

**HNRS 200**

**Spring 2007**

**Traffic Signal Control using a Neural Network**

**Research Performed by  
Ariel Kopel  
Dr. Helen Yu  
Dr. Art MacCarley**

## **Executive Summary**

Controlling the traffic lights at an intersection is a tedious and difficult control challenge. Modern traffic intersections are controlled using algorithms based on decision making and serial execution, similar to any software program. A new solution is purposed, in which a neural network is used instead to control the sometimes difficult to define intersection scenarios. In order to simulate a neural network, a neural network simulation package was created in C++ which allows the creation and training of any arbitrarily sized neural network. This simulation software can later be used to simulate the control of a traffic intersection, with the training data being collected from a commercially available traffic simulation package.

## **Introduction**

This report describes the methods used to develop the neural network simulation software, and how it can be used in the future to simulate a traffic intersection controller. Neural networks have been around for years, but only recently have they been applied to a variety of applications. Neural networks have recently been widely used for pattern recognition applications such as face and handwriting recognition, but have also started to appear in control based applications such as this one. This study will help answer the question of whether neural network based control is superior to the modern control algorithms applied to traffic intersections.

The goals of this research were to first develop a neural network simulation package in C++, which will allow the creation and training of any arbitrarily sized neural network. This simulation software can be used to create a neural network for any application, but specifically a traffic intersection controller. First, in order to develop the simulation software, a firm understanding of neural networks and the back-propagation training algorithm was obtained. Secondly, training data must be collected from a commercially available and reliable traffic simulation software package in order to train the neural network.

## **Background**

Neural networks got their name from the biology of a human brain. Neural networks, like the brain, are made up of neurons which are interconnected by synapses. Neural networks are useful because they learn by example, allowing the creation of a system that performs a desired function even if the input/output relationship is unknown. For example, it would be very difficult to write a program that performs facial recognition, because the relationship between the inputs (pixel values) and output (whose face is shown) are very difficult to define. A neural network is perfect for this job because it can learn to recognize a person if it is trained using many example pictures.

In order to develop a neural network simulation program, every little aspect of a neural network and the back-propagation training algorithm has to be completely understood. In order to understand how a neural network works, it is first necessary to understand how a neuron works. Figure 1 illustrates the components of a neuron, which are all real numbered values: inputs, weighted synapses, bias, and output. The two computation components of a neuron are the adder and the activation function.

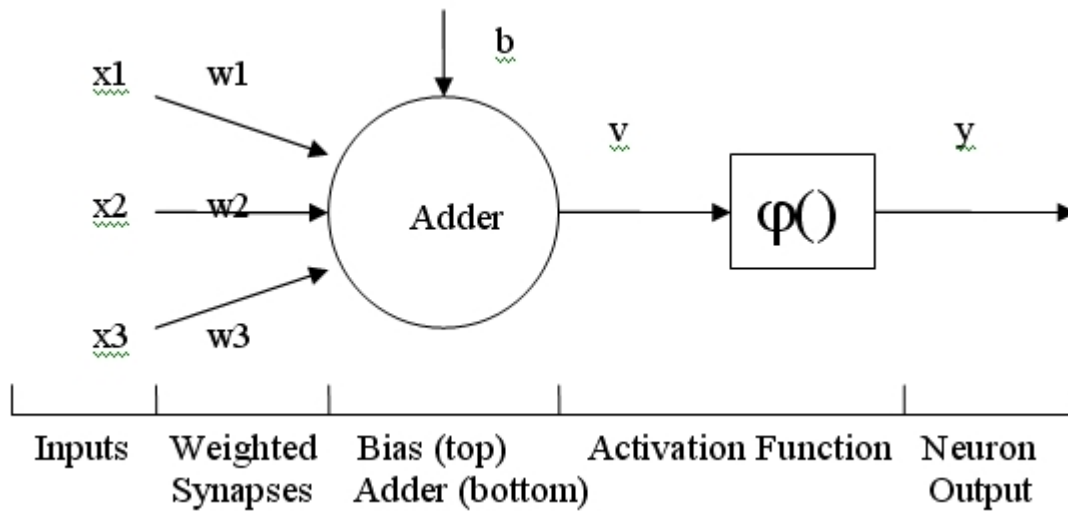


Figure 1 - The components of a Neuron

When calculating the output of a neuron, the weighted inputs are first summed by the adder to produce  $v$ , the induced local field. In this example, the neuron only has three inputs, but in general a neuron can have any number of inputs (each of which is connected to the adder by a weighted synapse). The relationship between  $v$ , the inputs, weights, and bias is shown below. After the induced local field is calculated ( $v$ ), it is plugged into the activation function to produce the output ( $y$ ).

$$v = b + (x_1)(w_1) + (x_2)(w_2) + (x_3)(w_3)$$

$$y = \varphi(v)$$

A typical activation function is shown in figure 2. There are many activation functions to choose from to generate a neuron output, each of which is suited for different tasks. The only requirement for an activation function is that it is continuous. This requirement is imposed by the back-propagation algorithm, which requires differentiating the activation function (impossible if the activation function is not continuous).

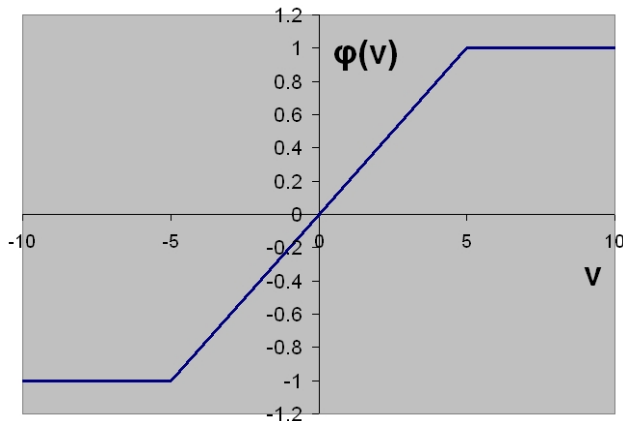


Figure 2 - A typical activation function

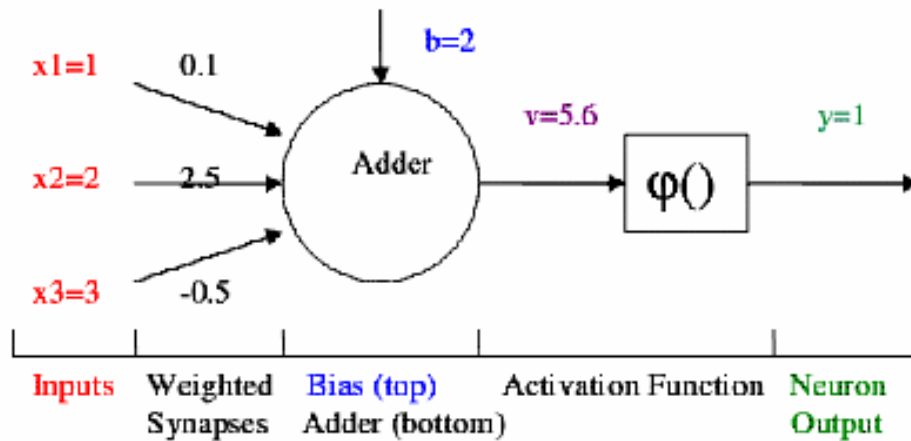
Figure 3 is an example of the output generated by the neuron in figure 1 given the shown inputs, weights, and bias. A detailed breakdown of how  $v$  and  $y$  were calculated is shown below:

$$v = 2 + (1)(0.1) + (2)(2.5) + (3)(-0.5)$$

$$v = 5.6$$

$$y = \phi(5.6)$$

$$y = 1$$



**Figure 3 - Neuron example**

Neural networks are made of anywhere from one to hundreds of neurons (and thousands of synapses). The output of one neuron can be connected to the inputs of another neuron. The utility of neural networks comes from their ability to learn. In this project, the back-propagation algorithm was used to train the neural networks. A detailed explanation of the back-propagation algorithm is very involved and takes up about a chapter in the reference textbook. It will be sufficient to understand the components necessary for the back-propagation algorithm to work.

A neural network is trained using a training set made up of many training samples. Each training sample contains a set of inputs and the corresponding set of desired outputs. The back-propagation algorithm works by minimizing error. Error ( $e$ ) is defined as the difference between the generated output ( $y$ ) and the desired output ( $d$ ):  $e = y - d$ . A neural network is trained one training sample at a time by generating an output ( $y$ ) given the inputs in the training sample. The output ( $y$ ) is compared with the desired output ( $d$ ) to generate an error ( $e$ ). This error is used to modify all of the synaptic weights and biases throughout the neural network in order to minimize the error. If a neural network is trained with enough training samples, it will be able to generate outputs close to those desired. An example of these errors for an arbitrary training set is shown in table 1.

**Table 1 – Back-propagation error example using neuron in figure 3**

Training samples			Generated output and error		
X1	X2	X3	d	y	e
1	2	3	0.5	1	0.5
1	1	1	0.25	0.82	0.57
2	1.5	2.2	-0.7	0.97	1.67

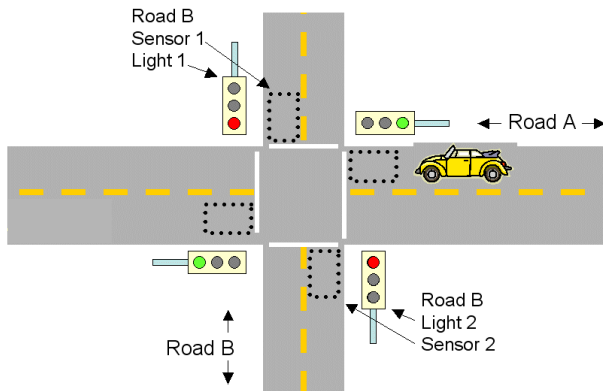
## Design/Theory

In order to simulate a neural network, all of the components described in the background section had to be modeled in software. An efficient way to do this is by using an object-oriented programming language such as C++. In an object oriented programming language, a programmer can create “classes” to represent any abstract object. A instance of a class, or object, contains data members and specialized functions that allow it to perform its specific functions. For this project, the following classes were created: Synapse, Neuron, NeuralNetwork. A brief description of each is provided in table 2.

**Table 2 – Class descriptions**

Class	Description
Synapse	Contains the weight of the synapse and stores what is connected to the synapse. If the synapse is connected to the inputs to the neural network, then a pointer to the input is stored. Otherwise, if the synapse is connected to the output of a Neuron, then a pointer to the neuron is stored. A Synapse object can be asked to retrieve its input.
Neuron	Contains a list of Synapse objects which are connected to the neuron’s inputs. By retrieving the inputs from each of the Synapse objects, a Neuron object can calculate its own output.
NeuralNetwork	Contains a list of Neurons and is in charge of interconnecting them. Once the neural network is set up, the NeuralNetwork class allows the user to perform such tasks as generating an output or training the network.

In order to control a traffic intersection using a neural network, the inputs and outputs need to be decided. Figure 4 shows a common setup of a four way intersection, containing sensors and traffic lights. The inputs to a traffic intersection controller neural network could be: the current state of the intersection (which lights are green) and all of the sensor values. The outputs would then be the next state of the intersection and perhaps the delay before the next state should go into effect. In order to train the neural network, a commercially available traffic simulation software package will be purchased and used to generate the training data.



**Figure 4 - A typical four way intersection**

## Methods

In order to test the neural network simulation software, the neural network was trained to perform one cycle of a sin function:  $y = \sin(x)$ . The neural network only has one input and one output. The desired and generated outputs are shown in figure 5, with blue representing the output (y) and pink representing the desired output (d). Each figure represents the output after a different amount of training. Figure 8 shows that the neural network was able to approximate a sin function very closely after 5000 training cycles.

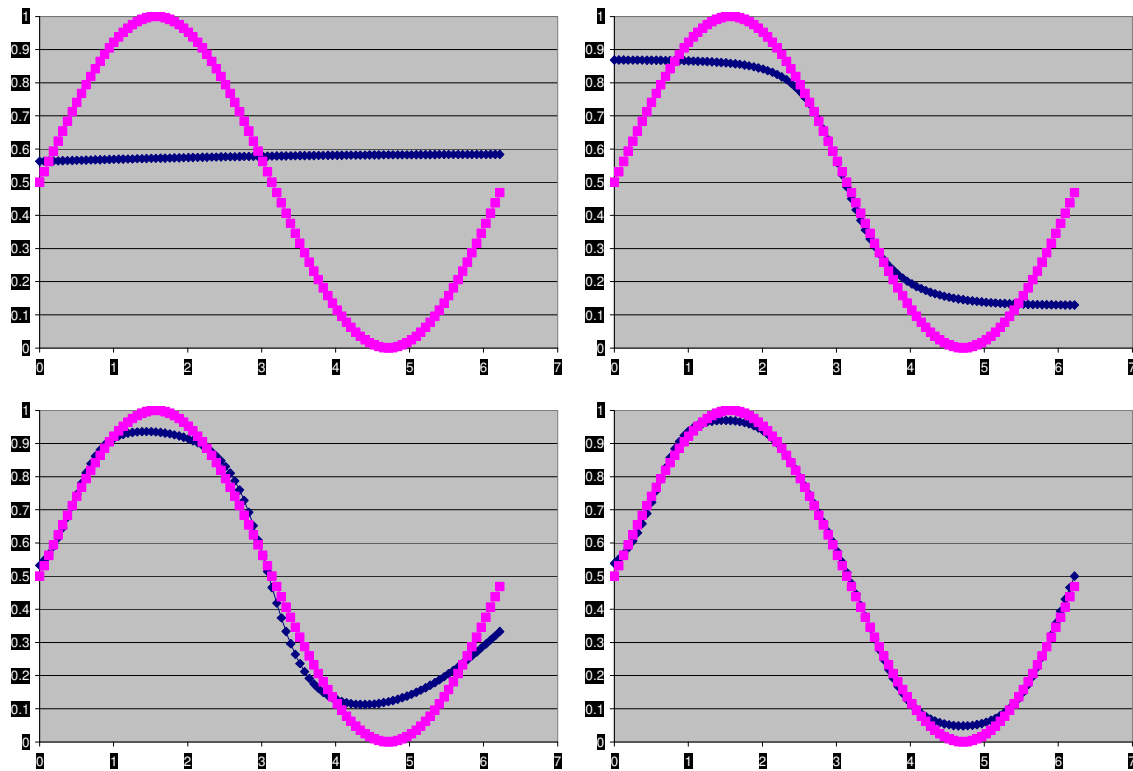


Figure 5 – Generated output after different number of training cycles (top left – 0, top right – 500, bottom left – 2000, bottom right – 5000).

## Conclusions

As shown in the Methods section, the neural network simulation package was successfully created using the C++ programming language. This software package can now be used to create any sized neural network which can then be trained to perform any desired function. A method for controlling a traffic intersection using a neural network was also created, by assigning the inputs and outputs necessary. The data necessary to create this controller is not yet available, but can be simulated using any of a plethora of commercially available traffic simulation software packages.

## Bibliography

- [1] Haykin, Simon. Neural Networks: A Comprehensive Foundation, Second Edition. Hamilton, Ontario, Canada: Pearson Education, 2006.