|  |  |  |
| --- | --- | --- |
|  | **Lab - Neural Network Training in Gridworld** |  |

# 

## Exercise 0 - Setting Up Lab

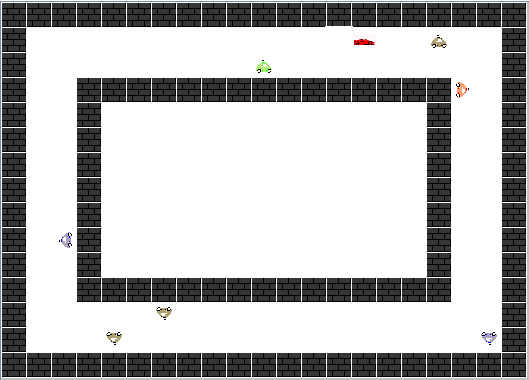
Before you can complete this lab you must set it up.

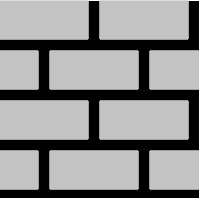
* Download and unzip eclipse-jee-mars-2-win32-x86\_64
* Download and unzip CarLab.zip
* Open Eclipse (Inside of the eclipse folder click the eclipse.exe file)
  + Select the workspace location. This should be in your home directory or on the desktop.
  + Click File→ Import → General → Existing Projects Into Workspace → Next button -> Select the unzipped CarLab folder → ok button → click Finish
  + Close the welcome window
  + If there is a red exclamation mark on the folder once it is installed. You do not have the most recent version of Java installed on your machine and must configure the JRE and Compiler. If you do not experience this problem skip the rest of this section.
    - Right click the CarLab folder → select Build Path → Configure Build Path → Libraries Tab → Add Library → JRE System Library → 1.7 → Finish → Apply and Close
    - On the Left click Java Compiler → At the Compiler Compliance Level field use the drop down menu to select your version of Java (1.7) → Click Apply

## What is the Neural Network Gridworld Simulation?

Gridworld is a retired lab from AP Computer Science A where students learned about programming topics such as object oriented programming and inheritance. You will not be programming the lab to work. This simulation will run without your input. Your job will be to train a self driving car to not crash(or at least not crash as often).

## The Grid will look similar to this...





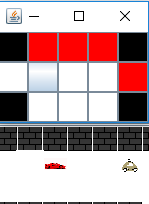
The **Walls** are auto generated every time the grid is created in a different size of track. They do not do anything but serve as an obstacle. The simple cars use the walls as guidance.



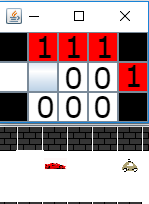
The small **Cars** will drive themselves but they are not smart. They work on a simple boolean decision tree. The lane they are in determines when they turn. If they encounter an obstacle they try to change lanes. If they cannot change lanes, or turn, they will crash.



The **Network Car** uses a neural network to decide where it will go. Like the small cars it will crash if it does not receive sufficient training from you. If this car crashes, a new **Network Car** is created.



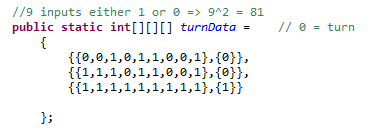
The **Car View** window shows you a representation of what the active **Network Car** can see. The gray square represents the car. The red squares are occupied by some object. Black squares are outside of the car’s view. The square behind the car is included in the car view but is not used in the neural networks that the car uses.



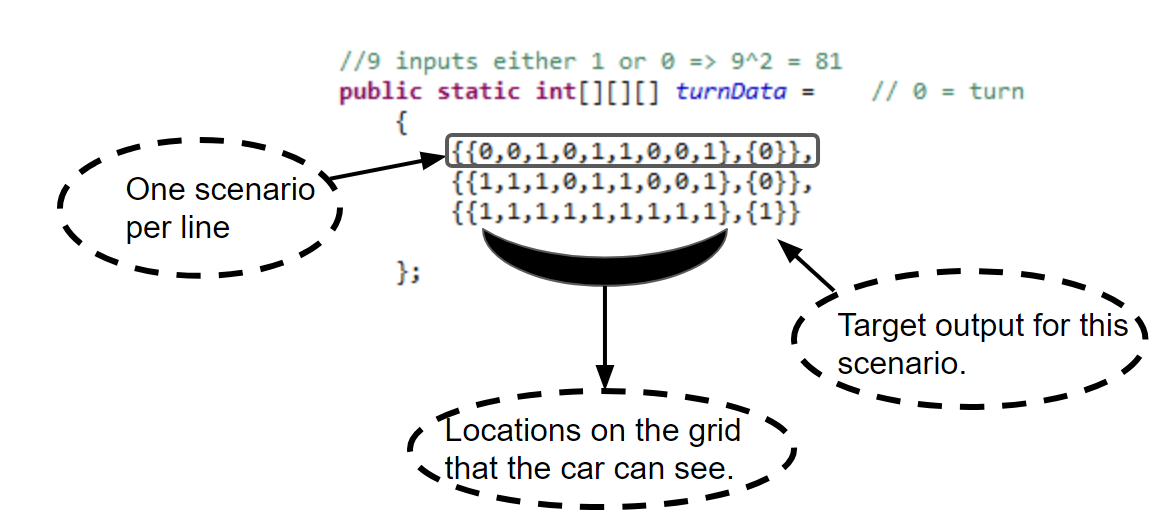
## The Neural Network

The neural network that you will be using is made up of perceptrons. In machine learning the perceptron network is the precursor to the sigmoid neuron. The main difference is that each neuron will only accept inputs of either 1 or 0 and each neuron will only output a 1 or a 0. Due to this an occupied location is represented by a 1 and an empty location is a 0.

## Exercise #1: Training the network (Turning)

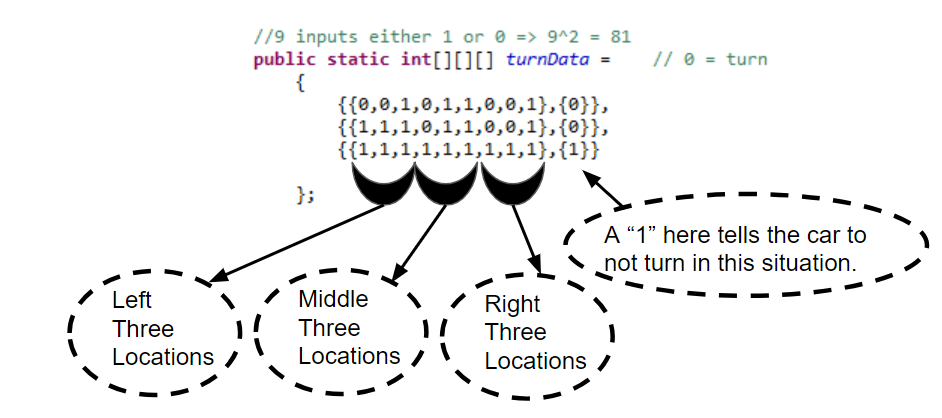


The training data consists of a three dimensional array. What you need to be concerned with is the inner two arrays. Each parring of inner arrays is a single training sample. The first represents the locations and the second is the target output of the network. When the weights for the network are being generated it will alter them until the network output matches all target samples.





Indexes for each training sample are in order from the left side, forward facing, and right side of the car.



For this exercise you will be training the **Network Car** to decide when to turn. The **Network Car** decision making process uses neural networks in a boolean tree in the following order.

If ( the **Turning Network** tells the car to ***turn*** ){

The **Network Car** ***turns***

}

Else if ( the **Forward Network** tells the car it should go ***forward***) {

The **Network Car** goes ***forward***

}

Else if ( The **Lane Change Network** tells the car to change lanes ***left***) {

The **Network Car** changes lanes to the ***left***

}

Else if ( The **Lane Change Network** tells the car to change lanes ***right*** ) {

The  **Network Car** changes lanes to the ***right***

}

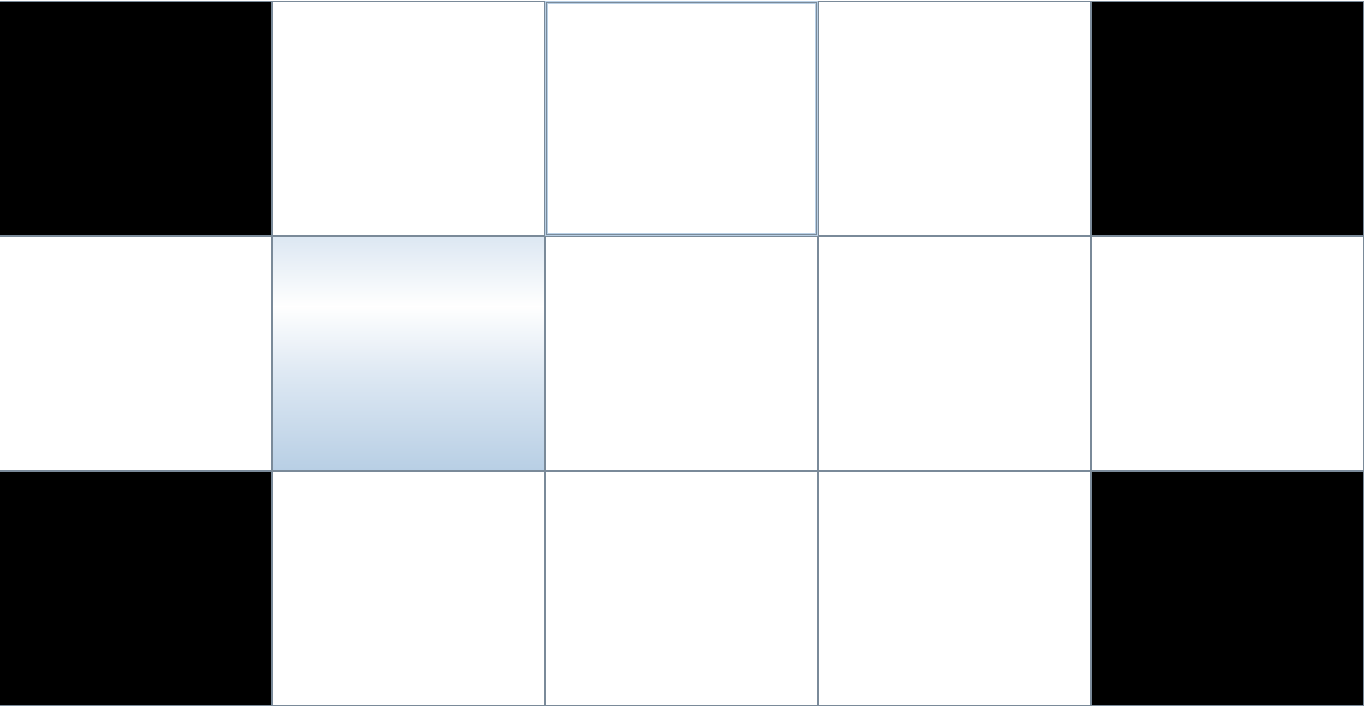
Else{

The **Network Car *stops***

}

As you can see from above there are three networks. The **Forward Network** is done for you. Using four scenarios the network has all of the information it needs to decide if it should go forward. To brute force the problem of the three input network would be easy since there are only eight total possible situations that the network can use. The neural network makes predictions based on the scenarios that it has been trained on to produce outputs for any situation that it has not seen before. Because of this the network does not need all eight possible scenarios to function effectively.

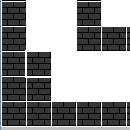
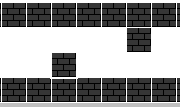
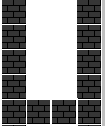
For the **Turning Network** it is not as easy. There are a total of 2^9 possible inputs or 512 potential configurations of inputs. Have no fear! The neural network is in charge of figuring this all out. You should not need more than 25 pieces of training data to run the network. You are given three of these to start with. So how will you determine what needs to be added? Trial and error.

* Run the network.
* If the car crashes or does not turn when you believe it should, record the car view information as training data so that the next time the network runs it will not crash. Making a handy chart will help you to record these findings and plug them into the network.
* Every time you add useful training data the car should theoretically crash less often. 

## Exercise #2: Training the network (Lane Changes)

Next you will complete the  **Lane Change Network**. This network is much simpler as it is identical in size to the **Forward Network**. The size is 2^3 or 8 total possibilities. You are not allowed to use more than six pieces of training data for this network.

* Remember that not all roads are perfect. By clicking on the grid and selecting **Road()** you can add obstacles that complicate the model but improve the network. Your car must be able to respond to these situations.

* If this causes errors in your **Turning Network** you will need to go back and correct that network as well.

## Congratulations at this point you should have a fully trained neural network and have programmed a self driving car!

Paste your working training data for each network below:

package club.westcs.ret.carlab;

public class NetworkCarTrainer {

public static int[][][] forwardData = // 1 = don't go forward

{

{{0,0,0},{0}},

{{0,0,1},{0}},

{{0,1,1},{0}},

{{1,0,0},{1}},

{{0,1,0},{0}},

{{1,1,1},{1}},

{{1,1,0},{1}},

};

public static int[][][] laneChangeData = // 1 = Don't change lanes this way

{

{{0,0,0},{0}},

{{1,1,1},{1}},

{{0,1,1},{0}},

{{0,1,0},{0}},

{{1,0,0},{0}},

};

//9 inputs either 1 or 0 => 2^9 = 512

public static int[][][] turnData = // 0 = turn

{

{{0,0,1,0,1,1,0,0,1},{0}},

{{0,0,0,1,1,1,0,0,0},{0}},

{{0,1,1,1,1,1,0,1,1},{0}},

{{1,1,1,0,0,0,0,0,0},{1}},

{{1,1,1,1,1,1,0,0,0},{0}},

{{0,0,0,1,0,0,1,1,1},{0}},

{{1,0,0,1,0,0,0,0,0},{0}},

{{1,1,1,0,0,1,0,0,0},{1}},

{{1,1,1,1,1,1,0,1,1},{0}},

{{1,1,1,0,1,0,0,0,0},{1}},

{{0,0,0,0,0,0,1,1,1},{1}},

{{1,1,1,1,0,0,0,0,0},{1}},

{{0,0,0,0,0,1,0,0,0},{1}},

{{0,0,0,0,1,0,1,1,1},{1}},

{{0,0,0,1,0,0,1,1,1},{0}},

};

}

For this lab, the car has a view of what is behind it but does not use that view. What possible use would a rearview input be to a car in a model such as this?

The rearview could be used to tell the car if it can turn around and get out of position and it could tell the car to change lanes.

Summarize in your own words what the “Show Network” button does? What is being displayed? (Hint: The answer is in the lab in the menus.)

In the real world a self driving car has many more variables than in this simulation. Take a moment and consider the additional considerations that this simulation has not taken into account.

List some variables a true self driving car would need to include in its programming.

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
7. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
8. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
9. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
10. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If this simulation were to be condensed into a single neural network what would be the most important thing that would need to change? (Hint: Think about what the network outputs.)

How would introducing cars of varying speeds change the network? What new inputs must be considered?

Consider one other application that this simple neural network could be used for. Describe that application and the details of how the network would be used to facilitate the decision making process. Include text and rough diagrams of what you are envisioning.