Deep Learning Lab Report

Imitation and Reinforcement Learning

Mohammadhossein Bahadorani
https://github.com/bigus-dl
4558449

Imitation Learning

Task 1,2) To do this exercise, I recorded 50,000 samples using the provided script. Considering the large amount of data, I was facing memory issues since all the samples are loaded at the same time. Looking at the code I found out that the input images are being loaded as ‘float32’ but they’re ‘unit8’. This reduced my memory consumption but still would occupy ~5GBs. This also meant the PC I was training on was a lot more likely to be restarted by someone since nothing works (beside my script).

I wrote my own DataLoader class. This made preprocessing and adding more channels much easier. I also used tensorboardX which is more straightforward to use and also allows asynchronous writes.

Collecting this data took a long time which also meant I got tired of playing the game and kept making mistakes. This complicated the idea of training and validation, especially since I’m using the last 10% of the collected data to validate i.e. what I recorded when I was most tired.

I assume overfitting is not that big of an issue in imitation learning. After all, the agent does have to imitate the expert. This is truer in my case where I actually don’t want the agent to imitate my validation set. There was one episode where I drove too far out and didn’t want the episode to count anymore, so I drove off the edge to end the episode. This encouraged many of my agents to do the same, every time they got a little too far from the road. Nice.

Task 3) I first trained an agent using a CNN with 3 conv. layers, followed by 3 linear layers. During experiments I also added dropout for some regularization. I also tried a smaller network with 2 conv. layers and 2 linear layers. I mostly used a mixture of these parameters :

Adam optimizer, Cross entropy loss
Batch size : 16, 32, 64
Learning rate : 1e-3, 1e-4
# epochs : 60, 100, 200
Activation function : relu

Large Network :
conv2d (input – 64, kernel size 5, stride 1, padding 1)
maxpool2d(kernel size 2, stride 2)
conv2d (64 – 64, kernel size 5, stride 1, padding 1)
maxpool2d(kernel size 2, stride 2)
conv2d (64 – 32, kernel size 5, stride 1, padding 1)
maxpool2d(kernel size 2, stride 2)
linear(32*11*11, 64)
dropout(0.5
linear(64, 16)
linear(16, 5)

Small Network:
conv2d (input – 64, kernel size 6, stride 2, padding 0)
conv2d (64 – 32, kernel size 4, stride 2, padding 1)

linear(32*22*22, 32)
dropout(0.5)
linear(32, 5)

The trained agents were mostly able to traverse the circuit without much problem. They could recover most of the times, if they slightly got on the grass. They did however as mentioned tend to replicate my mistakes as well. So far, my best agent has gotten a score of 876.947.

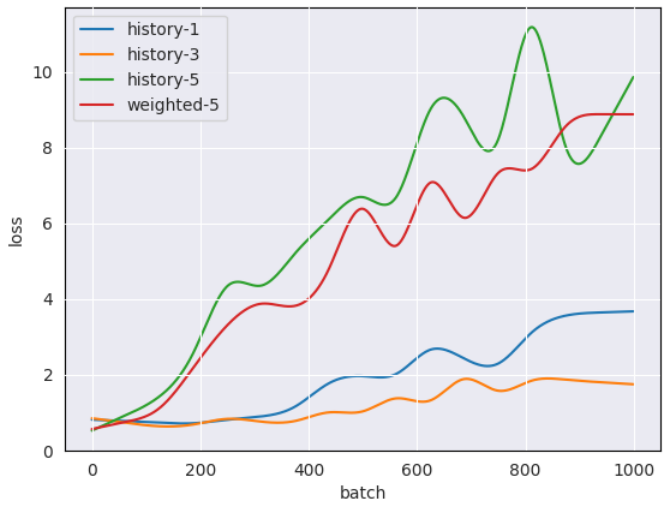
Task 4.1) To combat unbalanced training data, I decided to use a weighted cross entropy loss. To calculate the weights, I got the sample count of each class, got their median and divided the median by the class count:

	Left	*Right*	*Accelerate*	*Brake*	*Straight*
Count	8784	4038	13639	1118	22421
Weight	1	2.17533432	0.64403549	7.8568873	0.39177557

Task 4.2) For this exercise, I experimented with a lot of parameters and got some mixed results, shown in the table below:

History	*Weighted*	*Model*	*Max. Speed*	*Best*	*Worst*	*Mean*	*Std*
		me				**884.65**	
1	yes	small	1	923	557	**872.4**	89
3	yes	large	0.7	921	455	835.3	117
3	no	large	1	919	661	861.7	86
3	yes	large	0.7	926	439	826	147
5	no	small	1	922	731	**876.9**	43
5	yes	small	0.7	913	317	789.2	194
5	yes	small	1	922	531	802	134
5	no	large	0.7	916	520	856.9	109

I also tried some models with history=10 but didn't get good results. I think the reason there's a sweet spot when it comes to appending history is that we don't have an indication of when an episode starts in the training data. e.g. for history=10, the first frame of the episode is also affected by the last 9 frames from last episode.
There were a lot more parameters I could've experimented with but decided to move on to the reinforcement learning task instead.



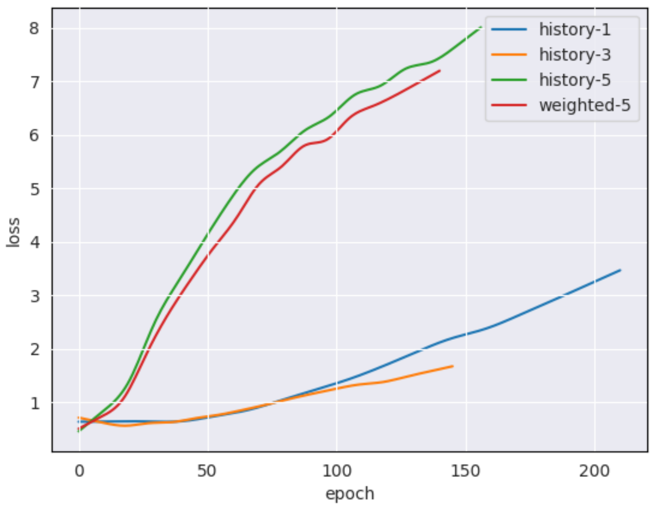


Figure 2 validation loss per batch

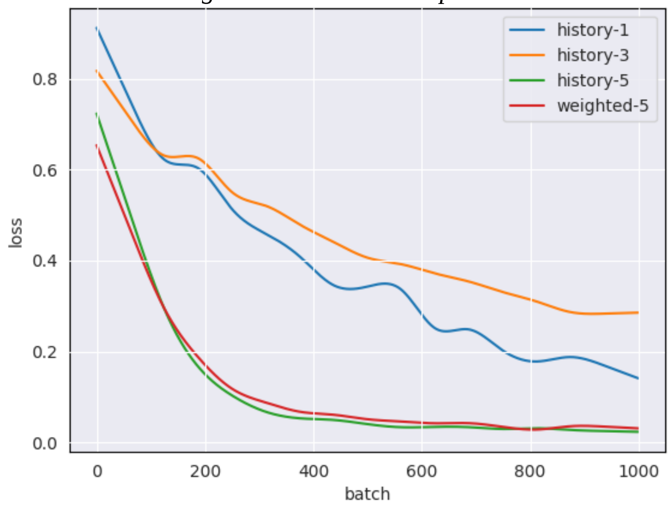


Figure 3 training loss per batch

Figure 1 validation loss per epoch

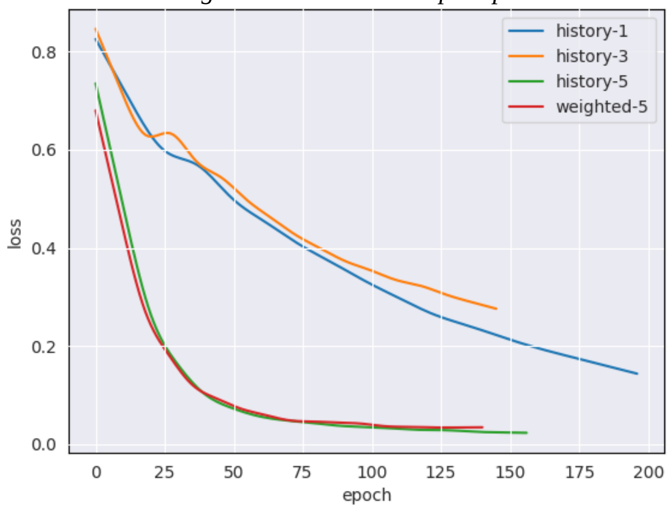


Figure 4 training loss per epoch

Reinforcement Task

Task 1

I completed the stubs and managed to get results for this task relatively easily. I tried both soft and hard updates for the target network. I used the following hyperparameters:

Hidden layer dimension: 300
Gamma: 0.95
Batch size: 64
Epsilon: 0.1
Tau: 0.01
Learning rate: 1e-4
Burn in: 256
Buffer capacity: 1e4

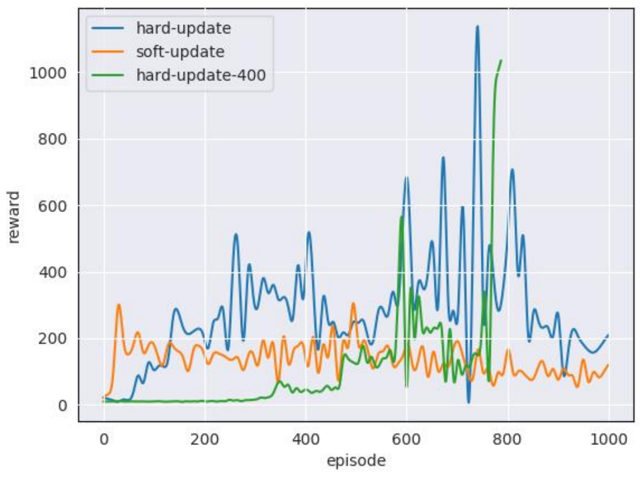


Figure 5 reward per episode

I added a burn in phase where the networks aren't trained, and the agent only explores, where it has a 50-50 chance of going left or right. I managed to get the optimum policy by using early stopping and hard updating the target network. The agent achieves maximum reward 1002 every episode (0 standard deviation).

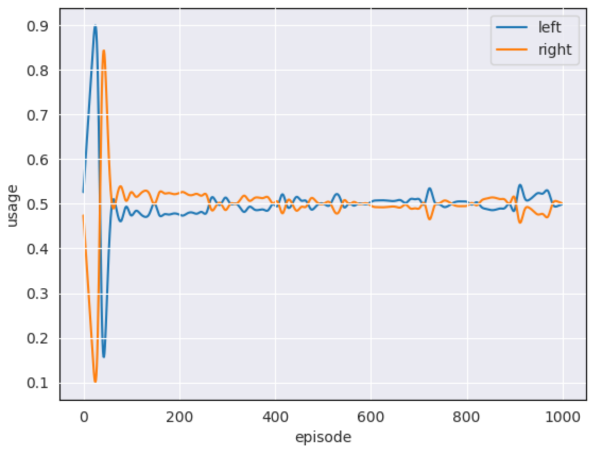


Figure 6 action distribution

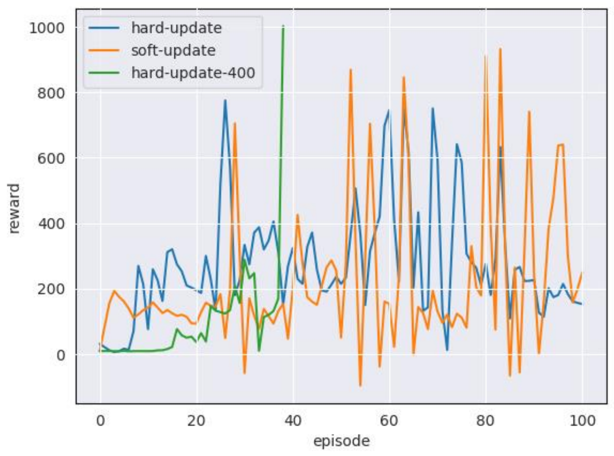


Figure 7 mean evaluation reward

The Q networks reach a stationary point when the left and right action distribution is split 50-50. I also tried a larger hidden layer but didn't get any good results.

Task 2

For this task I had to play around with different CNN sizes to get some results. The hard update produces faster results, but they were also instable. I ended up using the following hyperparameters:

History: 5
Gamma: 0.95
Batch size: 16
Tau: 0.01
Learning rate: 5e-4
Burn-in:16
Soft update
Network: :
conv2d(4 – 16, kernel size = 6, stride = 2)
batch norm(16)
conv2d(16 – 32 , kernel size = 6, stride = 2)
batch norm(32)
linear (21x21x32 – 256)
linear (256 – 5)

For exploration I used the same percentages I had from my imitation learning dataset. I changed them a bit to make it more balanced. The network was still very unstable, so I had to take some measures to stabilize training

Forcing Actions

I manually changed the action to acceleration for the first 10 steps of the episode. This pushed the agent to learn this behavior without having to figure it out. I did this after I noticed some of the evaluations had a constant negative mean. Setting rendering to true I saw that the agent was choosing actions other than accelerate for that state. This was biasing the replay buffer by filling it with the same. I was also planning to do something about breaking before corners, but I didn't have enough time to figure out how.

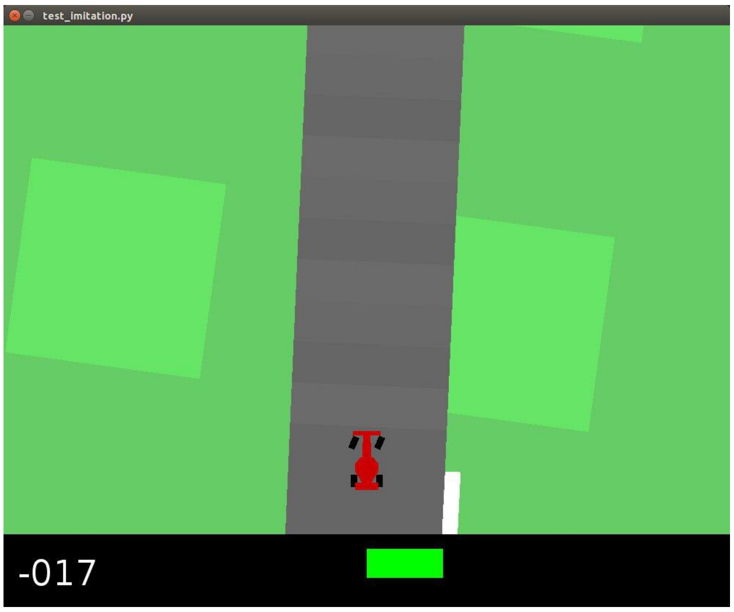


Figure 8 agent choosing right rather than accelerate

Epsilon Scheduling

Adding a burn-in phase was helping in the initial training stages but I needed something more long term and less deterministic. I added an epsilon scheduler that takes an upper bound and a lower bound and the number of episodes to run the scheduling. I used cosine annealing to reduce the epsilon value.
Epsilon upper bound: 1
Epsilon lower bound: 0.1
Episode count: 80

Skip Frames

Skipping frames helped the agent learn to perform an action repeatedly. This significantly improved training but I think was causing an issue for validation. I also 'scheduled' the number of frames the environment skips based on the current episode number.

Maximum Steps

I also scheduled the maximum number of steps the agent was allowed to train. I think this stopped early runs where the agent couldn't really increase its reward anyway and would just record a long episode on the grass, filling the replay buffer with useless samples.

Double Q Learning

I didn't really know about this algorithm because I haven't taken the RL course yet. Another classmate suggested it's a rather easy way to stabilize training and he was right.

Results

The agent managed to get an evaluation score of 843 ± 49 over 15 episodes. The agent still struggles with sharp corners which usually lead to it either getting lost on the grass or drifting, spinning and eventually going the wrong way.

I only started getting decent results yesterday, so I didn’t have time to evaluate all the agents. I only tested the best ones from the last run.

During one evaluation episode the agent got on the grass and started following its own tracks. I assume it had learned that going back on its own skid marks could lead to the road. Unfortunately if the agent makes too many skid marks, the earlier ones disappear so it started going in a circle on its old tracks.

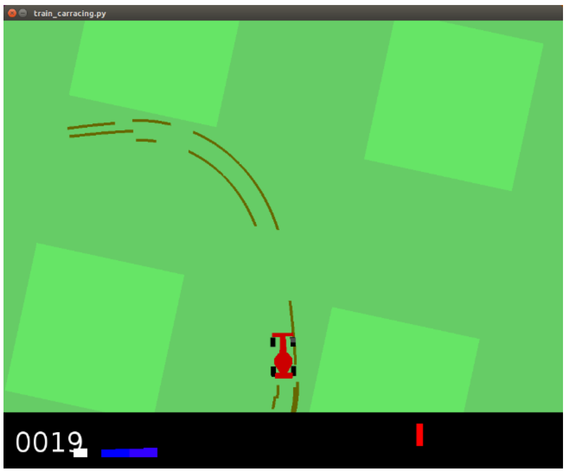


Figure 9 going in circles

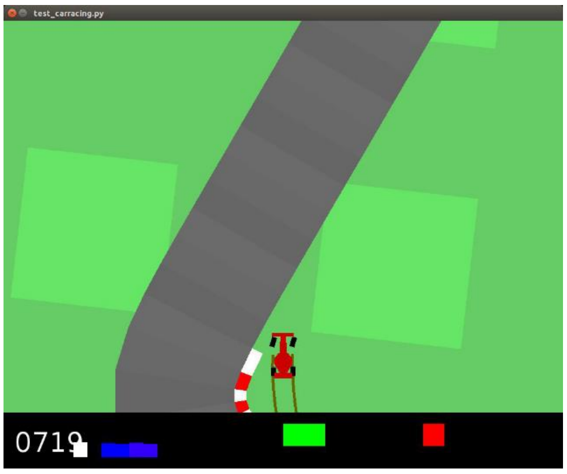


Figure 10 cutting corners

The agent also started cutting some corners. I don’t know why but maybe it learned missing some rewards is better than spinning out on the grass.

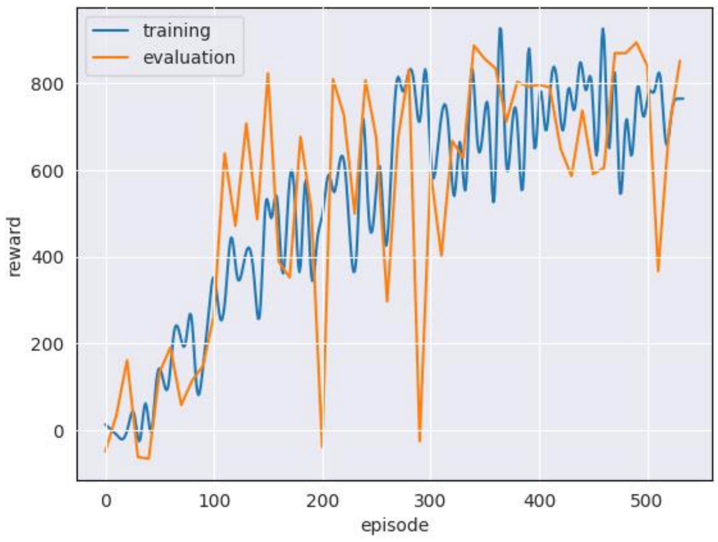


Figure 11 rewards