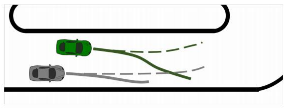
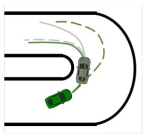
# Trajectory generation for autonomous cars in Competitive Scenarios using Deep Neural Networks

Stanford University

Mingyu Wang (mingyuw@stanford.edu)

## Introduction

In this project, we want to use deep learning approach to learn a trajectory planner for autonomous vehicles in competitive scenarios. We use a feedforward network to learn the planned trajectories.

Two examples of "target" trajectories in the dataset. The controlled vehicle take the state of both vehicles and track geometry as input, outputs a trajectory to win the racing game.

## Problem Formulation

### Dataset Description

The dataset is generated from a optimization-based trajectory planning algorithm which requires solving several iterations of non-linear optimization.

*Input*: $[x_e, y_e, vx_e, vy_e, x_o, y_o, vx_o, vy_o]$ which are positions and velocities of the two players.

*Output*:$[p_1, p_2, \dots, p_9, p_{10}]$ which are positional waypoints.

### Supervised Learning

We apply supervised learning approach on the given dataset. The neural network learns a policy $\pi$

$$\pi = \arg\min_{\Pi} \mathbb{E}_s[l(\pi, s)],$$

where loss function is

$$l(\pi, s) = \frac{1}{m}\sum_{i=1}^{m}(\pi_i^{*2} - \pi_i^2)$$

## Approach

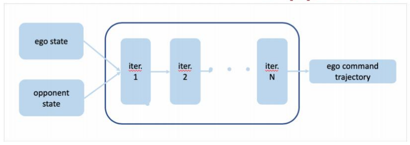
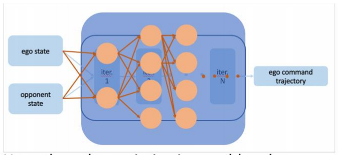


Diagram for trajectory generation. Each iteration solves a non-linear optimization problem. Solving one trajectory requires ~50ms, which is too slow for real-time systems.



We replace the optimization problem by neural network and learn the mapping from state to a trajectory.

## Approaches (cont'd)

- we implemented a feedward network with fully connect layers.
- ReLU and Leaky ReLU activations are used and compared.
- Adam optimizer is pick which achieves better results then Stochastic Gradient Descent.

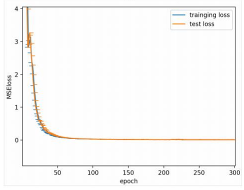
| hidden layer | type | # of neurons | activation |
|---|---|---|---|
| 1 | FC | 256 | ReLU / LeakyReLU |
| 2 | FC | 512 | ReLU / LeakyReLU |
| 3 | FC | 512 | ReLU / LeakyReLU |
| 4 | FC | 512 | ReLU / LeakyReLU |
| 5 | FC | 256 | ReLU / LeakyReLU |

Table 1: feedforward network structure

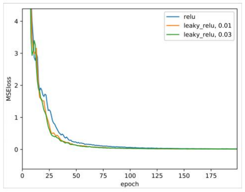
## Performance and Analysis

To evaluate the performance of the trained neural network, we validate the learned policy on test dataset.

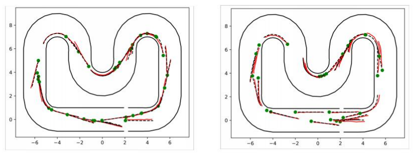
- loss on training set and test set



- training loss w/ different activation functions



- examples on test dataset. The black dotted lines are ground truth trajectories and red lines are learned trajectories.



- computation time is greatly reduced from ~50ms to 0.6ms.

The implemented supervised learning approach suffers from "covariate shift" when interacting with the environment. The challenge is that the training dataset is from expert policy, whereas the learned policy encounters state from a different distribution.
For future, we want to explore DAGGER which tackles this problem.