

Exploring Deep Q-Networks using MIT's DeepTraffic Self-Driving Car Machine Learning Competition

Udacity - School of AI
Machine Learning Nanodegree
Christopher Ohara
c.a.ohara@student.tue.nl
December 24, 2018

Abstract—In recent times, self-driving cars are becoming more feasible. With benefits being more eco-friendly, efficient, and safe, it is easy to see their growing popularity. However, a lot of progress still needs to be obtained prior to their mass integration into modern society. More specifically, there will be a transitional phase in which cars must first become semi-autonomous via the utilization of advanced driver-assistance systems (ADAS) that handle operations such as adaptive cruise control or providing take-over requests (TOR) to transition the semi-autonomous control back to the driver. The ability for autonomous vehicles to complete these tasks is in the realm of machine learning. Deep-reinforcement learning is a highly critical field of machine learning that is applied to solving the complex stochastic environmental conditions (semi)-autonomous vehicles and their drivers find themselves in. The goal of this project is to investigate the possibilities of deep-reinforcement learning (Deep Q-Learning) as a solution to autonomous navigation using the DeepTraffic neural network framework. The results of the project were that the reward quality rate increased from 0.52 to 2.27 and the average speed of the simulated vehicle increased from 51.69 mph to 74.48 mph.

Index Terms—DeepTraffic, DQN, Deep Learning, Deep Q-Learning, Self-Driving Cars, Autonomous Systems, AI, Neural Networks.

1 INTRODUCTION

1.1 Problem Statement

There will unlikely be a transitional phase prior to the advent of all vehicles having auto-pilot capabilities. More specifically, it is not feasible that on a single day all traditional (non-autonomous) vehicles will be replaced, resulting in a time period in which some of the vehicles have the capability while others do not. One approach to resolve this issue is the implementation of *Q-learning*, a reward-based deep-reinforcement learning technique. Initially, it can be assumed that less than 1% of vehicles will be autonomous, leaving the other 99% of vehicles with human drivers. Human drivers have completely stochastic and non-deterministic (almost arbitrary) behavior when driving; changing lanes or having a variable speed almost on a whim.

Q-learning can approach this problem on two levels. First, the technique can utilize a neural network architecture that is able to be used in the completion of path planning and state transitions. This allows for the autonomous vehicle to maneuver and avoid other vehicles that are on the path towards the goal. The second approach, which is less feasible due to the arbitrary behavior of humans, is to attempt at “predicting” human behavior and avoiding collisions as a result.

1.2 Background

Q-Learning is a machine learning technique under the umbrella of reinforcement learning. *Deep Q-Learning* (DQN) is a flavor of *Q-learning* that was designed by Google's *DeepMind* research group in 2014 (ref here). In *DeepMind*'s patent and research, they showed that machine learning agents were able to learn to play Atari 2600 games at a human equivalence to expert as a result. Since then, *DQN*'s have been used to teach robots to walk, play physical games (like football and ping-pong), as well as complete other tasks in which the robot is “self-taught.”

The commonly accepted notation for the *Q-Learning* value iteration update equation is given as:

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a))$$

where:

$Q^{new}(s_t, a_t)$ is the current (new) reward quality as a function of the state, action, and time.

$Q(s_t, a_t)$ is the previous (old) reward quality.

α is the learning rate.

γ is the discount factor.

r_t is the reward value.

$\max_a Q(s_{t+1}, a)$ is the estimation of the optimal future value.

In general, the goal of the agent is to maximize its total future reward by influencing the current actions based

on the current state and estimated maximum reward. This reward is given as the potential reward from the weighted sum of the expected values ($E(x)$) from *all* future steps including the current state. The state-action combination to update the reward quality is given by:

$$Q : S \times A \rightarrow \mathbb{R}$$

Each time the agent enters a state, it selects an action and observes the reward. The value of the reward determines the next state of the agent, and Q is updated accordingly, as to maximize the future potential. γ is a critical parameter (with range $0 \leq \gamma \leq 1$) that allows the agent to assign rewards that arrive "quickly" with a higher value than other options. This function (γ) can help the algorithm to quickly arrive at a maximized reward, but at the cost of "exploration," as the agent will continue to seek the highest reward. In the case of self-driving cars or walking Nao robots [3], this can lead to the algorithm prematurely determining the "optimal" path or sequence of actions.

1.3 DeepTraffic - Project Overview

DeepTraffic is a deep-reinforcement learning competition that is part of MIT's Deep Learning for Self-Driving Cars course. The course and materials are available to the general public, and the public is also allowed to physically participate in lectures. The goal of the project is to create a neural network that controls an autonomous vehicle through a densely populated (19 other vehicles) highway, while learning to navigate at the maximum possible speed (80 mph) while ensuring all safety specifications are met.

The network has been initialized with default hyperparameters and layers (described in the Benchmarking section). The network's task is to inform the autonomous vehicle (red) when it should change lanes or change its speed. Initially, the car is driving "blind," as it has not been instructed to receive information in its sensors. MIT has also noted that they "intentionally changed the network to perform badly," which is evident if the model is instructed to train and evaluate without changing any parameters or layers.

The simulation used *frames* as internal measure of time, meaning the results of training the network should be the same for any computer/network (which is ideal for validation and replication). The network itself is processed in the browser, utilizing the ConvNetJS library for in-browser neural networks [2].

1.4 Metrics

For evaluation, two metrics are be used. The first metric was to meet the performance requirements that MIT has set in order to receive credit for the assignment. This values is to have an average speed exceeding 65.0 mph. A second performance metric is to compare the results with the global competition winners. The three winners achieved average speeds of 74.48, 74.04 and 73.59 mph. Interestingly enough, all three winners won a scholarship to Udacity's Self-Driving Car Nanodegree Term 1. Therefore, as a bare minimum, 65.0 mph should be exceeded as a key performance indicator.

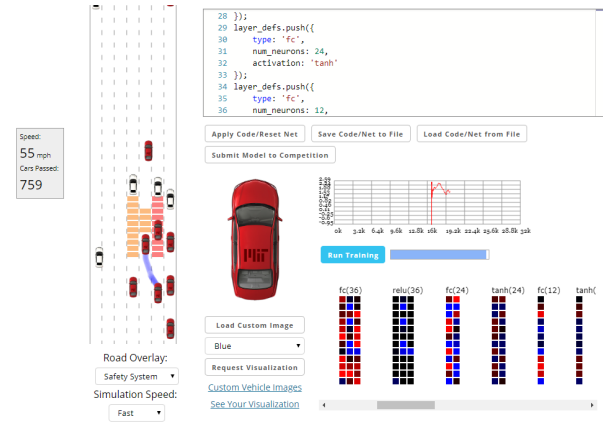


Fig. 1: DeepTraffic Learning Environment.

For more traditional metrics, the reward rate targeted with respect to iterations and time. In general, a higher reward rate is correlated with better performance (accuracy). Note that the same inputs were be used from both the naive benchmark model and the evaluation (improved) model.

DeepTraffic 1.0 Leaderboard

[DeepTraffic Game](#) - [About DeepTraffic](#) - [Back to Current Leaderboard](#)




	Purnawirman (74.48 mph) Winnings: Deep Learning book (Goodfellow, Bengio, Courville) - Udacity Self-Driving Car Engineer Program first term Comment: "I used a single hidden layer. The improvement seems to come from taking the data as a single snapshot (time window as 0). Spent some time on hyperparameter tuning, especially L2 regularization. Submitted the model several times, because the test scores have a big variance."
	Michael Gump (74.04 mph) Winnings: Udacity Self-Driving Car Engineer Program first term Comment: "I mainly played around with the L1 and L2 decay and that helped a lot. It was interesting how often the model would get stuck in suboptimal strategies and thought raising epsilon_min helped too."
	Jeffrey Hu (73.59 mph) Winnings: Udacity Self-Driving Car Engineer Program first term Comment: "I preprocessed to reduce the size of the input by taking the min of consecutive cells and fed that into a three layer fully connected network. Then I tried to make gamma as large as possible while playing with the other parameters to get the network to converge."

Fig. 2: Winners of the v1 DeepTraffic Competition with Scores.

2 ANALYSIS

2.1 Data Exploration

The machine learning field that the solution is being designed for is under that of reinforcement learning (specifically deep-reinforcement learning). For this project, MIT's DeepTraffic competition data is selected as an input. The goal of DeepTraffic is to alleviate the stress and number of hours vehicles are stuck in traffic, though as an extension, the problem statement above can be addressed. This will lead to improved safety and quality of life for drivers.

The state space can be generally considered as a regression problem, since while the other vehicles have discrete initial feature values (position, velocity, acceleration, and behavior), they operate with non-linear behaviors that are continuous in nature. *Q-learning* utilizes a model-free, off-policy structure within discrete spaces. The algorithms must find a policy with the maximum expect return based on the previous inputs. The inputs are user-developed, though the non-autonomous vehicles positions, accelerations, and

velocities will be the source of emphasis. The output will be a mapping policy that best returns the optimal path, with characteristics (outputs) such as move left, move right, slow down, or continue full-speed ahead.

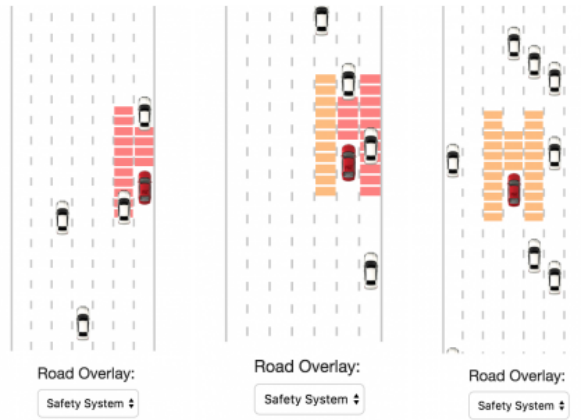


Fig. 3: Autonomous agent (red) sensory readings for nearby lanes.

DeepTraffic utilizes neural networks to train a vehicle to drive as quickly as possible through dense highway traffic. Using *Q-learning*, the vehicle learns how to navigate efficiently, while attempting to maintain the maximum speed limit. Safety protocols are ensured, making it unable to place the autonomous vehicle or other drivers at risk for an accident. The neural network hyperparameters and architecture is completely customizable, leaving all of the design choices up to the student, while enforcing the functional and quality requirements of the autonomous vehicles behavior. For the dataset, the manually driven cars are randomly spawned with various driving behaviors and velocities, making it an unlabeled dataset. The learning input range can also be changed, as the network can look far into the future, behind the car, or within specified limits.

In terms of data points, there are 20 other vehicles present at any given time with unique values for position, velocity, and acceleration. Note that the data range involves the numbers of cars present (and physically passed), which is a function of time and the number of iterations required to train the network. Therefore, the data range is variable that increases linearly with time, until the trained model is tested against the evaluation model.

In summary, the dataset contains data points of the pose, velocity, acceleration, and behavior of the other vehicles as well as the (semi-)autonomous vehicle during autonomous driving or teleoperation. Essentially, the dataset appears similar to a sensor fusion dataset.

2.2 Exploratory Visualization

One of the more tedious and difficult challenges in deep learning is parameter tuning. With a plethora of options, it can be a challenge simply isolating a parameter and changing the values (even with intuition). Furthermore, many parameters are not independent (i.e. they are a function of another parameter), as they relationship or ratio between two

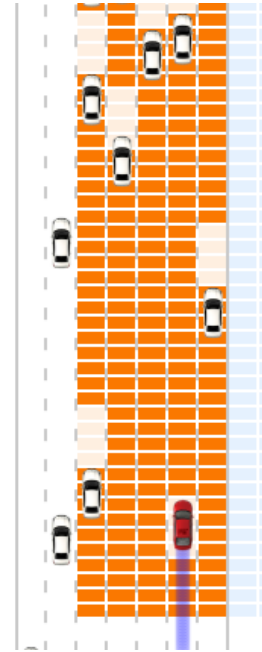


Fig. 4: Possible range of discrete points for the dataset including vehicle locations and mechanics.

parameters ultimately determines their behavior. Starting with the naïve model parameter, the input size was changed to find the impacts on the final speed. The model below in Fig.2 is a result of tuning multiple parameters while holding the speed and input size as dependent variables. As can be seen, there are several thresholds and a linear tendency to display improved behavior (higher maximum speed) up to a certain point. This is a key example of why increasing (or starting with the maximum) input size is not always beneficial to the final performance metric.

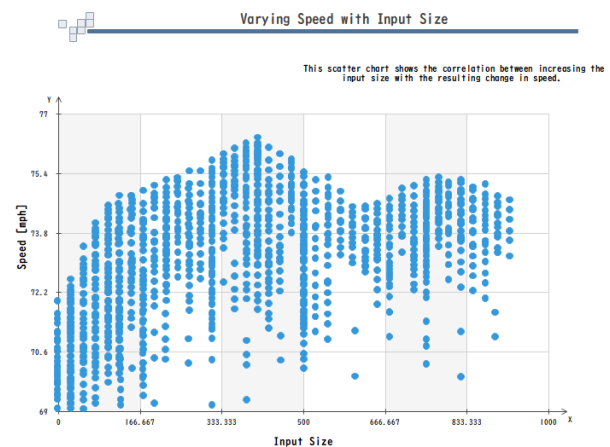


Fig. 5: Impact of Input Size on Maximum Speed

2.3 Algorithms & Techniques

Below, the primary algorithms and techniques are discussed. There are various other techniques and algorithms, though they are embedded (implicitly or optionally) within the following models:

2.3.1 Convolutional Neural Network

The primary algorithm used for this project is the convolution (in the CNN). The CNN uses a multilayering perception scheme and is bio-inspired from what was assumed a representation on how the brain works. CNNs are ideal for deep learning and computer vision problems, since they require minimal pre-processing. However, they do not innately have the ability to “remember,” which is a limitation seen within this project (and mitigated by saving the training sets over many iterations). Furthermore, CNNs are often trained over GPUs, which made this project feasible, as training locally would require a substantial amount of processing (which ultimately becomes a problem with time and energy consumption).

2.3.2 Recurrent Neural Network

One method to solve the issues with “memory” is to introduce a *Recurrent Neural Network (RNN)*. The RNN implements a long-short term memory (LSTM) technique to process sequences of inputs based on its internal state (effectively memory). RNNs are usually used in combination with CNNs (CNN encoder-RNN decoder) for speech recognition problems but due to their simple “memory” ability, they are suitable for trivial reinforcement learning problems. An RNN was briefly attempted in the project though there were issues with the states. Therefore, this will be discussed more in the *Conclusion* section.

2.3.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an incremental (iterative) method for optimizing around a cost feature. As this project uses rewards/costs, it is ideal to use an optimization process. The stochastic convention refers to the technique of randomly selecting samples instead of sequentially processing in a linear manner. The ConvNetJS processing model uses SGD with *Q-Learning* so that the agent learns to avoid states that lead to poor rewards over the course of time. SGD is used with *Backprop* (see below) in order to prevent an issue related to “normal” gradient descent where the *global minimum* of the *error function* becomes stuck in a *local minimum* (and never finds the *global minimum*).

2.3.4 Backpropagation

Backpropagation (Backprop) is utilized in order to implement *Temporal Difference Learning*. *Backprop* is usually used in systems/networks where the gradient needs weights to be calculated in order to minimize output errors via feedback. Usually, *Backprop* is used in supervised learning for modeling internal representations for arbitrary mappings of the input to the output. Generally, the *cost* feature is called the “*loss function*” for *Backprop*, but they are ultimately synonymous.

2.4 Benchmarking

2.4.1 Naïve Model

For a benchmark model, MIT has provided a naïve network, which is described below. Within this network, changing the default hyper-parameters without changing the network

will only return an average speed of approximately 51.5 *mph*, or can even cause the vehicle to stall in traffic. The initial model uses one *fully connected (FC)* layer, a *rectified linear unit (ReLU)* with one neuron, and a *regression* layer. The initial *batch size* is 64 and the network is trained for 10k iterations. The *learning rate*, *momentum*, and *L2 decay rate* are initialized to the default values used in most neural networks.

```
43 opt.temporal_window = temporal_window;
44 opt.experience_size = 3000;
45 opt.start_learn_threshold = 500;
46 opt.gamma = 0.7;
47 opt.learning_steps_total = 10000;
48 opt.learning_steps_burnin = 1000;
49 opt.epsilon_min = 0.0;
50 opt.epsilon_test_time = 0.0;
51 opt.layer_defs = layer_defs;
```

Fig. 6: Example of hyperparameters used in the naïve (default) model.

The network size is a function of the *number of inputs*, *number of actions*, and the *temporal window*. Adjusting any of these hyper-parameters will lead to highly variable results, so proper strategies are required in order to gain an improved performance and meaningful results (i.e. randomly changing parameters will not lead to meaningful results). As the result of the naïve model is 51.5 *mph*, this is the goal to beat (performance). In terms of typical machine learning-based metrics, the number of iterations over time are plotted against the *reward quality*, which results in 0.52 for the naïve model. This value (the reward quality) is also required to be improved in order to gain a higher performance, as the model’s maximum speed (performance) is *proportional* to the reward quality/rate.

Evaluation complete

Average speed: 51.57 mph

OK

Fig. 7: Average speed of the naïve model. Passing speed required is 65mph [performance metric].

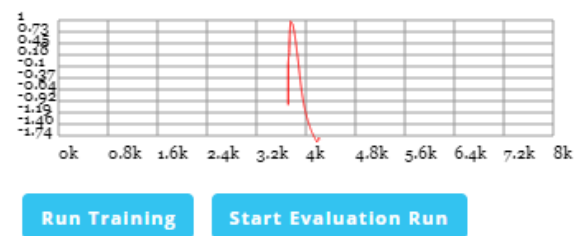


Fig. 8: Reward function characteristics (y-axis) of the naïve (default) model (max val = 1.00 and drops quickly).

In Fig.6 above, the x-axis is the number of iterations and the y-axis is the reward rate. Gaining a higher reward rate

will ensure a higher maximum speed. Fig.6 also shows an immediate plummet for the learning rate, showing that the network is incapable of learning any useful behavior.

3 METHODOLOGY

3.1 Data Pre-Processing

Normally, a machine learning problem would require some data pre-processing. However, in certain areas of machine learning (specifically deep reinforcement learning and computer vision), it is more ideal to input the raw data directly into the network. Naturally, this causes the network time to increase as well as the required available storage. For this project, pre-processing is not appropriate as the loss of data can drastically impact the results of the learning/reward, as well as potentially introduce unwanted bias.

Instead, a literature review was conducted to find appropriate values to initialize a general DQN. Initially, only the *learning rate*, *number of iterations*, and *batch size* were changed. In the *Refinement* section, the *L2 decay rate*, *momentum*, *gamma*, *epsilon*, *layer types* were adjusted to gain the maximum performance.

3.2 Implementation

After a bit of research, it was clear that several parameters would be required to change for a passing submission. Since the *gamma* and *epsilon* parameters do not have an intuitive (or explicit in visualization) representation, these were held a constants for the first initial implementation phase. Furthermore, it was decided to maintain the same learning parameters. Instead, the input size (as described above) was targeted and then the layer types (and sizes) were adjusted. A hyperbolic tangent (*tanh*) activation layer was added to gain so different features over the *ReLU*, but these layers must be used with caution (*tanh* is differentiable and monotonic).

To demonstrate the reasoning behind starting with the network layers, an image of the activation behavior for the neurons is shown below:

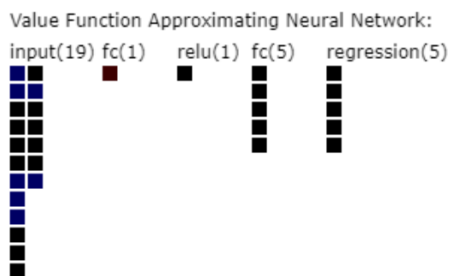


Fig. 9: Default network architecture with poor activation rates.

As can be seen, the network is minimal featuring only two *FCs*, one *ReLU*, one regression layer and the input layer. Also, the neurons are "dark" meaning the firing behavior is very poor. In order to improve this, some changes were made that will be discussed in the *Final Architecture* section, and shown briefly below:



Fig. 10: Improved network architecture with decent activation rates.

The *FC*, *ReLU* and input layer sizes were increased, as well as two *tanh* layers were added. The input layer matrix size was change to 385 based on the data was represented above in the *Exploratory Visualization* section.

Evaluation complete

Average speed: 68.9 mph

OK

Fig. 11: Average speed of the improved (personal) model. Passing speed required is 65mph [performance metric].

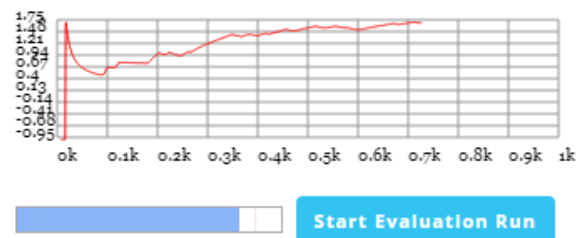


Fig. 12: Reward function characteristics (y-axis) of the improved (personal) model (max val = 1.75).

At this point, significant improvements had been made based on the specific performance metrics and functional requirements. However, it was decided to continue improving the model and hyperparameters (refinement) to meet higher level quality requirements.

3.3 Refinement

Now that a satisfactory network architecture had been derived, adjustments to the individual hyperparameters needed to be made. Due to previous experience with *DQNs*, it was decided to adjust the *learning step* amount, *burn in*, *epsilon*, and *gamma* values. The *starting learning threshold* and the *experience* size were experimentally found to have less of an impact on specific behavior, but did contribute to the overall final behavior (i.e., these parameters influence the maximum ranges and behavior of other hyperparameter combinations). Now, the trainer (learning) options were adjusted in a combination of trial-and-error along with

intuition and previous work review (such as options suggested by Google's DeepMind group). [7]

Learning Parameters:

learning - rate : 0.005
momentum : 0.5
batch - size : 32
l2 - decay : 0.05

```
57 para.temporal_window = temporal_window;
58 para.experience_size = 80000;
59 para.start_learn_threshold = 40000;
60 para.gamma = 0.95;
61 para.learning_steps_total = 200000;
62 para.learning_steps_burnin = 500;
63 para.epsilon_min = 0.05;
64 para.epsilon_test_time = .010;
65 para.layer_defs = layer_defs;
```

Fig. 13: Example of hyperparameters used in the refined (personal) model.

After a great deal of fine-tuning and analysis of trade-off behavior, a satisfactory parameter set was developed. Next, the model is trained and the activation neurons are observed.

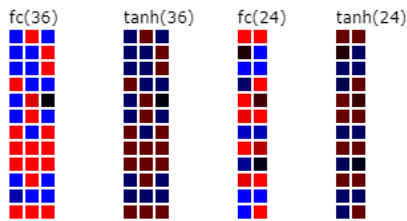


Fig. 14: Improved network architecture with satisfactory activation rates.

4 RESULTS

The final results of the refinement are displayed below. The results show a much improved reward (2.74 value / 2.21 average, or approximately 221 274% improvement) as well as a speed that far exceeds the minimal requirements (74.48 mph as compared to the initial 51.57 mph or nearly 50% improvement).

Evaluation complete

Average speed: **74.48 mph**

OK

Fig. 15: Average speed of the refined (personal) model. Passing speed required is 65mph [performance metric].

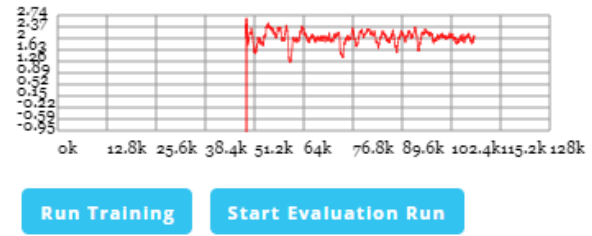


Fig. 16: Reward function characteristics (y-axis) of the refined (personal) model (max val = 2.74).

4.1 Model Evaluation & Validation

4.1.1 Final Architecture

As briefly displayed above, the final architecture is essentially a CNN. There is an input layer, four FCs, with two *ReLU*s and three *tanh* activation layers, a regression layer and an output layer. [6]

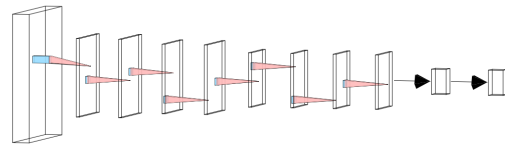


Fig. 17: Example of Final Model Architecture (approximate equivalence).

4.1.2 Local Evaluation

In terms of a discussion, it can be seen from Table 1 and Table 2 that the overall average speed and reward have decreased. This is due to the added stochastic behavior from nine other simultaneous autonomous agents navigating through the traffic. The algorithm is not optimized to handle other agents that act independently. As such, the agent of focus receives interference from the other agents and decreases the overall performance of all agents. However, this was found to be proportional to the number of other agents on the road. It is hypothesized that if the road was entirely filled with other agents, all vehicles would always perform at the maximum speed of 80 mph.

4.1.3 DeepTraffic v2.0 Official Competition

As an added challenge since the algorithm design seemed suitable, the multi-agent option was enabled and the algorithm was ran against MIT's evaluation test. The only difference in the algorithm is adding the capability for more agents. MIT has stated: "If you are officially registered for this class you need to perform better than 65 mph to get credit for this assignment.[1]" A screenshot of the submission (and successful completion) of the assignment is provided below.

The evaluation run takes place on a separate thread that simulates 500 runs at 30 seconds each. Afterwards, the average speed per run is returned for the final score, meaning the results that are provided locally are only an estimate of the actual score. MIT also checks each submission of

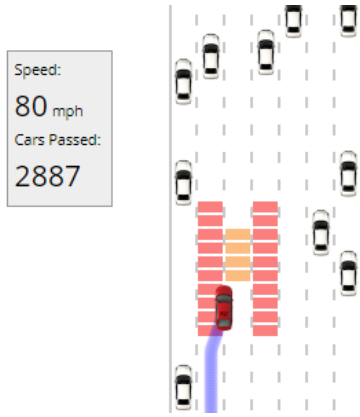


Fig. 18: Image of the autonomous agent (red) navigating at the maximum speed (80 mph) through traffic, having overtaken nearly 3000 vehicles.

User Info

Email: c.a.ohara@student.tue.nl

Highest Average Speed (DeepTraffic v2.0): 69.94

Highest Average Speed (DeepTraffic v1.1):

Visualization: [Visualization of Your Best Performing System](#)

Fig. 19: Results from Official DeepTraffic v2.0 Competition.

code for cheating, which will result in banishment from the competition and course.

At this time, it was decided to evaluate the current model and hyperparameter values against multiple agents as well. The results are shown below.

4.2 Model Comparison

For concise reporting, some model names (i.e., Mark Zero) were given to the agents. mk0 is the benchmark model, whereas mkll-v2 is with ten other agents (autonomous vehicles) present. mkll-v2 is listed in the tables since this model was used for the formal submission (validation and authenticity). Below, two tables are given to show the final results of each model and their average performance improvements.

TABLE 1: Average speed for trained networks.

Time	mk0	mkI	mkII	mkII-v2
1	51.69	68.10	70.05	68.46
2	51.69	68.48	70.38	70.52
3	51.69	66.98	70.96	69.77
4	51.69	66.72	70.51	68.88
5	51.69	68.90	73.96	68.90
6	51.69	69.67	72.92	69.67
7	51.69	69.94	72.84	70.38
8	51.69	66.72	74.48	69.92
9	51.69	66.87	72.92	70.18
10	51.69	69.94	73.84	69.94
Average	51.69	68.23	72.29	69.66

The bold numbers represent the maximum speed and best average speed achieved.

TABLE 2: Reward values for various trained networks.

Reward	mk0	mkI	mkII	mkII-v2
Low	0.73	2.07	1.82	1.63
High	0.31	2.07	2.52	2.37
Average	0.52	2.07	2.17	2.00

The bold numbers represent the maximum reward quality and best average reward quality achieved.

5 CONCLUSION

5.1 Free-Form Visualization

Due to poor network and parameter choices, the agent has absolutely abysmal performance. It is not often stated that some event is "off the charts" but meant with negative connotation. Below, two images are shown in which poor choices were attempted (learning rate too high/low, gamma at 1, etc.):

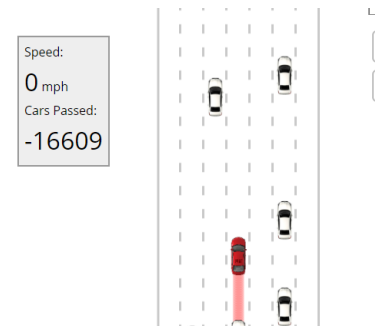


Fig. 20: Agent stopped during traffic allowing more than 16k cars to overtake it.

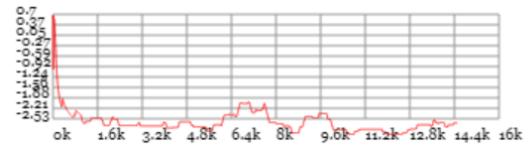


Fig. 21: Abysmal reward function over 14k iterations.

In Fig.21 above and Fig.22 below, we see the learning rate with the number of iterations. An important visualization here is, regardless of the number of iterations (i.e. the amount of time the network is allowed to train), if prerequisite selections do not meet satisfactory values, the network will never learn. This applies to both "poor" and "good" models. These values usually need to be tuned for each project too, as different assumptions and behaviors cannot usually experience "transfer learning." However, with enough practice and experience, engineers and scientists can intuitively predict problematic behavior and its causes.

Finally, it is important to consider the component of "luck" in machine learning. Below, an image (Fig.23) is shown in which the agent becomes stuck behind a number of vehicles. When this happens, the reward function will drastically decrease. If this happens at

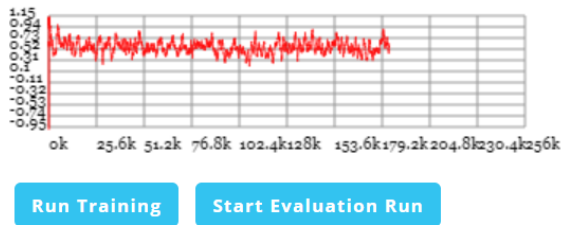


Fig. 22: Steady-state reward function over 160k iterations.

the beginning of the training sequence, the reward will plateau quickly and never improve, even with the same model/network/parameter selection. This is true for many classical (and rudimentary) models for machine learning. The initial training phases prove to be the most critical in final behavior. With dynamic training sets (such that the model can be saved and later updated with more iterations), a good model can become a poor model. Therefore, it is important to incorporate architectures that have a memory component.

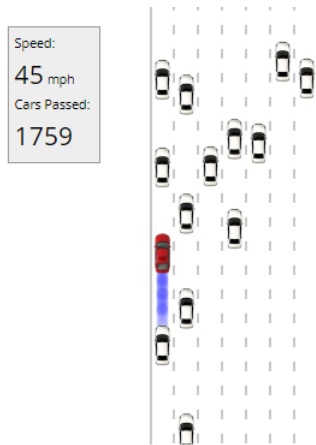


Fig. 23: Agent becoming stuck behind vehicles in a case of "bad luck".

5.2 Reflection

Overall, the project was quite successful. The total time took more than 40 hours, but I am very satisfied with the results. It could have been completed much faster but I was rather enjoying myself. If all engineering problems could be approached as a "game" with clear visualizations of the problems being solved (reward function?), I think that a great deal of progress would occur by appealing to broader students. Hopefully, with the advent of VR/AR, more interesting projects will recruit a new generation of technical students. In retrospect, I should have completed the visualizations with the agent that I edited to represent Udacity (HEX #34C5F9)

5.3 Improvement/Future Work

To minimize computational overhead and energy usage, an embedded version of the network architecture could



Fig. 24: MIT's flagship Agent painted with Udacity Blue.

be made (i.e., no more than one hidden layer is utilized such that matrix multiplication is kept to a minimum). Furthermore, it would be ideal it implement some form of "grandmother neuron" that would be able to remember ideal maneuvers (*if* stuck behind a vehicle, *then* move right, *else* move left), as the current implementation seems to unlearn ideal behavior and replace it with the current highest return of reward. This is currently a topic of *Progressive Neural Networks (ProgNN)* that are capable of solving one of the most important limitations here known as "catastrophic forgetting." [4]

Furthermore, it would be ideal to reevaluate the *RNN* capabilities with DeepTraffic, as it should be possible. *Asynchronous Actor-Critic Agents (A3C)* is also a State of the Art algorithm (also made by DeepMind) that allows for effective learning based on the the automatic selection of a reward policy. However, A3C currently requires TensorFlow, so it is not compatible with the DeepTraffic interface currently.[5]

6 REFERENCES

- [1] 4.1.3; MIT's DeepTraffic - About Page, Accessed: December 24, 2018 - [DeepTraffic - About](#)
- [2] 1.3; ConvNetJS - Home Page, Accessed: December 24, 2018 - [ConvNetJS Home Page](#)
- [3] 1.2; T. D. Le, A. T. Le & D. T. Nguyen, "Model-based Q-learning for humanoid robots," 2017 18th International Conference on Advanced Robotics (ICAR), Hong Kong, 2017, pp. 608-613. doi: 10.1109/ICAR.2017.8023674, doi: 10.1109
- [4] 5.3; Rusu, A. A., Rabinowitz, N. C., Desjardins, G., et al. 2016, arXiv:1606.04671
- [5] 5.3; Asynchronous Actor Critic Agents (A3C) [simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c](#)
- [6] 4.1.1; towardsdatascience.com - [activation-functions-neural-networks](#) [Activation Functions Neural Networks](#)
- [7] 3.3; Google's DeepMind, Playing Atari Games with Deep Reinforcement Learning, <https://arxiv.org/pdf/1312.5602v1.pdf>