Robot Learning

Exercise 1 – Imitation Learning

Kote Batonisashvili

Rob Cervone-Richards

1.1

a) Carla environment was created with town set to 01. Weather set to –speed 2.0 to have quicker changes in the environment, thus having better and more robust data. While we were able to spawn all 50 vehicles, approximately 35 pedestrians spawned because the environment did not have appropriate locations for all.
b) We decided to include the following sensors: RGB, collision detection, lane change detection, object detection, steering, throttle, brake. All this information gets stored within a csv file along with the current time stamp and frame data.
c) We collected several thousand images for training data under various conditions.
d) For good training data, the more covered scenarios and events, the more accurate the model. If you were to include as much information as possible, data from the most generalized cases, to the most fringe of situations, then the training agent will be able to learn from all situations and adjust as necessary. If data that is collected is only from perfect demonstrations, then just as we discussed in class, the agent would not know what to do in certain scenarios and cause catastrophic failures. Since one unknown situation can cascade into further unknown situations, the agent would not be able to respond appropriately given "expert" data. This is why the more information that can be fed to the training, the better the outcome. 6 hours of straight driving and stopping while accurate and dense, would not be nearly as important as 6 hours of different scenarios.

1.2

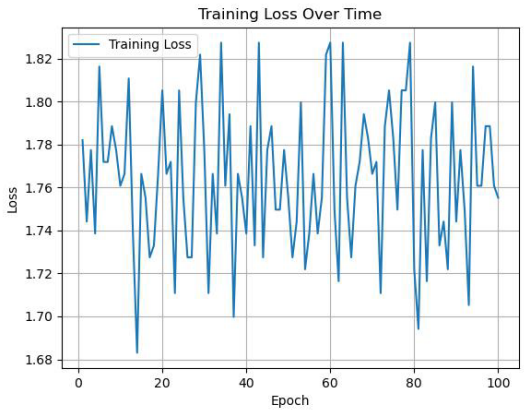
a) Dataset loader was added within the python file. We implement our thousands of images into tensor and use them as reference guides since they correspond to our drive frame data.
b) One-hot encoding has 9 classes – Steer left, steer straight, steer right, throttle left, throttle straight, throttle right, brake left, brake straight, brake right. These are all assigned a score of 0 to 1, except for steering, which is -1 to 1, to account for steer left and right.
c) We have 3 2D convolution layers, and 2 fully connected layers.
d) Forward Pass is implemented.
e) Breaking up data into batches has been proven to be more efficient. Several studies have confirmed that instead of having one long learning cycle, its best to divide up

the training so that the model updates and learns more frequently. An epoch is when a training model runs through its dataset in its entirety once.

f) Cross-entropy-loss was not from pytorch's built in model, and instead derived.

g) It looks like the training model was not able to converge and kept cycling back and forth between poor results. Incorporating more sensors would have fixed something like this.

```
Epoch  1  [Train] loss: 353.771240     ETA: +2775.950319s
Epoch  2  [Train] loss: 345.950806     ETA: +2615.136315s
Epoch  3  [Train] loss: 364.950806     ETA: +2546.276133s
Epoch  4  [Train] loss: 362.950775     ETA: +2499.762726s
Epoch  5  [Train] loss: 342.950806     ETA: +2461.460278s
Epoch  6  [Train] loss: 352.950775     ETA: +2427.472648s
Epoch  7  [Train] loss: 362.950806     ETA: +2395.242635s
Epoch  8  [Train] loss: 361.950775     ETA: +2366.293967s
Epoch  9  [Train] loss: 352.950806     ETA: +2337.095860s
Epoch 10  [Train] loss: 362.950806     ETA: +2308.468303s
Epoch 11  [Train] loss: 351.950806     ETA: +2281.085709s
Epoch 12  [Train] loss: 357.950806     ETA: +2253.044859s
Epoch 13  [Train] loss: 359.950806     ETA: +2226.247894s
Epoch 14  [Train] loss: 356.950806     ETA: +2199.779894s
Epoch 15  [Train] loss: 344.950806     ETA: +2172.965434s
```



1.3

a) The sensors within Carla are plentiful. Carla includes RGB sensors, IMU, GNSS, depth, collision, and about a dozen more. For our personal network, we incorporated the RGB, collision detection, lane change detection, and object detection sensors. Including any other sensor would assist the network in its learning model, however it would slow down the process significantly. Adding more data and elements to analyze adds complexity and thus longer run time.

b) Multiclass prediction

c) Since classification is tokenizing/labeling certain values, whereas regression is a continuous change, I believe for our model either approach would work, but it depends entirely on circumstances. For example, we have simple outputs of "left, right, gas, brake" which are all very discrete and singular outputs. However there are also situations where we would need continuous changing output, like partially steering or partially stopping. Regression models/networks would have the advantage of flexibility for more complex situations, however that complexity is exactly the disadvantage, where having more data could become too much to parse through. For our approach, it would be reasonable to set this up if we needed even more extensive training or for different reasons, for simple imitation, classification might be enough

d) Synthetically augmenting data would be extremely helpful to the network because it adds "useful noise". Having to deal with random situations such as an all white

image with no action inputs would train the model on what to do during simulation errors for instance. Another way to train the model would be to purposefully give it different observations (such as a different vehicle being controlled) with useful action data so that it can learn different ego's as well.

e) Concatenating previous frames adds more information to dynamic changes in the environment for the model. However just as before, this adds complexity and may not always yield the best information for the model (overfitting).

f) To add more tricks for performance, we could also incorporate attention/weighted values for our data. While this would be incredibly complicated to incorporate, our model would have a better way of prioritizing outputs/correct outputs. We could also incorporate Dr. Ohn-Bar's Learning by Watching techniques which according to his data, significantly improved performance.

1.4

Theorem 1.1 *Let* $\mathbb{E}_{s\sim d_{\pi^*}}[\ell(s,\pi)] = c$, *then* $J(\pi) \leq J(\pi^*) + T^2 c$.

a) Theorem 1.1 from what I can understand is the difference between the expert policy and the learned policy given a J(π*) and a learned J(π) respectively. The epsilon is the epoch and T the overall time. With the loss function defined by L(s, π), it follows the optimal/expert policy based on 1 – epsilon of the states. The second half of theorem 1.1 denotes the quadratic compounding effect of the T^2 variable, which calculates the error that becomes exponentially larger with each wrong move taken. Therefore, to minimize the loss function, and maximize performance of the model, we need a small enough epsilon increment, so that the policy J(π) can follow the expert model closely.

$$J(\pi) \leq J(\pi^*) + T^2 c.$$

To have DAGGER converge faster, we can provide the model an initial starting point so that it mimics the expert policy closer instead of starting on its own from scratch. Our project with imitation learning is a perfect way to enter the DAGGER policy because this way it already has a good starting location. However, DAGGER and Behavior cloning are extremely similar if the learned policy/state is similar to the expert state. Since DAGGER uses corrections based on expert policy, if the model is trained entirely by expert policy without any outliers or different error type scenarios, the behavior cloning will look just like DAGGER. There absolutely ARE common urban driving scenarios that lead to quadratic cost. The most obvious one are

complicated interchanges or intersections. Once a policy initiates an intersection improperly, the errors compound extremely quickly, and catastrophic errors are imminent. Another example are roundabouts, where teaching a policy how to react on a roundabout when human drivers don't properly know how to will immediately cause problems.

b) Our attempt at implementing DAGGER was not entirely successful because we ran into some issues. While we were able to follow the general outline for initializing it and copying the algorithm into python, we ran into the issue when getting an expert policy. We could not provide it with an expert policy since our data collection was done through CARLA's autopilot, which is already considered an expert. We were unable to record our own driving because of the way we have our data collection code set up, which focuses entirely on CARLAs autopilot as opposed to a human driver. This is where our code from DAGGER breaks down. Providing it the same input as our dataset would not change performance at all.

   The scenarios that would benefit the most form DAGGER would be if we had actually driven the CARLA vehicle ourselves and recorded the dataset, and then used CARLAs autopilot as the EXPERT in this case since it would most likely drive much better than a human. If we were to plot the regret, we would see a significant jump in performance immediately since we provide an expert policy with a head-start (the imitation learning part). While this would be slower overall because we would need to correct it with the expert policy, the regret would be lower and lower quicker.

1.5

a) Code initialized in setup and then model should run under run_step.