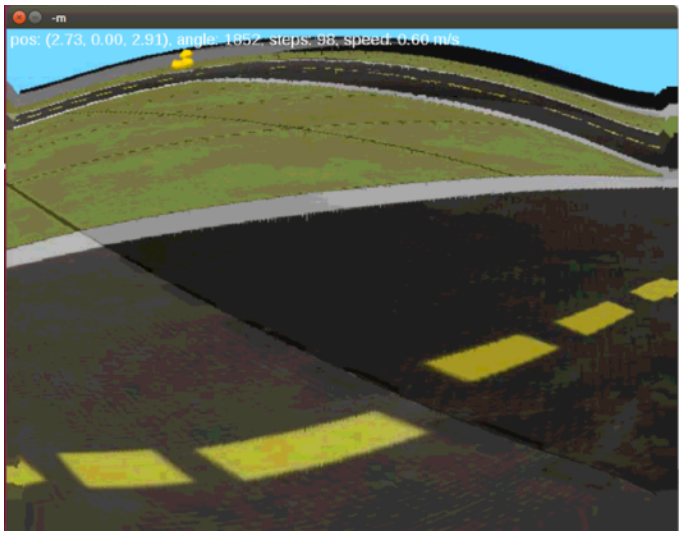
After training, your policies (models) will be saved at **duckietown_rl/pytorch_models**
With name:
Actor model: DDPG_(seed number)_actor.pth
Critic model: DDPG_(seed number)_critic.pth

Please edit the file **scripts/test_cnn.py line 11**
seed = <YOUR_SEED_NUMBER> (in this example, 123)

Test your trained policy

```
Laptop $ python3 -m scripts.test_cnn.py
```

A screen should shows up demonstrate the result.
The video might seems too fast. You can add a time pause after each step.



add time pause

At line 50, after `env.render()`, add following line

```
import time
Time.sleep(0.1)
```

Detail tutorial Link

Rule of Thumb:

- Don't set replay buffer size too big. If it costs more memory than your RAM, training process would become very slow. See file duckietown_rl/args.py

- You actually don't need that much time step. Set max_timestep to 5e5 or even lower. See file duckietown_rl/args.py

How can I make it better?:

- Check out the duckietown_rl/wrapper.py, modify rewards in class DtRewardWrapper (set them higher or lower and see what happens)
- Check out src/gym-duckietown/gym_duckietown/simulator.py. See function compute_reward and try play around with it.
- Use other network ex: ResNet
- Cut off the horizon from the image (and correspondingly change the convnet parameters).
- Try making the observation image grayscale instead of color. And while you're at it, try stacking multiple images, like 4 monochrome images instead of 1 color image

Scoring Criteria:

- Performance (100%):
  - Run baseline solution.
  - Send your "wrapper.py", "ddpg.py" and trained model (both actor and critic)
  - Zip all file in one file and name it like "DLP_Duckietown_yourstudentID_name"
  - Alternate way
    - Submit your solution model to AIDO website. (see appendix tutorial)
    - Send screenshot of your performance

Reference:

[1] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
[2] Lever, Guy. "Deterministic policy gradient algorithms." (2014).
[3] ShangtongZhang et al. "Highly modularized implementation of popular deep RL algorithms in PyTorch." Retrieved from Github: https://github.com/ShangtongZhang/DeepRL.
[4] pemami4911 et al. "Collection of Deep Reinforcement Learning algorithms." Retrieved from Github: https://github.com/pemami4911/deep-rl.
[5] bhairavmehta95 et al. "challenge-aido1_LF1-baseline-RL-sim-pytorch" Retrieved from Github: https://github.com/duckietown/challenge-aido1_LF1-baseline-RL-sim-pytorch
[5] Duckietown Library https://docs.duckietown.org/DT19/AIDO/out/index.html

Appendix:

Tutorial on how to submit your solutions to AIDO.

 # Get accounts
Docker Hub account: Create an account here.
Duckietown account: Create an account here.

 # Software requirements
Docker: Install Docker from these instructions.
Duckietown Shell: Install the Duckietown Shell by following the *Installation* instructions in the README.

 # Duckietown Shell and token authentic
Check if Duckietown shell is installed successfully.
Laptop $ dts version

Set the Duckietown authentication token. This combine Duckietown Shell to your personal Duckietown account
Laptop $ dts tok set

See if you have a good authentication token
Laptop $ dts challenges info

 # Set Docker Hub information
Set your Docker Hub username using
Laptop $ dts challenges config --docker-username <YOUR_DOCKER_USERNAME>

Login to Docker Hub
Laptop $ docker login

 # Submit the RL baseline solution you just trained

Put files and models to the right direction
- challenge-aido_LF-baseline-RL-sim-pytorch/duckietown_rl/ddpg.py rename to challenge-aido_LF-baseline-RL-sim-pytorch/model.py
- challenge-aido_LF-baseline-RL-sim-pytorch/duckietown_rl/pytorch_models/DDPG_(seed_number)_actor.pth rename to challenge-aido_LF-baseline-RL-sim-pytorch/models/model_actor.pth
- challenge-aido_LF-baseline-RL-sim-pytorch/duckietown_rl/pytorch_models/DDPG_(seed_number)_critic.pth rename to challenge-aido_LF-baseline-RL-sim-pytorch/models/model_critic.pth
- challenge-aido_LF-baseline-RL-sim-pytorch/duckietown_rl/wrappers.py to just challenge-aido_LF-baseline-RL-sim-pytorch/wrappers.py

After that, uncomment Line 22 23 in solution.py , otherwise the solution won't use your model.

Before the submission, add this line to submissiom.yaml
challenge: ['aido2-LF-sim-validation']

Finally, submit your solution
Laptop $ dts challenges submit

Check out dashboard for your submission
The video won't show up instantly, wait for a couple time for the server to evaluate.

You can edit submission.yaml to customize your "user-label" and "description"
Do dts challenges submit again after editing to re-submit your solution

Further information here.
AIDO dts challenges CLI here.