Simulating Automated Car Navigation Using Pygame and Machine Learning

MMANU CHATURVEDI

Summary

- A very simple car game using the cross-platform Python library for making video-games, Pygame [1]
- Pseudo-randomly positioned (in y direction) cars approach a user controlled car from right to left
- User controlled car can move vertically or stay still
- Collected data about the cars and trained a neural network
- Let the trained neural network 'navigate' the car

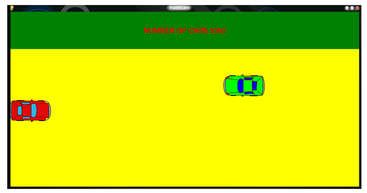
Background

- NavLab at CMU had proposed ALVINN, an autonomous land vehicle in a neural network [2]
- They used simulated road images to train their neural network
- Their neural network had one layer of hidden units
- We take inspiration from them and use a single hidden layer neural network to navigate the user controlled car in our simple game

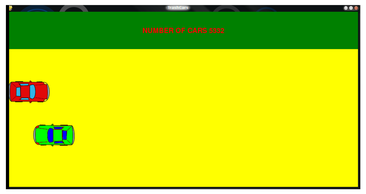
Steps Involved

- **Gameplay**
  - User-controlled car has three states: still, moving up and moving down
  - Detect car-crashes
  - Detect road boundaries
- **Creating pseudo-randomly positioned cars**
  - Decides the difficulty of navigation
  - Ensure that a reasonably easy navigation is possible
  - In our opinion critical for 'good training'
- **Data collection**
  - Collect the position of the user-controlled car and the state (moving up/moving down/still) its in
  - Collect the coordinates of the other cars
- **Neural network implementation**
  - Used the neural network classifier code used in class [3]
  - Used the data collected to train a neural network
  - Used the predicted state to move the car

Method and Results



Car moving down



Car moving up

- Neural Network Description:

Inputs

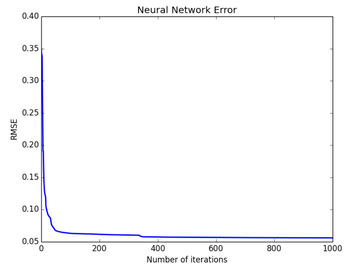
- Difference of x and y coordinates of the other cars w.r.t. the user controlled car
- Distance of the y coordinate of the user-controlled car from the boundaries
- Number of samples = 747 (Training/Testing = 80/20)

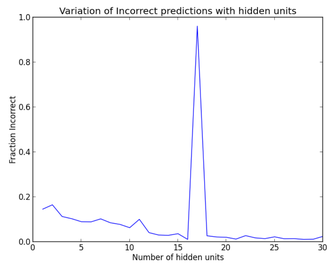
Output

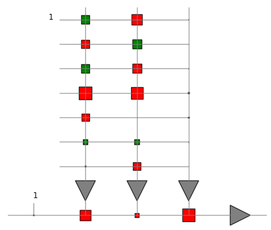
- State of the car, moving up/down

Number of units in the hidden layer : 3

Number of iterations : 1000







Schematic of the neural network

PARAMETERS

SCALE_SPEEDS = 3
TIME_BETWEEN_LOGGING_MS = 500/SCALE_SPEEDS
TIME_BETWEEN_CAR_CREATION_MS = 7000/SCALE_SPEEDS
CAR_SPEED = 3*SCALE_SPEEDS
USER_CAR_SPEED = 2*SCALE_SPEEDS

Observations

- 'Over-prediction' of the still state
- Although lesser error occurs for more number of hidden units, practically, the car navigates better for fewer number of them
- When the coordinates of cars are given directly without taking the relative distance, the NN doesn't work very well
- We keep the car always moving
- Polling frequently and predicting the most occurring state doesn't help too much
- Filler values are important
- When the speeds of the approaching cars are changed, NN controlled car doesn't navigate very well
- When the car creation frequency is increased, our game doesn't work very well

Future Work

- **Object related:**
  - Different car speeds
  - Bends on the road
  - Cars could move vertically
  - Different size objects
  - Other props like friction – loss of control
- **Prediction related:**
  - Reinforcement Learning
  - Create the pseudo-random objects depending on the position of the user-controlled car, like in real games
  - Explore ways to make the neural network not overtrain for the still state
  - Speeds of the cars can be given as another input

References

[1] http://www.pygame.org/wiki/about

[2] Dean A. Pomerleau. Advances in neural information processing systems 1. chapter ALVINN: An Autonomous Land Vehicle in a Neural Network, pages 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989. ISBN 1-558-60015-9. URL http://dl.acm.org/citation.cfm?id=89851.89891.

[3] http://www.cs.colostate.edu/~anderson/cs545/index.html/doku.php