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 roadway.

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 markets is likely several years out.

# Literature Review

In the meantime, autonomous racing is ‘gaining traction’ across 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 through reinforcement learning algorithms.

## Introduction to Deep Reinforcement Learning

Reinforcement learning is a supervised learning algorithm that tries to guide an *agent* through an *environment*. As the agent performs *actions* the *reward function* expresses satisfaction with the behavior through numerical scores. The agent constructs a *policy* that maps the expected reward obtained by transitioning from one *state* to another.

A baby (agent) might have the objective to walk across the room (environment). During each step (action) its brain will (1) collect sensor readings (state); and (2) determining if that step moved them (a) closer to dad or (2) caused them to fall (reward function). Actions such as leaning too far forward, resulting in tipping over, are later avoided (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 a reinforcement algorithm.

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

Each frame is traversed through the network and then uses the reward function as a mechanism to determine if the predicted action equals the desired action. Using a mathematical transform called ‘backpropagation’, the hidden layer’s edge weights (called gradients) are adjusted to improve the accuracy of future predictions.

Once the training has completed, the network will identify certain features of the image such as the centerline and edges of the track. While the computer has no concept of what these features are it does understand that certain patterns infer the desired action is *left 30 degrees* over *right 15 degrees*. These actions are then transmitted to mechanical systems the execute the state change.

## 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 solutions to challenges ranging from classification to computer vision.

Li et al describe a system called Multi-Task Learning (MTL) 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).

Each camera frame is passed through a series of convolution layers that extract higher level constructs. For instance, the first layer might provide edge detection, the second corners and contours, the third object parts, and finally actual objects.

Deeper the network can make smarter predictions at the cost of requiring exponentially more training data. Having a reward function and classification operate on top of the extracted features enabled the researchers to reduce that requirement and improve action selection. These recommendations fed into a lateral control plane that 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 hundreds to thousands of billions of hours of video analysis. To process these enormous collections of videos will require new algorithms that exploit extreme levels of parallelism.

Transfer learning attempts to address these challenges by making segments of the neural 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 the network would predict object parts instead of objects. This suggests that networks for similar domains could be reusable in a different context. Clearly, not having to recompute base networks, has 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 training hours was accomplished in 1 wall hour.

However, the parameter is a single point of failure and susceptible to being overwhelmed by the agent nodes. It might be possible to mitigate this issue by tiering the parameter servers at the cost of increased complexity merging agent results.

Another constraint is called the ‘vanishing gradient problem.’ When an agent updates 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 sharing a global state.

## Quasi Steady-State Approach to Race Car Lap Simulation

The racing line is defined as the path that minimizes the curvature distance around the track 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 the goal of optimizing the gravitational speed around the corners. 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 computing time. The journal states if calculating the QSS of a course takes 60 seconds, then TO would be on the order of 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 solving algorithms. 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 waypoints with each independently measured by the solvers.

At each step, the results of the different algorithms are compared, and the most efficient route selected. The first solver uses the QSS approach, and the others introduce the idea of a Genetic Algorithm and Particle Swarming.

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 continue ‘shuffling the route parameters’ until the ‘best hand’ converges and expresses the optimal route.

The third approach used particle swarming to express a graph that represents the available paths to the next waypoint. 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 that pairs Amazon’s physics engine (Amazon Robomaker) with their machine learning platform (Amazon SageMaker); to teach students about autonomous driving.

## Configuring Deep Racer

Amazon provides automation to instantiate all resources inside the customer’s account. This included storage buckets, unit tests, security roles, and associating shared docker containers with the Virtual Private Cloud (VPC). Afterward, a reward function can be submitted to a realistic 1/18th scale simulated car, that learns to drive through reinforcement learning.

The authored reward function is called after each frame and passed a collection of parameters such as (1) the orientation of the steering wheel; (2) the current speed and heading; and (3) various track position metrics. From these parameters, it is possible to determine the car is in the correct lane and heading in the right direction.

## Measuring the Base Line

Deep Racer comes with 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 navigate 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 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 results in the car occasionally driving off-road. If the vehicle leaves the track, then the evaluation is terminated early.

## 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 asking the car to drive at least 0.5mps was replaced with an angular momentum algorithm. This used the waypoint coordinates to compute the curvature between around the track and the theoretical maximum velocity.

The car was panelized for driving too fast or too slow based on the ratio of desired speed versus the actual speed. Despite this working within the local unit tests, the algorithm could not be deployed to Deep Racer, as it lacked support for the ‘numpy’ library. It might be possible to mitigate this limitation by precomputing the curvatures and rendering the results directly into the reward function. This would have limited use in the real world but could be advantageous within the context of the Amazon Deep Racer League contest.

Following the advice of Brayshaw and Harrison, basic geometry was used to approximate the QSS curvature between waypoints. 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 confidence. However, high-quality solutions can still be created with simple geometry that approximates the race line.

Investigations and research are required for the stability of the vehicle. Currently, the Deep Racer simulation zigzags across the lane and wastes valuable time. Most notably these challenges are visible around certain tight corners. To address these issues, better reward functions need to be designed to approximate and follow the race line and maximum speeds. Real world racing uses the strategy of ‘enter slow and exit fast,’ perhaps this idea can be encoded into the reward.

There is also an open topic to reduce the convergence of training. If the system can prune undesirable paths sooner, then it can spend more time exploring rewarding paths. This may be a matter of playing with parameters that control the ratio of breath-first versus depth-first searching across the policy gradient.

There is also research that suggests that instead of reinforcement learning more efficient solutions might exist with Recurrent Neural Networks (RNN). RNN supports the learning across sequential sequences of actions. This can be paired with Long Short-Term Memory (LSTM) networks to more accurately represent the time series nature of driving. These methods were not investigated due to the finite amount of time available for this assignment.