Autonomous Vision Group
Prof. Dr.-Ing. Andreas Geiger

Eberhard Karls Universität Tübingen

Self-Driving Cars

Ex. 1 – Coding Challenge: Imitation Learning

Deadline: 17. November 2022; 3pm

For this exercise you need to submit a **.zip** folder containing your pre-trained model as a **.pth** file, the filled out submission form (*submission.txt*) and your code (namely the files *network.py, training.py, demonstrations.py*). Please use the provided code templates for these exercises. The given `main.py` file will be used for evaluating your code.
Submissions with changes to this main file, to the gym environment or additionally installed packages will not be considered. Your code must be able to run within the provided singularity container from exercise 00 to participate in the challenge. You may choose discrete or continuous actions as output of your agent. You are allowed to collect your own data using different augmentation methods. We recommend to collect the data inside the singularity container to ensure reproducibility. You must provide a single model, i.e., ensembles are not allowed. Your method must be a pure IL method, i.e., you are not allowed to use reinforcement learning or handcrafted (i.e., PID-) controllers based on waypoint predictions, but your model must instead directly output the action. Comment your code clearly, use docstrings and self-explanatory variable names and structure your code well.
For the challenge we only evaluate your trained model. But we also provide some questions and tasks to deepen your understanding of the topic.

1.1 Network design

Let $\mathcal{S}$ be the state space and $\mathcal{A}$ the action space. The gym environment encodes a transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ that maps a (state, action) - pair to a new state. It also provides a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which we will only use for evaluation this time.
The aim of this exercise is to design a network that learns a policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$, parameterized by θ, to predict the best 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.

a) Some demonstrations from an expert are given in *data/teacher*. Implement the function *load_demonstrations* in *demonstrations.py*. Given the folder of the demonstrations, it should load all `observation` and `action` files into two separate lists `observations` and `actions` which are returned.

b) 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 45 to 51 do?

c) To start with, we formulate the problem as a classification task. Have a look at the actions provided by the expert demonstrations, there are three controls: steer, gas and brake. Which values do they take? Since they are not independent (accelerate and brake simultaneously does not make sense), it is reasonable to define classes of possible actions, like {steer_left}, {steer_right and brake}, {gas} and so forth.
Define the set of action-classes you want to use and complete the class-methods *actions_to_classes* and *class_scores_to_action* in *network.py*. The former maps every action to its class representation and the latter retrieves an action from the class scores predicted by the network.

d) Design and implement a small first network architecture in *network.py*. Start with 2 to 3 2D convolution layers on the images, followed by 2 or 3 fully connected layers (linear layers) to extract a 1D feature vector. Let each layer be followed by a ReLU as the non-linear activation (see code snippets below).
The output of the network should function as a controller for the car and predict one of the action-classes for each given state. Use the torch CrossEntropyLoss (combines LogSoftmax and NLLLoss) to calculate the Loss.

```
torch.nn.Sequential(
   torch.nn.Conv2d(in_channels, out_channels, filter_size, stride=*arg),
   torch.nn.LeakyReLU(negative_slope=0.2))

torch.nn.Sequential(
   torch.nn.Linear(in_size, out_size),
   torch.nn.LeakyReLU(negative_slope=0.2))
```

e) Implement the forward pass for your network, which is the function *forward* in *network.py*. Given an observation, it should return the probability distribution over the action-classes predicted by the network. You can decide whether you want to work with all 3 color channels or convert them to gray-scale beforehand. Motivate your choice briefly.
Train your network by running *python main.py --train*. Afterwards, enjoy watching its performance by running *python main.py --test* on your local machine. Can you achieve better results when changing the hyper-parameters? Can you explain this?

f) *demonstrations.py* provides some code to record new demonstrations. Complete the function *save_demonstrations* and start driving yourself by running *python main.py --teach* on your local machine. What is 'good' training data? Is there any problem with only perfect demonstrations?

1.2 Network Improvements

Now that your network is up and running, it's time to increase its performance! Each of the following tasks adds to its architecture. It is up to you to choose which of them you use for participating in the competition, we just provide some ideas.

a) **Observations.** The training data of the network actually contains more information than just the image from the car in the environment. Look at the class method *extract_sensor_values* in *network.py*. What does it do? Incorporate it into your network architecture. How does the performance change?

b) **MultiClass prediction.** Design a second network architecture that encodes a multi-class approach by defining 4 binary classes that represent the 4 arrow keys on a keyboard and stand for: steer right, steer left, accelerate and brake. Since those don't all exclude each other, let the network learn to predict 0 or 1 for each class independently.
You will need to implement another loss function and might find a sigmoid-activation function useful. Again, compare the results to the previous classification approach.

c) **Classification vs. regression.** Formulate the current problem as a regression network. Which loss function is appropriate? What are the advantages / drawbacks compared to the classification networks? Is it reasonable to use a regression approach given our training data?
Hint: You can control the top speed of your car by changing the `acceleration` variable in *ControlStatus* from *demonstrations.py*. But be aware that this also changes the actions you are recording.

d) **Data augmentation.** As discussed in the lecture, the more diverse the training data is, the better generally. Investigate two ways to create more training data with synthetically modified data by augmenting the (observation, action) - pairs the simulator provides. Does the overall performance change?

e) **Fine-tuning.** What other tricks can be used to improve the performance of the network? You could think of trying different network architectures, learning rate adaptation, dropout-, batch- or instance normalization, different optimizers or mitigating class imbalance of the training data.

1.3 Competition

With each coding exercise sheet you are welcome to participate in our competition! The winners for each of the 3 exercise sessions will be given the opportunity to present their approach in the very last lecture. The evaluation works as follows: the models are tested on a set of validation-tracks. For each track, the reward after 600 frames is used as performance measure and the mean reward from all validation tracks then forms the overall score. For the final ranking, we will run that evaluation on a set of secret tracks for every submission. The reward is -0.1 every frame and +1000/N for every track tile visited, where N is the total number of tiles in track. Good luck!

1.4 References

[1] https://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf
[2] https://arxiv.org/pdf/1604.07316.pdf