Undergraduate Directed Studies – Carleton University
Date of Submission: Wednesday, April 10, 2019

Goal-Directed Obstacle Avoidance Using Neural Network

MAAE 4917 Undergraduate Directed Studies Report – Winter 2019

Liam Gritters 100976928
liamgritters@cmail.carleton.ca

Alex Ellery
Alex.Ellery@carleton.ca

A series of neural networks are characterized to implement rover navigation and demonstrate a sophisticated task central to the realization of self-sufficient in-situ resource utilization. The neural network architecture follows that of a multilayer perceptron (MLP) with three layers. The network was trained within a simulator in which the user could command a simple two-wheeled robot to navigate through obstacles to a destination. The network was continuously fed three distance sensor values and a destination represented by the radius and heading from its initial state. The network outputs a direction in which it should move dependent on its current situation.

1 Introduction

The exploration beyond the extraction of water and oxygen from lunar and asteroid resources proposed in most in-situ resource utilization schemes is not often sought after. In particular, the construction of electronic computing devices from lunar resources in a manner that does not require the stringent manufacturing methods of solid-state electronics is explored. Lunar resources can be exploited to create thermionic vacuum tubes as a substitute for discrete transistors. These devices may be composed of fused silica glass encapsulation (from lunar anorthite), aluminum (from lunar anorthite), Kovar wiring (from nickel-iron meteorites), tungsten filament cathodes (from nickel-iron meteorites) coated with calcium oxide (from anorthite), and nickel anodes and control grids (from nickel-iron meteorites) [1]. Rather than constructing traditional CPU-based architectures with these bulky vacuum tubes, more compact neural network circuits based on analogue neurons is proposed. Neural networks are Turing-complete as they are able to recognize or decide data-manipulation rule sets [2]. In order to explore this possibility, a series of analogue neural networks are characterized to implement rover navigation and demonstrate such circuits conducting a sophisticated task central to the realization of self-sufficient in-situ resource utilization. This paper focuses on the software neural network that will implement this rover navigation.

2 Multilayer Perceptron

The type of neural network used for navigation and obstacle avoidance was a multilayer perceptron.

A perceptron is a single neuron model that acts as a precursor to larger neural networks. A neuron is the building block of neural networks and are simple computational units that have a weighted input signal and produce an output signal [3]. In this standard model, multiple inputs are multiplied by weight values and are summed together as shown in Figure 1.

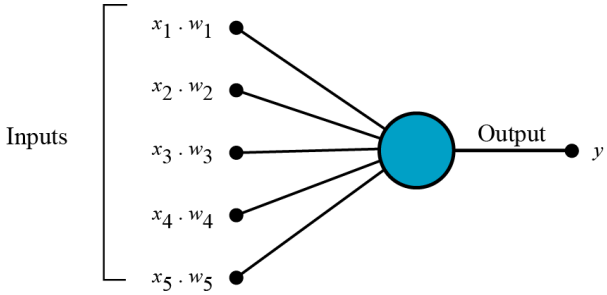

Figure 1: Neuron Diagram

The weighted sum produced passes through an activation function which applies non-linearity to the neural network. Non-linearities help model more complex systems and by having multiple neurons in series, higher order systems can be replicated [4]. The output of a neuron is described by Equation (1), where y_k represents the output at k, φ is the activation function, w_{kj} is the weight between k and j, x_j is the input, and b_k is the bias.

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j + b_k\right) \quad (1)$$

In a multilayer perceptron model, neurons are arranged into a network with multiple rows in series as shown in Figure 2.

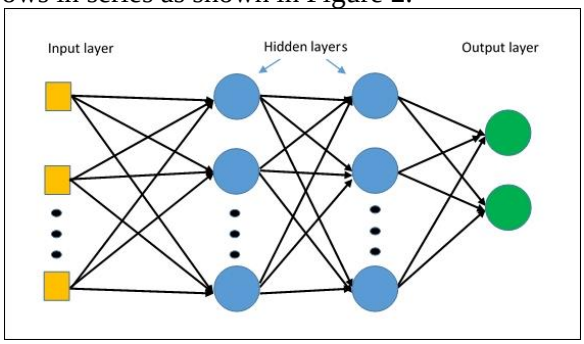

Figure 2: Multilayer Perceptron Network Example [5]

A row of neurons is called a layer with the first layer referred to as the input layer which receives data into the network and the last layer referred to as the output layer which yields a prediction. The layers in between both of these layers are called the hidden layers because they are not directly exposed to the input or output [3].

The network is fed data and calculates a classification. The classification is tuned to more accurately label the data through the training phase by inputting labelled data into the network and adjusting the weight values using the backpropagation training algorithm. The improved weight value is calculated using Equation (2).

$$\theta(t+1) = \theta(t) - \alpha \frac{\partial E(y,\theta(t))}{\partial \theta} \quad (2)$$

Where θ is the weights and biases, α is the learning rate, y is the output, and $\partial E(y,\theta(t))$ is the error function with respect to the weights and biases. The error function used in the network is the Softmax-Loss function and can be calculated from Equation (3).

$$E(y,\theta(t)) = log\left(\sum_{j=1}^{m} e^{\theta_j}\right) - \theta_y \quad (3)$$

3 Architecture Design

The architecture of the network is also referred to as the network topology [5]. The design of the neural network trained for goal-directed obstacle avoidance consisted of two hidden layers.

The input layer receives 5 input values consisting of three distance sensor values, a radius to goal distance, and an angle to goal value in radians. The three distance sensor values are normalized by the max sensing distance, which is then subtracted from 1.0 as shown in Equation (4).

$$Sensor\ Input = 1.0 - \frac{Distance}{Max\ Distance} \quad (4)$$

This represents the percent of an object sensed with respect to the rover and improves training of the network as zero input values are less likely.

The angle to goal input is also normalized by a value of pi, with a forward direction expressed as 0°, a right-angle direction expressed as +90° and a left-angle direction expressed as -90°. The radius to goal input is normalized by the initial value, and as the robot approaches the goal this input shrinks to zero. Thus, the input represents the distance, in terms of percent, to the goal.

This data is then passed to the first hidden layer which consists of 4 neurons. The initial random weight values in the network are generated using a method called Xavier Weight Initialization. This method improves the convergence of the network by limiting the range of the randomly generated values [6]. Doing so, keeps the variance the same between each layer and prevents the signal from exploding to a high value or vanishing to zero as it passes through the network. The variance for the Xavier Initialization formula can be calculated from Equation (5), where w represents the weights and N represents the number of neurons in the input and output.

$$Var(w) = \frac{2}{N_{in}+N_{out}} \quad (5)$$

The activation function used within this network is a sigmoid function which is shown in Equation (6).

$$\varphi(x) = \frac{1}{1+e^{-x}} \quad (6)$$

The data passes through a second hidden layer which also has 4 neurons and is modelled the same as the first hidden layer.

The modified data is then outputted to the output layer which classifies the direction the robot should move given its current scenario as described by the input data. There are four actions that are classified and used to control the movement robot; forward, left, right, and stop. The actions are classified using the SoftMax function shown in Equation (7).

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \; for \; i = 1, \dots, k \qquad (7)$$

4 Implementation

The robot was modelled in a robotic software program called Webots. It comprises 3 distance sensors with a max distance of 100 cm. The left sensor and right sensor are angled -5° and +5° from the center line of the robot. The robot consists of 2 differential wheels and a front cast wheel as shown in Figure 3.

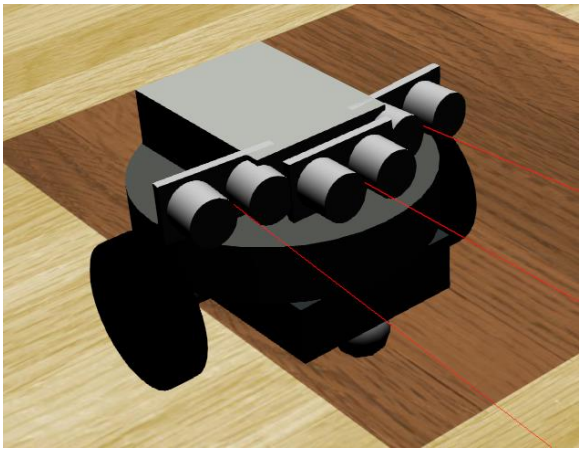

Figure 3: Robot Simulation Model

The turning rate of the robot is 5° per turning command. Decreasing this rate from 8° per turning command to 5° improved the robot's navigation through obstacles as more information was processed between commands.

The simulated world consists of randomly generated objects in the shape of cylinders, spheres, and rectangles. The world is 5x10 meters in size as shown in **Error! Reference source not found.**.

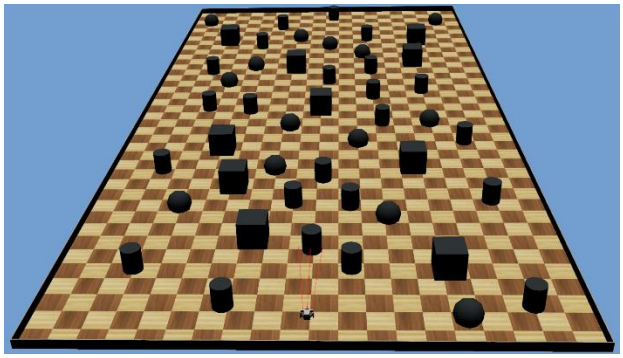

Figure 4: Simulated World

Encoder noise and distance sensor noise is also incorporated into the simulator through a gaussian noise generating function [7].

5 Training

Initially the training data was generated by manually driving the network through the simulated world to the destination. However, this generated an imbalance in the training data as the majority of the time the robot was directed to move forward. To remedy this, a data generation tool was created which randomly generated sensor distance values within 100 cm and a goal within 10 meters. The user than provided a direction the robot should move given the randomly generated scenario. The display of the data generation tool is shown in Figure 5. The white lines represent the distant sensors and the grey circle represents the goal the robot is attempting to navigate to.

This training data was more balanced and provided more instances of difficult situations.

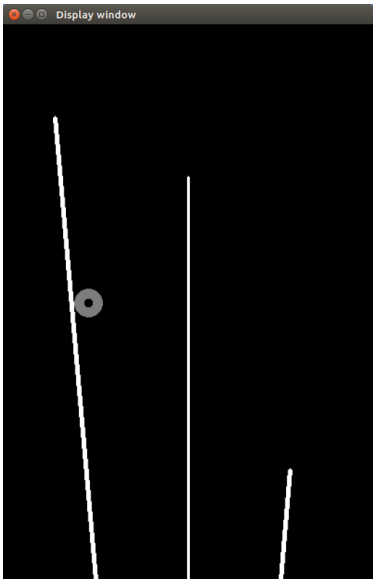

Figure 5: Display of Data Generation Tool

6 Next Steps

Now that the neural network model has been trained and tested within a virtual simulation, it needs to be constructed as an analog neural network and tested in reality. More research is still required in order to achieve this goal.

7 Conclusion

A multilayer perceptron neural network was trained to navigate through obstacles to a specified goal. This was achieved with 3 layers, two of which are hidden layers with 4 neurons.

The network was successfully tested in a virtual simulator given different goals represented as a radius and angle.

Further work must be done to recreate this network in reality and model the weights in an analog circuit. By doing so, the neural network could then, in theory, be constructed on the moon using only the resources there. Further research can also be conducted on training the analog neural networks without the use of software.

References

[1] A. Ellery, "Extraterrestrial 3D printing & in-situ resource utilisation to sidestep launch costs," *JBIS - Journal of the British Interplanetary Society*, vol. 70, pp. 337-343, 2017.

[2] "Turing Complete," 21 11 2014. [Online]. Available: http://wiki.c2.com/?TuringComplete. [Accessed 20 04 2019].

[3] J. Brownlee, "Crash Course On Multi-Layer Perceptron Neural Networks," Machine Learning Mastery, 12 05 2016. [Online]. Available: https://machinelearningmastery.com/neural-networks-crash-course/. [Accessed 20 04 2019].

[4] A. S. Walia, "Activation functions and it's types," Towards Data Science , 29 05 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f. [Accessed 21 04 2019].

[5] G. Zaccone, "Multi Layer Perceptron," O'Reilly, [Online]. Available: https://www.oreilly.com/library/view/getting-started-with/9781786468574/ch04s04.html. [Accessed 20 04 2019].

[6] P. Remy, "Xavier Initialization," 21 03 2016. [Online]. Available: http://philipperemy.github.io/xavier-initialization/. [Accessed 19 04 2019].

[7] "Webots Reference Manual," Cyberbotics, 2019. [Online]. Available: https://www.cyberbotics.com/doc/reference/positionsensor. [Accessed 19 04 2019].