COSC-071 Project 4: Design and Implementation of a Class for Artificial Neurons

Marcus A. Maloof
Department of Computer Science
Georgetown University
Washington, DC 20057
maloof@cs.georgetown.edu

November 3, 2003

Neuroscience is the study of the brain. Naturally, neuroscientists have limited ability to study live brains, which consist of billions of cells called neurons. However, computers have given such scientists an invaluable tool, for we can write programs that simulate neurons based on our knowledge of the brain. Scientists can then study and manipulate artificial neurons in ways that are impossible for real ones. Project 4 involves writing a class for a simple, artificial neuron.

A simple neuron is a cell with dendrites and an axon, as shown in Figure 1. The neuron receives signals from all of its dendrites. Some signals are strong, some are weak. Some are inhibitory, some are excitatory. In the cell body, the accumulated effect of the strong and weak, inhibitory and excitatory signals results in a state of activation. If the cell's state of activation surpasses some threshold, then the neuron fires by sending an excitatory

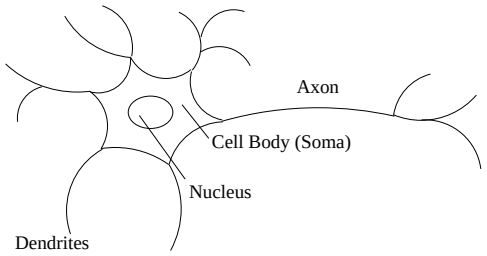

Figure 1: A simple neuron.

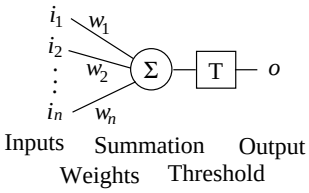

Figure 2: Diagram of an artificial neuron.

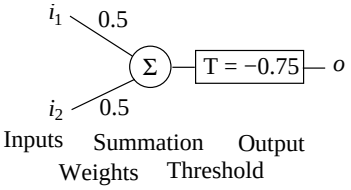

Figure 3: Artificial neuron for the two-input OR function.

signal down its axon to a synapse and to the dendrites of other neurons.

One way to model a simple biological neuron is to use -1 and $+1$ as incoming inhibitory and excitatory signals, respectively. We can use weights for each signal to increase or decrease their strength. The state of activation, in this case, would be simply the weighted sum of the neuron's inputs. Then, if the weighted sum surpasses some threshold, then the output of the neuron would be $+1$; otherwise, the output would be -1. A diagram of this model appears in Figure 2.

Mathematically, we can represent the set of inputs as the vector $\vec{i}$. In general, there will be n such inputs. For n inputs, we will need n weights—one for each input—which we will store in the vector $\vec{w}$. The threshold, T, is simply some real number. Given a set of inputs, $\vec{i}$, a set of weights, $\vec{w}$, and a threshold, T, the output, o, of a neuron is given by

$$o = \begin{cases} +1 & \text{if } \sum_{j=1}^{n} i_j w_j > T; \\ -1 & \text{otherwise,} \end{cases}$$

where

$$\sum_{j=1}^{n} i_j w_j = i_1 w_1 + i_2 w_2 + \cdots + i_n w_n.$$

Of interest to computer scientists is how we might use artificial neurons to realize simple Boolean functions, such as OR and AND. Figure 3 shows an artificial neuron realizing the two-input OR function. To propagate the

Table 1: Inputs, sums, and outputs for the two-input OR neuron.

i_1	i_2	$\sum$	$\sum > T$	o
-1	-1	-1	`false`	-1
-1	$+1$	0	`true`	$+1$
$+1$	-1	0	`true`	$+1$
$+1$	$+1$	$+1$	`true`	$+1$

input $\vec{i} = [+1 - 1]$ through the neuron, we multiply each input with the appropriate weight: $(+1)(0.5) + (-1)(0.5) = 0$. Since $0 > -0.75$, the output of the neuron is $+1$. Table 1 shows the inputs, sums, and outputs for the two-input OR neuron.

For Project 4, you are to implement a class for an artificial neuron. Neuroscientists will want to store configurations (i.e., a threshold and weights) in a file and load the file during object construction or during the program's execution. They will also want to save such configurations. There should be accessor and observer methods for changing and inspecting both the threshold and weights. There should be methods for propagating an input through the neuron. A method should print the neuron's configuration to the console in an understandable format.

The senior software engineers on this project have already completed the high-level design for the `Neuron` class, which they have graciously provided to you before heading out to the golf course. The high-level design (aka class definition) appears in Figure 4. Descriptions of the class methods follow.

To demonstrate your implementation of the `Neuron` class, write a simple driver program in `main`. The driver program should create neurons for the two-input AND function and for the three-input OR function. Do not use the method `Neuron::load` to accomplish this. The construction of these neurons should be hard coded in the main function using `Neuron::setThreshold` and `Neuron::setWeights`. Print to the screen the configurations of these neurons. Write code to propagate all possible inputs through each neuron, and print the truth table to the console (i.e., the outputs for the given inputs). Note that `cout.setf(ios::showpos)` will force the display of the plus sign for positive numbers. `cout.unsetf(ios::showpos)` returns `cout` to the default. Finally, write the configurations to files named `and.out` and `or.out`, respectively. The program should then terminate.

```
class Neuron
{

  public:
    Neuron();
    Neuron(const double t, const vector<double> &w);
    Neuron(const string &filename);
    int size();
    void setThreshold(const double t);
    double getThreshold();
    void setWeights(const vector<double> &w);
    vector<double> getWeights();
    void print();
    bool read(const string &filename);
    bool write(const string &filename);
    int propagate(const vector<int> &input);

  private:
    double threshold;
    vector<double> weights;

}; // Neuron class
```

Figure 4: Class definition for the `Neuron` class.

Method Descriptions

`Neuron()` — Default constructor.

`Neuron(const double t, const vector<double> &w)` — Constructor. Constructs a `Neuron` by assigning `t` to `threshold` and `w` to `weights`.

`Neuron(const string &filename)` — Constructor. Constructs a `Neuron` by loading a configuration from the file `filename`.

`int size()` — Returns the number of inputs of the neuron.

`void setThreshold(const double t)` — Accessor. Sets `threshold` to `t`.

`double getThreshold()` — Observer. Returns `threshold`.

`void setWeights(const vector<double> &w)` — Accessor. Sets `weights` to `w`.

`vector<double> getWeights()` — Observer. Returns `weights`.

`void print()` — Prints information about the neuron in an understandable format.

`bool read(const string &filename)` — Reads a neuron configuration from the file `filename`. Returns `true` if the operation is successful; returns `false` otherwise.

`bool write(const string &filename)` — Writes a neuron configuration to the file `filename`. Returns `true` if the operation is successful; returns `false` otherwise.

`int propagate(const vector<int> &input)` — Propagates `input` through the neuron, returning `-1` or `+1`. Returns `0` if operation fails.