Self-Training Autonomous Driving System Using An Advantage-Actor-Critic Model

Elise McEllhiney

Introduction

This project covers a journey from manual model implementations to reinforcement learning in a simulated environment

Emphasis on bridging theory and real-world application in the field of autonomous driving

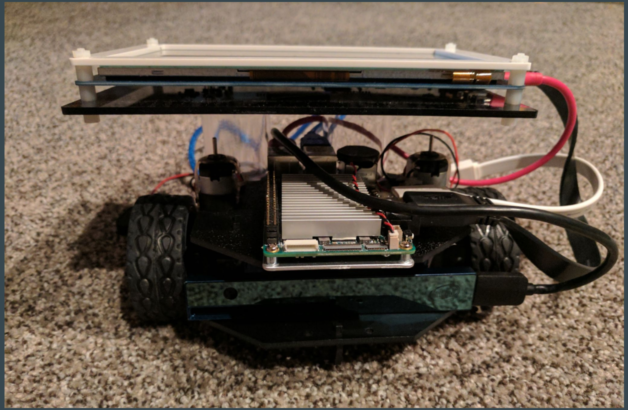
Manual Model on a Physical Car (Roachbot)

Early exploration of autonomous driving on a minimal embedded system.

Two-wheel robot car with an Intel Up and an Intel RealSense R200 IR camera.

Challenges faced with a manually designed model and noisy depth-map sensor data.

Initial attempt at decision-making based on the depth-map sensor data.



Manual Model on a Physical Car (Roachbot) Contd.

Improvements in the manual model, including averaging depth-map columns.

Challenges in handling different materials and lighting conditions.

Issues with convex shapes and corners in the algorithm.

Motivation for exploring machine learning algorithms.

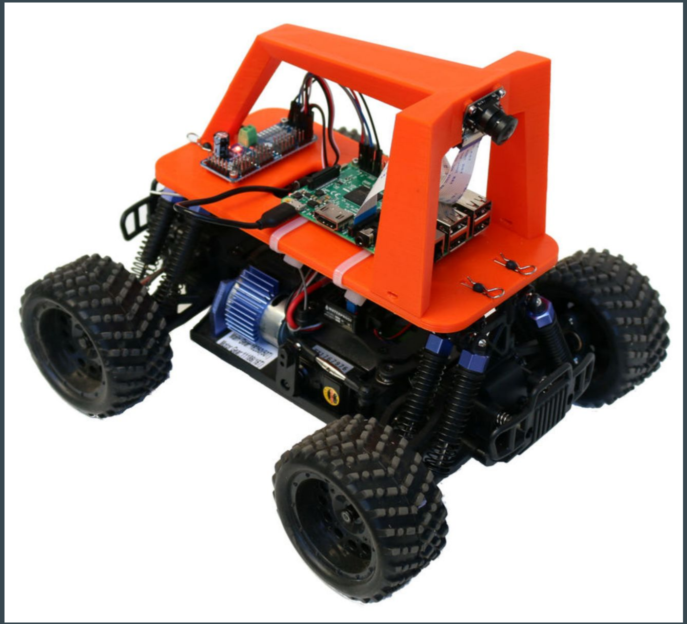
Supervised Model on a Physical Car (Donkeycar)

Transition to a supervised model based on Donkeycar implementation.

Replacement of the IR camera with a Raspberry Pi 3 camera.

Introduction of a servo for steering control.

Training the model using video frames recorded during human driving.



https://www.donkeycar.com/

Supervised Model on a Physical Car (Donkeycar) Contd.

Challenges of overfitting due to consistent training data.

Strategies to address overfitting, including diverse training data and longer tracks.

Factors impacting model performance, such as battery charge and lighting conditions.

The need for a self-adapting model to account for varying conditions.

We attempted to fit Donkeycar with a reinforcement learning model, but ran into difficulties due to the inconsistencies inherent in the physical construction of the robot and track.

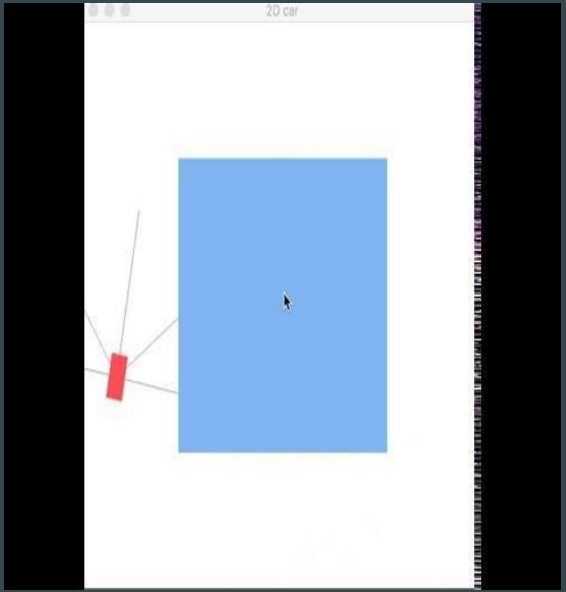
Reinforcement Model in Simulation (Simulation Architecture)

Transition to a simulated environment for experimentation.

Simulated architecture, including distance sensors and car actions.

Simplifications made in the simulation compared to physical cars.

Simulation allowed for controlled experiments that would otherwise be tainted by noise introduced because of physical components.



Reinforcement Model in Simulation (Q Learning)

Q Learning is a reinforcement learning model that allows the training to learn delayed rewards such as those present in driving or in video games.

“Playing Atari with Deep Reinforcement Learning”, is research around having a Machine Learning program take in graphical inputs and play several different ATARI games.

This research was the basis for using the Advantage Actor Critic model as it was an improved version of this model that could potentially take in visual inputs and train on time delayed rewards.

$$\pi(s) = argmax_a Q(s, a)$$

$$\pi : predicted\ reward$$

$$s : state$$

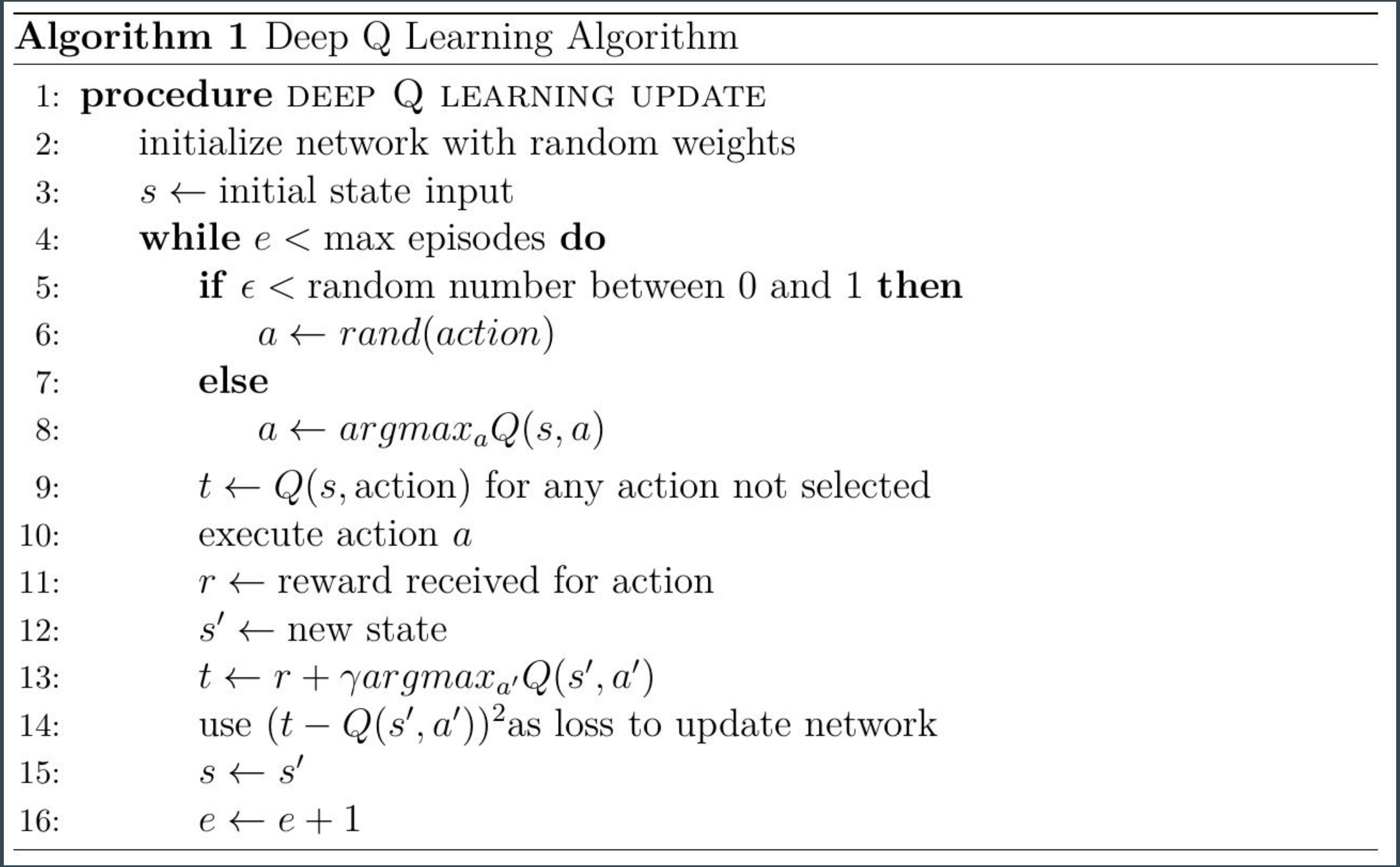
$$a : action$$

$$Q(s, a) = r + \gamma argmax_{a'} Q(s', a')$$

$$\gamma : discount factor$$

$$a' : nextaction$$

$$s' : nextstate$$



Algorithm 1 Deep Q Learning Algorithm

```
procedure DEEP Q LEARNING UPDATE
    initialize network with random weights
    $s \leftarrow$ initial state input
    while $e <$ max episodes do
        if $\epsilon <$ random number between 0 and 1 then
            $a \leftarrow rand(action)$
        else
            $a \leftarrow argmax_a Q(s, a)$
        $t \leftarrow Q(s, \text{action})$ for any action not selected
        execute action $a$
        $r \leftarrow$ reward received for action
        $s' \leftarrow$ new state
        $t \leftarrow r + \gamma argmax_{a'} Q(s', a')$
        use $(t - Q(s', a'))^2$as loss to update network
        $s \leftarrow s'$
        $e \leftarrow e + 1$
```

Reinforcement Model in Simulation (Model)

Utilization of five distance sensors as input for state information.

The model we used was an advantage actor-critic network.

Two neural networks: actor (for action selection) and critic (for predicting rewards).

Actions decided stochastically based on advantages.

Diagram of Advantage Actor Critic Model

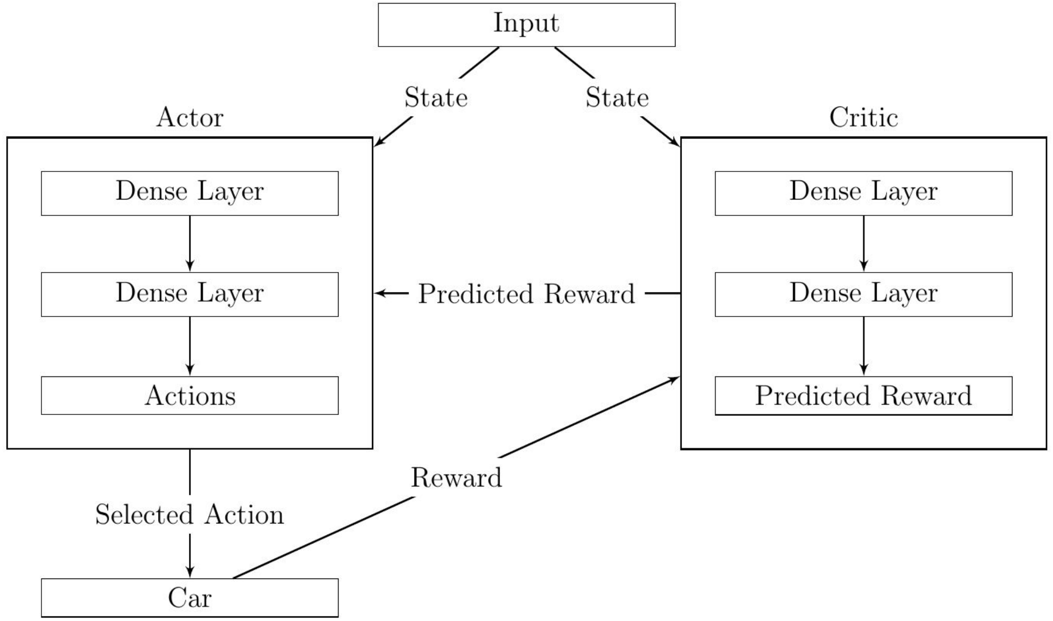
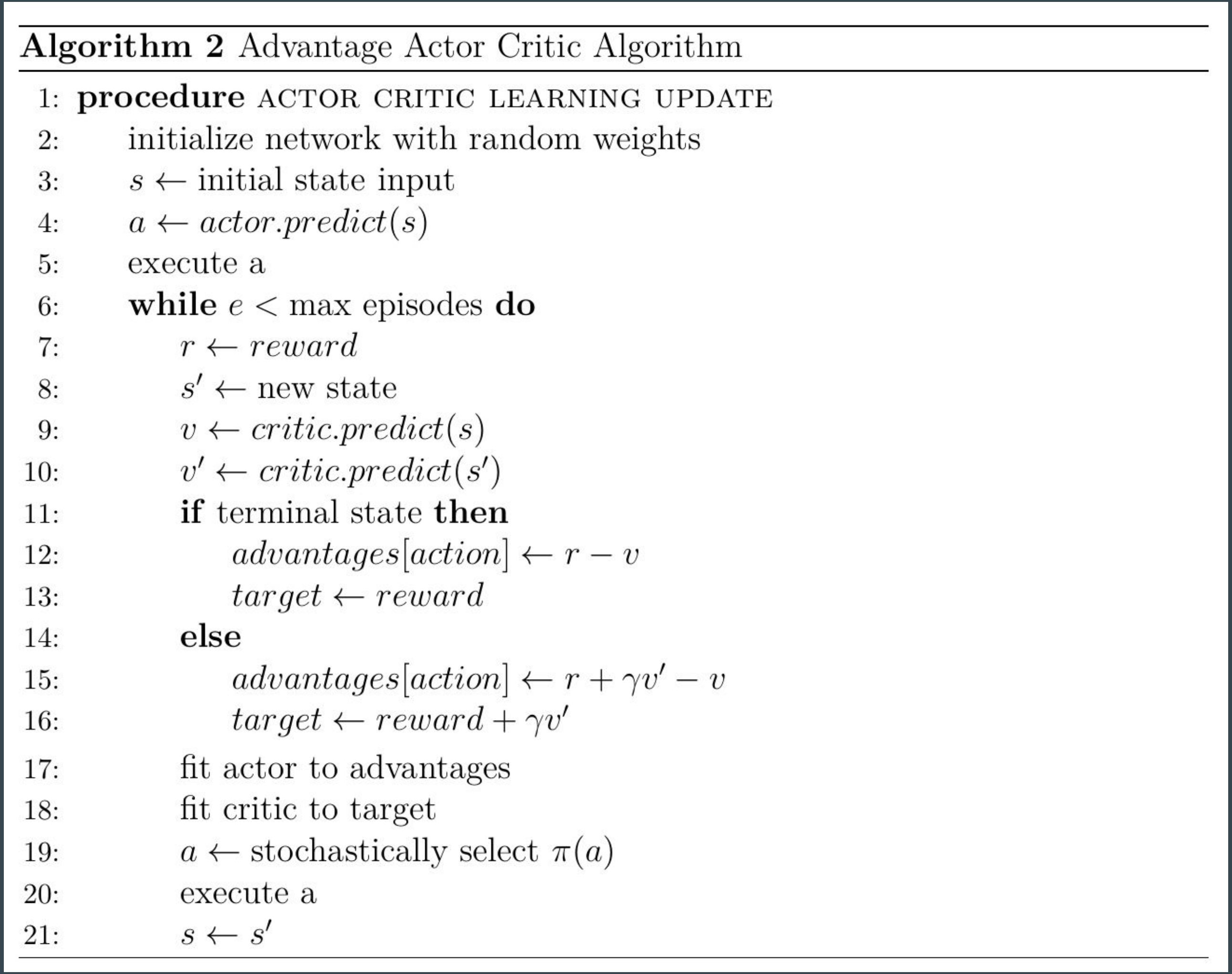


Figure (3.1) Diagram of Advantage Actor Critic Model



Algorithm 2 Advantage Actor Critic Algorithm

```
procedure ACTOR CRITIC LEARNING UPDATE
    initialize network with random weights
    $s \leftarrow$ initial state input
    $a \leftarrow actor.predict(s)$
    execute a
    while $e <$ max episodes do
        $r \leftarrow reward$
        $s' \leftarrow$ new state
        $v \leftarrow critic.predict(s)$
        $v' \leftarrow critic.predict(s')$
        if terminal state then
            $advantages[action] \leftarrow r - v$
            $target \leftarrow reward$
        else
            $advantages[action] \leftarrow r + \gamma v' - v$
            $target \leftarrow reward + \gamma v'$
        fit actor to advantages
        fit critic to target
        $a \leftarrow$ stochastically select $\pi(a)$
        execute a
        $s \leftarrow s'$
```

Reinforcement Model in Simulation (Hyperparameters)

Hyperparameters in machine learning models are parameters used to define the training details of the model.

Hyperparameters tested in the research: actor learning rate, critic learning rate, discount factor, and reset strategies.

Impact of hyperparameters on model convergence and training time.

Reinforcement Model in Simulation (Hyperparameters contd.)

Learning Rate - Tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

Actor Learning Rate - Learning rate of the Actor Network

Critic Learning Rate - Learning rate of the Critic Network

Discount Factor - Importance of future rewards to the current state. Discount factor is a decimal value between 0 and 1.

Reinforcement Model in Simulation (Hyperparameters Contd.)

Analysis of the effects of actor and critic learning rates on convergence and training time.

Discount factor on balances immediate and future rewards.

Hyperparameter tuning allowed for efficient model convergence.

Driving Success Criteria

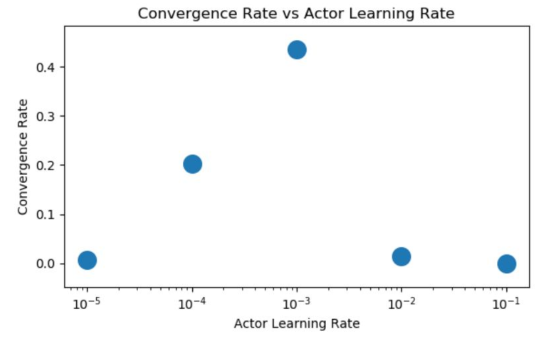
There is substantial difficulty in determining when a reinforcement model is successful. For this particular project, since driving is subjective we needed a quantitative metric for determining when the model was finished training.

We consider the model to have converged (successfully learned to drive) if it achieves 10 sequential 10,000 step episodes without colliding with the track walls.

This is not a standard metric, but one chosen to make determining whether the model has been trained successfully both consistent and able to be determined programmatically.

We allowed for a maximum of 500 episodes consisting of 10,000 steps each before determining that the model had not converged.

(a) Convergence by Actor Learning Rate



(b) Convergence by Critic Learning Rate

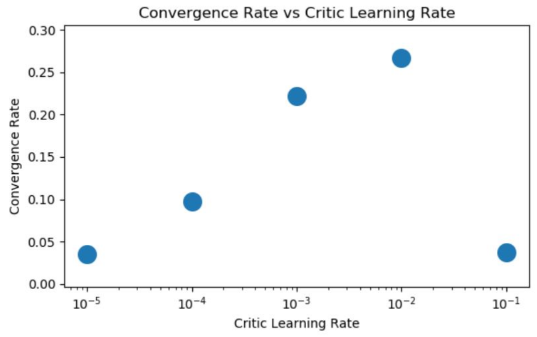


Figure (3.2) Average Training Time and Convergence Rate by Actor and Critic Learning Rates

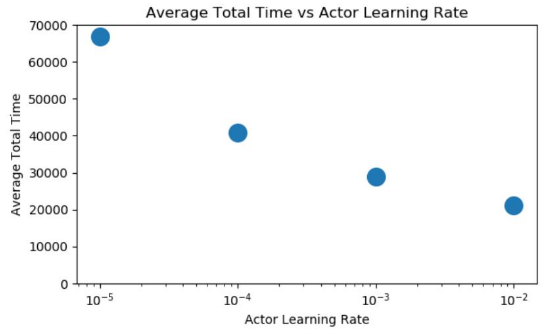
Learning Rate - Determines the step size at each iteration while moving toward a minimum of a loss function.

Convergence - The model has met the predefined criteria to be categorized as a successfully trained model.

This chart shows that there is a fairly narrow band where Actor and Critic learning rates can be set to achieve a model that will successfully train. Outside of these values the models rarely learned to drive.

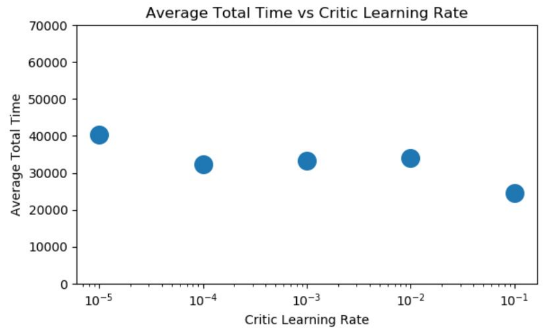
At optimal values, models would successfully train about 30-40% of the time.

(a) Average Training Time by Actor Learning Rate



Converged Models Only

(b) Average Training Time by Critic Learning Rate



Converged Models Only

Figure (3.3) Average Training Time and Convergence Rate by Actor and Critic Learning Rates

Learning Rate - Determines the step size at each iteration while moving toward a minimum of a loss function.

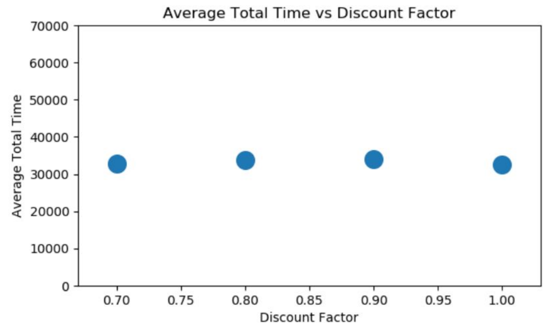
Training Time - Number of simulation steps required for the model to have learned to drive.

This only considers successfully trained models.

Higher actor learning rates appear to train the model faster.

Critic learning rates seem to have little effect on the average training time.

(a) Average Training Time by Discount Factor



Converged Models Only

(b) Convergence by Discount Factor

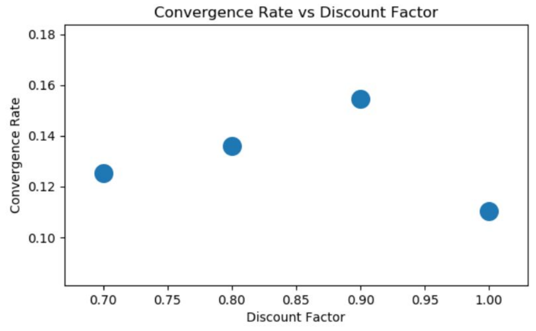


Figure (3.4) Average Training Time and Convergence Rate by Discount Factor

Discount Factor - Importance of future rewards to the current state. Discount factor is a value between 0 and 1.

Training time does not appear to be affected by discount factor.

Convergence rate appears to be highest when the discount factor is around 0.9.

Reinforcement Model in Simulation (Hyperparameters contd.)

Reset Strategies - These values were decided upon due to the nature of the driving problem. The model was becoming overtrained on the inputs at the starting point of the track. These were experimental strategies to resolve this issue and determine the effectiveness of the different strategies.

- Full Reset Per Episode - Resets car to static position for each episode.
- Reverse Distance - After episode concludes due to collision car is reversed for a set distance and next episode is started from that position.
- Periodic Full Resets - Combination of above two reset strategies. Car uses reverse strategy for a set number of episodes and fully resets to initial position after the set number of episodes.

Reinforcement Model in Simulation (Reverse Distance)

Strategy to reset the car by moving it backward in the simulation.

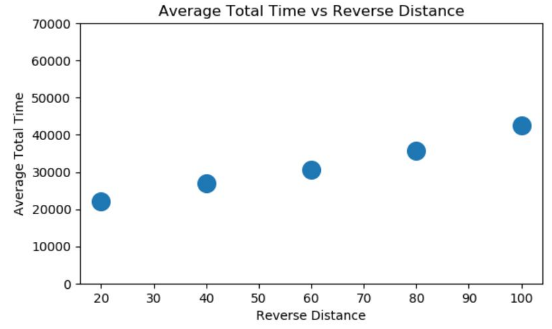
Comparison of different reverse distance values and their impact on convergence and training time.

The balance between allowing the model to react and repeating challenging situations.

Testing the combination of reverse distance reset strategy and periodic full resets to the original placement.

The practicality of reducing human intervention for resetting the car.

(a) Average Training Time by Reverse Distance



Converged Models Only

(b) Convergence by Reverse Distance

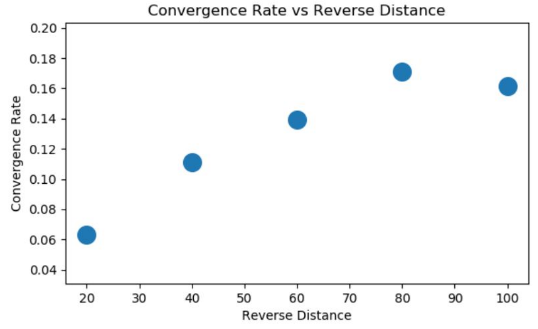


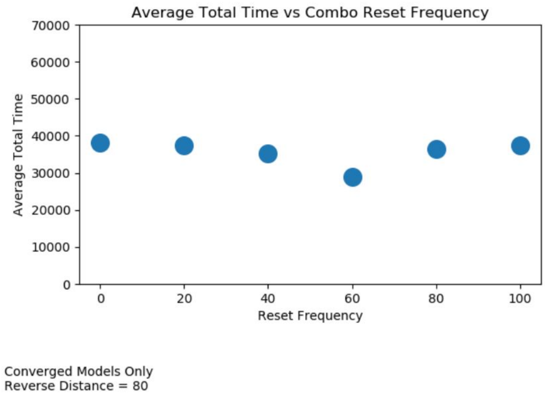
Figure (3.5) Average Training Time and Convergence Rate by Reverse Distance

Reverse Distance - Distance the car was moved backwards after a collision.

Short reverse distances allowed the car to see failure states more rapidly and shorten the training time.

Short reverse distances also caused the car to get stuck repeating the same collision and models had trouble converging.

(a) Average Training Time by Reset Rate



(b) Convergence by Reset Rate

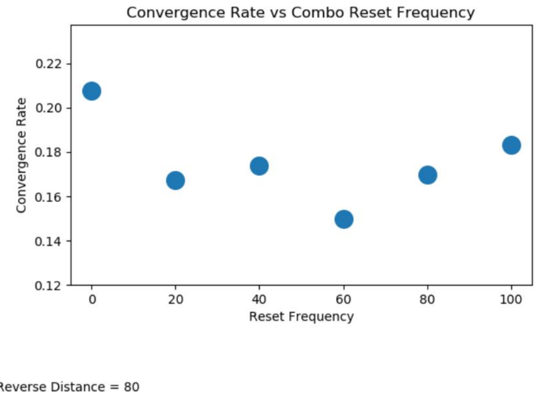


Figure (3.6) Average Training Time and Convergence Rate by Reset Rate

Reset Rate - How often the car was returned to the starting point of the track. This is how many reversal resets were allowed to run before the car was returned to the default position.

Reset Rate had very little impact on training time hovering around requiring 40,000 steps to complete.

Convergence shows that the less frequently the car was reset, the more dependably it converged.

The highest convergence was achieved when not resetting the model to the default position.

Conclusion

Successfully implemented two physical robots with autonomous driving models.

Established viability of implementing a reinforcement model on a physical car.

Successfully developed a neural network model using a simulated environment.

Identified crucial variables, hyperparameters, and strategies for practical implementation of autonomous driving.

Simplified the training process by utilizing a reset strategy through reversing the car enabling continuous training sessions.

Demonstrated the new reset strategies had comparable training times and convergence rates as setting the car in a default position, reducing the need for human intervention during the training of a physical system.