Nanyang Technological University

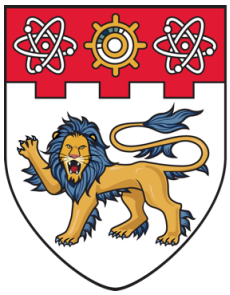
Reinforcement Learning for Self-Driving Cars

Ho Song Yan

School of Computer Science and Engineering

2018

Nanyang Technological University



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SCE17-0434

Reinforcement Learning for Self-Driving Cars

25 March 2018

Submitted in Partial Fulfilment of Requirements
for the Degree of Bachelor in Computer Science
of the Nanyang Technological University by

Ho Song Yan

U1420448A

Supervisor: Assoc Prof Xavier Bresson

Examiner: Asst Prof Zhang Hanwang

School of Computer Science and Engineering

2018

Abstract

This project presents the implementation of deep learning model to act as a self-driving car-agent to maximize its speed on a multilane expressway. This project includes the development of traffic environment simulation, the design of neural network model, and the implementation of reinforcement learning algorithm. The proposed model uses the minimal sensory input collected from the environment. The model was trained with reinforcement learning algorithm in the simulation environment to simulate traffic condition of seven-lane expressway. The model successfully learns and applies the optimal policy. The model was tested under three different traffic conditions to determine its performance statistically. The best model is the model with neural network configuration that approximate to the optimal Q-learning function. The source code of this project can be found on https://github.com/songyanho/Reinforcement-Learning-for-Self-Driving-Cars.

Acknowledgement

The student would like to express his sincere gratitude to the following people:

First of all, the student would like to thank his project supervisor, Professor Xavier Bresson for his guidance and support throughout the whole project. The student has received full support from Professor Xavier to improve the project with Professor Xavier's knowledge and experiences.

Second, the student would like to express appreciation to his family for boundless love and support when the student faced challenges along the project.

Last but not the least, the student would like to thank his fellow mates for providing useful ideas and suggestions to improve the project.

Table of Contents

Abstract i

Acknowledgement ii

Table of Contents iii

List of Notations iv

List of Figures v

List of Tables vi

1. Introduction 1
- 1.1 Background 2
- 1.2 Objective 2
- 1.3 Scope 2

2. Literature Review 4
- 2.1 Reinforcement Learning 4
- 2.2 Markov Decision Process 4
- 2.3 Q-Learning 5
- 2.4 Convolutional Neural Networks 5
- 2.5 Deep Q-Learning Networks (DQN) 6
- 2.6 Double Deep Q-Learning (DDQN) 7

3. Project Timeline 8
- 3.1 Actual Project Timeline 9

4. Algorithm 11

5. Model Architecture 13

6. Experiments 15
- 6.1 Experimental Setup 15
- 6.2 Evaluation Metrics 17

7. Results 18
- 7.1 Average speed 18
- 7.2 Score 19
- 7.3 Training loss 20
- 7.4 Evaluation 21

8. Discussion 24

9. Conclusion 25

10. Recommendations 26

References 27

List of Notations

Notation	Description
S	Set of possible environment states
A	Available actions
R	Reward function in Markov Decision Processes
$\mathbb{P}$	Transition probability in Markov Decision Processes
γ	Discount factor of the future reward
$\pi(s, a)$	State-action pair policy
$V(s, a)$	Value function of Q-learning
$Q(s, a)$	Q-value function of Q-learning
θ	Parameter of the neural network
a_t	Action selected at time step t
s_t	Environment state at time step t

List of Figures

Figure 1: The illustration of convolutional layers and fully-connected layers forming convolutional neural network……... 6

Figure 2: Gantt chart of proposed project timeline ……... 9

Figure 3: Gantt chart of revised project timeline …….. 10

Figure 4: Baseline model with CNN architecture with one convolutional layer and one fully-connected layer……..... 13

Figure 5: Proposed model with CNN architecture with 2-layer fully-connected for action stack and two convolutional layer and two fully-connected layer... 14

Figure 6: Screenshot of simulator running in macOS High Sierra……. 15

Figure 7: Average speed of agent-controlled car during training…….. 18

Figure 8: Score of self-driving car at the end of episode during training……... 19

Figure 9: Log-scaled loss against number of training episode……... 20

Figure 10: Average speed against average number of emergency brake applied under Condition 1……..... 22

Figure 11: Average speed against average number of emergency brake applied under Condition 2……..... 22

Figure 12: Average speed against average number of emergency brake applied under Condition 3……..... 23

List of Tables

Table 1: Proposed project timeline 8

Table 2: Revised project timeline 9

Table 3: Actions for skipped frames with respect to action at k^{th} frame 12

Table 4: Traffic condition in the simulation 16

Table 5: Driving behaviour of non-agent-controlled car in the simulation 16

Table 6: The list of hyperparameters and their values used in the experiments 16

Table 7: Experiments with different configurations 17

Table 8: Comparison of the average speed of agent with different configuration during evaluation 21

1. Introduction

Driving a car requires precision and active attention. Any distraction to the driver could lead to accidents and losses. According to the data provided by Singapore Police Force, there were 94.34% of the road accidents led by human errors in year 2016 which could be avoided, for instance, failing to keep proper lookout, failing to have proper control and change lane without due care. Other road users are affected by the road accidents even though they follow the rules and regulations. Lately, drivers divert their attention to their smartphones to answer calls, send messages or navigate to their destinations. It increases the chances of road accident when drivers are not focusing on the road condition. The automotive industry is innovating on the smart features of car to further improve the safety and driving experience for users. The automatic braking system or collision avoidance system is one of the innovation that uses sensors such as radar to detect and mitigate collision. The innovation in preventive safety mechanism catalyses the development process of autonomous vehicle that can drive to the destination autonomously and then reduce the number of road accidents that were caused by human errors by prioritising the safety of passengers and other road users including motorcyclists and pedestrians.

Self-driving car is the future development in automotive industry. The breakthrough in performance of parallel processing hardware and innovation in neural network design allows an artificial intelligence(AI) agent to control a vehicle autonomously. An AI agent is more capable of sustained attention and focus than humans. However, building an autonomous vehicle is a long-standing goal and complex task. Driving requires three main capabilities including recognition, prediction and planning[1]. First of all, an AI agent requires to recognize the environment with sensors. Convolutional neural networks (CNN) are the most successful neural network model that is scalable and able to extract important spatial features from spatial data. Moreover, an AI agent has to predict the future states of the environment and perform best action at the moment. Furthermore, the planning component integrate the ability of model to understand the environment (recognition) and its dynamics (prediction) to plan the future actions to avoid unwanted situations (penalties) and drives safely to its destination (rewards). The video feed from camera and output of Light Detection and Ranging(LIDAR) are mainly used as the visual feed to AI agents.

1.1 Background

An autonomous vehicle requires to carry out multiple tasks to drive on the road with dynamic traffic condition. The tasks include following the road marking, keeping a safe distance with vehicles, reacting to emergency situation accordingly and navigating to the destination. Udacity has partnered with Didi Chuxing to organize the first self-driving car challenge in year 2017 to come up with the best way to detect obstacles using camera and LIDAR data. It could ensure self-driving cars prioritize the safety of road users and pedestrians. Besides that, Udacity has organized another challenge to predict the steering angle with deep learning based on the image inputs from a camera mounted to the windshield.

However, the quality of the driving skill and its efficiency in getting passengers to the destination are often neglected while most of the researches and designs focus on the basic features of self-driving car. The driving skill of driving agent becomes the priority to deliver the best transportation service compared to a manned vehicle. For instance, a self-driving car-agent should be capable to switch to the optimal lane which maximize its speed rather than follow slow-moving vehicles in a predefined lane all the time. It is important because passengers expect to reach its destination by taking the shortest journey and shortest time along with safety.

1.2 Objective

The objective of this project is to design and implement a self-driving car-agent to maximize the its speed. A simulation environment software is required to simulate and visualize the actual traffic environment. A neural network model is essential to build a self-driving car-agent that interact with complex traffic condition. Reinforcement learning technique is used to train the model to adapt to the dynamic environment with infinite action-state sequences. The agent is required to learn the optimal policy to choose the optimal available option which maximizes the speed of vehicle.

1.3 Scope

The scopes of this project are to maximize the speed of a self-driving car on an expressway with seven lanes in a simulated environment. The simulated environment is built with *pygame* framework in Python environment. The autonomous vehicle is assumed to function perfectly to steer along the road. It simulates the road conditions where a self-driving car-agent on a

seven-lane roads with the other non-agent-controlled cars with several human driving behaviours.

2. Literature Review

2.1 Reinforcement Learning

Reinforcement learning is one of the learning approach for neural networks. Reinforcement learning implements differently compared to supervised learning and unsupervised learning. Reinforcement learning makes an agent to generate the optimal policy in an environment and maximises the reward [2]. An agent can learn the rewards given by taking action in respective state and subsequently learn the proper action for each state without having any predefined rules and knowledge about the environment [3]. The agent learns by trials and errors, or in another words, the actions with higher reward are reinforcement while actions with penalties are avoided in the future. It is different with supervised learning because supervised learning requires huge number of labelled dataset to train the model [4]. The training process is repeated with same set of data until the model converges. It requires efforts in labelling the data and the process is error-prone[5]. Reinforcement learning is also different with unsupervised learning because reinforcement learning aims to maximise the reward while unsupervised learning does not.

2.2 Markov Decision Process

Markov Decision Process (MDP) can make a sequence of decisions based on the utilities of environment state. MDP [1] is defined by (S, A, R, $\mathbb{P}$, γ), in which a set of possible environment states(S), a set of possible actions(A), discount factor(γ), reward function (R), and state transition probability($\mathbb{P}$). Environment states S are sequence of states with initial state s_0. Actions A are available actions for environment states S. Reward function R is a map of immediate reward given (state, action) pair. State transition probability $\mathbb{P}$ is the probability of transition from states $s_0 \dots s_i$ to state s_{i+1}. However, there are infinite state sequences in the environment and calculation of transition probability for all state sequences is not feasible[6]. Thus, MDP follows Markov assumption in which the transition probability to state s_{i+1} is only depends on s_i and not previous states [7].

$$\mathbb{P}(s_{i+1}|s_i, a_i, \dots, s_0, a_0) \approx \mathbb{P}(s_{i+1}|s_i, a_i) \quad (1)$$

Discount factor γ discounts the rewards of future states contributing to the cumulative reward [2]. The discounted future rewards can be formulated with the formula below.

$$R_t(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n = \sum_{t \geq 0} \gamma^t R_{a_t}(s_t, s_{t+1}) \quad (2)$$

The value function can estimate the value of a state sequence start with state s, as shown follows:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^{t} r_{t}\right] \quad (3)$$

The optimal value function is the value functions that give the highest value for all states.

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \quad (4)$$

2.3 Q-Learning

Q-learning is a reinforcement learning mechanism that compare the expected utility of available actions given a state. Q-learning can train a model to find the optimal policy in any finite MDP [8]. The Bellman equation suggests the Q-value function[9] as shown in equation 5.

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a') \quad (5)$$

The difference of Q-value function in equation 7 with value function in equation 3 is Q-value function estimates the maximum future reward for taking action a given a state s. The calculation of Q-value function takes into account of the maximum discounted future reward for an agent to move from state s to state s'. The Q-value function can be iteratively converged to the optimal Q-value function [10] by the difference of the estimated utilities between current state s_t and next state s_{t+1} with learning rate α, at time step unit t, as shown in equation 6.

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \max_{a} Q_t(s_{t+1},a) - Q_t(s_t,a_t)\right) \quad (6)$$

Then, Q-learning function can substitute the value function in equation 4, as shown below.

$$V^{*}(s) = \max_{a} Q^{*}(s,a) \quad (7)$$

The optimal policy can be retrieved from the optimal value function with equation 8.

$$\pi^{*}(s) = \arg\max_{a} Q^{*}(s,a) \quad (8)$$

2.4 Convolutional Neural Networks

The efficiency and learning ability of brain have inspired the innovation of the convolutional neural networks (CNN). The ability of animal to perceive features from complex visual map and generate responses has biological guided the construction of CNN [11]. The local correlation can be extracted spatially by allowing a local connectivity pattern between neighbouring cells of adjacent layers. Figure 1 shows that each filter is applied across the entire input map and generate a feature map. The output of the feature map is connected to the adjacent layer with features extracted. Each feature map explores unique features regardless of the location across visual field.

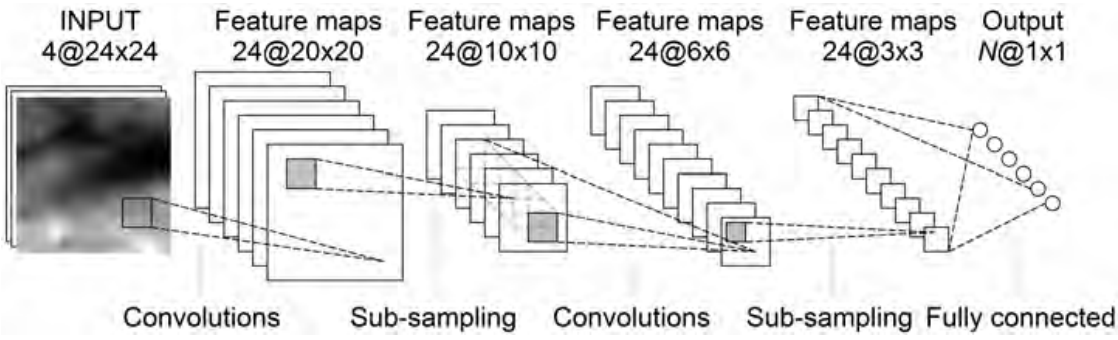


Figure 1: The illustration of convolutional layers and fully-connected layers forming convolutional neural network suggested in [12].

The neuron in convolutional layer of CNN at location (i, j) with kernel w accept synaptic input u as shown in equation 9.

$$u(i,j) = \sum_{l}\sum_{m} x(i+l, j+m)w(l,m) + b \tag{9}$$

$$w = \{w(l,m)\}_{l,m=-\frac{L}{2},-\frac{M}{2}}^{\frac{L}{2},\frac{M}{2}} \tag{10}$$

The synaptic input of neuron at (i, j) of the convolutional layer is processed with activation function $f(u)$ to produce the output, $y(i, j)$. The pooling layer reduces the dimensions of convolutional layer. Pooling down-samples the input. The final layers of deep CNN consist of one or more fully connected layers with *softmax* regression.

2.5 Deep Q-Learning Networks (DQN)

Neural network is utilized as the function approximator of the Q-learning algorithm to form the Deep Q-learning network (DQN). The learning goal of DQN is to find the best settings of the parameter θ of $Q(s, a; \theta)$ or the weights w of the function approximator [13]. The objective of DQN is to minimise the Mean Square Error (MSE) of the Q-values. Besides that, experience replay technique uses first-in-first-out(FIFO) method to keep experience tuples in replay memory for every time step. The replay memory stores experience tuples for several episodes to ensure that the memory holds diversified experiences for different states. The experience tuples are sampled from replay memory randomly during Q-learning updates. The implementation of experience replay can remove correlations in the observation sequence and smoothing over changes in the data distribution by randomizes over the data [13]. In practice, the most recent N experience records are sampled uniformly and randomly from replay memory. The random sampling gives equal probability to all experience tuples.

2.6 Double Deep Q-Learning (DDQN)

In DQN, a batch of experience tuples are sampled in each learning iteration. The Q-learning network is updated based on the difference of the estimated future rewards. The sampling process is random and the frequent update of the network destabilises the learning process. The network is facing difficulties to reach convergence. Thus, a separate target network, $\hat{Q}$ is used to improve the stability of the main neural network [14]. The target network $\hat{Q}$ is used to estimate the discounted future rewards given the subsequent state s_{t+1} and action a_{t+1} at unit time step t. The target network $\hat{Q}$ is synchronized with the main network Q for every C Q-learning updates on the main network.

3. Project Timeline

A project timeline was planned and proposed as an overview of the planning and execution of this project. The project timeline in Table 1 shows the tasks need to be taken, its description and task deadline. The project timeline is visualized in Gantt chart as shown in Figure 2.

ID	Task	Description	Expected start date	Expected completion date	Dependencies
T1	Gathering project requirements	Arrange meeting and discussion with project supervisor to understand project objectives and clarify scopes of project	Aug 14, 17	Aug 20, 17	-
T2	Project planning	Set up a project plan	Aug 21, 17	Aug 27, 17	T1
T3	Research and exploration	Research on neural networks, CNN, reinforcement learning	Aug 28, 17	Sep 24, 17	-
T4	Implementation: CNN	Implement CNN	Sep 25, 17	Oct 22, 17	T3
T5	Implementation: RI for self-driving car	Implement reinforcement learning to maximize speed of self-driving car	Oct 23, 17	Nov 19, 17	T3, T4
	Exam break	Suspend project activities during examination period	Nov 20, 17	Dec 10, 17	-
T6	Data analysis	Analyze and verify collected statistics	Dec 11, 17	Jan 7, 18	T4, T5
T7	Interim report	Submit project progress	Jan 8, 18	Jan 14, 18	T6
T8	Formulate conclusion	Deduce a conclusion from data	Jan 15, 18	Jan 28, 18	T6
T9	Report submission	Submit report consisting methodologies and findings	Jan 29, 18	Mar 4, 18	T8
T10	Report amendment	Amend final report after review	Mar 5, 18	Apr 22, 18	T9
T11	Oral presentation	Present project to panel of judges	May 14, 18	May 16, 18	-

Table 1: Proposed project timeline.

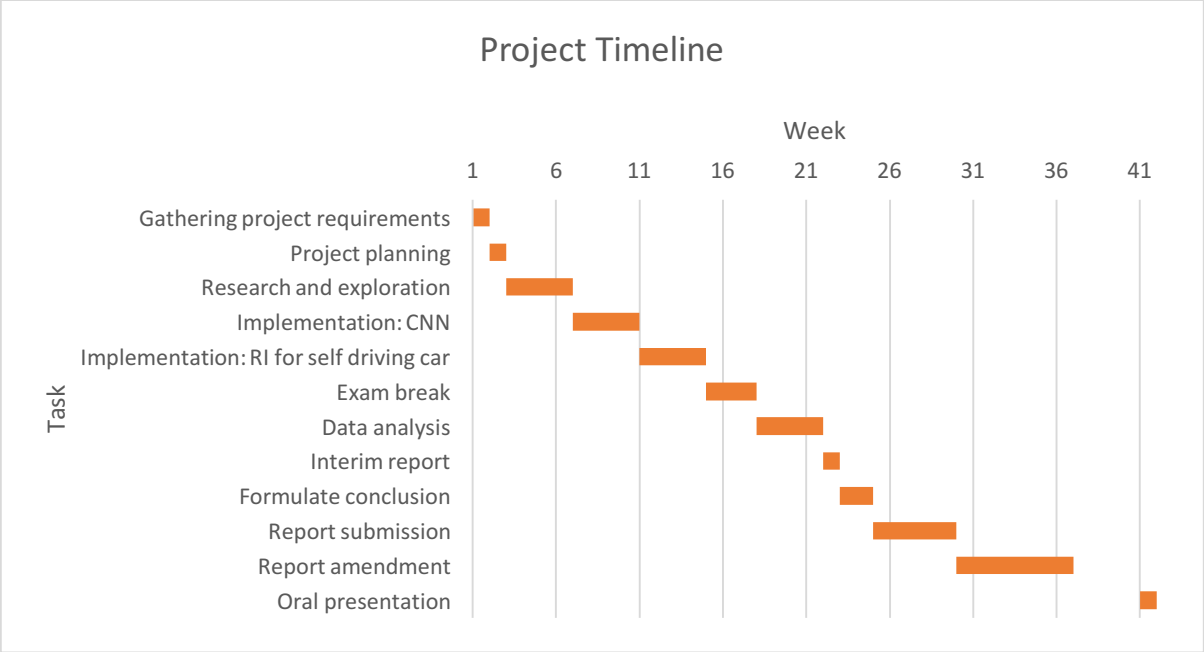


Figure 2: Gantt chart of proposed project timeline.

3.1 Actual Project Timeline

The actual project timeline deviated from the proposed project timeline at Task T6: Data analysis. The project timeline was then revised as shown in Table 2 and Figure 3.

ID	Task	Description	Expected start date	Expected completion date	Dependencies
T1	Gathering project requirements	Arrange meeting and discussion with project supervisor to understand project objectives and clarify scopes of project	Aug 14, 17	Aug 20, 17	-
T2	Project planning	Set up a project plan	Aug 21, 17	Aug 27, 17	T1
T3	Research and exploration	Research on neural networks, CNN, reinforcement learning	Aug 28, 17	Sep 24, 17	-
T4	Implementation: CNN	Implement CNN	Sep 25, 17	Oct 22, 17	T3
T5	Implementation: RI for self-driving car	Implement reinforcement learning to	Oct 23, 17	Nov 19, 17	T3, T4

		maximize speed of self-driving car			
	Exam break	Suspend project activities during examination period	Nov 20, 17	Dec 10, 17	-
T6	Data analysis	Analyze and verify collected statistics	Dec 11, 17	Feb 11, 18	T4, T5
T7	Interim report	Submit project progress	Jan 8, 18	Jan 14, 18	T6
T8	Formulate conclusion	Deduce a conclusion from data	Feb 12, 18	Mar 11, 18	T6
T9	Report submission	Submit report consisting methodologies and findings	Mar 12, 18	Mar 26, 18	T8
T10	Report amendment	Amend final report after review	Mar 27, 18	Apr 22, 18	T9
T11	Oral presentation	Present project to panel of judges	May 14, 18	May 16, 18	-

Table 2: Revised project timeline.

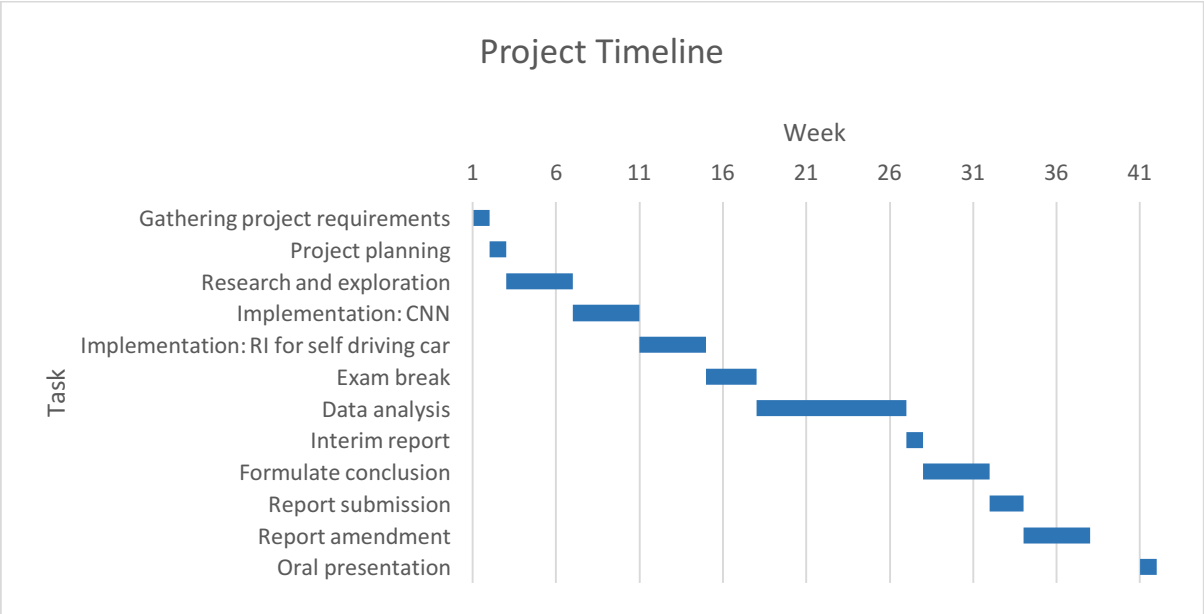


Figure 3: Gantt chart of revised project timeline.

The deviation of the project timeline was caused by the unexpected time taken for the learning process. A complete learning and evaluation process of a model took around 48 hours on one core of graphic processing unit (GPU). Although the learning and evaluation process can be accelerated with the help of GPU, most of the algorithm and simulation logics run on CPU. The time taken for computation with Tensorflow was shortened and the bottleneck of the learning process cannot be accelerated.

4. Algorithm

Reinforcement learning is the learning approach used in this project. Algorithm from [8] is used to interact with the simulator and use the surrounding environment states as the input of the algorithm. The S number of the most recent states s_t at time step t are supplied to as input. The available action options are Accelerate (**A**), Decelerate (**D**), Left (**L**), Right (**R**) and Maintain (**M**). Epsilon greedy exploration is implemented to maximize the exploration and ensure agent experiences more possible states in the state domain. An action a_t is randomly chosen from the action options as the agent's decision when the probability is lower than the current epsilon-greedy probability. The epsilon-greedy probability is uniformly decreases with number of processed states. Otherwise, an action with highest Q-value among the other options is chosen. The chosen action is then relayed to the simulator and the resulting state s_{t+1} and reward r_t is recorded. Reward r_t is the difference of score at time step t and $t-1$. The experience tuple (s_t, a_t, r_t, s_{t+1}) is appended to the replay memory $\mathcal{D}$ with capacity N. When the replay memory is full of experience tuples, the replay memory acts as a queue and replace the least recent experience tuple with most recent experience tuple.

The learning algorithm starts at F frames so that the learning process does not over fit the limited experiences in the replay memory $\mathcal{D}$. Experience replay is deployed to apply Q-learning update gradually. Learning process samples the most recent N experience rows in the replay memory. The goal of the reinforcement learning agent is to perceive the environment in the simulator and the optimal action that maximises the future reward is chosen. The optimal Q-value function $Q^*(s, a)$ can be modelled by a deep learning model as the nonlinear function approximator of the Q-learning algorithm. Neural networks are used to approximate the optimal Q-learning network. Thus, the parameterized neural network is used to approximate the optimal Q-learning network. The optimal target value, y can be formulated as below.

$$y = r + \gamma \max_{a'} Q^*(s', a';\ \theta_i^-) \tag{11}$$

Double Deep Q-learning Network (DDQN) is implemented to stabilize the learning process. It is because the Q-learning update under DQN may lead to the divergence of the action-value function. DDQN suggests having a separate network to generate target y_i in the Q-learning update. The target network $\hat{Q}$ generates the target y_i and synchronizes with the main network Q every C updates.

Frame-skipping technique [15] is adopted during training. The agent reacts with the surrounding environment every k^{th} frame to increase the response time of agent's reaction. In this project, the actions for skipped frames are determined in Table 3.

Action at k^{th} frame	**Action for skipped frames**
Accelerate (A)	Accelerate (A)
Decelerate (D)	Accelerate (A)
Left (L)	Maintain (M)
Right (R)	Maintain (M)
Maintain (M)	Maintain (M)

Table 3: Actions for skipped frames with respect to action at k^{th} frame.

5. Model Architecture

The proposed model is influenced by the model proposed in [8] and [16]. The baseline model follows the network design listed in [8]. As shown in Figure 4, the baseline network takes the environment mapping of four most recent states as input, which is a set of 4×36×3 state. The state consists of Boolean values, in which true (1) if there is obstacle in the vision field and false (0) if there is not. The first hidden layer convolves 16 filters of 3×3 and stride 2. Rectified Linear Unit (ReLU) is applied to the outputs of the first hidden layer. The outputs of first hidden layer are flatten and passed to the fully-connected layer. The fully connected linear layer comes with one output for five actions.

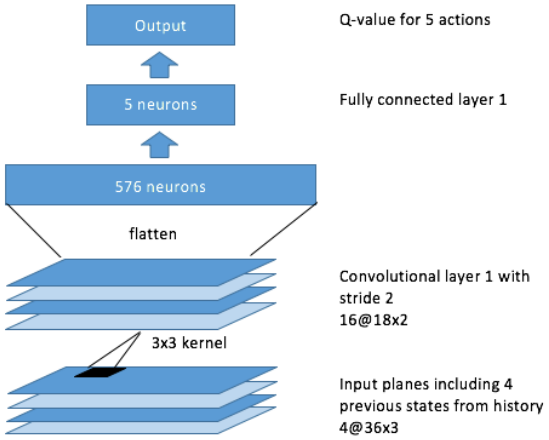


Figure 4: Baseline model with CNN architecture with one convolutional layer and one fully-connected layer.

The proposed model is implemented as shown in Figure 5. The network is enhanced with additional convolutional layer and fully-connected layer. The second convolutional layer compresses the output from the first convolutional layer to extract important features from the inputs. It passes the important features to fully-connected layer. The two fully-connected layers of the model brings the ability to approximate nonlinear function of action-value function. The additional sub-network is added to accept action stack as input. Action stack is a stack of action memory which stores the four most recent actions taken. The rational is to improve the stability of the agent from switching lane at high frequency.

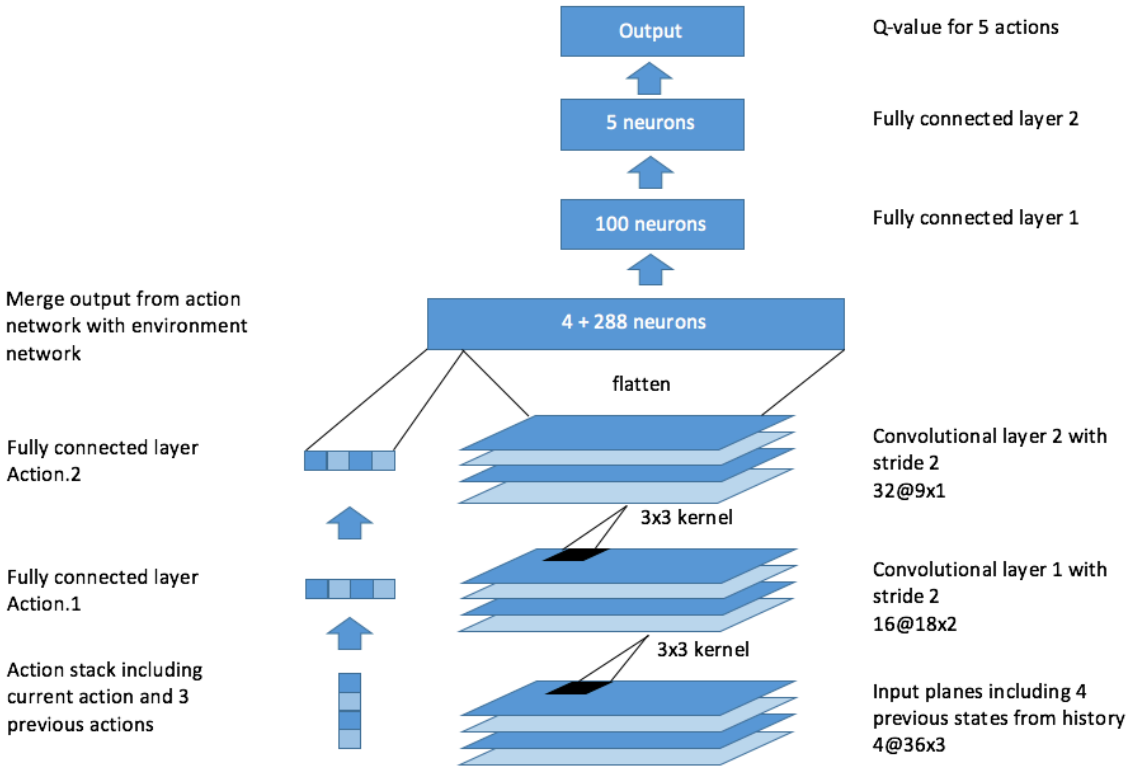


Figure 5: Proposed model with CNN architecture with 2-layer fully-connected for action stack and two convolutional layer and two fully-connected layer.

6. Experiments

6.1 Experimental Setup

The experiment is carried out in simulation program as shown in Figure 6. The simulator simulates seven-lane traffic condition with at most N number vehicles at time step t. Cars with different driving behaviour are deployed randomly to interact with agent-controlled car. It helps agent to generalize strategy and policies to maximize speed of self-driving car. Each model is trained with 2000 episodes and each episode consists of at most 4000 frames. During training, the agent makes decision every 4^{th} frame and during evaluation. It allows agent to react to environment with the human response rate. The aim of the experiment is to find the hyperparameters which give the best performance to the self-driving car-agent. The source code of the simulator can be retrieved from [17].

The visual input of the model is environment state of current lane and the adjacent lanes on the left and on the right. The visual input is sufficient for agent to make decision from an ego-centric perspective[18]. The visual input detect vehicle up to 20 meters ahead and 10 meters behind.

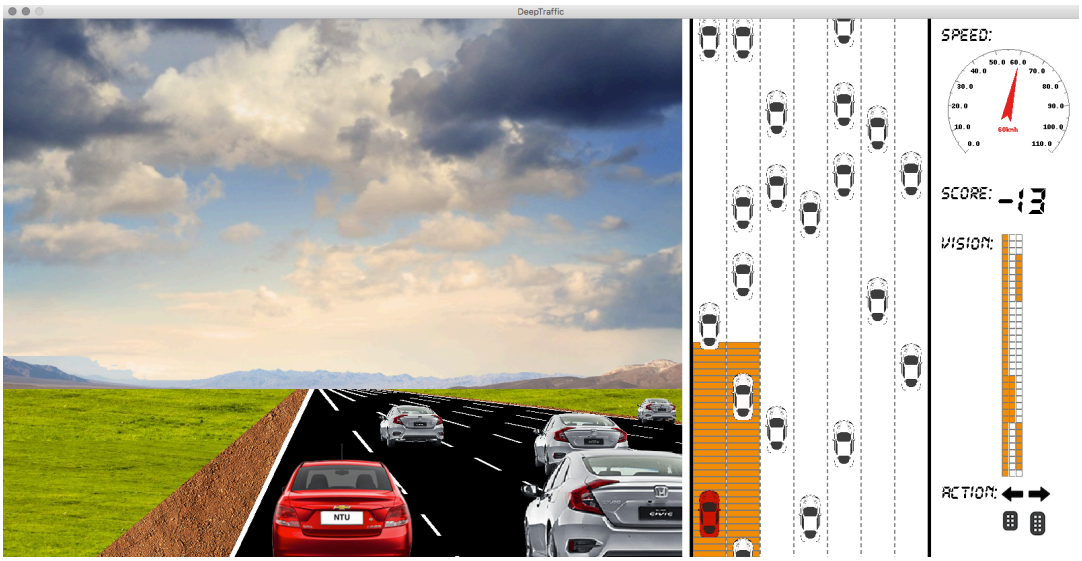


Figure 6: Screenshot of simulator running in macOS High Sierra. The left part of the simulator shows the third-person view of the agent-controlled car(red). The middle part of the simulator shows the bird's eye view of the road with agent-controlled car (red car). The right side of the simulator shows the speed of the agent-controlled car, current score of the episode, vision input to model, and action taken by the agent.

Traffic condition	Description
1. Light traffic	Maximum 20 cars are simulated with plenty room for overtaking.
2. Medium traffic	Maximum 40 cars are simulated with lesser chance to overtake other cars.
3. Heavy traffic	Maximum 60 cars are simulated to simulate heavy traffic.

Table 4: Traffic condition in the simulation.

Driving behaviour	Description
Stick to certain speed	The car follows certain speed within the acceptable speed and does not speed most of the time
Stick to the maximum speed	The car follows the maximum allowed speed.
Switching lane regularly	The car follows the maximum allowed speed and regularly switch lane to further speeding.

Table 5: Driving behaviour of non-agent-controlled car in the simulation.

Hyperparameter	Value	Description
Minibatch size	32	Number of experiences selected for each iteration.
Replay memory size	1000000	Replay memory size to store the most recent experience tuples, replace the least recent memory with latest memory when it is full.
Agent history length	4	Number of past states experienced that are given to the network.
Target network update frequency	10000	Update frequency of the target network after the main network is updated.
Discount factor	0.99	Discount factor to discount future rewards in Q-learning update.
Learning rate	0.0005	RMSProp learning rate.
Gradient momentum	0.95	RMSProp gradient momentum.
Initial exploration	1	Initial value of exploration probability in epsilon greedy exploration
Final exploration	0.1	Minimum value of exploration probability in epsilon greedy exploration
Final exploration frame	2000000	Number of frames over in which exploration value decreases linearly.
Replay start size	100000	Size of experience tuples in replay memory before learning starts.
Number of cars in scene	40	Maximum number of non-agent-controlled cars appear in the simulation at a time. (Traffic condition 2)
Constant penalty	0	Penalty applied for each time step.
Reward	1	Reward applied when agent overtake one car.
Emergency brake penalty	0	Penalty applied when agent applies emergency brake.
Line switching penalty	-0.00001	Penalty applied when agent chooses direction action (left or right) alternatively within the most recent action history of size 4.

Table 6: The list of hyperparameters and their values used in the experiments.

The baseline model of the experiment is model with default hyperparameters listed in Table 6. The default hyperparameters are inspired by [13] and modified based on the context this project. Table 7 shows the difference in configuration for each experiment with baseline model.

Experiment	Attribute	Value
1	Learning rate	0.0001
2	Learning rate	0.00025
3	Learning rate	0.0005
4	Learning rate	0.00075

Table 7: Experiments with different configurations.

6.2 Evaluation Metrics

The evaluation of models can be categorized into evaluation during training and evaluation during testing. The loss of the model for each training is one of the main evaluation metrics. It is to ensure the model reach convergence throughout the training. Besides that, the average speed of agent during training is another metrics to measure the optimal learning rate to reach convergence. The score of each episode is another crucial metric to measure the capability of model to learn the optimal policy[19].

For the evaluation during testing, the models are evaluated by its average speed over 200 episodes under each traffic condition as listed in Table 4. The number of cars simulated in scene are varied during the testing to evaluate the ability of the agent to maximise its speed in different traffic condition. Moreover, the average number of emergency brake applied during testing phase shows reflects the user experience for the journey in each episode. An emergency brake is counted when the difference of the speed of the agent's car and the car in front is greater or equal than 20km/h and emergency brake is applied to prevent collision and keep a safe distance from any car in front of agent's car.

7. Results

Several experiments with different configuration were conducted in the simulator. The model of each experiment was trained under identical simulation environment.

7.1 Average speed

Based on Figure 7, the average speed of agent for each episode increased gradually over episode for all models. The models were able to improve the maximum speed over training episodes. It also shows that Experiment 4 with the highest learning rate managed to increase its speed earlier than other models. Besides that, the performance of baseline model was better than Experiment 1. It is because the baseline model is less complex than the advanced model in Figure 5. All models managed converged at the end of training episodes.

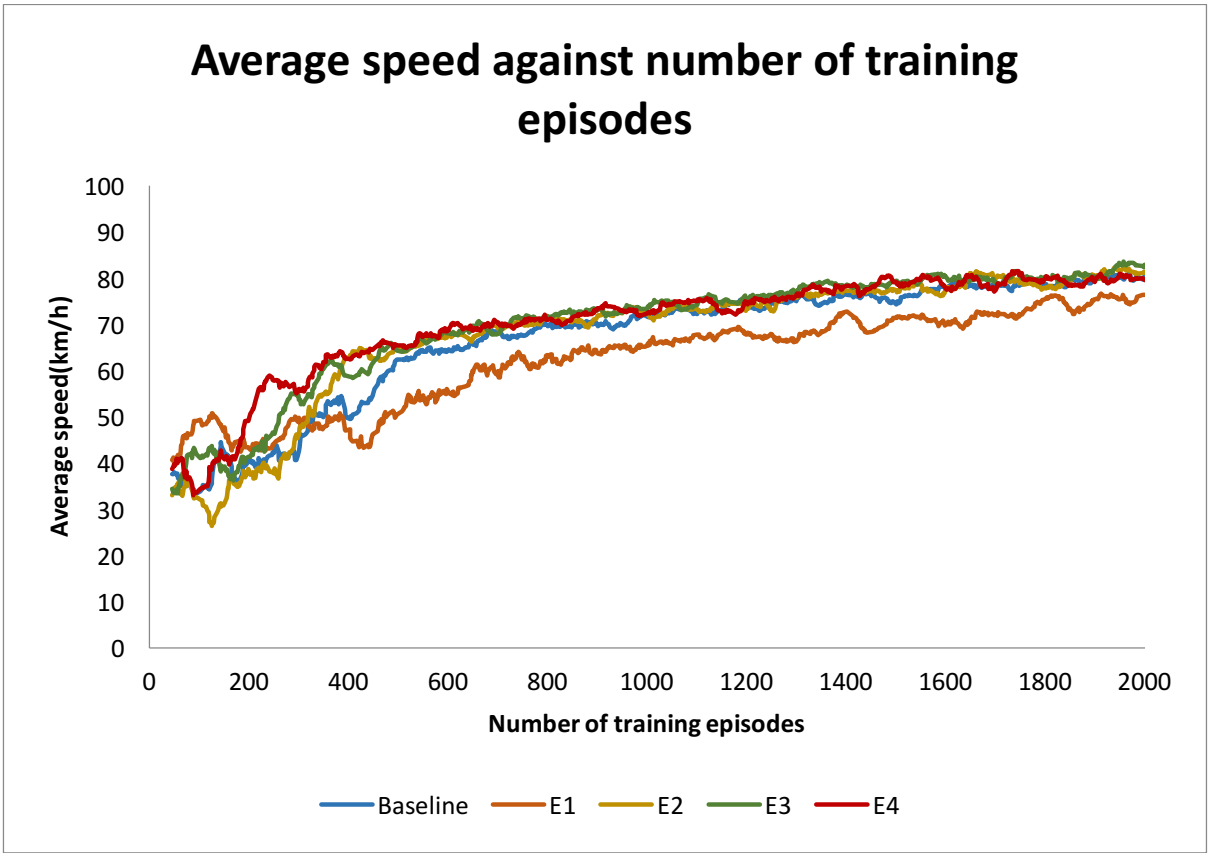


Figure 7: Average speed of agent-controlled car during training.

7.2 Score

Score of each episode is cumulative rewards received. Agent receives reward (+1) when it overtakes a car and receives penalty (-1) when it is overtaken. Other penalty such as line switching penalty is applied when agent takes alternative direction action in the most recent action history. Based on Figure 8, the score of model in each experiment increases over training episodes. The score increases from negative scale to positive scale when model learnt the strategy to maximize the rewards and avoid penalties. Score of Experiment 4 increased rapidly at the early stage of training. Most of the models managed to maximize the score greater than 100 except baseline model. Baseline model took more time to maximize the score but its score was the lowest among the experiments.

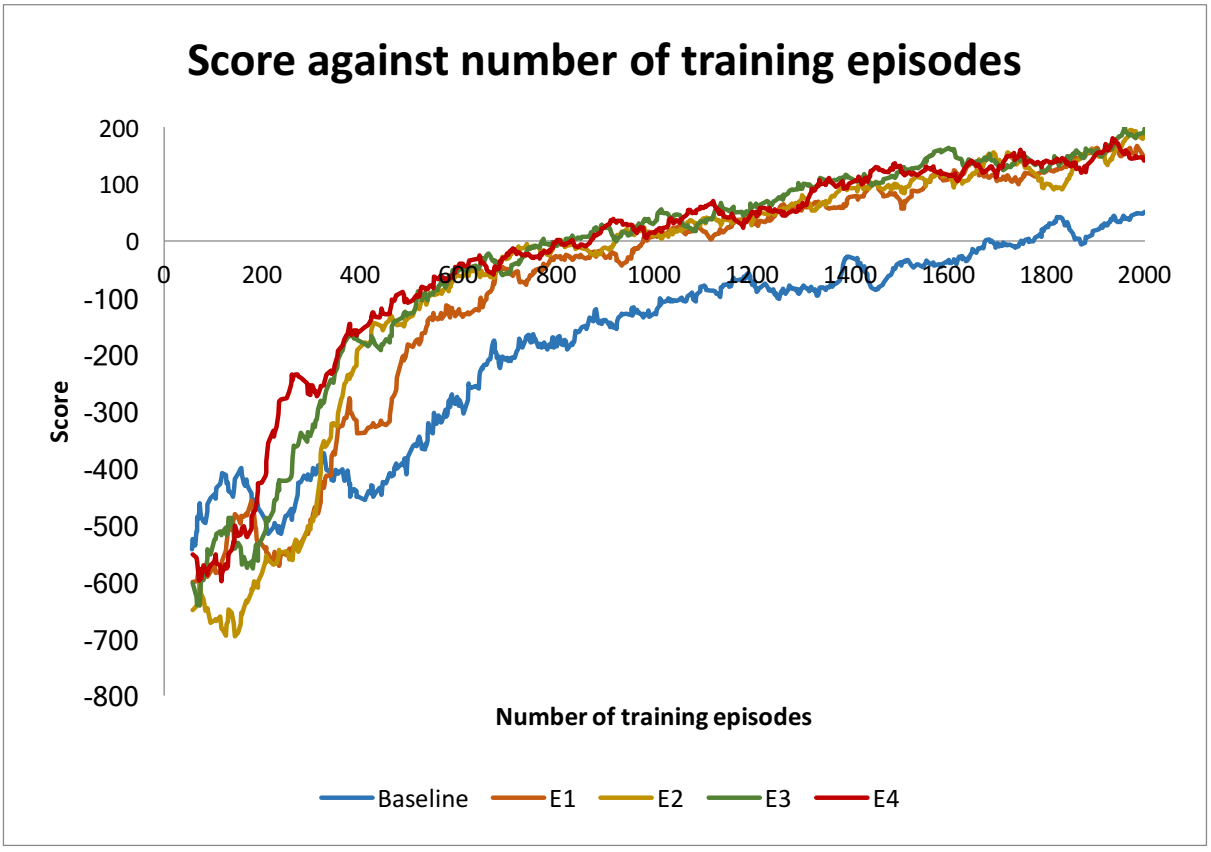


Figure 8: Score of self-driving car at the end of episode during training.

7.3 Training loss

Losses over training is another indicator to show how close a model behaves to the environment. Figure 9 shows the training losses decreases in overall. The spikes in the training losses was resulted by the regular update of the target network to the main network. The models were designed with DDQN methodology and a separated target network was implemented during training to improve the stability of the network. The target network was synchronized with the main network every 10000 trainings.

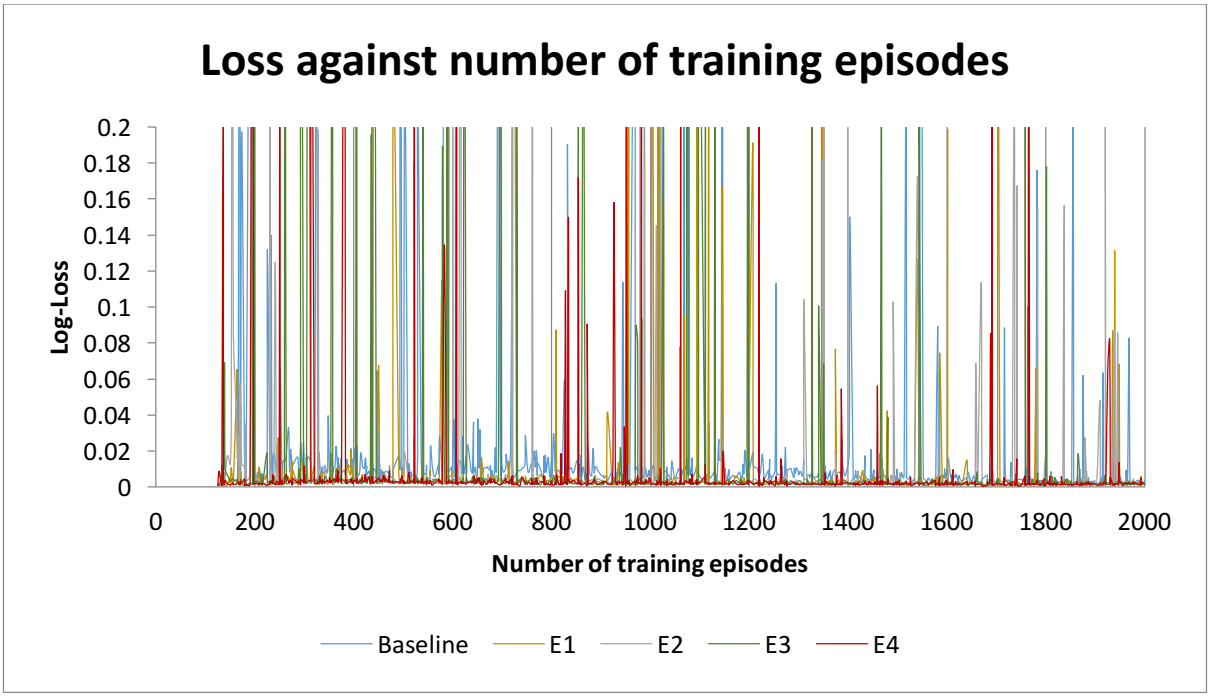


Figure 9: Log-scaled loss against number of training episode.

7.4 Evaluation

The models were examined during evaluation phase. The models were tested with the average speed of the agent and the average number of emergency brake applied. Evaluation was carried out under 3 different traffic conditions as shown in Table 4. The results are tabulated as shown in Table 8. Based on the result of experiments, it shows that the average speed decreases and number of emergency brakes increases when traffic condition is crowded. Compared to baseline model, the average speed of baseline model is lower but it applied less number of emergency brake. It shows that the models from experiments are more likely to accelerate to maximize its speed and hence, the average number of emergency brake applied increases.

Experiment	**Average speed during evaluation (km/h)**			**Average number of emergency brake applied**		
	Traffic condition 1	**Traffic condition 2**	**Traffic condition 3**	**Traffic condition 1**	**Traffic condition 2**	**Traffic condition 3**
Baseline	71.98	64.28	60.62	2.99	4.70	6.72
1	**103.42**	**87.96**	**72.94**	8.56	22.67	23.92
2	102.64	86.39	72.76	7.98	20.96	22.71
3	98.87	85.64	71.10	7.02	19.55	19.755
4	98.19	81.66	70.48	**6.73**	**16.705**	**19.68**

Table 8: Comparison of the average speed of agent with different configuration during evaluation.

The chart in Figure 10, Figure 11 and Figure 12 show the correlation between the average speed of the model and the average number of the emergency brake applied. It shows that the baseline model was underperformed compared to the other models in the experiment. The model from Experiment 4 is the most stable model compared to the other models in the experiment in term of its average number of emergency brake applied. However, its performance in term of average speed is at most 8.6% lower than the fastest model in three different traffic conditions.

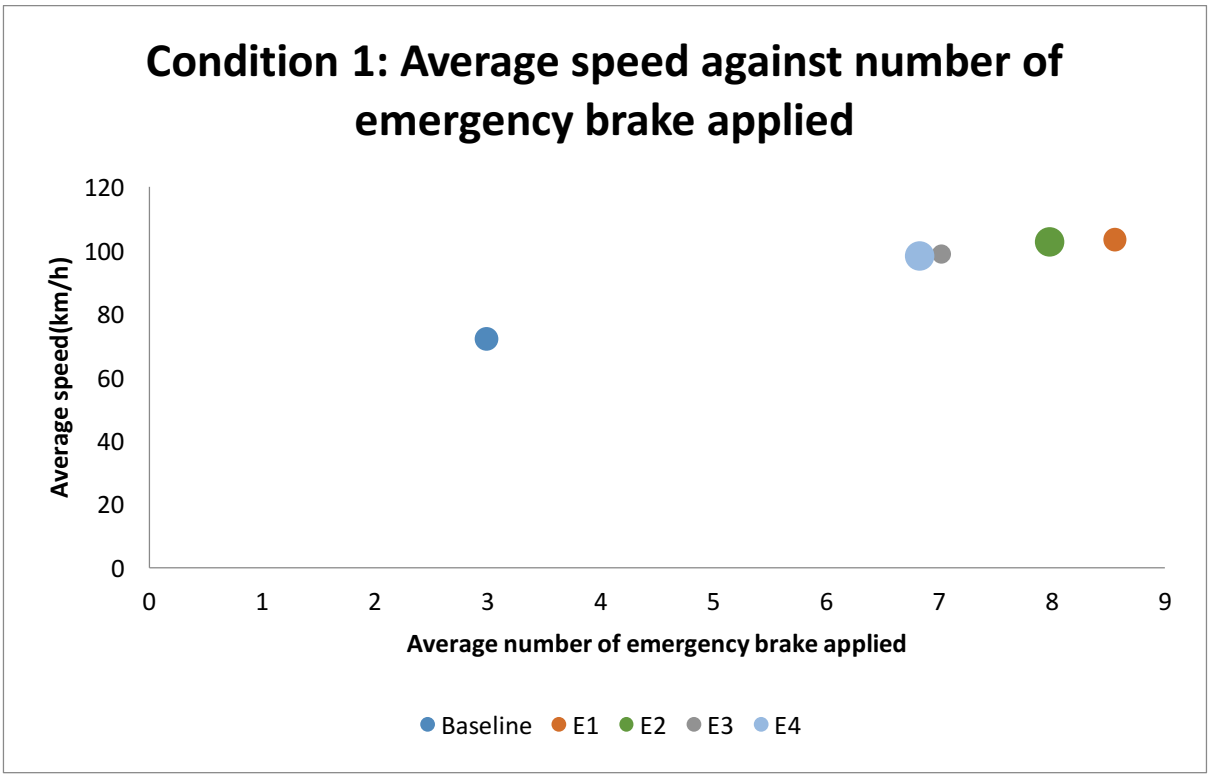


Figure 10: Average speed against average number of emergency brake applied under Condition 1.

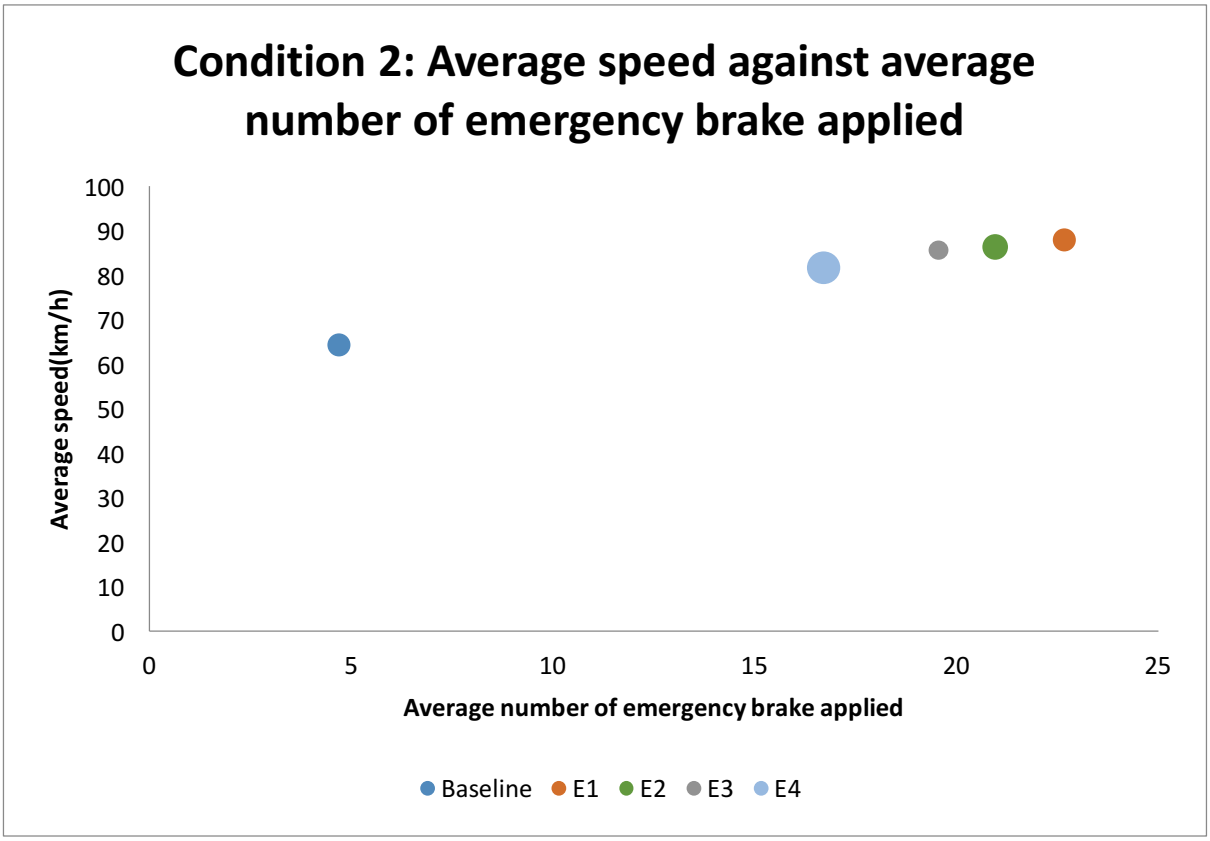


Figure 11: Average speed against average number of emergency brake applied under Condition 2.

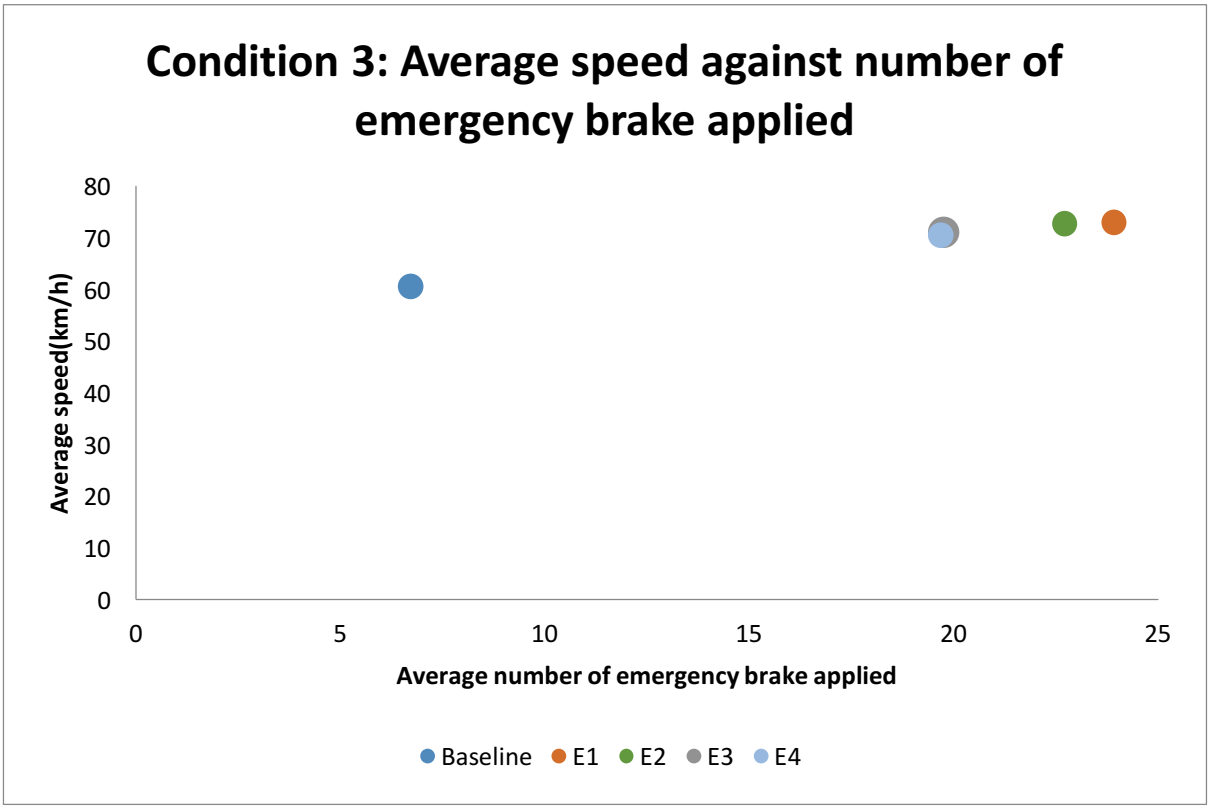


Figure 12: Average speed against average number of emergency brake applied under Condition 3.

8. Discussion

Based on the training data shown in Section 7, it shows that the model in each experiment managed to learn the optimal policy to maximize the speed of the autonomous car-agent. It discovered the correlation between the speed of the autonomous vehicle and the scoring system of the simulation. In order to maximize the score of the agent for each game, it learnt and performed the optimal action at a given environment state to maximize its score. The metrics measured during training phase are used to monitor the learning process of models to convergence. The models from the experiments show that each model converges to generate the optimal policy for the autonomous car-agent.

Based on the evaluation data, the baseline model is a simple convolutional neural network (CNN) model with two convolutional layers and one fully-connected layer. It used its limited capabilities in term of the network size to generate the optimal policy that maximizes its objective based on the information from the complex environment. It was underperformed compared to the other models in term of the average speed under three simulated traffic conditions. However, it has the lowest average number of emergency brake applied during the evaluation phase. It is due to the low average speed of the model led to lesser incidents in which the difference of the speed of the agent's car and the car in front is greater and equal to 20km/h.

The models from the experiments show that they have average speed from the range from 81.66km/h to 87.96km/h under traffic condition 2. These models have higher average speed compared to baseline model under three traffic conditions. The proposed model used in the experiments is more complex to capture features from the environment states in the simulation. The model from Experiment 1 has the highest average speed compared to the rest of the models under three traffic conditions. The model from Experiment 1 was trained with lowest learning rate compared to other models. The lower learning rate in Experiment 1 made the model achieve generalization of the simulation environment because the learning process was less sensitive to changes and avoid overfitting situation during training phase. It shows that the model learnt the most optimal policy to maximize its reward, and subsequently produce the highest average speed. However, the average number of emergency brake applied is higher when the average speed of the autonomous car is higher. It shows the direct proportionality between these two properties. The higher speed of the car led to higher occurrences of emergency brake, which will affect the user experience of autonomous car.

9. Conclusion

This paper demonstrated the implementation of deep learning model trained with reinforcement learning to maximize speed of self-driving car in multi-lane expressway. This implementation proposed the use of state-of-the-art deep learning technique to approach complex tasks with clear objective. The generated model from Experiment 1 is equipped with high performance in achieving its objective to maximize the speed of autonomous car. The model can be applied with artificial intelligent agents to control an autonomous car in the dynamic environment. The generated models were trained in the simulation environment in which the sensors collect complete and correct data as sensory input. The models are not tolerated to noises from input data. However, the data from sensory input has higher probability to be interfered, modified, corrupted or lost due to the limitations of sensors in real world environment. Besides that, the proposed model placed the speed maximization as the main objective for the reinforcement learning. However, the user experience of the autonomous car is another important aspect to be prioritized as one of the objectives in reinforcement learning.

10. Recommendations

The future work can focus on improving user experience as one of the objectives in the implementation of self-driving car-agent. A penalty can be added to the reinforcement learning algorithm to penalize the learning process when user experience degrades during the training phase. The suggested metrics such as average number of emergency brake and average number of lane switching alternatively can be used to portray the user experience of the self-driving car.

Besides that, the future work can also focus on accepting corrupted data as the model input. It is crucial when the external factors such as weather can affect the precision of sensors and noises in collected input. It can improve the stability and accuracy of the model to interact with the dynamic environment on the road. Denoising autoencoders can be implemented to remove noise and extract important features from sensory input with unsupervised learning.

References

1 Sallab, A.E., Abdou, M., Perot, E., and Yogamani, S.: 'Deep reinforcement learning framework for autonomous driving', Electronic Imaging, 2017, 2017, (19), pp. 70-76

2 Littman, M.L.: 'Markov games as a framework for multi-agent reinforcement learning': 'Machine Learning Proceedings 1994' (Elsevier, 1994), pp. 157-163

3 Mahadevan, S., and Connell, J.: 'Automatic programming of behavior-based robots using reinforcement learning', Artificial intelligence, 1992, 55, (2-3), pp. 311-365

4 Srivastava, N., Mansimov, E., and Salakhudinov, R.: 'Unsupervised learning of video representations using lstms', in Editor (Ed.)^(Eds.): 'Book Unsupervised learning of video representations using lstms' (2015, edn.), pp. 843-852

5 Donmez, P., Carbonell, J.G., and Schneider, J.: 'Efficiently learning the accuracy of labeling sources for selective sampling', in Editor (Ed.)^(Eds.): 'Book Efficiently learning the accuracy of labeling sources for selective sampling' (ACM, 2009, edn.), pp. 259-268

6 Ng, A.Y.: 'Shaping and policy search in reinforcement learning', University of California, Berkeley, 2003

7 Hermanns, H.: 'Markov Chains': 'Interactive Markov Chains' (Springer, 2002), pp. 35-55

8 Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M.: 'Playing atari with deep reinforcement learning', arXiv preprint arXiv:1312.5602, 2013

9 Sutton, R.S.: 'Learning to predict by the methods of temporal differences', Machine learning, 1988, 3, (1), pp. 9-44

10 Boyan, J.A., and Moore, A.W.: 'Generalization in reinforcement learning: Safely approximating the value function', in Editor (Ed.)^(Eds.): 'Book Generalization in reinforcement learning: Safely approximating the value function' (1995, edn.), pp. 369-376

11 LeCun, Y., Kavukcuoglu, K., and Farabet, C.: 'Convolutional networks and applications in vision', in Editor (Ed.)^(Eds.): 'Book Convolutional networks and applications in vision' (IEEE, 2010, edn.), pp. 253-256

12 Bakas, S., Zeng, K., Sotiras, A., Rathore, S., Akbari, H., Gaonkar, B., Rozycki, M., Pati, S., and Davazikos, C.: 'Segmentation of gliomas in multimodal magnetic resonance

imaging volumes based on a hybrid generative-discriminative framework', Proceeding of the Multimodal Brain Tumor Image Segmentation Challenge, 2015, pp. 5-12

13 Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., and Ostrovski, G.: 'Human-level control through deep reinforcement learning', Nature, 2015, 518, (7540), pp. 529-533

14 Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N.: 'Dueling network architectures for deep reinforcement learning', arXiv preprint arXiv:1511.06581, 2015

15 Bellemare, M.G., Veness, J., and Bowling, M.: 'Investigating Contingency Awareness Using Atari 2600 Games', in Editor (Ed.)^(Eds.): 'Book Investigating Contingency Awareness Using Atari 2600 Games' (2012, edn.), pp.

16 Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., and Zhang, J.: 'End to end learning for self-driving cars', arXiv preprint arXiv:1604.07316, 2016

17 https://github.com/songyanho/Reinforcement-Learning-for-Self-Driving-Cars

18 Chen, C., Seff, A., Kornhauser, A., and Xiao, J.: 'Deepdriving: Learning affordance for direct perception in autonomous driving', in Editor (Ed.)^(Eds.): 'Book Deepdriving: Learning affordance for direct perception in autonomous driving' (2015, edn.), pp. 2722-2730

19 Yu, A., Palefsky-Smith, R., and Bedi, R.: 'Deep Reinforcement Learning for Simulated Autonomous Vehicle Control', Course Project Reports: Winter, 2016, pp. 1-7