COMP 423/523: Computer Vision for Autonomous Driving Homework 1

Due: April, 17 2022

Introduction

In this assignment, you will work first answer some questions based on the concepts and the algorithms we learned in the class. Then, you will implement three approaches to self-driving that we learned in the class:

1. (**20 points**) Pen-and-paper part
2. (**25 points**) Conditional imitation learning
3. (**25 points**) Conditional affordance learning
4. (**30 points**) Deep RL using ground-truth affordances
5. (**Bonus 10 points**) Extend Deep RL (4) to use predicted affordances (3)

You will be using the CARLA simulator for training and testing your solutions, which can be installed either on your local machine or on the university HPC cluster as we discussed in the class. If possible, we advise you to use your local machine for development, and the university cluster for running the final code. For information on how to install CARLA, you can look at the tutorial slides we previously shared with you.

Note that the code we provide is very lengthy but you do not need to understand or modify the parts we do not specifically ask you to modify.

In addition to the CARLA simulator and the code we provide, you will need the expert dataset we provide to you. The dataset is hosted on our university's HPC cluster under the path **/userfiles/eozsuer16/expert_data/**. If you wish you can also download this dataset to your local machine. (you can download a subsection of it for development purposes if you do not wish to make such a big download.) This dataset was collected using a hand designed expert autopilot which has access to the simulator ground truth data. As the autopilot drove around town, a sample was saved every 0.5 seconds. The collected samples consist of an 512x512 RGB image saved as a .png file, and a .json file that contains relevant measurements for driving.

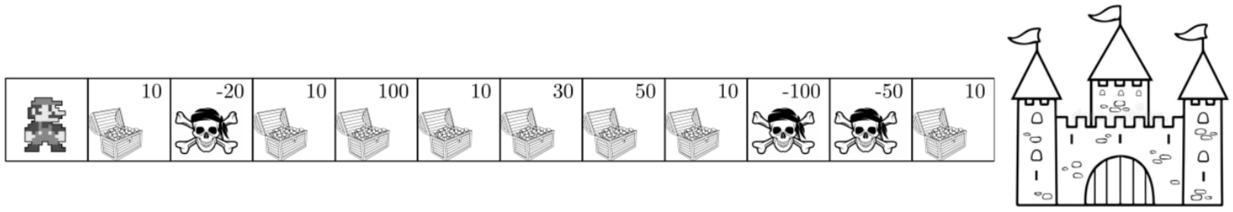
Important Note: This is an individual assignment, not group work. You can discuss your solution with each other but you are supposed to work on the code on your own.

1 Pen-and-Paper Questions

1. (**1 point**) In imitation learning or reinforcement learning, what is the goal of policy network?
2. (**1 point**) What does it mean for a state to be partially or fully observable. Give some examples for each.
3. (**1 point**) What is the main assumption in Behavior Cloning? Explain how this assumption leads to the distribution shift problem.
4. (**1 point**) Describe the two solutions that we have seen in the class for the distribution shift problem.
5. (**1 point**) What is the problem of behavior cloning at the intersections? What is the solution that we have seen in the class to this problem? How does conditional imitation learning improve behavior cloning in these cases?
6. (**1 point**) What is the role of affordances in self-driving? What kind of affordances can be useful for highways driving or inner-city driving or both? Compare the two by suggesting some specific example affordances for each case.
7. (**1 point**) What are ideal properties of visual abstractions in self-driving?
8. (**1 point**) When we use semantic segmentation as an intermediate representation, we learned that fewer classes and coarsely annotated images perform better in self-driving. What could be the reason for that?
9. (**1 point**) In visual abstractions, we said that typically $n_r \ll n_a$ where n_r is the number of images annotated with semantic labels and n_a is the number of images annotated with expert actions. Besides the cost of labelling, is there any other intuition behind this assumption?
10. (**1 point**) What is the difference between online and offline evaluation? Why does relying on offline evaluation only can be dangerous? Why do offline metrics fail to represent online metrics?

11. (**10 points**) Mario wants to reach princess Peach. Mario's home and the palace of Peach are located on a 1D discrete coordinate system and the palace is always on the right side of Mario's home. Mario starts his journey from his home and he can move two or three steps to the right. At each discrete coordinate, there are several coins that Mario can collect or a thief who robs a portion of Mario's coins. Mario wants to collect many coins as Mario wants to buy a new dress for Peach. Help Mario to collect as many coins as possible during his journey.

    Mario knows where he is located. The state can be denoted as $s_t = x_t$ where x_t is his location at time t. Note that he holds 100 coins at the beginning. Mario will enter the palace by moving over the last square.



- Mario is greedy but a fool so he always moves to a square where he makes more profit (or less loss). Define the policy $a_t = \pi(s_t)$ of greedy but fool Mario.
- Roll out his trajectory plus his coin status until reaching the palace.
- Calculate the state-value function $V^\pi(s_t)$ and the action-value function $Q^\pi(s_t, a_t)$ of the greedy policy π assuming a discount factor of 1.
- Find an optimal policy π^* via Q-learning. Let $Q_k(s_t, a_t)$ be the k-th updated $Q(s_t, a_t)$ function by the Bellman equality equation. To speed-up the calculations, we make two changes to the original Q-learning algorithm: Instead of random initialization, initialize $Q(s_t, a_t)$ to the greedy policy π. Furthermore, during each iteration, update all elements of the Q-table simultaneously based on the TD error update rule, instead of executing a behavior policy with ϵ-greedy exploration and updating only a single entry. Compute $Q_k(s_t, a_t)$ until the induced policy becomes the optimal policy.
- Induce the optimal policy from the optimal state-value function $Q^*(s_t, a_t)$.

2 Imitation Learning

The aim of this exercise is to design a network that learns a policy $\pi_\theta : S \to A$, parameterized by θ, to predict the expert action for a given state. We formulate it as a supervised learning problem, where the objective is to follow the observed demonstrations of an "expert" driver.

2.1 Loading the Data

In order to use the expert data we have provided to you, you must complete the implementation of the **ExpertDataset** class in the file **expert_dataset.py**

You must complete the **__init__** function such that it constructs the dataset instance using the **data_root** parameter. Then you must complete the **__getitem__** function such that it returns an RGB image and the corresponding measurements when given an index. The measurements are stored in .json files as dictionaries. Each item in the json file is explained below:

- "speed": Current speed in m/s (float)
- "throttle": Expert throttle action (float)
- "brake": Expert brake action (float)
- "steer": Expert steer action (float)
- "command": High level command (int). The integer values and their meanings are as follows: LEFT = 0, RIGHT = 1, STRAIGHT = 2, LANEFOLLOW = 3
- "route_dist": Ego vehicle distance from optimal path in meters (float)
- "route_angle": Signed angle between optimal path and ego vehicle heading (float)
- "lane_dist": Ego vehicle distance from nearest lane center in meters (float)
- "lane_angle": Signed angle between nearest lane heading and ego vehicle heading (float)
- "hazard": Whether there is a vehicle/pedestrian ahead, this is always false (bool)
- "hazard_dist": Distance to nearest vehicle/pedestrian ahead of us, always 25.0 (float)
- "tl_state": 0 if light is green or there is no light, 1 if light is yellow or red (int)
- "tl_dist": Distance to nearest traffic light ahead of us, 45.0 if nothing detected (float)
- "is_junction": Whether we are at a junction or not (bool)

Note: The images from the carla dataset are saved using opencv, which uses BGR encoding by default. When you open the images, they might appear as if the red and blue channels have swapped. This channel order does not matter to the neural network **as long as you make sure the training and inference images have the same channel order!**

2.2 Model

Next you must complete the **CILRS** class in the file **model/cilrs.py**. Your model must follow the branched architecture in CILRS [1]. As in the paper, your model must take an RGB image, vehicle speed and a high level command as input, and output driving actions along with a speed prediction. However we advise that instead of ResNet50, you use ResNet18 as the backbone of your architecture. Note that you shouldn't re-implement the ResNet architectures yourself, you can import and use the pretrained PyTorch implementations.

2.3 Training

In this part you must complete the **train** and **validate** functions in the file **cilrs_train.py**. You must train your model using the expert dataset. Note that you will not need all the measurements of the dataset, only the high level commands, speed values and the expert actions are required for the training. You can experiment with different optimizers(Adam, SGD), hyper-parameters, and loss functions(L1, L2). If you would like a starting point, you can look at the CILRS [1].

2.4 Evaluation & Discussion

Now you must evaluate the driving performance of your agent. To test this, you must complete the **load_agent** to initialize your model and load its weights. You must also complete the **generate_action** function in the file **cilrs_eval.py**. After you are done, you can run the file with the command **python cilrs_eval.py** to evaluate the driving performance of your agent in the simulator.

Discuss your results in your report. Include plots of your train and validation losses. Optionally, you can try to improve your model performance using some of the following techniques:

- Data augmentation
- Weight regularization
- Dropout

3 Affordance Prediction

In this part, you will again use the expert dataset we have provided to you and train a model that takes RGB images as input and predicts a pre-defined set of affordances. The affordances we would like you to predict are:

- Lane distance
- Route angle

- Traffic light distance
- Traffic light state

3.1 Loading the Data

Depending on your implementation of the **ExpertDataset** class, it might not be directly usable for both the imitation learning and the affordance prediction tasks. If that is the case, you must modify your previous implementation in **expert_dataset.py** such that it works with **both tasks**.

3.2 Model

For this part, you will complete the model class **AffordancePredictor** in the file **models/affordance_predictor.py**. Your model must take a single RGB image as input, and output 4 affordances. Note that while lane distance, route angle and traffic light distance values are floating point values, traffic light state is a binary signal that becomes 1 when there is a red/yellow light, and 0 otherwise.

The architecture of your model is up to you, but if you are not sure where to start, you can use a modified version of your model from the first part or refer to CAL [2]. In the CAL paper, they use high level command information along with visual input when predicting relevant affordances such as lane distance and route angle. For command independent affordances such as traffic light state, they use only the visual input.

3.3 Training

Next you must complete the **train** and **validate** functions in the file **pred_train.py**. You should think about how you are going to calculate the loss for a mixture of continuous and binary inputs.

3.4 Evaluation

There is no separate evaluation code for this task. Discuss in your report what kind of architecture you used and whether it worked well for all affordances. Which affordances was your model least successful in predicting? Include plots of your loss functions.

4 Reinforcement Learning

In this part, you will train an RL agent using the TD3 algorithm. TD3 is an improvement over the DDPG algorithm we have seen in the class. For a good explanation of the TD3 algorithm, its changes and how to implement it, you can check out Spinning Up: TD3.

4.1 Loss computation

In this part you must fill the **_compute_q_loss** and **compute_p_loss** functions inside the file **rl/td3.py**. You can refer to the spinning up pseudocode for reference.

4.2 Reward function

Next you must design the reward function for the RL algorithm by completing the **get_reward** function in the file **carla_env/managers/reward_manager.py**. You can use any value available to you inside the **state** dictionary to generate your reward function. Some values that might be helpful in desigining your reward:

- speed
- route_dist
- route_angle
- tl_state
- tl_dist
- is_junction
- command

Remember that reward functions are much more flexible than loss functions since they don't need to be differentiable, and desigining a reward function is more art than science, so let your creativity run free. Or you can check out the reward function of the Implicit Affordances paper [3].

4.3 Affordance Selection

In this section you must choose which ground truth features you will give to your policy and q networks by completing the function **_extract_features** in the **rl/td3.py** file. Same as the reward function, you are free to access any ground truth information available in the state dictionary.

4.4 Policy Network

Next you must design your policy network in **models/policy.py**. Since our input is a small vector of scalar features rich in semantic information, a compact MLP network should be fine. Your output should be a tensor of shape [B, 2], which will be used as an acceleration and steering value. Both actions must be in the range [-1, 1]. The first action stands for acceleration, with -1 being full braking, and 1 being full throttle. The second action stands for steering, with -1 steering fully to the left, and 1 steering fully to the right

4.5 Q Network

Your policy network will be very similar to your policy network, with two differences.

- You will need to take actions as input along with the features you selected.
- You will be outputting a tensor of shape [B, 1] since you are outputting only a single q-value.

4.6 Training

To train your RL agent, you can use the **td3_train.sh** shell script. Since the CARLA simulator can sometimes crash during RL training, this script (crudely) ensures your training will continue from the last saved checkpoint.

If you would like to change the hyperparameters for RL training, you can do so by changing the file **configs/td3.yaml**

Your training will be logged to tensorboard automatically, which you can use to monitor your training if you wish.

4.7 Evaluation

When you are satisfied with your RL agent, you can stop the training and evaluate its success with the command **test_rl_agent.py**. Discuss your results, compare it to your imitation learning agent. Which fared better? What causes of termination do you observe in the two different agents?

Submission

Your report should be in the pdf format and your code should be hosted as a github repository. Include the link to your github project at the start of your report. You will then only submit your pdf file to blackboard.

References

[1] Felipe Codevilla et al. "Exploring the Limitations of Behavior Cloning for Autonomous Driving". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[2] Axel Sauer, Nikolay Savinov, and Andreas Geiger. "Conditional Affordance Learning for Driving in Urban Environments". In: *Conference on Robot Learning (CoRL)*. 2018.

[3] Marin Toromanoff, Émilie Wirbel, and Fabien Moutarde. "End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.