An ALVINN Inspired Neural Network Autonomous Driving Simulation

Xiyuan Liu

Carnegie Mellon University
`xiyuanl1@andrew.cmu.edu`,

Abstract. This report describes the use of a simple neural network to drive a vehicle dynamic simulation autonomously by looking at the 2D road image and output the steering command.

Keywords: neural network, autonomous driving, vehicle dynamics

1 Introduction

The technology of autonomous driving received increased publicity recently due to its application to commercial usage. However, research on computer controlled vehicles goes back to the year of 1984 at Carnegie Mellon University. The Autonomous Land Vehicle In a Neural Network [1], or ALVINN, developed in 1986, described the use of a 3-layers back-propagation neural network to perform the task of road following.

This report describes a project inspired by ALVINN, with the intention to replicate the capability of road following using a modular structured neural network implementation, and demonstrate the results in real time with a 2D vehicle dynamic simulation.

2 Implementation

2.1 Vehicle Dynamics Simulation

In this section, I will discuss the details of the vehicle simulation used in this project.

The simulation was initially developed for the purpose of my capstone project, Evasive Maneuvers and Drifting for Autonomous Driving. In the context of complex maneuvers, the conventional kinematic vehicle model is not capable of capturing the realistic movement of a vehicle. For the purpose of controlling such highly non-linear motion behaviors, a dynamic vehicle model is needed. The theoretical foundation of this dynamic simulation includes contact patch tire dynamic model and 3-states bicycle vehicle model [2,3].

Since this is not the main focus of the project, I will only discuss important components that are implemented. Details about the theories and derivations can be found in the papers from Stanford Dynamic Design Lab [2,3].

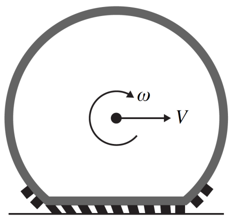


Fig. 1. Brush Tire Model

Tire Dynamics The tire dynamic model used in this simulation is a brush tire model, which utilizes the concept of contact patches. Given the parameters of the tires, along with the longitudinal wheel slip and lateral slip angle, the tire dynamic model will provide the longitudinal and lateral forces on the tires, which will later be used in the vehicle model.

$$\gamma = \sqrt{C_x^2(\frac{\kappa}{1+\kappa})^2 + C_\alpha^2(\frac{\tan\alpha}{1+\kappa})} \tag{1}$$

$$F = \begin{cases} \gamma - \frac{1}{3\mu F_z}(2 - \frac{\mu_s}{\mu}\gamma^2 + \frac{1}{9\mu^2 F_z^2}(1 - \frac{2\mu_s}{3\mu}\gamma^3) & \gamma \leq 3\mu F_z \\ \mu_s F_z & \gamma > 3\mu F_z \end{cases} \tag{2}$$

$$F_x = \frac{C_x}{\gamma}(\frac{\kappa}{1+\kappa})F \tag{3}$$

$$F_y = -\frac{C_\alpha}{\gamma}(\frac{\tan\alpha}{1+\kappa})F \tag{4}$$

where μ is the peak friction coefficient, μ_s is the friction coefficient while sliding, C_x and C_α are the tire stiffness parameters in longitudinal and lateral direction. κ and α denote the longitudinal wheel slip and lateral slip angle.

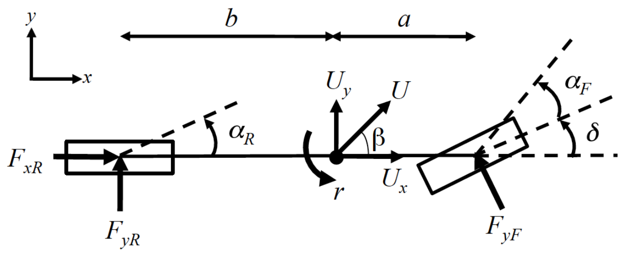


Fig. 2. Bicycle Vehicle Model

Vehicle Dynamics The vehicle dynamics in this project is model based on a 3-states bicycle model with states [U_x, U_y, r]. U_x, U_y denote the longitude and lateral velocity of the vehicle in vehicles frame, and r denote the yaw rate of the vehicle. The model assume a rear-wheel mechanism with control input of wheel speed and steering angel [ω,δ]. The following equations are derived from [3] to provide accurate modeling for larger sliding angle. The implementation of the vehicle dynamics includes more conditions to cover even more extreme scenarios.

$$\dot{U_x} = \frac{F_{xR} - F_{yR}\sin\delta}{m} + rU_y \tag{5}$$

$$\dot{U_y} = \frac{F_{yF}\cos\delta + F_{yR}}{m} - rU_x \tag{6}$$

$$\dot{r} = \frac{aF_{yF}\cos\delta - bF_{yR}}{I_z} \tag{7}$$

where F terms denote the tire forces of the two tires in longitudinal and lateral direction, a and b represent the distance from front and rear wheel axle to the center of gravity of the vehicle.

2.2 Neural Network Implementation

The neural network used in this project is a ground-up implementation in MATLAB with modularity in mind. The basic structure of the network follows a traditional pattern recognition neural network structure, with one input layer, certain number of hidden layers and a softmax output layer.

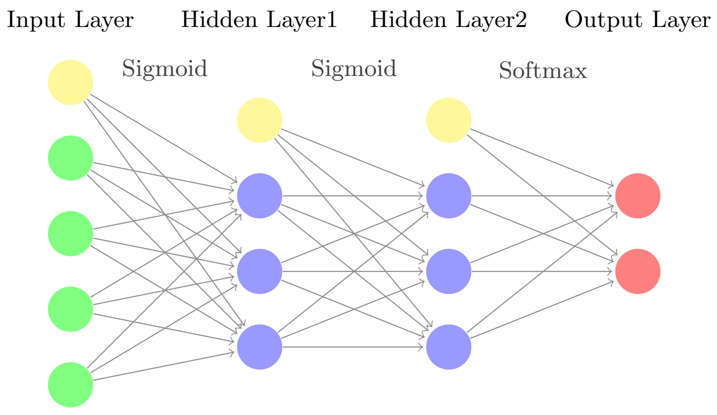


Fig. 3. Traditional Pattern Recognition Structure

Function Signatures In order to support modularity in terms of number of hidden layers and number of neurons in each layer, the implementation includes the following functions.

```
function [W, b] = InitializeNetwork(layers)
```

This function initializes the weights and biases for a fully connected neural network. This input `layers` is defined as a vector containing the size of each layer in the neural network, the length of this vector also defines the number of layers.

It should return the cell arrays `W` and `b` which contain the randomly initialized weights and biases for this neural network.

```
function [output, act_h, act_a] = Forward(W, b, X)
```

`Forward(..)` performs forward propagation on the input data `X`, using the network defined by weights and biases `W` and `b` (as generated by `InitializeNetwork(..)`).

The function should return the final softmax output layer activations in OUT, as well as the hidden layer post activations in `act_h`, and the hidden layer pre activations in `act_a`.

```
function [grad_W, grad_b] = Backward(W, b, X, Y, act_h, act_a)
```

`Backward(..)` computes the gradient updates to the deep network parameters and returns them in cell arrays `grad_W` and `grad_b`.

```
function [W, b] = UpdateParameters(W, b, grad_W, grad_b, learning_rate)
```

This function computes and returns the new network parameters `W` and `b` with respect to the old parameters, the gradient updates `grad_W` and `grad_b`, and the learning rate.

2.3 Data Collection and Training

In order to have the neural network to perform road following, the network first need to be trained with a training set that describe road following behaviors. In this implementation, input data will be road images with resemblance of the ones used with ALVINN. The difference however, is that in this implementation, road images will be a top-down view of the 2D simulated roads instead of a 3D projective view. The labeling of the data will be the commanded steering angle sent to the simulation corresponding to the road image at that moment.

Data collection is done by manually driving the vehicle simulation with a joystick. While driving the simulation, the vehicle moves with a restrained viewport attached to the vehicle, displaying a 32-by-32 meter square view. The viewport is also restrained to the vehicle's heading angle, leaving the curvature of road the only variable in the view. Road images are collected by using MATLAB's

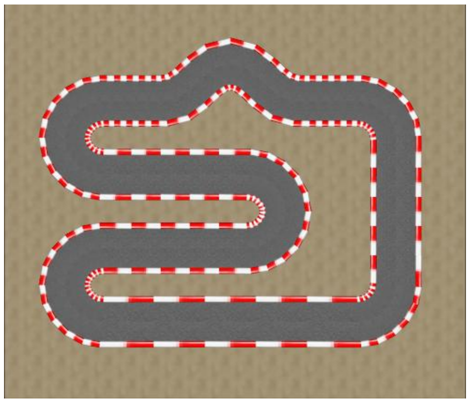


Fig. 4. Simulation Racetrack

`getframe(..)` function while the steering commands from the joystick are stored in a vector. The dataset used for training was acquired by running the simulation at a time step of 0.05 second for several laps on the racetrack, which generated over 6000 training samples.

The raw data of each road image is around 300×300×3 pixels, which is too complex to be used directly as inputs to the neural network. In the training dataset used to generate the results, the images are turned into gray-scale images and then resized to 32×32 pixels images.

The labels are also processed in order to be used with the softmax network. The recorded values are single scalars of the steering angle, while the network expects distributions of the different classes. In this implementation, steering angles are divided into 31 classes e.g. [-0.3:0.03:0.3]. Two different methods were attempted for assigning labels. The first method is done by assigning 1 only to the class of the recorded steering angle, while the second method assigns a Gaussian distribution around the recorded steering angle. The Gaussian method, which is inspired by ALVINN, was later discovered to provide better and more consistent results.

In this implementation, the network initialization is done according to [4]:

$$W_{ij} \sim U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}] \tag{8}$$

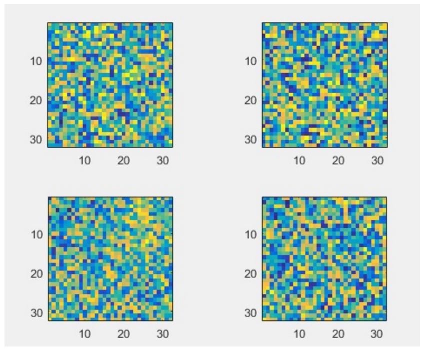
where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and n is the size of the previous layer (the number of columns of W).

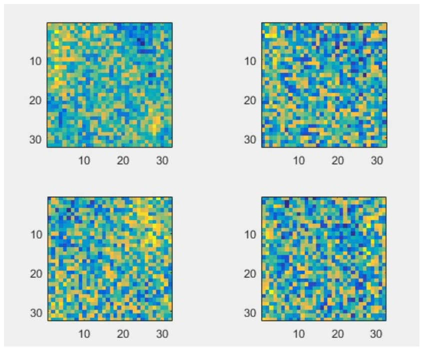
The implementation also uses stochastic learning methods:

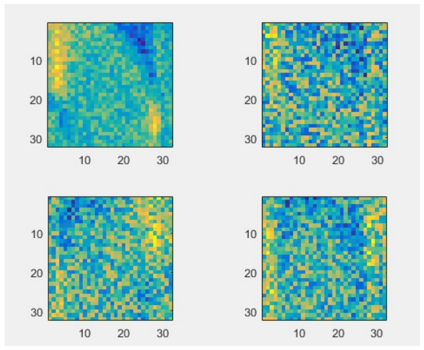
$$w := w - \eta \bigtriangledown Q_i(w) \tag{9}$$

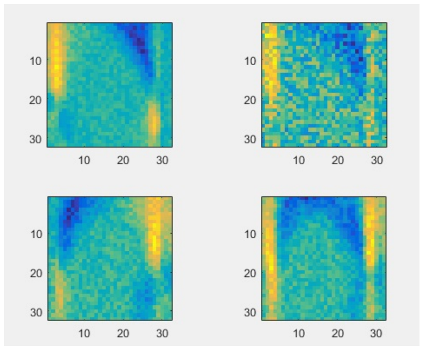
for $i = 1, 2, \ldots, k$, where k is the number of training samples.

The learning rate in this implementation is adaptive to the accuracy and cross-entropy loss in similar fashion as described here [5].


(1)


(2)


(3)


(4)

3 Results

After the network is trained, a new script is used for the neural network to driving the simulation described with the following pseudocode.

```
fig = initialize_graphics();
[W,b] = load_network_parameters();
dt = 0.05;
while(true)
{
    raw_input = capture_drive_scence(fig);
    input = preprocess(raw_input);
    output = classify(W,b,input);
    steer = postprocess(output);
    simulate_car(steer,dt);
    fig = update_graphic();
}
```

The testing is conducted on several different models with different number of layers and neurons, along with different labeling methods.

A noticeable difference appears while comparing the 'all-or-none' labeling method with Gaussian labeling method.

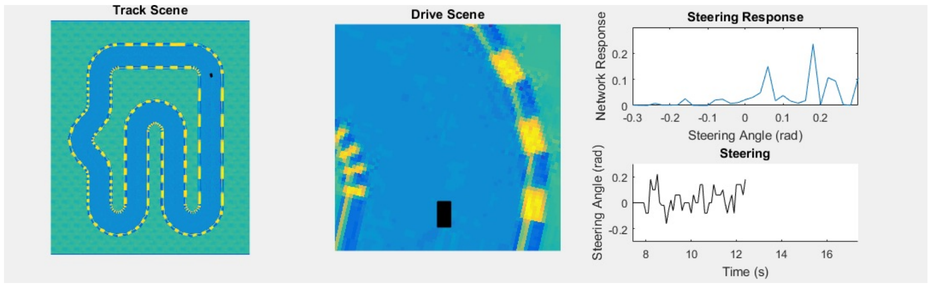


Fig. 5. Driving with an all-or-none Model

The models trained with all-or-none labeling has very frequent unnecessary movements indicated by the noisy steering response and steering recording. These models has the tendency of over-fitting, as during training the models minimize loss strictly to the input commands. These input commands may not be perfect and given the same road image there may be different input commands, which may confuse the neural network. Most of these models failed to follow the track at some point.

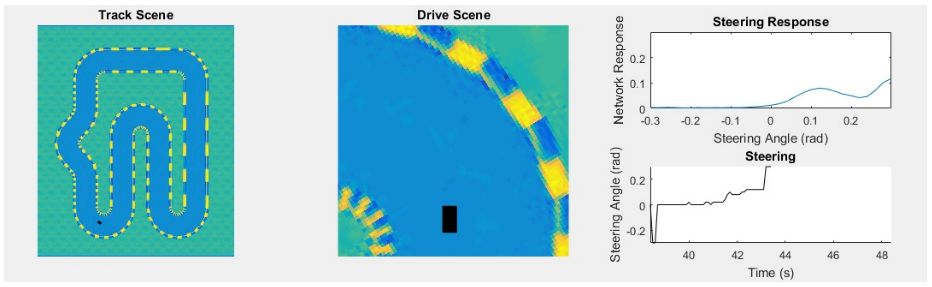


Fig. 6. Driving with a Gaussian Model

The models trained with Gaussian labeling performs much better, as can be seen from figure 6. The Gaussian labeling allows the neural net to have some freedom in terms of fitting the steering response, resulting much smoother and more stable steering responses. Models trained with these methods can perform road following as long as the script is running.

The number of hidden neurons or layers has less impact compare to the labeling methods. The road following problem also does not require too many units to be solved. The one shown in figure 6 uses only 4 hidden neurons and has very good performance.

References

1. D. A. Pomerleau, “Advances in neural information processing systems 1,” ch. ALVINN: An Autonomous Land Vehicle in a Neural Network, pp. 305–313, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989.
2. R. Y. Hindiyeh, *Dynamics and Control of Drifting in Automobiles.* PhD thesis, Stanford University, 2013.
3. C. E. Beal, *Applications of Model Predictive Control to Vehicle Dynamics for Active Safety and Stability.* PhD thesis, Stanford University, 2011.
4. X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.
5. “Gradient descent with adaptive learning rate backpropagation.”