Self-Driving Car for Highway Intersection in Mixed Traffic

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By

Raghava Devaraje Urs (raghavadevaraje.urs@sjsu.edu) – Team Lead
Shiv Kumar Ganesh (shivkumar.ganesh@sjsu.edu)
Kumuda B G Murthy(kumuda.benakanahalliguruprasadamurt@sjsu.edu)

Dec 2021

Copyright © 2020
Raghava Devaraje Urs, Shiv Kumar Ganesh, Kumuda B G Murthy

ALL RIGHTS RESERVED

APPROVED

Jahan Ghofraniha, Ph.D., Project Advisor

EXECUTIVE SUMMARY

Self-Driving Car for Highway Intersection in Mixed Traffic

By
Raghava Devaraje Urs, Shiv Kumar Ganesh, Kumuda B G Murthy

A self-driving car is an autonomous vehicle that has the ability to sense its surroundings and move safely without any human intervention. This technology has gained high recognition in the 21st century. The earliest experiments conducted in this field dates to the 1920s. In 1977, the first semi-automated vehicle was developed in Japan which needed a specific street marking and cameras to interpret the markings. As time passed, AI technologies are used power self-driving cars. Image recognition along with machine learning and neural networks fuel the growth of self-driving cars. There is ongoing research in this field to improve the accuracy to 100 percent and reduce the any accident incidents to zero.

We consider dense traffic to learn architectures for behavioral planning. These architectures consist of varying number of nearby vehicles and are independent to the ordering of the vehicles. An attention-based architecture that satisfies all the above stated properties at the same time accounts for various interactions between the traffic participants is proposed in this project. This approach provides a significant gain in the performance and captures interaction patterns which helps in visualization and interpretation.

Deep-Q Networks was first proposed in 2015 by DeepMind which combines the advantages of deep learning with Reinforcement learning. Reinforcement learning focuses on training agents to take any actions in an environment to maximize the rewards. It then tries to train a model to improve itself and its choices by observing the rewards through interaction with the network.

Using DQN algorithm, a Reinforcement learning based simulated self-driving car is implemented in this project. we plan to define a generative model for states, rewards, and observations and tune some algorithm parameters to adapt it to a highway environment. This makes it useful as a preliminary analysis tool for deciding which control approach to implement on an actual vehicle. Using various reinforcement methods, feature scope expansion is fine-tuned, and the implementation can be extended to different highway environment with automatic parking when the parking spot is vacant. The model is then evaluated using a challenging intersection-crossing task which involves up to 15 vehicles. The evaluations shows that the proposed solution improves quantitatively, and it helps to capture interaction patterns that can be interpreted visually.

Acknowledgments

The authors are deeply indebted to Professor Jahan Ghofraniha, Ph.D. for his invaluable comments and assistance in the preparation of this study.

Table of Contents

Contents

Chapter 1. Project Overview

Background and Introduction

In the last few years, the behavioral planning problem has received the least attention from the research community. The behavioral planning problem is nothing but the high-level of decision-making respect to the context involved in autonomous driving. This field has also seen the least progress compared all other robotics field. Majority of the existing systems completely rely on the hand-crafted rules in Finite State Machines (FSM). A finite state machine is an abstract model of computation that is used to model logic. A language is considered regular if and only if it can be recognized by an FSM. This results in only a particular set of use cases being addressed. Also, these methodologies do not scale for complex scenarios where the decision making involves interacting with other human drivers whose actions cannot be predicted.

All these observations have made the AI community to investigate methodologies based on learning which promises the usage of leveraging data to automatically learn complex driving policy. In the imitation learning approach, a policy can be trained in a supervised manner to imitate human driving decisions. A computer system that achieves AI through a machine learning technique is called a learning system. In contrast to rule-based systems, learning systems have a very ambitious goal. The vision of AI research, which turns out to be more a hope than a concrete vision, is to implement general AI through the learning capability of these systems. Hence, the hope is that a learning system is in principle unlimited in its ability to simulate intelligence. The ability to learn causes adaptive intelligence, and adaptive intelligence means that existing knowledge can be changed or discarded, and new knowledge can be acquired. Hence, these systems build the rules on the fly. That is what makes learning systems so different from rule-based testing. A neural network is an instance of a learning system.

Beyond the choice of reinforcement learning algorithm, the formalization of the problem as a Markov Decision Process plays an important part in the design of the system. Indeed, the definition of the state space involves choosing a representation of the driving scene. In this project, we focus on how the vehicles are represented. We claim that the two most-widely used representations both suffer from different drawbacks: on the one hand, the list of features representation is compact and accurate but has a varying-size and depends on the choice of ordering. On the other hand, the spatial grid representation addresses these concerns but in return suffers from an accuracy-size trade-off.

We propose an attention-based architecture for decision making involving social interactions. This architecture allows to satisfy the variable-size and permutation, invariance requirements even when using a list of features representation. It also naturally accounts for interactions between the ego-vehicle and any other traffic participant. We then evaluate our model on a challenging intersection-crossing task involving up to 15 vehicles perceived simultaneously. We show that our proposed method provides significant quantitative improvements.

Problem Statement

We study the design of learning architectures for behavioral planning in a dense traffic setting. Such architectures should be prepared to deal with a varying number of vehicles in the surrounding at the same time be invariant to the ordering chosen to describe them, while staying accurate and compact.

Using Deep Q Networks algorithm, a Reinforcement learning based simulated self-driving car is implemented in this project. we plan to define a generative model for states, rewards, and observations and tune some algorithm parameters to adapt it to a highway environment. This makes it useful as a preliminary analysis tool for deciding which control approach to implement on an actual vehicle. Using various reinforcement methods, feature scope expansion is fine-tuned, and the implementation can be extended to different highway environment with automatic parking when the parking spot is vacant. The model is then evaluated using a challenging intersection-crossing task which involves up to 15 vehicles. The evaluations shows that the proposed solution improves quantitatively, and it helps to capture interaction patterns that can be interpreted visually.

Purpose and Motivation

The behavioral planning problem has received the least attention from the research community. The behavioral planning problem is nothing but the high-level of decision-making respect to the context involved in autonomous driving. This field has also seen the least progress compared all other robotics field. This results in only a particular set of use cases being addressed. Also, these methodologies do not scale for complex scenarios where the decision making involves interacting with other human drivers whose actions cannot be predicted. Also, the cost of human driving data collection at large scale can be prohibitive, another promising approach is training a policy in simulation using reinforcement learning.

In contrast to rule-based systems, learning systems have a very ambitious goal. The vision of AI research, which turns out to be more a hope than a concrete vision, is to implement general AI through the learning capability of these systems. Hence, the hope is that a learning system is in principle unlimited in its ability to simulate intelligence. The ability to learn causes adaptive intelligence, and adaptive intelligence means that existing knowledge can be changed or discarded, and new knowledge can be acquired. Hence, these systems build the rules on the fly. That is what makes reinforcement learning based systems so different from rule-based testing.

Chapter 2. Project Architecture

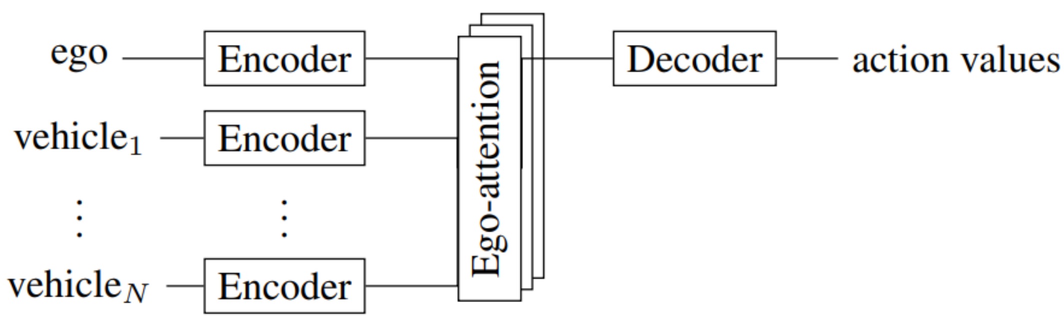
The attention architecture was introduced to enable neural networks to discover inter-dependencies within a variable number of inputs. It has been used for pedestrian trajectory forecasting in Vemula et al. with spatiotemporal graphs and in Sadeghian et al. with spatial and social attention using a generative neural network. In Sadeghian et al. attention over top-view road scene images for car trajectory forecasting is used. Multi-head attention mechanism has been developed in Vaswani et al. for sentence translation. In Messaoud et al. a mechanism called non-local multi-head attention is developed. However, this is a spatial attention that does not allow vehicle-to-vehicle attention. In the present work, we use a multi-head social attention mechanism to capture vehicle-to-ego dependencies and build varying input size and permutation invariance into the policy model.

To apply a reinforcement learning algorithm such as DQN to an autonomous driving problem, a state space S must first be chosen, that is, a representation of the scene. When social interactions are relevant to the decision, the state should at least contain a description of every nearby vehicle. A vehicle driving on a road can be described in the most general way by its continuous position, heading and velocity. Then, the joint state of a road traffic with one ego-vehicle denoted s0 and N other vehicles can be described by a list of individual vehicle states known as list of features:

$$s = (s_i)_{i\in[0,N]} \qquad \text{where} \qquad s_i = [x_i \quad y_i \quad v_i^x \quad v_i^y \quad \cos\psi_i \quad \sin\psi_i]^T$$

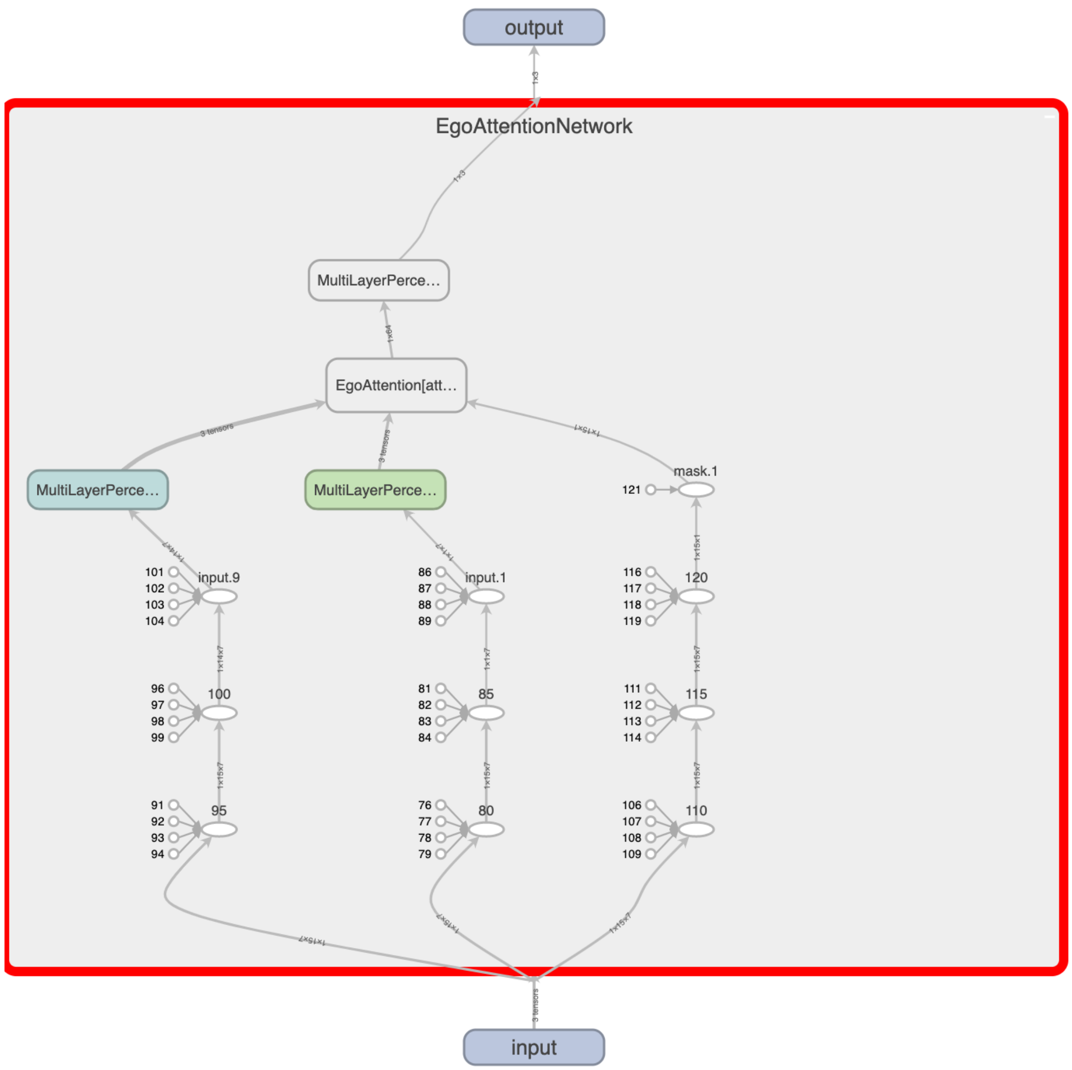
In each complex driving condition, the model should be able to filter information and consider only what is relevant to make its decision. In other words, the agent should pay attention to vehicles that are close or conflict with the planned route.

The proposed Deep Q Network (DQN) architecture is represented using three layers - Linear encoding layer, Ego attention layer and Linear decoder.



These layers are used to represent the Q-function that will be optimized by the DQN algorithm. The model is composed of a first linear encoding layer whose weights are shared between all vehicles. At this point, the embeddings only contain individual features of size dx. They are then fed to an ego-attention layer, composed of several heads stacked together. The ego prefix highlights that it is like a multi-head self-attention layer

The outputs from all heads are finally combined with a linear layer, and the resulting tensor is then added to the ego encoding as in residual networks. We can easily see that this process is permutation invariant: indeed, a permutation will change the order of the rows in keys K and values V have kept their correspondence. The result is a dot product of values and key-similarities, which is independent of the ordering.



Graph diagram of our model architecture, showcasing several linear identical encoders, a stack of ego-attention heads, and a linear decoder.

Chapter 3. Methodology

We applied multiple algorithms in order to achieve our goals. Deep Q-Networks stood out and gave us decent performance when compared to all other algorithms.

Deep Q-Networks (DQN)
Deep Q-Networks use a neural network to estimate/approximate the value of the Q-Function. The DQN takes state as input and generates all possible Q-values as the output. In the case of self-driving cars the state space must be chosen carefully. The state space represents the scene in which cars are running. The state also contains the description of the nearby vehicle. This is required for social awareness of the car in question.

To apply a reinforcement learning algorithm such as DQN to an autonomous driving problem, a state space S must first be chosen, that is, a representation of the scene. There are few basic steps involved in DQN which can be summarized as follows:

1. History is maintained in memory
2. Maximum value of the Q-Network plays an important role in determining the next action.
3. Mean Squared Error between the predicted and target value acts as the loss function.

Bellman Equation is followed as an important identifier for the optimal action-value function.

The optimal action-value function $Q^* = \max_\pi Q^\pi(s)$ satisfies the Bellman Optimality Equation:

$$Q^*(s,a) = (\mathcal{T}Q^*)(s,a) \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{s' \sim P(s'|s,a)} \max_{a' \in A} [R(s,a) + \gamma Q^*(s',a')]$$

A car driving on the road can be identified by three vectors namely the position i.e continuous, heading and the velocity component. When social interactions are relevant to the decision, the state should at least contain a description of every nearby vehicle. In our case we have multiple vehicles on the road and the main vehicle in question will be referred to as an ego-vehicle.

Deep neural networks come in a variety of architectures. In this specific case we will use the **Attention mechanism** along with a deep neural network in order to implement our DQN. Attention layer provides us with a benefit by discovering inter-dependencies between various inputs across time. This helps in determining proper relationship between two or more inputs and thus helping us determine proper Q-Value.

As described above, this approach will help us to figure out the dependencies between the vehicle and ego-vehicles on the road and also will help us process the inputs of varying sizes using the policy model. In order to increase the accuracy and capture more context, we will be using Multi-Headed Attention.

Chapter 4. Implementation and Results

Implementation

Agent: Car
Environment: Highway Environment- Intersection
State: Position, heading, velocity
Action: Slower, faster
Rewards: Drive with speed, avoid collision with negihbouring vehicle.

Differentiator and Contribution

The existing state of art for self-driving reinforcement learning algorithms are based on Monte Carlo Method, Fitted Q Iteration and Deep Q-Network.

Monte Carlo Method

The Monte Carlo method for reinforcement learning learns directly from episodes of experience without any prior knowledge of MDP transitions. Here, the random component is the return or reward.
One caveat is that it can only be applied to episodic *MDPs*. It's fair to ask why, at this point. The reason is that the episode must terminate before we can calculate any returns. Here, we don't do an update after every action, but rather after every episode. It uses the simplest idea – the value is the mean return of all sample trajectories for each state.
Similar to dynamic programming, there is a policy evaluation (finding the value function for a given random policy).

Policy Evaluation
The goal here, again, is to learn the value function vpi(s) from episodes of experience under a policy pi. Recall that the return is the total discounted reward:

$$S1, A1, R2, \ldots Sk \sim pi$$

Also recall that the value function is the expected return:
We know that we can estimate any expected value simply by adding up samples and dividing by the total number of samples:

- i – Episode index
- s – Index of state

The question is how do we get these sample returns? For that, we need to play a bunch of episodes and generate them.
For every episode we play, we'll have a sequence of states and rewards. And from these rewards, we can calculate the return, which is just the sum of all future rewards.

Fitted Q Iteration

We will take the most popular extension of Q-Learning to batch reinforcement learning setting called Fitted Q Iteration or FQI for short. This method was developed around 2005 and 2006 in a series of papers by Ernest and co-workers, and by Murphy. One noticeable point here is that Ernest and co-workers considered time stationary problems, where the Q-Function does not depend on time. And many published research in reinforcement learning literature deal with an infinite horizon Q- Learning where a Q- function is independent of time.

The Fitted Q Iteration method works with continuous valid data. And therefore, we can now bring the model formulation back to a general continuous state space case, as was the case in our Monte Carlo searching for the dynamic programming solution. But If we want to stick to a discrete space formulation, Fitted Q iteration can be used in essentially the same way. The only difference would be in a specification of basic functions that the method needs to use. So, to retaliate, the FQI method works by using all Monte Carlo paths or historical paths for the replication portfolio simultaneously. This is very similar to how we solve the problem using Dynamic Programming with Monte Carlo method for the case of known dynamics.

In this method we averaged over all scenarios at time C and t plus one simultaneously, by taking an empirical mean of different pathways, Monte Carlo scenarios. Conditioning on that information set f_t, time t was implemented as conditioning on old Monte Carlo paths up to time C. Now in the setting of batch-mode reinforcement learning, the only thing we need to change in such Monte Carlo settings is the structure of input output data. In the setting of Dynamic Programming when the model is known, the inputs are simulated or real for the paths of the state variable extreme. And the outputs are the optimal actions, an action policy and optimal Q-function. That is the negative option price. The optimal action, a star, is determined by maximization of the optimal Q-function and instantaneously words are computed in the course of backward recursion for the optimal action and optimal Q-function.

Based on our study we found that using the Deep Q-Network algorithm we are able to achieve better performance. Thus, we have picked DQN as our optimal algorithm for our modeling.

In the project experimentation we explored multiple policies like Greedy, Boltzmann and Epsilon Greedy. Executed multi head attention head by using different learning rate and optimizer types like ADAM, RMS Prop and Ranger.

We trained the DQN, multi-layer perceptron with an ego-attention head and found the best results for below parameters. We modified batch size, epochs and trained for gamma 0.95, 0.96 and 0.98

```json
{
    "__class__": "<class 'rl_agents.agents.deep_q_network.pytorch.DQNAgent'>",
    "model": {
        "type": "MultiLayerPerceptron",
        "layers": [128, 128]
    },
    "gamma": 0.95,
    "n_steps": 1,
    "batch_size": 64,
    "memory_capacity": 15000,
    "target_update": 512,
    "exploration": {
        "method": "EpsilonGreedy",
        "tau": 15000,
        "temperature": 1.0,
        "final_temperature": 0.05
    }
}
```

```json
{
    "base_config": "configs/IntersectionEnv/agents/DQNAgent/baseline_98.json",
    "model": {
        "type": "EgoAttentionNetwork",
        "embedding_layer": {
            "type": "MultiLayerPerceptron",
            "layers": [64, 64],
            "reshape": false,
            "in": 7
        },
        "others_embedding_layer": {
            "type": "MultiLayerPerceptron",
            "layers": [64, 64],
            "reshape": false,
            "in": 7
        },
        "self_attention_layer": null,
        "attention_layer": {
            "type": "EgoAttention",
            "feature_size": 64,
            "heads": 2
        },
        "output_layer": {
            "type": "MultiLayerPerceptron",
            "layers": [64, 64],
            "reshape": false
        }
    }
}
```

Reward Function and Objectives

The qualitative objectives in solving this problem are to reach the target lane as quickly as possible and increase the comfort and safety of both the ego and the other nearby vehicles. The reward function is focused on two features: a vehicle should

- progress quickly on the road;
- avoid collisions.

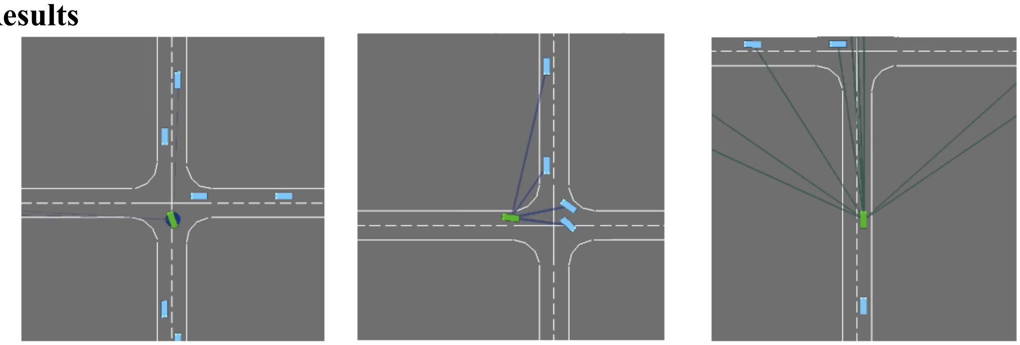
Thus, the reward function is often composed of a velocity term and a collision term

$$R(s,a) = a\frac{v - v_{\min}}{v_{\max} - v_{\min}} - b\,\text{collision}$$

where v, vmin, vmax are the current, minimum and maximum speed of the ego-vehicle respectively, and a, b are two coefficients.

A reward is generated for each step in the target lane, and a cost is accrued for each action. The agent is rewarded by 1 when it drives at maximum speed, 0 when it drives at minimum speed and by -5 when a collision occurs.

Results



Gamma = 0.95 Gamma = 0.98 Gamma = 0.99

We observed better results for gamma 0.98 and applying modifiers and fine-tuning batch size. We could have achieved better results by training a large number of episodes, due to hardware constraints, we have performed training for 50 episodes.

Model Evaluation

DQN is trained 50 episodes. The score for each episode is displayed.

```
evaluation.train()
```

```
[INFO] Episode 0 score: 3.0
[INFO] Episode 1 score: -3.1
[INFO] Episode 2 score: -0.3
[INFO] Episode 3 score: -1.1
[INFO] Episode 4 score: 7.9
[INFO] Episode 5 score: -3.0
[INFO] Episode 6 score: 1.1
[INFO] Episode 7 score: -1.3
[INFO] Episode 8 score: 1.9
[INFO] Episode 9 score: 0.1
[INFO] Episode 10 score: -2.2
[INFO] Episode 11 score: -2.9
[INFO] Episode 12 score: 2.9
[INFO] Episode 13 score: -2.2
[INFO] Episode 14 score: 1.9
[INFO] Episode 15 score: 0.9
[INFO] Episode 16 score: -2.0
[INFO] Episode 17 score: -1.2
[INFO] Episode 18 score: 4.6
[INFO] Episode 19 score: 0.0
[INFO] Episode 20 score: -2.1
[INFO] Episode 21 score: 3.1
[INFO] Episode 22 score: 3.0
[INFO] Episode 23 score: 8.7
[INFO] Episode 24 score: 7.7
[INFO] Episode 25 score: 4.6
[INFO] Episode 26 score: -0.1
[INFO] Episode 27 score: 1.8
[INFO] Episode 28 score: -1.1
[INFO] Episode 29 score: 1.1
[INFO] Episode 30 score: -0.1
[INFO] Episode 31 score: 6.8
[INFO] Episode 32 score: 5.9
[INFO] Episode 33 score: 2.9
[INFO] Episode 34 score: 0.1
[INFO] Episode 35 score: 6.9
[INFO] Episode 36 score: 3.9
[INFO] Episode 37 score: 8.8
[INFO] Episode 38 score: 5.8
[INFO] Episode 39 score: -5.0
[INFO] Episode 40 score: 8.9
[INFO] Episode 41 score: 4.8
[INFO] Episode 42 score: 9.8
[INFO] Episode 43 score: 2.7
[INFO] Episode 44 score: 1.1
[INFO] Episode 45 score: -5.0
[INFO] Episode 46 score: 6.9
[INFO] Episode 47 score: 7.8
[INFO] Episode 48 score: -3.1
[INFO] Episode 49 score: 6.9
[INFO] Saved DQNAgent model to /content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211203-055249_60/checkpoint-final.tar
INFO: Finished writing results. You can upload them to the scoreboard via gym.upload('/content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211203-055249_60')
```

Then the model is evaluated

```
[ ] evaluation.load_agent_model('/content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211202-062759_60/checkpoint-final.tar')
```

```
[INFO] Loaded DQNAgent model from /content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211202-062759_60/checkpoint-final.tar
```

```
evaluation.test()
```

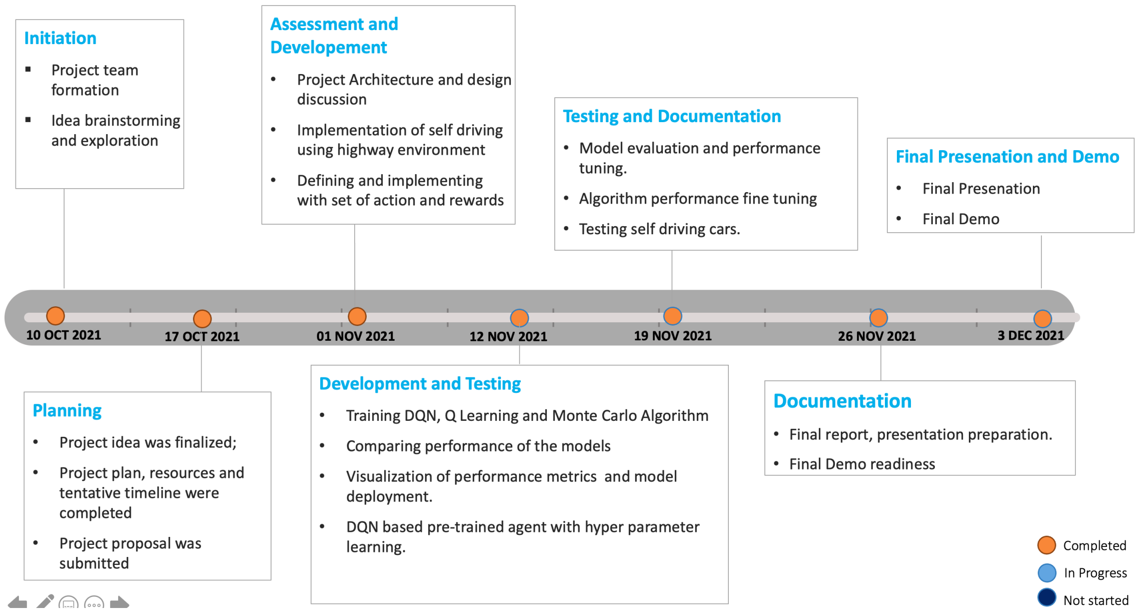
```
[INFO] Episode 0 score: 0.3
[INFO] Episode 1 score: 9.9
[INFO] Episode 2 score: 10.0
[INFO] Episode 3 score: 10.0
[INFO] Episode 4 score: 9.0
[INFO] Episode 5 score: 1.0
[INFO] Episode 6 score: 8.9
[INFO] Episode 7 score: -0.1
[INFO] Episode 8 score: -0.9
[INFO] Episode 9 score: 0.0
```

Testing the tainined model with gamma value of 0.95

```
env = load_environment(env_config)
env.configure({"offscreen_rendering": True})
agent = load_agent(agent_config, env)
evaluation = Evaluation(env, agent, num_episodes=4, recover=True)
evaluation.test()
show_videos(evaluation.run_directory)
```

```
INFO: Making new env: intersection-v0
[WARNING] Preferred device cuda:best unavailable, switching to default cpu
[INFO] Loaded DQNAgent model from out/IntersectionEnv/DQNAgent/saved_models/latest.tar
INFO: Creating monitor directory out/IntersectionEnv/DQNAgent/run_20211203-055406_60
INFO: Starting new video recorder writing to /content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211203-055406_60/openaigym.video.6.60.video000000.mp4
[INFO] Episode 0 score: -2.9
INFO: Starting new video recorder writing to /content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211203-055406_60/openaigym.video.6.60.video000001.mp4
[INFO] Episode 1 score: -5.0
INFO: Starting new video recorder writing to /content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211203-055406_60/openaigym.video.6.60.video000002.mp4
[INFO] Episode 2 score: 0.0
INFO: Starting new video recorder writing to /content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211203-055406_60/openaigym.video.6.60.video000003.mp4
[INFO] Episode 3 score: 10.0
INFO: Finished writing results. You can upload them to the scoreboard via gym.upload('/content/rl-agents/scripts/out/IntersectionEnv/DQNAgent/run_20211203-055406_60')
```

Project Schedule



Task by Team Members

Time	Raghava	Shiv	Kumuda
Oct 10th	• Team formation • Project idea brainstorming • Explored on ideas relating to self-driving cars	• Team formation • Project idea brainstorming • Explored ideas relating to Atari games.	• Team formation • Project idea brainstorming • Explored ideas relating to games like tic tac toe, 4connect etc.
Oct 17th	• Project idea finalized with self-driving car simulation. • Explored different published papers to understand the domain. • Formulation of deep reinforcement learning architecture toward autonomous	• Project idea exploration on self-driving car simulation. • Robust traffic merging strategies for sensor-enabled cars using time geography(link) • Merge Maneuver by Autonomous Vehicle using Reinforcement Learning in Dense Traffic(link)	• Performed exploration of the different available libraries for performing car simulation. • Explored different published papers to understand the domain. • Worked on completing the project proposal. • Cooperative decision-making for mixed traffic: A ramp merging example (link) • Project proposal document preparation.

	driving for on-ramp merge(link) ● Project proposal document preparation.	● Project proposal document preparation.	
Nov 1st	● Detailed design preparation ● Infrastructure setup for project execution ● Sample code setup to test the project setup	● Project code setup ● Exploration of self-driving car UI simulation	● Study of Highway Environment ● Exploring the various actions and observation spaces ● Project code setup
Nov 12th	● Exploration of the self-driving car using Q Learning. ● Project status update document preparation	● Project status updates document review and completion. ● Highway driving scenario definition and implementation of a set of actions and rewards.	● Project status updates document review and completion. ● Comparing DQN and Q-Learning started
Nov 19th	● DQN Implementation started ● Hyperparameter Tuning started	● DQN Implementation started ● Hyperparameter Tuning started	● Visualization and Graph generation started ● Exploring PPO and MAPPO started ● Project status updates document review and completion.
Nov 26th	● Project Report Preparation ● Presentation Preparation	● Project Report Preparation ● Presentation Preparation	● Project Report Preparation ● Presentation Preparation
Dec 3rd	● Final Presentation/ Final Demo	● Final Presentation/ Final Demo	● Final Presentation/ Final Demo

Chapter 5. Conclusions

In this work, we showed that the list of features representation, commonly used to describe vehicles in autonomous driving literature, is not tailored for use in a function approximation setting, with neural networks. These concerns can be addressed by the spatial grid representation, but it comes at the price of an increased input size and loss of accuracy. In contrast, we proposed an attention-based neural network architecture to tackle the issues of the list of features representation without compromising either size or accuracy. This architecture enjoys a better performance on a simulated negotiation and intersection crossing task and is also more interpretable thanks to the visualization of the attention matrix. The resulting policy successfully learns to recognize and exploit the interaction patterns that govern the nearby traffic.

References

1. James Bagnell, David Bradley, David Silver, Boris Sofman, and Anthony Stentz. Learning for autonomous navigation. IEEE Robotics and Automation Magazine, 17(2):74–84, 2010.

2. Haoyu Bai, Shaojun Cai, Nan Ye, David Hsu, and Wee Lee. Intention-aware online pomdp planning for autonomous driving in a crowd. Proceedings - IEEE International Conference on Robotics and Automation, 2015:454–460, 06 2015.

3. Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst, 2018.

4. Richard Bellman. Dynamic programming and lagrange multipliers. In Proceedings of the National Academy of Sciences of the United States of America, 1956.

5. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon

6. Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao,and Karol Zieba. End to End Learning for Self-Driving Cars. arXiv preprint, 2016.

7. Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09, pages 1179–1186. ACM, 2009.

8. Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. Socially aware motion planning with deep reinforcement learning. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sep 2017.

9. Felipe Codevilla, Matthias Miiller, Antonio Lopez, Vladlen Koltun, and Alexey Dosovitskiy. End-to-End Driving Via Conditional Imitation Learning.

10. Proceedings - IEEE International Conference on Robotics and Automation, pages 4693–4700, 2018.

11. Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-End Deep Learning for Steering

12. Autonomous Vehicles Considering Temporal Dependencies. In Workshop on Machine Learning for Intelligent Transportation Systems, NeurIPS 2017, 2017.
13. Lex Fridman, Jack Terwilliger, and Benedikt Jenik. Deeptraffic: Crowdsourced hyperparametertuning of deep reinforcement learning systems for multi-agent dense traffic navigation. In Deep

14. Reinforcement Learning Workshop at NeurIPS 2018, 2018.

15. Enric Galceran, Alexander G. Cunningham, Ryan M. Eustice, and Edwin Olson.

16. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment. Autonomous Robots, 2017.

17. Tobias Gindele, Sebastian Brechtel, and Rudiger Dillmann. Learning driver behavior models from traffic observations for decision making and planning. IEEE Intelligent Transportation Systems Magazine, 7(1):69–79, 2015.

18. David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A Review of Motion

19. Planning Techniques for Automated Vehicles. IEEE Transactions on Intelligent Transportation Systems, 2016.

20. David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio,

21. H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 2450–2462. Curran Associates, Inc., 2018.

Appendices

Appendix A. **https://github.com/vrmusketeers/RL_Self_Driving_Car_Intersection/blob/main/Self_Driving_Car_Intersection_Presentation.pdf**

Appendix B. **https://github.com/vrmusketeers/RL_Self_Driving_Car_Intersection**