Section 3: Week 8: Autonomous Driving

Nate Bachmeier

TIM-8110: Programming Languages and Algorithms

May 19, 2019

North Central University

# Autonomous Driving

Some of the most challenging problems of algorithmic study can be found in autonomous vehicles. This is due to driving being less like chess and more like a conversation, in that the context is continuously evolving and formal rules are difficult to define (Fridman, 2017). To correctly maneuver through this conversation the vehicle needs to identify objects and their likely path. In addition to other cars, these vehicles need to react to the unexpected such as a child chasing a ball or debris falling onto the road way.

There is a huge potential to improve the safety, cost, and performance of transportation though autonomous driving. Many newer vehicles are already including ‘driver assisted technologies’ such as lane detection, adaptive cruise control, and automated parallel parking. Despite the advantages of pure autonomy, the broad adoption across mainstream consumer is likely several years out.

# Literature Review

In the meantime, autonomous racing is gaining traction within academia and the industry. This allows research to continue with fewer safety risks as the system under test is enclosed. To further reduce costs much of this research takes place within the context of physics simulators. These simulators are often controlled driven by reinforcement learning algorithms.

## Introduction to Deep Reinforcement Learning

A reinforcement algorithm is a supervised learning algorithm which tries to guide an *agent* through an *environment*. As the agent performs *actions* the *reward function* scores and behavior and signifies satisfaction through numerical values. The agent uses the reward values to construct a *policy*, that maps the expected value of transitioning from one *state* to another.

A baby (agent) might have the objective of walking across the room (environment). During each step (action) its brain is collecting sensor readings (state) and determining if that step moved them closer to dad or caused them to fall (reward function). Actions such as leaning to far forward cause them to tip over and are avoided later (policy). Through enough repetition (training) the baby eventually learns to complete the objective with a high degree of reliability.

## Introduction to Deep Learning and Self-Driving Cars

Computers can use a similar mechanism to learn complex skills such as how to drive a car. A common pattern is to attach a camera to the front of the car and then stream each frame into the reinforcement algorithm.

The frame is converted to an array of numerical pixels with each value becoming an input node to a connected graph, called a neural network. The input layer will be paired with one or more hidden layers that eventually connect to an output result (e.g. desired steering angle or speed).

After each frame is traversed through the network the system uses the reward function as a mechanism to determine if the predicted action is the desired action. Using a mathematical transform called ‘backpropagation’, the edge weights are adjusted such that similar future examples result in more accurate predictions.

Once the training has completed the network will identify certain features of the image such as the center line and edges of the track. While the computer has no concept of what these features are it does understand that certain patterns infer the desire of *left 30 degrees* over *right 15 degrees*. These actions are then transmitted to mechanical systems the execute the state change, like pressing the gas peddle and turning the wheel.

## Deep Learning Based on Lateral Control

*The Open Racing Car Simulator* (TORCS) is an annual contest during the IEEE World Congress on Computational Intelligence. Since 2008 it has been a reoccurring theme for attendees to demonstrate new and innovative approaches to challenges ranging from classification to computer vision.

Li et al describe a system called Multi-Task Learning that attempts to drive a car by jointly solving N related tasks. This exploits a correlation between related challenges such that predictions are more accurate due to additional evidence. Their implementation chose tasks of (1) feature selection from camera frames; (2) optimal steering commands; and (3) classification of track curvature (e.g. left, right and straight).

Then each camera frame was passed through a series of convolution layers with each providing a higher construct of information. For instance, the first layer might provide edge detection, the second corners and contours, the third object parts, and the forth actual objects.

The deeper the network the smarter it can become at the cost of requiring exponentially more data. Having a reward function and classification operate on top of the extracted features enabled the researchers to bypass that requirement and make accurate recommendations. These recommendations fed into a lateral control plane which attempts to keep the car in the lane and pointed the correct direction.

## Distributed Reinforcement Learning for Autonomous Driving

One of the challenges with reproducing the results of TORCS is that training a model is very time consuming. To completely train a vehicle for public roads could require between hundreds to thousands of billions of hours of video feeds. Processing such an enormous collection of videos, will require new algorithms that exploit extreme levels of parallelism.

Transfer learning attempts to address these challenges by making segments of the network reusable. Consider the previous example where multiple convolution layers are overlaid to extract edges, object parts and then objects. If the last layer was omitted, then network would predict object parts instead of objects. This suggests that networks for similar domains could be reusable in different context. Clearly such by not having to recompute base networks there is the potential for huge savings.

To improve the performance of calculating the shared base network the researchers also propose a distributed architecture where N agents report to a *parameter* *server*. The server is responsible for sending work to the agents and collecting their results. When the solution is paired with economically priced cloud resources it becomes possible to run a high number of iterations rather quickly. In their scenario 140 hours of training was accomplished in 1 wall hour.

One of the challenges encountered by this architecture is the load applied on the parameter server as it is a single point of failure. It might be possible to mitigate this issue by tiering the parameter servers and increasing the complexity of merging agent results.

Another constraint is called the ‘vanishing gradient problem.’ When the agent is making updates to the network weights it is possible for infinitely large or small values to skew the significance of a single node within the hidden layers. The likelihood is compounded by distributed agent scenarios due to the sharing of global state.

## Quasi Steady State Approach to Race Car Lap Simulation

The racing line is the optimal path that minimizes the curvature distance and maximizes the gravitational speed through each corner. A racer that closely follows this line is more likely to achieve the theoretical best lap time on the course.

Quasi Steady State (QSS) modeling can be used to approximate the line racers, by breaking the track into multiple discrete segments. Each segment is processed in parallel with a goal of optimizing the gravitational speed through the corner. Some systems leverage an even more crude approximation by using basic geometry.

Transient-optimal (TO) control systems can calculate significantly more efficient racing lines. This is accomplished by finding the optimal path with respect to the angular momentum, engine torque, tire model, and aerodynamic forces. These additional features come at the cost of increased compute time. The journal states that QSS can be evaluated in under 60 seconds versus TO requiring more than 24 hours.

## Particle Swarm for Path Planning in a Racing Circuit Simulation

Bevilacqua and Starr provide a novel approach for calculating the racing line by using three minimum curved path solver algorithms. At each step the results of the different algorithms are compared and the most efficient selected. The first solver uses the same approach as QSS, and the other introduce the idea of a Genetic Algorithm and Particle Swarming.

The implementation begins with encoding the track into a 2D plane of x and y coordinates. Next the track was broken into a collection of way points and each way point independently measured by all three solvers.

An analogy for their generic algorithm would be to deal cards and then chose the best poker hands. The best hands are reshuffled and redealt while the worst hands are omitted from the deck. They used a similar approach with route parameters as a mechanism to converge on the optimal paths.

The third approach uses particle swarming by creating a graph with nodes to represent the path available to the next way point. Though a Monte Carlo simulation different paths are chosen and the weights between the nodes updated to reflect the required time to traverse. After enough iterations the most efficient path is equal to the shortest distance across the particle graph.

# Reproducing the Study

After acquiring a rudimentary understanding of reinforcement learning it was possible to revisit the primary literature on TORCS. Due to the time constraints of the course, the efforts were reproduced using Amazon Deep Racer. Deep Racer is a managed service that teaching students about autonomous driving by pairing Amazon’s physics engine (Amazon Robomaker) with their machine learning platform (Amazon SageMaker).

## Configuring Deep Racer

Amazon provides automation to provision all resources within a customer’s account. This included storage buckets, unit tests, security roles, and associating shared docker containers with the private virtual cloud. Afterwards a reward function can be submitted to the simulation environment, where a realistic 1/18th scale car will learn to drive using a camera attached to the virtual hood.

The user authored reward function is called after each frame and passed a collection of parameters such as the orientation of the steering wheel, the current speed, and various track position metrics. From the parameters it is possible to tell e.g. the car is in the lane and heading the correct direction. This is beneficial to completing the race in the shortest time and is rewarded with a positive score. Naturally if the car has driven off the road or going the wrong direction that is penalized with a negative score.

## Measuring the Base Line

Deep Racer comes with a several example reward functions written in Python. The vanilla hello-world script attempts to drive down the center of the road at a maximum speed of 1mps. During the training phase a video feed shows the car’s view as it makes random decisions and evaluates the policy. After roughly one hour the car can successfully drive the *Reinvent 2018* track within 66 seconds.

## Improving the Runtime

Different hyperparameters to the ‘HelloWorld script’ were evaluated such as increasing the number of gears and decreasing the maximum turning radius. These results provided a general understanding of the vehicle mechanics. However, the best lap time remained fixed at roughly 68 seconds.

The reward function was rewritten to score based on (1) the speed is statically above 0.5mps; (2) the car is near the center line; and (3) the heading is toward the next waypoint. With a maximum speed of 2mps the lap time reduced to 41 seconds. Using a maximum speed of 3mps further reduced the time to 30 seconds.

However, this also decreased stability and resulted in the car driving off road. If the vehicle leaves the track, then the evaluation is terminated, and the partial result displayed to the user.

## Improving the Stability

According to the simulated video stream the car would veer off the track due to high speeds around the corners. The naïve reward for the car being between 0.5 to MAX\_SPEED was replaced with an angular momentum algorithm. It used the waypoint coordinates to compute the curvature between each point and the theoretical maximum velocity.

A ratio of the actual speed to the maximum speed was used to penalize the car for driving too fast or to slow. Despite this working within the local unit tests, the algorithm was not run within Deep Racer due to a lack of support for the ‘numpy’ library. To mitigate the limitation the curvatures could be computed out of band or a different approximation method used.

Following the advice of Brayshaw and Harrison, basic geometry was used to approximate the QSS curvature to the next way point. A discrete reward function was implemented to use this value and the current speed. This (1) improved reliability; (2) enabled a maximum speed of 4mps; and (3) decreased the best lap time to 23.8 seconds.

# Conclusions

Autonomous driving is a challenging problem space that requires new algorithms to optimize the actions chosen by the computer. The state-of-the-art solutions leverage neural networks, which teaches the system desired behavior through examples. To specify which states are more desirable a reward function needs to be provided.

There are many ways to model the reward function, each providing different levels of performance. The most advanced solutions use Multi-Task Learning to provide evidence of choices and boost the confidence. However, high quality solutions can still be created with simple geometry that approximates the race line.