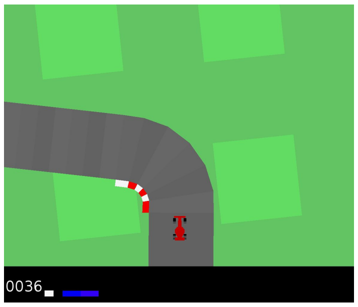
DL-LAB Exercise 3 R_NR Solution

Name: Shashank Agnihotri   Matriculation Number: 4775315
email: agnihotr@tf.uni-freiburg.de
shashanksagnhotri@gmail.com
Github repository: www.github.com/shashankskagnihotri/DL-LAB

Exercise 3.1:

Setting Up:

Setting up the experiment was simple. The time taking part was going through the documentation of tensorflow. Though it has been used before, yet, this experiment had extensive use of tensorflow and understanding most of its functions and classes was time taking but at the same time very interesting and enlightening.



Exercise 3.2:

Behavioral Cloning:

Running the car drive_manually.py was quite difficult.
I first started with collecting data for just 10000 steps, but while training they seemed very less, and so I then collected upto 20,000 steps, then 25,000 upto 70,000 steps, training the agent on all of them and reviewing the performance of test_agent.py each time.

Some of the results when I acted as the expert were:

mean_all_episodes	**std_all_episodes**
460.09999999990924	215.57351414311077
737.333333333294	44.25534493771355
750.8999999999619	34.978898400801796
751.4142857142473	32.40865971449659

And more…

Getting familiar with Gym also took some time, but was simple and fun.

Preprocessing was difficult due to limitation of memory, and thus I divided the preprocessing data also in mini batches, that is, I preprocessed data after dividing the entire data into mini batches and then preprocessed these mini batches.

Handling Imbalance:

To handle the imbalance, I tried various approaches. Some of them are:

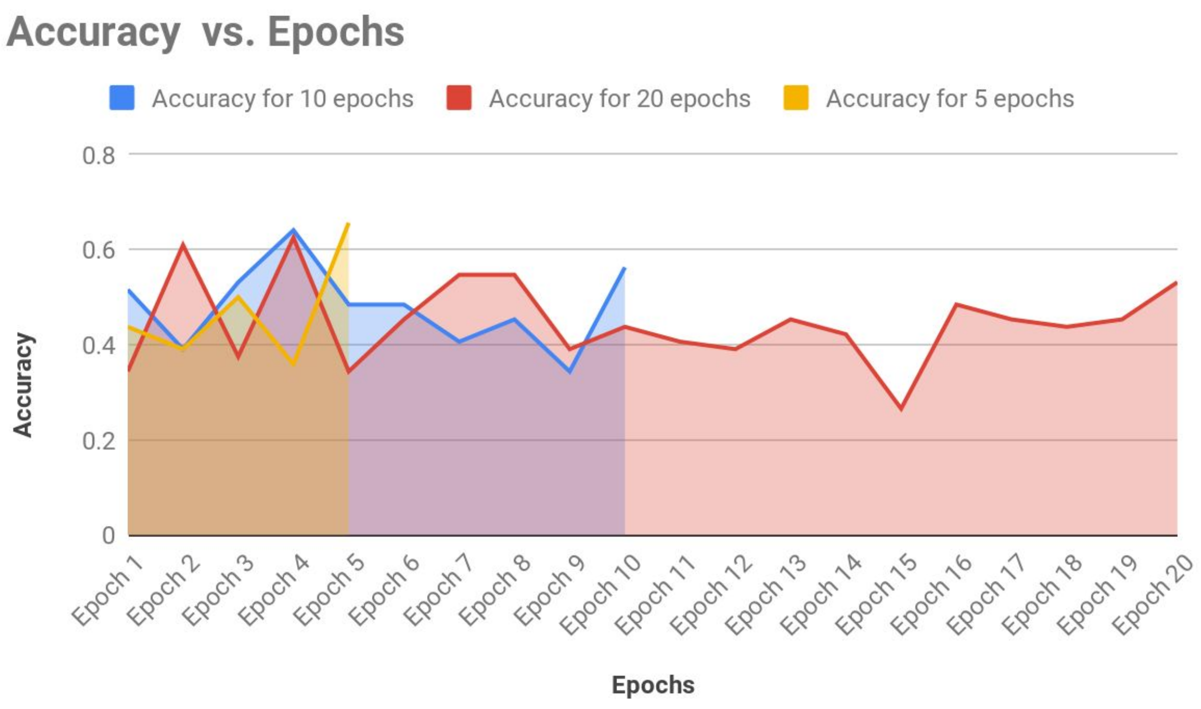
- Forcefully converting all the STRAIGHT actions to ACCELERATE:
    - This one was not very helpful, as in test_agent, the car was just accelerating, without taking any turns until it reached the green area and then it was just revolving in a circle.
- Forcefully converting some of the STRAIGHT actions to ACCELERATE:
    - This had two modes, either the car was not moving at all as it was only predicting STRAIGHT, or on increasing the number of STRAIGHT actions converted to ACCELERATE the car was just accelerating.
- Forcefully converting some STRAIGHT actions to some other action randomly:
    - I finally went with this approach as in this the car was taking actions other than just STRAIGHT and ACCELERATE.

Hyperparameter Optimisation:

I tried optimising the learning rate, number of epochs and history length.
When the learning rate was very low, like 0.0001, the agent was not learning anything and the training accuracy was really low, as low as 5%-10%
On increasing the learning rate till around 0.01 the training accuracy increased, after which it was either invariant or started decreasing.

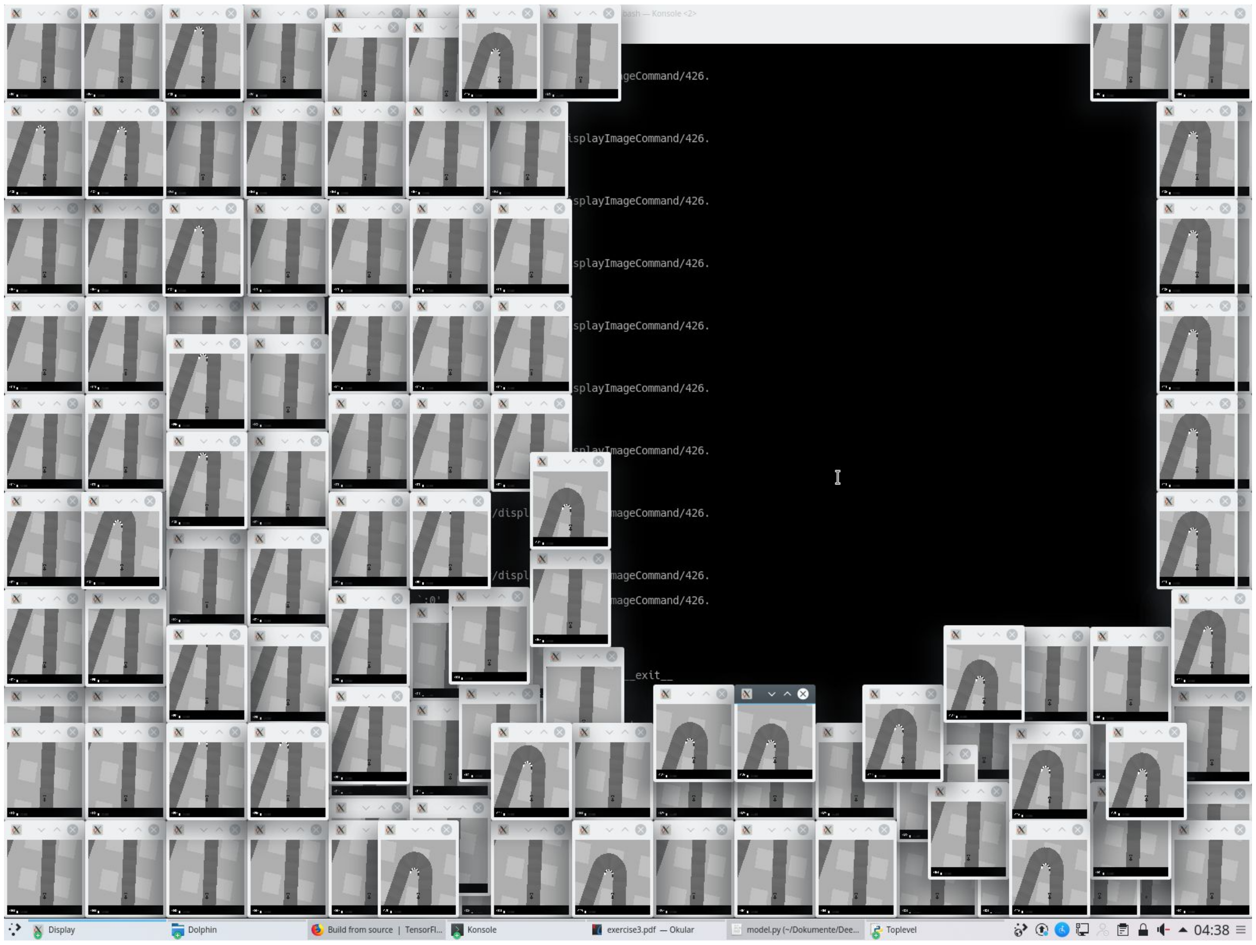
Comparison over Number of epochs:

The agent on training seems to seems to lose accuracy as the number of epochs are increased, and highest accuracy was found for 5 epochs and that was set as default number of epochs for various history lengths.



Test Agent :

The following are the states as seen by the agent in the first episode. Seeing these states gave me an idea of how exactly the agent was receiving a state and how the next state looked like for the agent.

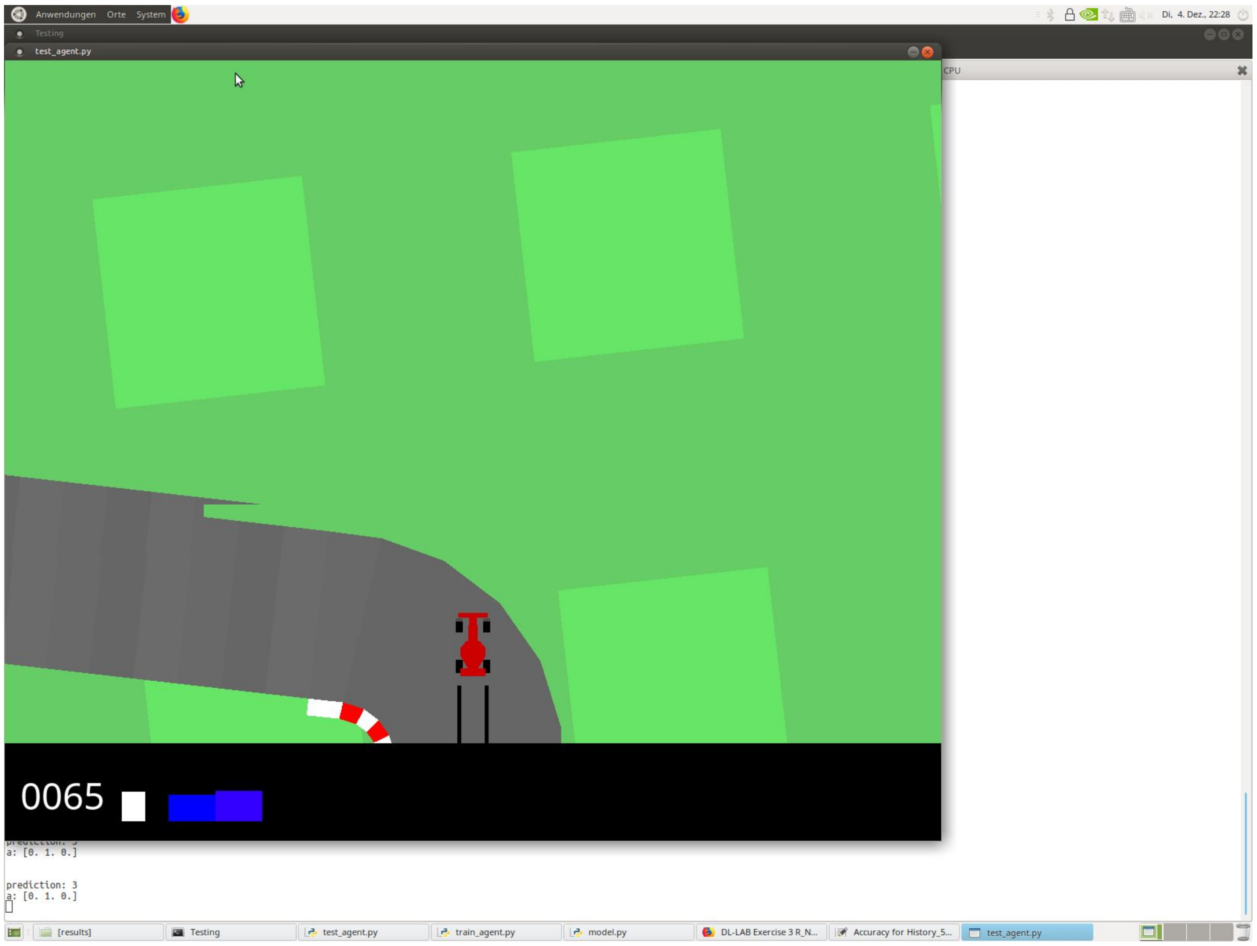


```
ageCommand/426.
isplayImageCommand/426.
splayImageCommand/426.
splayImageCommand/426.
splayImageCommand/426.
splayImageCommand/426.
splayImageCommand/426.
/displ          ageCommand/426.
/displ          ageCommand/426.
                ageCommand/426.
__exit__
```

One of the problems faced in test_agent was the agent predicting STRAIGHT as the action, as always the first state was a straight track and thus to overcome this issue I forced the first few actions of the agent to be ACCELERATE everytime a new episode started.

But after further optimizing the code, making changes to functions in utils.py, I removed this forced ACCELERATE action, as the agent was making correct predictions even without it.

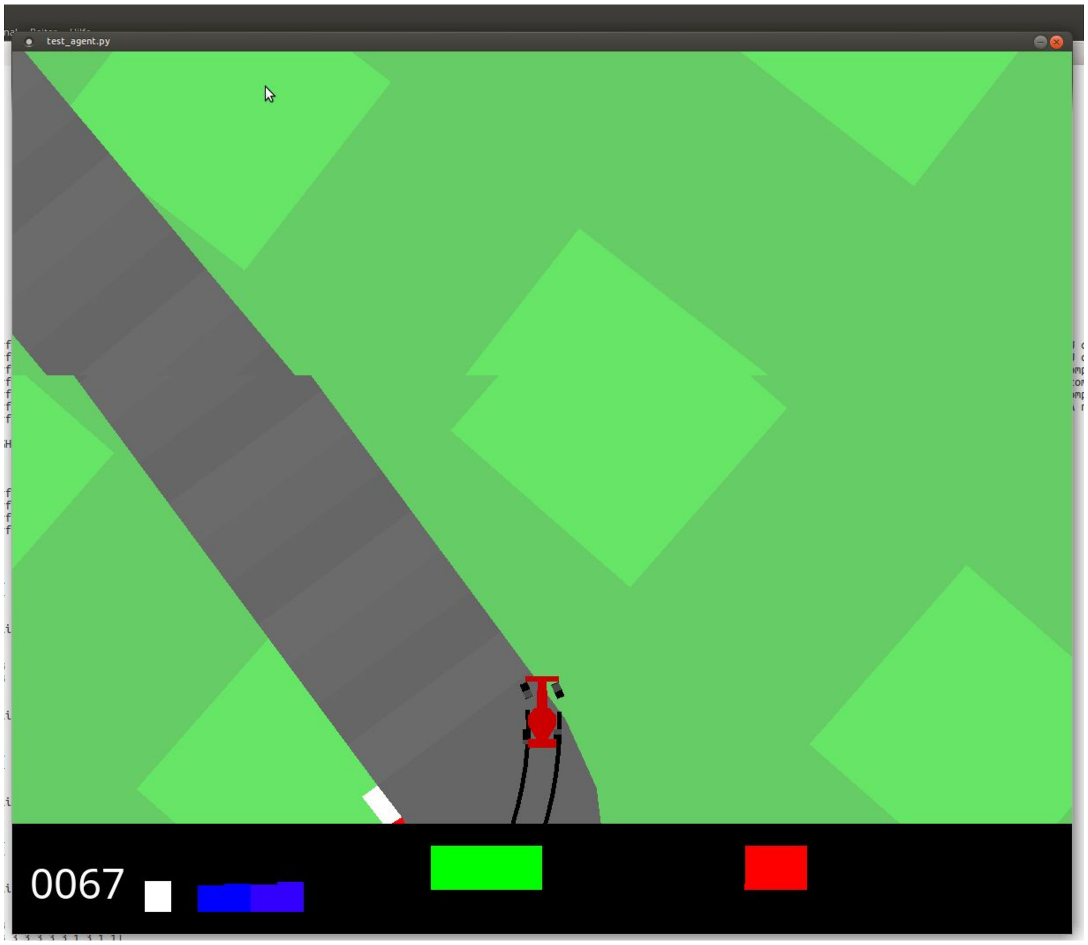
The following is a snapshot of the car moving on the track, but due to bad prediction, not turning at the turn.



```
prediction: 3
a: [0. 1. 0.]

prediction: 3
a: [0. 1. 0.]
```

The following is a snapshot of good prediction making the car to turn:

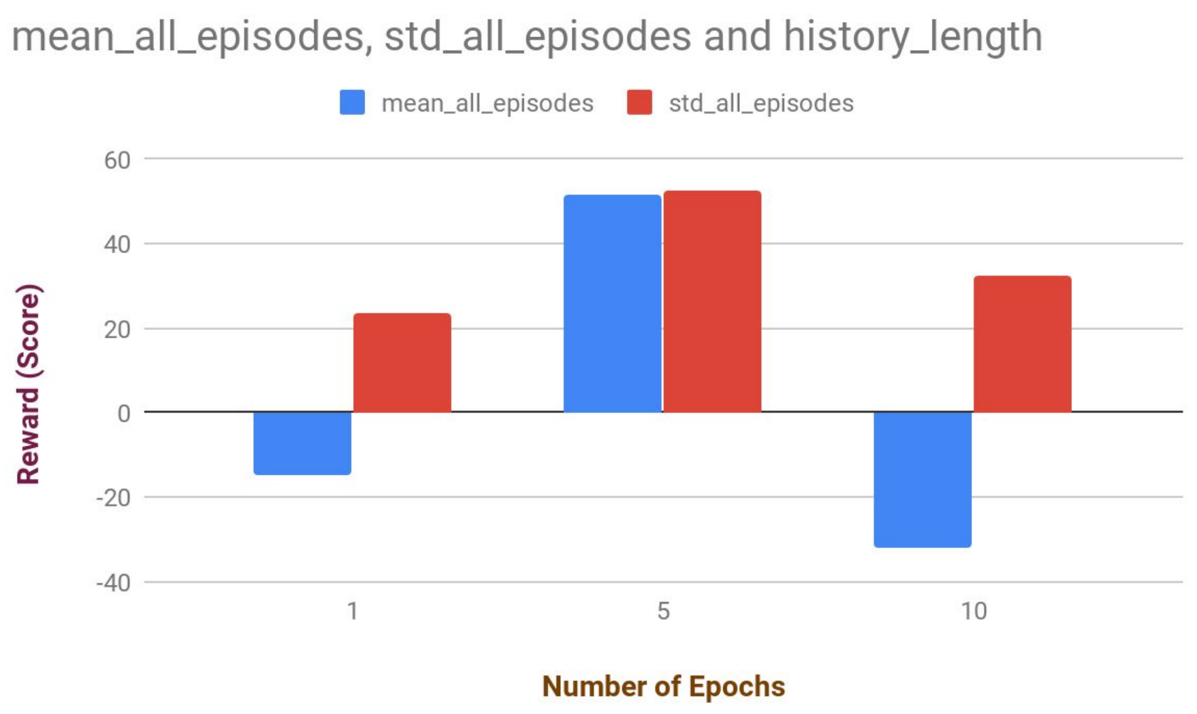


Analysing Effect of Number of epochs on final performance:

Comparison over 10 episodes for number of epochs:

mean_all_episodes	std_all_episodes	history_length	Number of Epochs
-14.49499622933225	23.80415259266346	1	1
51.37002838790922	52.354978094591054	1	5
-31.92413443114764	32.5847650675919	1	10

These results can be better analysed using the bar graph below:



So, the results obtained in test_agent reflects the observations made in the training. We see good predictions when number of epochs are increased to ‘5’ and the performance drastically decreases when the number of epochs are increased to ‘10’.

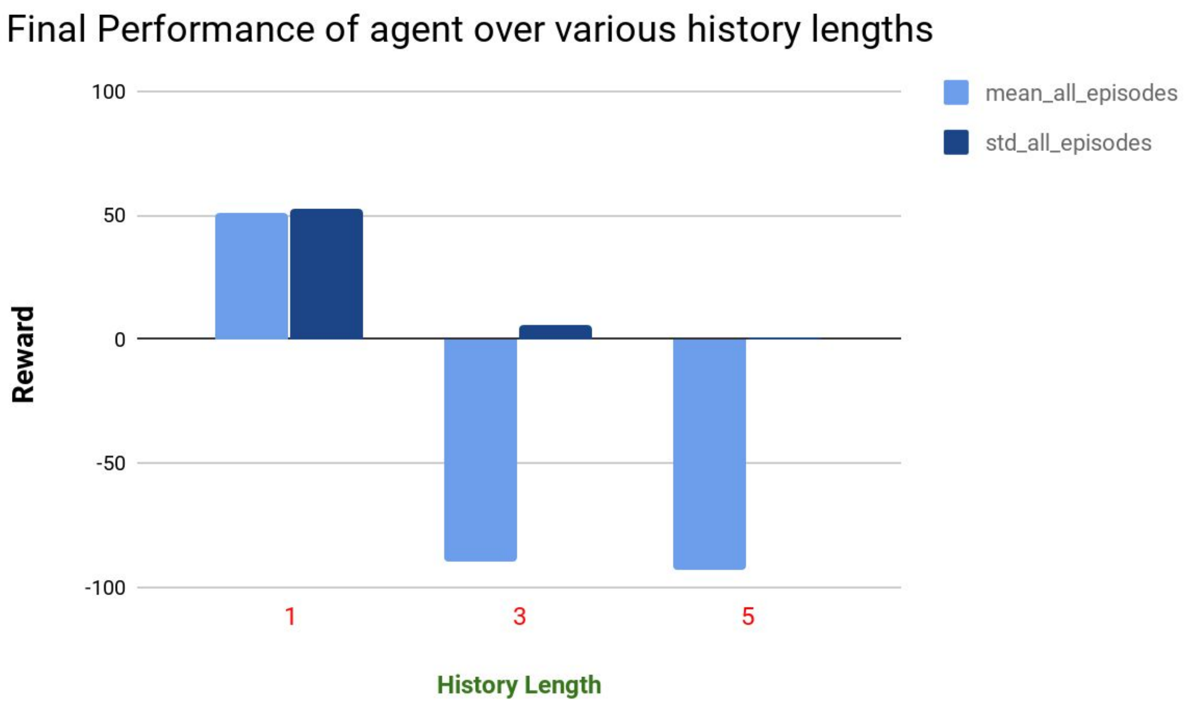
Increasing the history length:

Comparison over 10 episodes for number of epochs = 5 :

mean_all_episodes	std_all_episodes	history_length
51.37002838790922	52.354978094591054	1
-89.80153887533454	5.55639260614373	3
-93.29133235413941	0.48919847286196694	5

On increasing the number of N or the history length from ‘1’ to ‘3’, we observe that there is a drastic drop in the performance of the agent. This trend seems to continue when the history length is increased to ‘5’.

This can be better understood by the bar graph below:



Exercise 3.3 :

I tried to implement LSTM out of curiosity to see if the model trains better with LSTM layers, I initially added one layer which can be seen in the commented section in model.py. I had the idea of adding the LSTM layer at the end of all the pools and fully connected layers, so that LSTM works not superficially but in the core.
Unfortunately, the network was getting too complex for me to understand and I decided to drop that approach.

Exercise 3.4 :

I was simply too late to contest in the competition.

References:

1. https://pythonprogramming.net/cnn-tensorflow-convolutional-nerual-network-machine-learning-tutorial/
2. https://www.tensorflow.org/
3. https://www.tensorflow.org/api_docs/python/tf/nn/rnn_cell/LSTMCell
4. And ofcourse the very helpful: https://stackoverflow.com/