Synapse: A Neural Simulation Language

COMS W4115: Project Proposal

Jonathan Williford

Introduction

Before mathematical models were used in neuroscience, models have mainly been limited to imprecise word models. Such word models that have sounded reasonable in the past have turned out to be inconsistent and unworkable when trying to convert to a mathematical model [Abbott]. Simulation enables precise models to be tested on large interconnected networks. The proposed language Synapse is a language specifically for modeling and testing neural networks.

While every neuron in the brain executes in parallel, most languages are written for architectures that execute sequential. Even as parallel computing becomes more important, parallel support is usually added as an after-thought. For example, CUDA relies on extending C and C++ so that it can take advantage of nVidia's graphic cards and OpenMP adds C preprocessor commands to enable, among other things, parallel for-loops. One of the goals of Synapse is to create a language that is targeted for parallel execution from the ground up.

The name Synapse pays homage to the critical role of the synapse in the brain [LeDoux].

The Language

Modules

The brain contains structures such as micronetworks, which typically contain 100 to 1000 neurons, and hypercolumns, which contain on the order of 10 to 100 micronetworks [Boothe]. A neural network defined by Synapse is structured into modules, which may be nested and can contain any number of neurons.

The following is an example of a definition of a module:

```
module Foobar ( in[5,5], a, b ) >> ( out, x, y )
{
      // ...
}
```

In this definition of `Foobar`, there are 27 declared input neurons and 3 declared output neurons.
The first parameter `in[5,5]` means that there is a 2d array of 25 neurons inside the module which can accept “action potentials” externally.

`Foobar` could be used as an instantiation of the module or can be used to instantiate multiple instances.

For example:

```
$1 >> Foobar.in;
```

Reads the data from the first program parameter and inputs it into an instantiation of `Foobar` with the same name. In this case the input video must consist of images that are 5x5 pixels.

```
Foobar foo[ dim($1,1), dim($1,2)];
```

creates an instance of `Foobar` for every pixel of the first program parameter in a 2d array `foo`.

Activation Functions and Connections between Neurons and Modules

All of the activation functions will be defined in standard libraries. They are based on the integrate-and-fire model. Future activation functions may, however, have memory; which means that previous input or output values may impact the current result.

Current activation functions include:

```
step( expression )
sigmoid( expression )
```

Neurons are connected by using an activation function like:

```
postsynaptic << function( expression );
```

or

```
postsynaptic << expression;
```

where `postsynaptic` is the neuron receiving the inputs, `function` is the activation function and `expression` is an "activating expression" that uses the presynaptic neurons (the neurons that send information to the postsynaptic neuron). Local interneurons and neurons between modules are connected the same way.

Activation functions can also be defined for entire arrays:

```
postsynaptic[1:n] << function( expression[n] );
```

Where `expression[n]` is an expression that results in an array of n values, for example:

```
postsynaptic[1:n] << function( .5*x[1:n]+.5*y[1:n] );
```

Selected elements of an array can also be defined, for example:

```
postsynaptic[1:2:end] << function( x[1:n]-y[1:n] );
postsynaptic[2:2:end] << function( y[1:n]-x[1:n] );
```

Each neuron in a module, including the output neurons, are defined by one and only one activation function inside the containing module. Input neurons are also defined by one and only one activation function, although they are defined outside of the module.

Macros

```
dim( array, d )
```

This macro returns the size of an array along the `d` dimension.

```
wsum( array1, array2 )
```

This macro multiplies each of the elements of array1 with the corresponding element of array2 and then

sums all of the multiplied elements.

`for`

This macro makes it easier to define activation functions, since this language is intended to allow the simulation of very large networks. For example, if we wanted to connect two arrays in reverse order using the sigmoid activation function the following code could be simplified from

```
out[1] << sigmoid( in[3] );
out[2] << sigmoid( in[2] );
out[3] << sigmoid( in[1] );
```

to

```
out[x] << sigmoid[ end - x + 1 ] for x=1:end;
```

when `end` is used as an index, as it is in `sigmoid[ end - x + 1 ]`, it stands for the last index. When `begin` or `end` is used in the `for` expression, it stands for the first or last value respectively that results in a valid activation function. The syntax for indices is borrowed from MATLAB, mean that `1:10` means 1 through 10 and `1:2:10` means 1 through 10 in increments of 2 (1,3,5,7, and 9).

External Input and Output Files

The input and output files are not defined in Synapse but can vary by compiler and can be defined at runtime. I plan on supporting sequences of images as the input and output of a compiled program. The size of the input may be defined in the program, resulting in an error if any other size is given to it. The program will feed one input (or image) per time step. The output will start to be written as soon as all of the output variables are defined (all of the neural pathways have been executed). Since there may be feedback loops, each instance may depend on previous instances.

Example Program

The following program creates a network that models a network of on-center surround cells being applied to an image.

```
module OnCenterSurround ( ker[7,7] ) >> ( out )
{
      out << sigmoid( wsum( ker,
                        [
                            0  ,  0  ,-.036,-.036,-.036,  0  ,  0  ;
                            0  ,-.036,-.036,-.036,-.036,-.036,  0  ;
                          -.036,-.036,  0  , +.2 ,  0   -.036 -.036;
                          -.036,-.036, +.2 , +.2 , +.2  -.036 -.036;
                          -.036,-.036,  0  , +.2 ,  0   -.036 -.036;
                            0  ,-.036,-.036,-.036,-.036,-.036,  0  ;
                            0  ,  0  ,-.036,-.036,-.036,  0  ,  0  ;
                        ]  ) );
}

OnCenterSurround on[ (dim($1,1)-6)/2, (dim($1,2)-6)/2 ];

on[x-3,y-3].ker[:,:] << $1[x:x+6,y:x+6] for x=1:2:end, y=1:2:end;
$2[x,y] << on[x,y].out for x=begin:end, y=begin:end;
```

References

[Abott] Larry Abbott, “Theoretical Neuroscience Rising,” Neuron, 60(3), pp. 489 – 495, 2008.

[Boothe] Ronald Boothe, Perception of the Visual Environment. Spring-Verlag, 2002.

[LeDoux] Joseph LeDoux, Synaptic Self: How Our Brains Become Who We Are. Penguin Group, 2002.