RESEARCH REPORT - Spring 2019

Aarati Noronha
SafeAI Lab, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15217
anoronha@andrew.cmu.edu

Abstract

This report provides a concise summary of the research conducted by Aarati Noronha during the Spring 2019 semester at Carnegie Mellon University. The research was divided into two sections; each of which are explained by a separate section of the paper. The first section focuses on using reinforcement techniques for predicting interactions between cyclists and pedestrians for autonomous vehicle path planning. The second section delineates the progress made with regards to data collection and management.

1. Navigating intersections through reinforcement learning

1.1. Introduction

Twenty percent of all accidents occur at intersections for human drivers. Intersections have been selected as a problematic zone for autonomous vehicle navigation. Particularly, interactions with pedestrians and cyclists have been sparsely documented. Primarily, there are three different ways of handling behaviour at intersections: online learning, offline learning and imitation learning.

Imitation learning techniques [1] such as conditional imitation learning or guided cost learning might work well when encountered scenarios that have already been taught through past trajectories. However, agents are unable to handle exceptions which is the case at most intersections. Online learning requires the generation of an accurate generative model. Learning generative models is a very hard task and existing methods often fail when it comes to visual quality, temporal coherence and bridging the gap between real world data complexity and synthetic data sets [6].

This report therefore focuses on an offline learning method - reinforcement learning for navigation. As Varaiya suggested [7], planning for intelligent driving is divided into four hierarchical classes: route planning, path planning, manoeuvring planning and trajectory planning. Route planning is out of the scope of this report. Instead it focuses on the latter three.

The research demonstrates results from two different training algorithms: deep Q learning and maximum entropy inverse reinforcement learning. The first one works well in small state spaces but takes an infinitely large amount of time in real world settings and can scarcely account for limited perception of the ego agent. The second one is a novel idea that is sometimes overtly cautious and computationally more expensive. But it relates better to scenarios at intersections and is ultimately more feasible.

1.2. Literature Review

David Isele et.al [4] deployed Deep Q-Networks for the specific problem of handling intersections. It surpassed a number of heuristic approaches when it came to identifying occlusion scenarios and was better able to generalize to novel domains. However, it does occasionally result in collisions which are very expensive in the real world.

In his paper in 2016, D. Zhao et al. [8] introduced an on-policy reinforcement learning method. SARSA learning combined with deep learning to solve more complicated control problems, and as a result it outperforms deep Q-learning methods in convergence rate. The key algorithm is for every update process, SARSA learning takes the next action a' for updating the current state-action value, and the quintuple (s, a, r, s', a') will be derived in sequence. I have based my approach on the latter method when implementing the deep SARSA agent.

B. Ziebart et al. [9] proposed Maximum Entropy Inverse Reinforcement Learning that resolves ambiguities by choosing distributions that have no additional preferences beyond matching feature expectations. In locally normalizing probabilistic IRL models, bias exists that paths with more branches will be assigned lower probability mass. However, maximum entropy IRL avoids this bias by assigning probability according to expected reward, higher probability for higher reward behavior. This model has been applied to model the real-world navigation and driving behaviors.

In 2010 Henry, Peter, et al. [3] use inverse reinforcement

learning (IRL) to enable robots to navigate through crowded environments with dynamics and partially observed features. The results is an imitated human pedestrian behavior given an environment. It is an extension of maximum entropy inverse reinforcement learning with more difficult scenarios. They use a crowd motion simulator that generate realistic motion patterns to evaluate their work. The two approaches mentioned above are used in combination with mean models of features generated through Gaussian regression to enable better performance of the maximum entropy agent.

1.3. Dataset deployed

The dataset in question is the Stanford drone dataset [5]. It consists of eight unique scenes and six different types of agents. The number of videos in each scene and the percentage of each agent in each scene is reported in 1. An excellent feature of the dataset is that it has a large percentage of pedestrians and cyclists.

This project has focused its learning algorithms on the roundabout instance called 'deathCircle' on the Stanford campus. The ground truth of the dataset is shown in 2. The 'deathCircle' spans over 200m in both directions in a horizontal plane. Training data includes 1307 cyclists, 831 pedestrians and 109 cars from 5 videos. Other agents are ignored.

Scenes	Videos	Bicyclist	Pedestrian	Skateboarder	Cart	Car	Bus
gates	9	51.94	43.36	2.55	0.29	1.08	0.78
little	4	56.04	42.46	0.67	0	0.17	0.67
nexus	12	4.22	64.02	0.60	0.40	29.51	1.25
coupa	4	18.89	80.61	0.17	0.17	0.17	0
bookstore	7	32.89	63.94	1.63	0.34	0.83	0.37
deathCircle	5	56.30	33.13	2.33	3.10	4.71	0.42
quad	4	12.50	87.50	0	0	0	0
hyang	15	27.68	70.01	1.29	0.43	0.50	0.09

Figure 1: Stanford drone dataset statistics

1.4. Deep Q Learning agent (DQN)agent

1.4.1 Designing the basic algorithm

The neural network in question has 3 layers. It is designed to input an array of 4 values containing the state of the system. The state of the system is basically the feed received from sensors at 4 points of the agent i.e. at the front, rear, left and right. The neural network has 2 fully connected hidden layers each consisting of 24 nodes with activation of the form relu. The output layer has 9 nodes each representing a different action which are all defined within the reward function. The action space includes 9 actions: forward, back, left, right, forward and left, forward and right, back and left, back and right. The training process involves feeding input and output pairs into the neural network in order to understand and predict rewards based on the data received from the sensors. After the training process is completed, the model is able to effectively predict the reward of the current state. The algorithm was executed for 300 episodes and the gist is delineated in figures 3 and 4.

1.4.2 Computing the loss function for the network

The loss function is of the mean square error form and Adam optimizer is utilized. Consider the agent in a given state s. It can perform action a, receive reward r and move to the next state s. The value function as a result of performing a given action is denoted by $Q(s, a)$. The Q-value of all possible actions for a given state are considered and the maximum is selected. It is then multiplied by a discount factor γ and added to the reward in the current state to get the target value. The loss function as shown in equation 1 is computed as the difference between the target value and the Q-value predicted by the model.

$$Loss = (r + \gamma Q(s, a') - Q(s, a))^2 \tag{1}$$

1.4.3 Computing the reward function and deciding an action

A parameter called exploration rate is defined. In the initial stages of training, when the value of the exploration rate is high, the agent in a given state selects actions randomly based on this parameter. Once it learns through several interactions with the environment, it picks the action with the largest Q-value. The exploration rate is updated after every 50 episodes and becomes 99.5% of the previously exploited exploration rate value. This way, the agent gets to explore different sets of states and rewards and hence learn the surrounding environment in a better and more complete way. The reward is basically is a function of the aggressiveness or passiveness of approach of the agents in the vicinity. In other words, $\Delta d/\Delta v$ is taken into consideration. The reward is delineated in 2. Success is defined if the goal or target position is attained is attained. Collision is defined if the agent is within 0.5m of another agent.

$$r = \begin{cases} 100, & \text{if } success \\ -100 & \text{if } collision \\ \sum_{i=1,..4} 5\frac{\Delta d_i}{\Delta v_i} & \text{otherwise} \end{cases} \tag{2}$$

1.4.4 Introducing experience replay

Neural networks do not have a memory and they tend to replace older experiences with newer ones. Every action

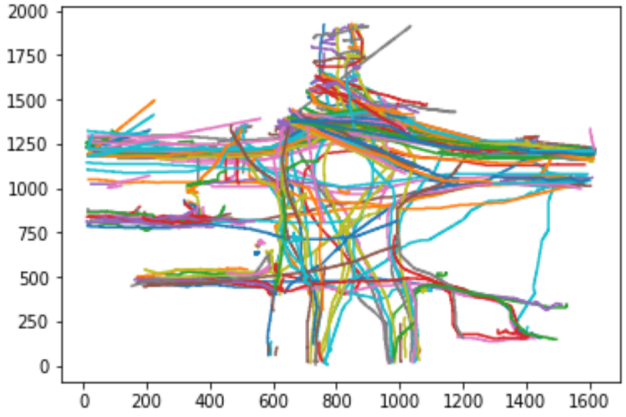

Figure 2: Ground truth over death circle

Algorithm 1: Deep Q-learning with Experience Sampling
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, **do**
Initialize state s
  for step = 1, **do**
  with probability = Ɛ
    select a random action a
  otherwise
    select $a = max_a$ Q(s, a′)
  Execute action a, observe reward r and state s'
  Set s′= s
  Store transition s, a, r, s' in D
  Sample random minibatch of transitions $s, a, r, s'$from D
  Compute reward from equation 2
  Compute loss from equation, perform a gradient descent step
  end for
end for

Figure 3: Learning algorithm for deep Q learning with experience replay

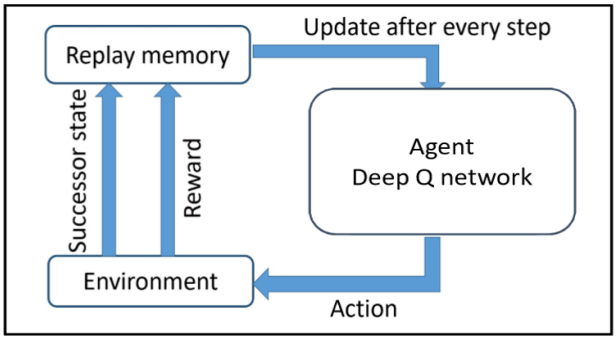

Figure 4: Gist of Deep Q learning

affects the next state. This outputs a sequence of experience tuples which can be highly correlated. If the network is trained in sequential order, the agent is at risk of being influenced by the effect of this correlation. This does not favor optimum performance during training. This algorithm allocates a certain amount of memory to store experiences and observations of the past. At specific instances during training of the model, a batch of experiences is sampled and fed into the neural network. This ensures greater efficacy of the agent in the long run. Correlation is broken and the action values do not oscillate or diverge catastrophically. In training simulations, the batch size was 64.

1.5. Maximum entropy inverse reinforcement learning agent

1.5.1 Highlights of Original framework

The original maximum entropy framework assumes trajectories in the existing data set are reasonable and uses them as input demonstrations in order to teach an ego agent ideal behaviour maneuvers. It generates a mean reward at the end of training and a policy is calculated on the basis of this reward. Another assumption of the conventional framework is that it assumes that the ego agent has complete knowledge of the features of the entire environment. Also, the features in the state space are constant for a given trajectory and across different trajectories.

This model implies that a path or trajectory τ through states s_i and actions a_{ij} has a cost which is a linear combi-

nation of features parameterized by θ as shown:

$$cost(\tau) = \sum_{a_{i,j} \in \tau} \theta \cdot f_{a_{i,j}} \tag{3}$$

For a fixed start and end point, this results in a maximum entropy distribution over paths shown in equation 4.

$$P(\tau|\theta) = \frac{1}{Z(\theta)} e^{\sum_{a_{ij} \in \tau} -\theta \cdot f_{a_{i,j}}} \tag{4}$$

1.5.2 Modified Learning algorithm

This scenario differs from the original algorithm because the features are not constant. They vary within a path and also between paths. Secondly, the ego agent has limited perceptual range of his surrounding. He has knowledge of the states around him to the extent allowed by his sensory system in the case of a pedestrian or sensors as in the case of a car.

In this case we update the features in the ego agent's perceptive field and perform gradient descent for every time step as shown in equation 5. The other features beyond the scope of the agent's detection are set to mean model values computed in section 1.5.4.

$$\nabla F = \mathbf{f}^t - \sum_{a_{ij} \in H} D_j^t f_{a_{ij}}^t \tag{5}$$

Where D_j is the expected state visitation frequency of all actions entering the state during the given time step. Calculating state visitation frequency using the method in [3] is computationally expensive. It has been modified such that a matrix is formed using values from the density mean model. This matrix is updated with values using equation6 every time a new state is encountered. It is a function of the policy Q_{ij} computed through regular value iteration and the transition probability p_{ij}. It is conditioned on weight vector θ as follows:

$$D_j^t = \sum_{a_{ij} \in H} D_i^t Q_{ij}^t p_{ij} \tag{6}$$

The weights are updated as follows where γ is the discount factor:

$$\theta_{n+1} = \theta_n e^{-\gamma \nabla F} \tag{7}$$

A reward is computed for every time increment H and a policy is calculated on the basis of this reward. The ego agent is made to follow this policy. The algorithm is delineated in figure 5

For a given trajectory, the maximum entropy distribution is as follows where T is the duration for which the trajectory lasts:

$$P(\tau|\theta) = \frac{1}{Z(\theta)} e^{\sum_T \sum_{0<h<H} -\theta \cdot f_{a_{i,j}}} \tag{8}$$

1.5.3 Feature Estimation

The environment which spans over 200m is represented by a grid state space with m states in total. Actions describe transitions between these states. Features are basically functions of all actions entering the given state. In other words, the feature vector for a given state is representative of the neighboring states.

It is assumed that all maneuvers of a virtual agent are planned on the basis of the number of agents in the vicinity as well as the passiveness or aggressiveness of their approach. In such a scenario, time to collision or the ratio of relative distance to relative velocity would have been considered an ideal feature metric. However, in order to learn better rewards, this approach keeps distance static and considers relative velocity and density of neighboring states.

Consider n possible actions into a given state or in other words, n neighboring states. The dynamic features for each state are divided into $3n + 1$ bins. The real valued density is designated n bins and is a whole number representing the number of people in each neighboring state. In addition, the velocity component in the x and y direction are designated n bins each. Relative velocity is computed for each of the neighboring states. The velocity features also successfully capture orientation in this manner. Finally, a single bin has been introduced at the end which is a function of distance from the target position of the virtual agent. Intuitively speaking, a smaller target distance means a smaller weight learned and equivalently a smaller reward for the given state.

1.5.4 Gaussian representation of mean model features

The original feature matrix is constructed based on values from mean density and velocity models. As mentioned earlier, the state space is two-dimensional grid across which the virtual agent travels. The underlying assumption is that the Stanford drone data-set is a record of trajectories that are considered ideal. In other words, zero collisions occurred during the recorded duration. For each time instant in a recorded trajectory, there exists a position (x, y) with a corresponding density and velocity component.

To formulate a density model, this report looks to Gaussian mixture models in combination with the expectation maximization technique. Several gaussian kernels are fitted across positions in the state space as a means of identifying and distinguishing dense and sparse regions. The mean and covariance matrix of each mixture model are arbitrarily initialized, and the log likelihood is computed. Posterior probabilities are calculated, and initial parameters and corresponding log likelihood are re-estimated. The steps are repeated until convergence occurs. The number of Gaussian components allowed to exist is evaluated by minimizing

Input: Ground truth of trajectories from an 84 min video at the death circle intersection
Start point and goal of virtual agent
Compute mean models of density, velocity in x and velocity in y positions at the intersection
Compute initial feature matrix based on mean models - f
While target not reached **do**:
For time = 1, ... t - H, t **do**:
Initialize weight vector θ
While not converged **do**:
Update estimates of locally observable features in original feature matrix - **f**
For actions $a \in t + H$, compute expected state visitation frequency - D_a
Compute the gradient $\nabla F^t = \mathbf{f}^t - \sum D_a f^t$
Update the weights
end while
Compute reward $r = \mathbf{f} \cdot \theta$
Compute policy for virtual agent, perform action with maximum reward
end for
end while

Figure 5: Max entropy inverse reinforcement learning

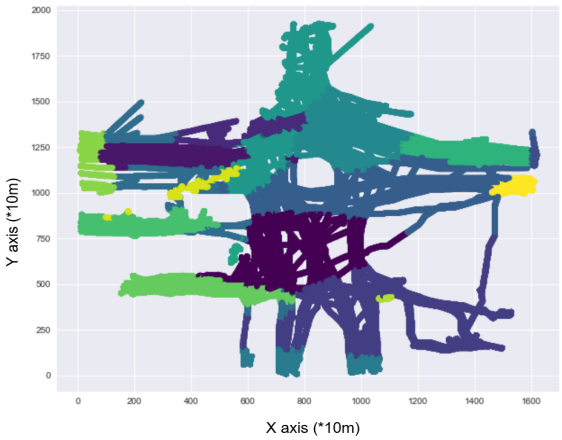

Figure 6: Density model for identifying dense and sparse regions across the state space

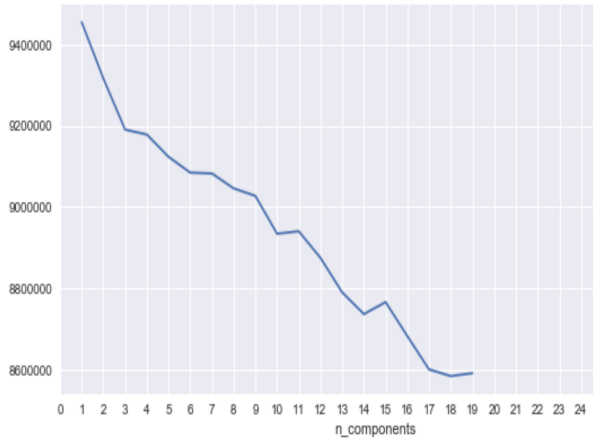

Figure 7: Akaike Information criterion for cluster number selection

information lost per model through the Akaike Information Criteria. This is shown in figure 6 To compute mean density at a particular point (x_1, y_1), the value at (x_1, y_1) in the density model above is divided by the visitation frequency of that state. This value is rounded off to the nearest whole number to get mean density.

To formulate mean velocity models as shown in figures 8 and 9 , velocity from the pre-recorded data at several positions is divided into its x and y components. Separate Gaussian kernels representing mean velocity in x and y directions are learnt. The kernels are a combination of inverse radial bias functions that represent medium irregularities and white noise. The hyperparameters are tuned to maximize negative log likelihood until convergence occurs. Given a set of observations at a given instance, this approach subtracts observed features from mean model features to get a set of deviations from the mean. A new Gaussian kernel is constructed that is a smooth representation of these deviations. The deviated values are then added to the original feature model which represents the updated feature matrix . This method gives extra weightage to the features within the observable range of the agent at a given time instant.

1.6. Experiments

1.6.1 Comparison between the two agents

Deep Q learning (DQN) and max entropy inverse reinforcement learning (MAX-ENT) have been used to train a virtual agent in a simplified environment setting. In this

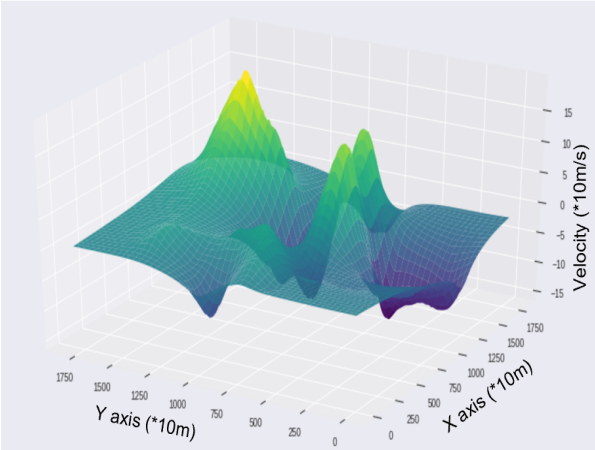

Figure 8: Velocity mean model in X direction across state space: X and Y axis (metres)*10, Z axis is mean velocity (metres*10/sec)

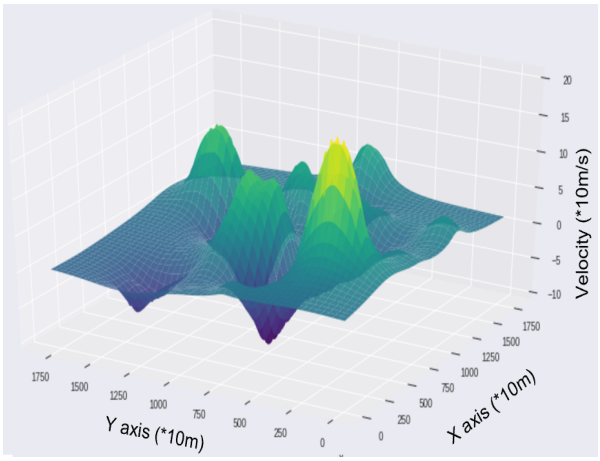

Figure 9: Velocity mean model in Y direction across state space: X and Y axis (metres*10), Z axis is mean velocity (metres*10/sec)

case, the virtual agent is a pedestrian but the approach can be extended to cars and cyclists as well. The other agents in play in the environment are extracted from the ground truth of the Stanford drone dataset. The start and end positions of both virtual agents are fixed. A comparison is performed between the two agents. It is imperative to note that the environment is its most primitive for the experiment at hand.

For the sake of convenient comparison, the state space has been discretized into an 8 by 8 matrix. The action space includes only 5 actions: forward, back, left, right and stay. With regards to the training time, the DQN agent took a large amount of time to train before demonstrating a near optimum path. This means that a DQN agent must be taught its behavior through extensive simulation before being allowed to navigate on the road in real time. Even so, it is unlikely to have encountered every scenario during training and is destined to collide and fail in case of anomalies.

Reward generation is very different for both agents and is shown in figures 10 and fig:rewardIRL. Trajectory followed by each agent is also different and is shown in figures 12 and 13

1.6.2 Fine tuning the maximum entropy agent

The star represents the ego agent in . The simulation environment is a replica of the death Circle intersection in Stanford. The other agents in play denoted by colored dots are from a 22 minute video recorded at the intersection. The action space has 9 actions: stay, forward, back, left, right, forward and right, forward and left, back and right, back and left. If the agent chooses to leave the current state, it will move 1.5m every 1.2 seconds. The virtual agent is made to navigate the environment. The trajectory taken by it is shown in blue in figure 14 It avoids all collisions. It learns optimum rewards every time instant. However, it faces problems on occasion when translating the learned rewards to a successful policy and may not always focus on reaching the target position.

1.7. Conclusion

The research successfully compared direct and inverse reinforcement learning techniques. It demonstrated the efficacy of the latter over the former for successful navigation at traffic intersections in real time with dynamic features. The former technique requires extensive training which is extremely time consuming. It is also not equipped to handle any deviations from the behavior it was trained on. Reinforcement learning through deep Q learning is therefore deemed unfit for navigating intersections.

The latter technique deploying maximum entropy inverse reinforcement learning eliminates the need for training and is equipped to handle anomalies in the behavior of other road users. It can also be deployed at every time instant to calculate policies in real time. This approach has also deemed the original method of computing expected state visitation frequency computationally inefficient for large state spaces and has replaced it with a much faster approach.

1.8. Future work

The technique in this report can be extended successfully to any type of virtual agent from a cyclist to a car. However, it is pertinent to note that the mean models used for constructing the feature matrix may not always best represent the agents that were in minority during the recorded trajectories. It is not very successful in reaching the target position. Future work will delve into representing the

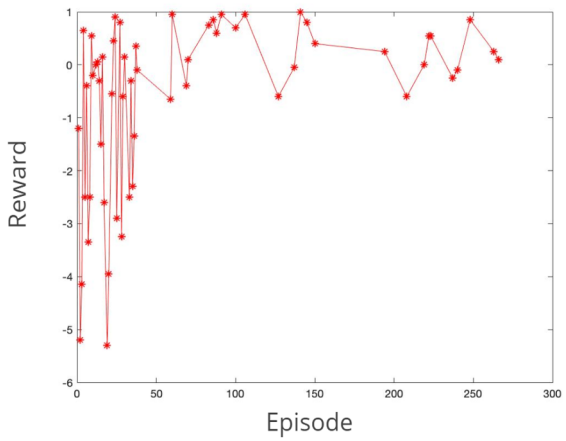

Figure 10: Reward of the DQN Agent over time during training

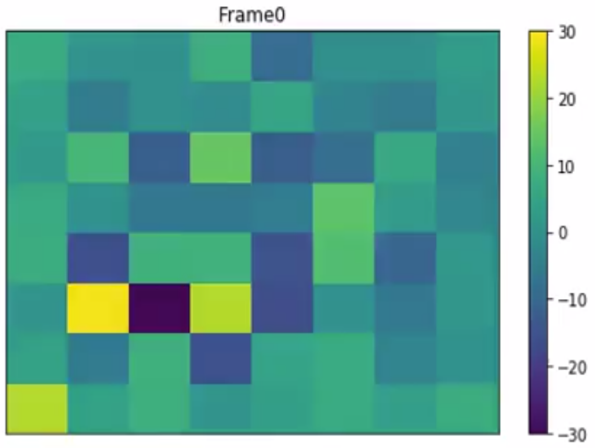

Figure 11: Reward of the MAX-ENT Agent for the first frame across the state space

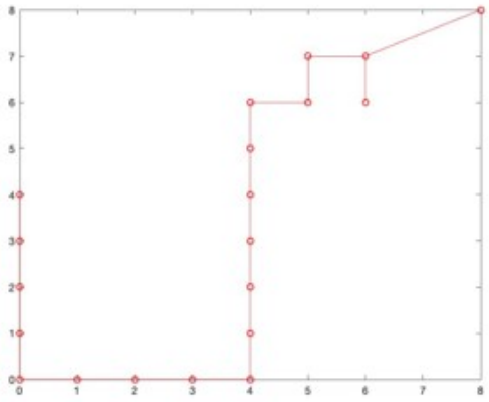

Figure 12: Trajectory of the DQN Agent

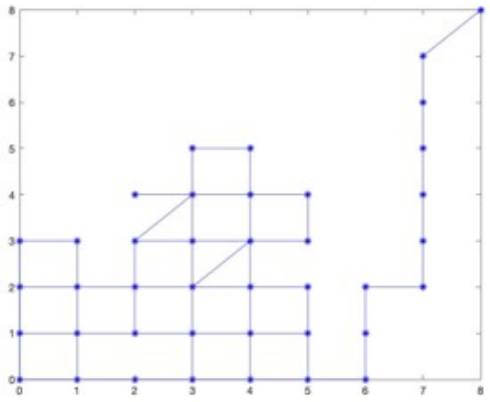

Figure 13: Trajectory of the MAX-ENT Agent

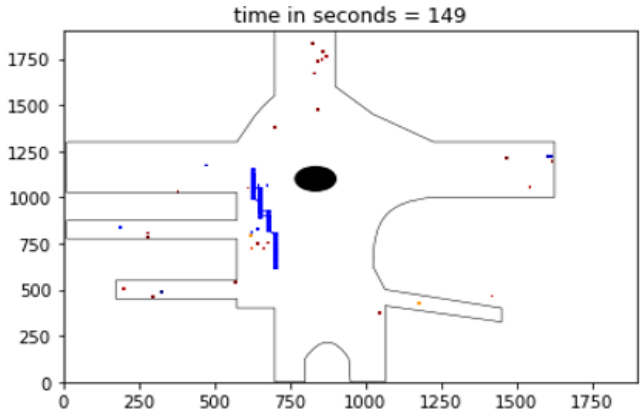

Figure 14: Trajectory taken by IRL agent after fine tuning

target position better in the feature matrix so that the learning algorithm gives it greater importance whilst computing weights. Minor changes can also be included to account for velocity in the given action space.

2. Progress in Database management

This section provides a brief and concise summary of my work in mySQL this semester. After successfully completing a course in Udemy, I proceeded to install darknet on the main computer at the instruction of Jia Cheng. I also downloaded the KITTI dataset and uploaded it to the mySQL Workbench. Modifications had to be made to existing python code for conversion off data into acceptable SQL format. This code was KITTI dataset centric. I also generalized the python program so that it could deal with other datasets. Currently I am in possession of data from the KITTI dataset , Safety Pilot Model Deployment Database

and the Stanford Drone Dataset.

3. Work in Progress and Future directions

This section highlights future work which includes implementing Bayesian recurrrent neural networks to generate a set of control points of distance and velocity for pedestrians and cyclists. It also involves the implementation of a cyclist motion model that identifies trajectory patterns and discrete transitions at several intersections using Dirichlet Process Gaussian Proceeses specified by Yutaon Han [2] in his 2019 paper.

References

[1] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[2] Y. Han, R. Tse, and M. Campbell. Pedestrian motion model using non-parametric trajectory clustering and discrete transition points. *IEEE Robotics and Automation Letters*, 2019.

[3] P. Henry, C. Vollmer, B. Ferris, and D. Fox. Learning to navigate through crowded environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 981–986. IEEE, 2010.

[4] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2039, 2018.

[5] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. In *European conference on computer vision*, pages 549–565. Springer, 2016.

[6] T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

[7] P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on automatic control*, 38(2):195–207, 1993.

[8] D. Zhao, H. Wang, K. Shao, and Y. Zhu. Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–6. IEEE, 2016.

[9] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.