Advanced Programming Practice

HW6 Imitation Learning

2023 Fall-CSE4152

Sogang University

Slides Credit by Geiger

How to install GYM and pytorch

Please download GYM we provided from cyber campus.

Highly recommend you to use conda environment (minconda/anaconda).

Please install the GYM environment by following the setup slide.

Experiment overview

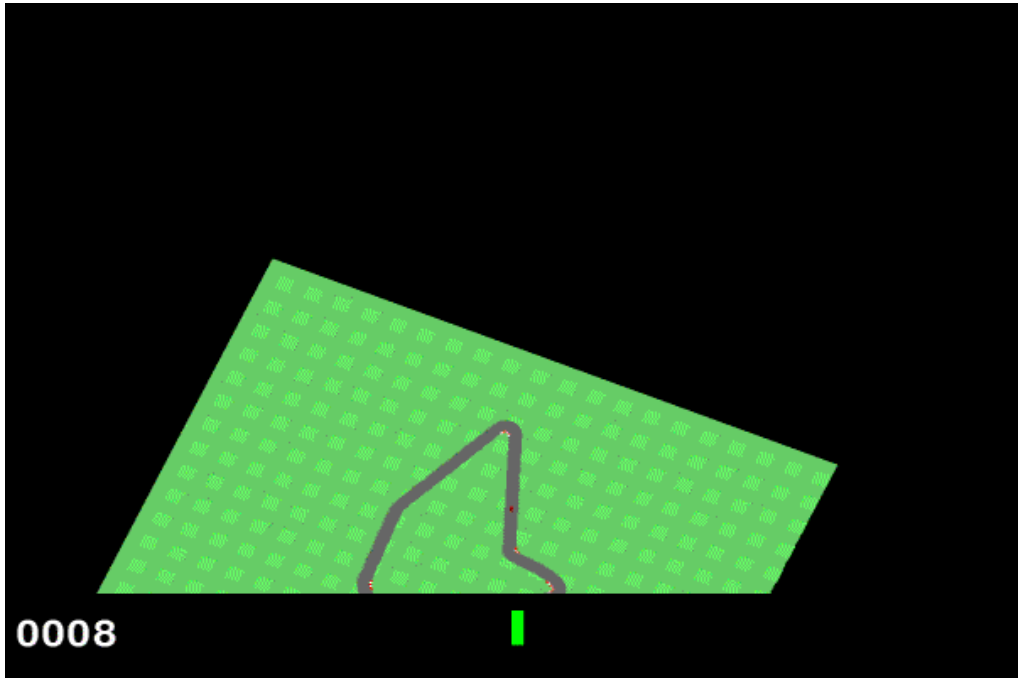
Goal: implement reinforcement learning framework that trains a driving agent.

Simulator: **OpenAI GYM**

- https://www.gymlibrary.ml/
- We will use Box2D-CarRacing
- Please install gymSG

CarRacing information

- Action: steering, acceleration, brake
- Sensor input: 96x96x3 screen
  - It shows car's states and path information.



Skeletons

Network.py: a agent network should be implemented

demonstration.py: record/load the demonstration

training.py: the main part of behavior cloning (completed)

Evaluate.py: test your agent on unexperienced track (completed)

main.py: the main console to run the learning framework (completed)

Four tasks

A. Implement loading demonstrations

Given the folder of the imitations, it should load all observation and action files into two separate lists observations and actions which are returned. Implement the function `load_demonstrations` in `demonstrations.py`

B. Understand training

The module training.py contains the training loop for the network. Read and understand its function train. Why is it necessary to divide the data into batches? What is an epoch? What do lines 69 to 77 do? **Please write on report precisely.**

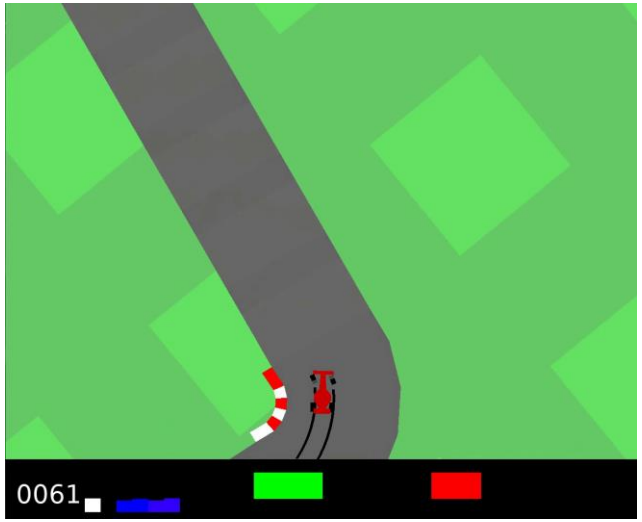
C. Conversion from action to classes and selection of action from scores

Define the set of action-classes you want to use and complete the class-methods `actions_to_classes` and `scores_to_action` in `network.py` The former maps every action to its class representation using a one-hot encoding and the latter retrieves an action from the scores predicted by the network.

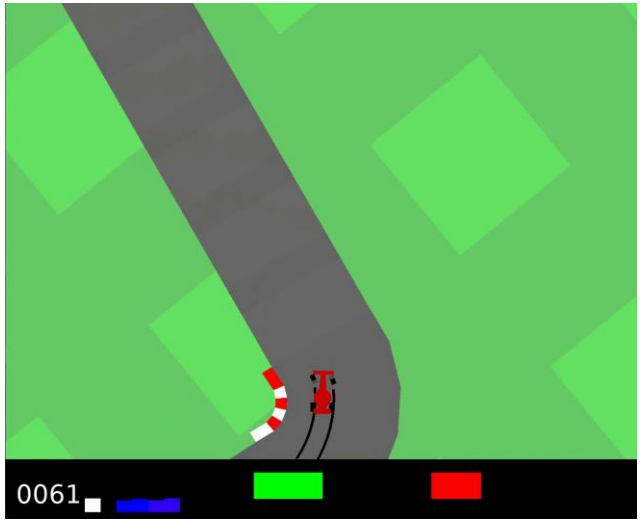
D. Network Implementation

- Design and implement an simple network architecture in `network.py`.
- Convolution layers on the images, followed by 2 or 3 fully connected layers to extract a 1D feature vector. Let each layer be followed by a ReLU as the non-linear activation.

Pipeline overview




```
Epoch     1     [Train] loss: 10.484694         ETA: +13251.256037s
Epoch     2     [Train] loss: 8.430299  ETA: +6689.617068s
Epoch     3     [Train] loss: 6.944557  ETA: +4499.477992s
Epoch     4     [Train] loss: 6.765140  ETA: +3403.183179s
Epoch     5     [Train] loss: 6.035543  ETA: +2746.988570s
Epoch     6     [Train] loss: 6.392803  ETA: +2304.578091s
Epoch     7     [Train] loss: 5.718839  ETA: +1988.900927s
Epoch     8     [Train] loss: 5.171953  ETA: +1751.453157s
Epoch     9     [Train] loss: 5.026197  ETA: +1566.527530s
Epoch    10     [Train] loss: 4.874210  ETA: +1417.454919s
Epoch    11     [Train] loss: 4.240147  ETA: +1295.563473s
Epoch    12     [Train] loss: 3.777738  ETA: +1193.880279s
Epoch    13     [Train] loss: 3.421766  ETA: +1109.131542s
Epoch    14     [Train] loss: 3.222221  ETA: +1035.915575s
Epoch    15     [Train] loss: 3.113763  ETA: +971.666078s
Epoch    16     [Train] loss: 2.785062  ETA: +915.625350s
Epoch    17     [Train] loss: 2.750206  ETA: +866.223965s
Epoch    18     [Train] loss: 1.884841  ETA: +822.092400s
Epoch    19     [Train] loss: 1.779317  ETA: +782.940820s
Epoch    20     [Train] loss: 1.484417  ETA: +747.180350s
Epoch    21     [Train] loss: 1.393778  ETA: +714.822199s
Epoch    22     [Train] loss: 0.943727  ETA: +686.009632s
Epoch    23     [Train] loss: 0.716615  ETA: +659.375737s
Epoch    24     [Train] loss: 0.581006  ETA: +634.877513s
Epoch    25     [Train] loss: 0.445800  ETA: +612.310310s
Epoch    26     [Train] loss: 0.317822  ETA: +591.575134s
```



Demonstration: record expert's play

- **python main.py --teach**

Track information is stored in "data/teacher" folder

Train: train an agent

- **python main.py --train**

It loads data from "data/teacher"
Agent file is stored in "data".

Test: evaluate the agent

- **python main.py --test**

Demonstrations

➢**python main.py --teach**

s – store the current demonstration. They are stored in “data/teacher”

r – restart

q - quit

User

up	**press**
down	not
left	press
right	not

Keyboard input

[-1, 0.5, 0]

steer angle (회전각)

throughput (출력)

brake (감속)

Action

C. action to class

There are 9 common actions.

Other actions are weird (ex. left and right together)

Each action becomes a unique **action class**.

```
self.classes = [[-1., 0., 0.],  # act.class 0: left
                [-1., 0.5, 0.],# act.class 1: left and accelerate
                [-1., 0., 0.8],# act.class 2: left and brake
                [1., 0., 0.],  # act.class 3: right
                [1., 0.5, 0.], # act.class 4: right and accelerate
                [1., 0., 0.8], # act.class 5: right and brake
                [0., 0., 0.],  # act.class 6: no input
                [0., 0.5, 0.], # act.class 7: accelerate
                [0., 0., 0.8]] # act.class 8: brake
```

```python
def register_input():

    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                action[0] = -1.0
            if event.key == pygame.K_RIGHT:
                action[0] = +1.0
            if event.key == pygame.K_UP:
                action[1] = +0.5
            if event.key == pygame.K_DOWN:
                action[2] = +0.8  # set 1.0 for wheels to block to zero rotation
            if event.key == pygame.K_r:
                global retry
                retry = True
            if event.key == pygame.K_s:
                global record
                record = True
            if event.key == pygame.K_q:
                global quit
                quit = True

        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT and action[0] < 0.0:
                action[0] = 0
            if event.key == pygame.K_RIGHT and action[0] > 0.0:
                action[0] = 0
            if event.key == pygame.K_UP:
                action[1] = 0
            if event.key == pygame.K_DOWN:
                action[2] = 0
```

Event handling of keyboard

C. action to class

In “scenarios”, there are recorded plays (screen, action)

To train the network, we need to convert recorded actions to action classes so that we train the classification network that outputs the **action class** (index).

```
[[0, 0, 0],
 [1, 0, 0],
 [-1, 0, 0.8],
 [0, 0.5, 0],
 [1, 0, 0],
 [1, 0.5, 0],
 [0, 0, 0.8]]
```

actions

```
[6,
 3,
 2,
 7,
 3,
 4,
 8]
```

action classes

C. score to action

The classification network outputs scores of actions.

We select the action with the highest score

Scores for a given state
(game screen)

[-12, 2, 10, 27, -12, -4, -18, 1, 2]

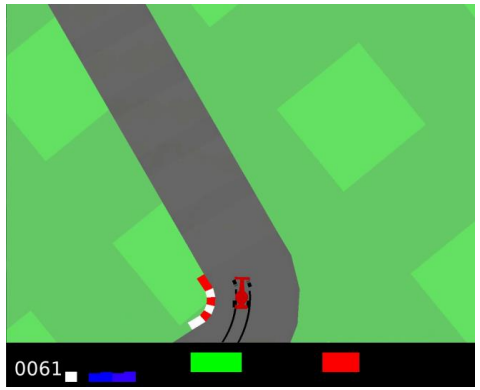
Action of the action classes
with the highest score

self.classes[3]

```
[1., 0., 0.],  # act.class 3: right
```

Network Design

Classification Network



Input: 게임 화면

π_θ

MLP-based Classifier

0	0.0
1	0.0
2	0.1
3	0.2
4	0.0
5	**0.6**
6	0.0
7	0.0

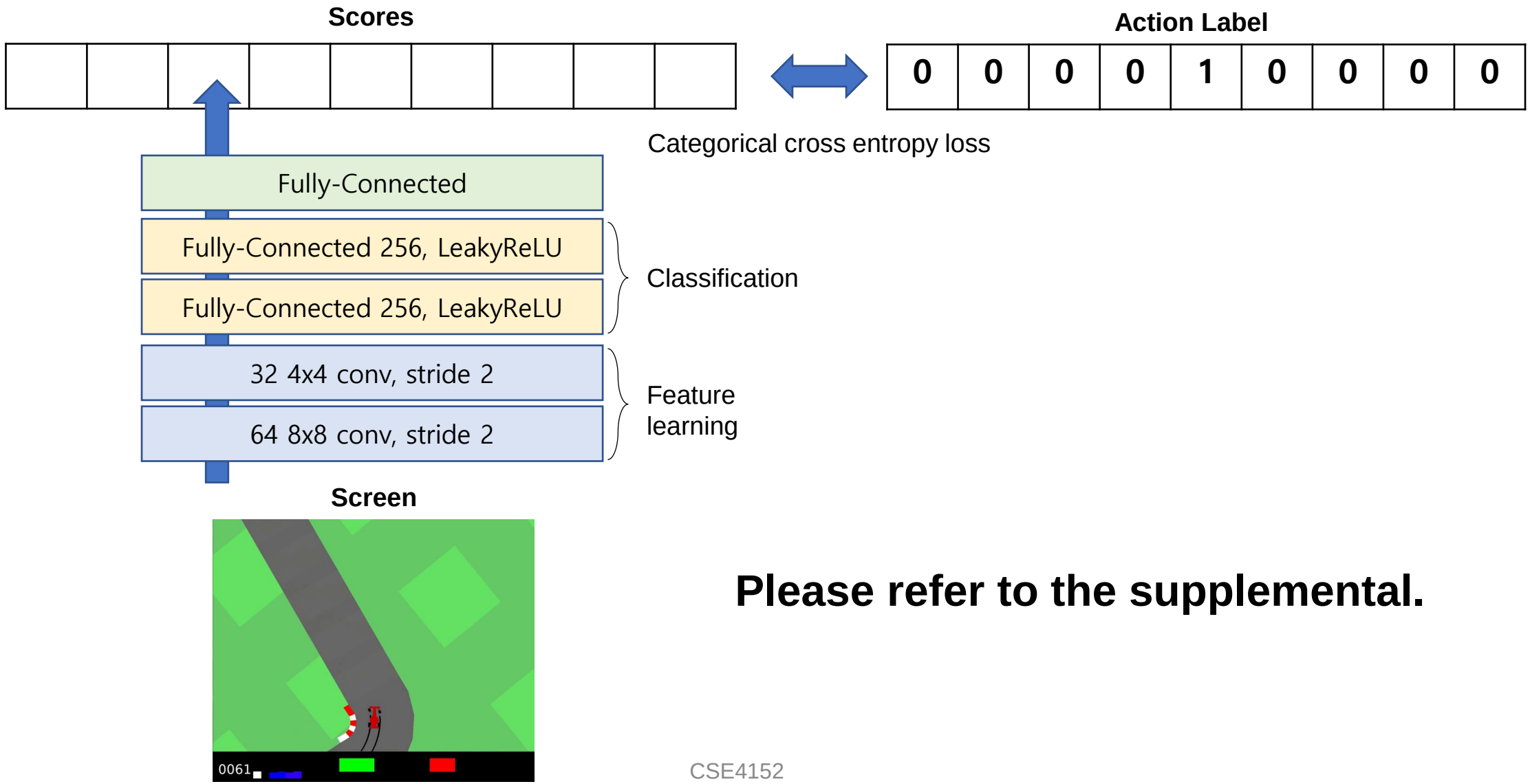
Output: scores of each action
(8x1 real-number vector)

MAX

action class 5
(브레이크)

- 2 to 3 Convolutional layers + leakyReLU
- 2 to 3 Fully connected layers + leakyReLU

Network overview



Please refer to the supplemental.

Network Specification Reference

- In part of “Playing Atari with Deep Reinforcement Learning”,

> We now describe the exact architecture used for all seven Atari games. The input to the neural network consists is an $84 \times 84 \times 4$ image produced by ϕ. The first hidden layer convolves 16 8×8 filters with stride 4 with the input image and applies a rectifier nonlinearity [10, 18]. The second hidden layer convolves 32 4×4 filters with stride 2, again followed by a rectifier nonlinearity. The final hidden layer is fully-connected and consists of 256 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions varied between 4 and 18 on the games we considered.

You can change the inner network specification as you want. There is no standard answer.

https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

Agent network

- Input: game screen(Dim: Mx96x96x3, M: the number of batches)
- Output: action score (Dim: Mx9)

- The agent selects an action with the max score.

- Analogous to the classification network
  - You can use a convolutional neural network
  - Reference: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

HW6 -submission

Deadline: ~2023.11.19 (SUN) on Cyber campus

Submission:

- Python code: your implementation
- Report: write answer of “task B. Understand training” on report precisely
- Agent file: When training runs successfully, the model agent is automatically saved
- Zip file: include three files above

File format:

- Python MLP code: demonstration.py, network.py
- Report: CSE4152_학번_HW06.txt
- Agent file: agent.pt
- Zip file: CSE4152_학번_hw6.zip

If you have any questions, feel free to ask in the cyber campus Q&A or send them to the TA's email(kangpiljae@gmail.com)

Tips

- Before starting the project, please read codes in the folder carefully.