Synapse Language Reference Manual

Jonathan Williford
jw2389 @ columbia.edu

http://synapse-lang.googlecode.com

Contents

1 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 simulating 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 afterthought. 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. Synapse is a language that being created for parallel execution from the ground up.

The source code and documentation (including the LaTeX source for PDFs) may be downloaded from:

2 Document Conventions

Literals are denoted with **monospace**. Syntatic categories are denoted with *italics* and are all lowercase. Identifiers, integers, and floats are represented by *Id*, *Int*, and *Float* respectively. Optional items are indicated with "opt" in subscripts following the item, ex. $optional\text{-}item_{opt}$. Sometimes the syntatic categories are enumerated in the suffix, ex. *item-1*, for ease of reference. Section numbers to the right of the productions indicate the location of syntatic categories not defined in the same subsection.

3 Lexical Conventions

3.1 Comments

Comments begin with the characters /* and continue until */.

3.2 Identifiers

Identifiers consist of letters, digits, and underscores. The first character must be a letter. Identifiers are case sensitive.

3.3 Keywords

The following identifiers are reserved keywords and may not be used for any other purpose.

```
module
size
for
begin
end
kernel
pi
e
sin
cos
exp
pragma
input
```

3.4 Constants

There are two types of constants, int constants and float constants.

3.4.1 Integer Constants

An int consists of one or more digits.

3.4.2 Float Constants

A float consists of a decimal point and at least one digit. The precision of the float is compiler dependent and may even be implemented as an integer using scaling. An int can be implicitly casted as a float, but not vice versa.

3.4.3 Built-in Constants

e and *pi* are built-in constants which are approximately 2.71828 and 3.14159 respectively. The accuracy depends on the precision of float used by the compiler.

3.5 Program Parameters

The input / output sources of the program are specified by $1, $2, etc. For command-line applications $1 corresponds to the first parameter, $2 the second, etc. How they are used will determine whether they are input or output. They may not be both. Every input must be declared with its dimension.

input-decl:
    input *Param dimensions* **;** §4.2

4 Program

A program consists of module definitions, module declarations, kernel function definitions, activation function definitions, and synaptic connections.

program:

/ nothing */*	
input-decl program	§3.5
module-def program	§4.1
module-decl program	§4.2
activation-def program	§4.3.1
kernel-def program	§4.3.2
synap-connection program	§4.4

4.1 Module Definition

Neurons can only be defined in modules. Multiple instances of a module can be used with module declarations (section 4.2). There are three exclusive types of neurons in a module: input neurons, output neurons, and inner neurons. Input neurons receive external signals, output neurons send external signals, and inner neurons are encapsulated in the module.

module-def:

module *Id neurons-1* >> *neurons-2* **{** *module-body* **}** — *Id* is the name of the module. *neurons-1* and *neurons-2* are the list of input and output neurons respectively.

neurons:

Id
Id **,** *neurons*

The inner and output neurons are defined with an activation expression inside of the module using *neuron-def*. The activation expression of an input neuron is defined by synaptic connections outside of the module using *synap-connection*.

All modules are globally defined, however, instances of a module can be declared inside another. Recursive instantions are not allowed.

module-body:

/ nothing */*	
neuron-def module-body	Within a module
synap-connection module-body	Between modules, §4.4
module-decl module-body	§4.2

neuron-def:

Id $dimensions_{opt}$ << *expression* **;**	
Id $dimensions_{opt}$ << *expression* **for** *for-list* **;**	§7.2

The variable iterators can only be used in *expression*. *dimensions* is used to specify the size of the array of neurons and must be equal in size of the expression that is being iterated over.

Modules may be used directly:

```
modulename1.input << modulename2.output;
```

or may be instantiated:

```
modulename1 mods[2];
mods[1].input_neuron << mods[2].output_neuron;
```

4.2 Module Declaration

Once a module has been defined, multiple instances can then be declared.

module-decl:
    Id-1 Id-2 dimensions$_{opt}$;

where *Id-2* is a new instance of the module *Id-1*.

```
module module1 x >> y
{
y = act( .5 * x );
}

module module2 input[100] >> output
{
module1 mod[100];
mod[i].x << input[i] for i = 1:end;
output << act( mod[i].y, gauss );
}
```

dimensions:
    [*const-int-list*]

const-int-list:
    const-int-expr §5.1
    const-int-expr , *const-int-list*

Modules may also contain instantiations of other modules.

4.3 Function Definitions

There are two types of functions allowed in Synapse: activation functions and kernel functions. Activation functions take and returns a scalar while kernel functions generate matrices that fit the context referenced.

4.3.1 Activation Function Definitions

Activation functions take a single scalar and returns a single scalar.
activation-def:
    Id-1 (*Id-2 fparams*$_{opt}$) = *expression* ;

Id-1 is the name of the function and *Id-2* is the name of the local input scalar. *fparams* may be used to define additional optional floats along with their default values.
fparams:
    / nothing */*
    ; *fparam-list*

fparam-list:
    Id = *Float*
    Id = *Float* , *fparam-list*

4.3.2 Kernel Function Definition

Kernel definitions may only be used directly to the right of a convolution operation (§5.2.1).
kernel-def:
    kernel *Id* (*id-list fparams*$_{opt}$) = *expression* ; §4.3.1,5

id-list contains the names for the indices that may be referenced in *expression*. The first index refers to the first dimension (the row if 2D), the second index refers to the second dimension, etc. If w is the number of cells in a dimension, then the indices are enumerated from $-\frac{w-1}{2}$ to $\frac{w-1}{2}$ while incrementing by 1. Therefore, if the dimension is even, then the index values will not be an integer.

The Gabor filter can be implemented as:

```
kernel gabor(x,y,lambda=0,theta=0,psi=0,sigma=1,gamma=0) =
   exp(-((x*cos(theta) + y*sin(theta))^2+gamma ^ 2 * (-x*sin(theta)
         + y * cos(theta))^2)/(2 * sigma^2))
 * cos( 2*pi*(x*cos(theta) + y*sin(theta))/lambda+psi);
```

4.4 Inter-Module Synaptic Connections

The synaptic connections are used to connect the input and output neurons between modules.

synap-connection:

neuron-scoped dimensions$_{opt}$	<< *expression* ;	§4.2
neuron-scoped dimensions$_{opt}$	<< *expression* **for** *for-list* ;	§7.2

See §8 for the definition of *neuron-scoped* and for information on scoping.

5 Expressions

The subsections below appear from highest to lowest precedence. Operators within a subsection have equal precedence.

expression:

primary-expression	
(*expr* **)**	
expr **+** *expr*	
expr **-** *expr*	
expr ***** *expr*	
expr **/** *expr*	
expr **^** *expr*	
- *expr*	
expr ***** *kernel-call*	§6.2
pi	
e	
exp **(** *expr* **)**	
sin **(** *expr* **)**	
cos **(** *expr* **)**	

5.1 Primary expressions

Primary expressions include the below syntatic category plus kernel function calls. Kernel function calls can only appear to the right of a convolution operator (5.2.1).

primary-expression:

Float	
Int	
indexable-expression indices$_{opt}$	
activation-call	§6.2

indexable-expression:
    Param
    Id
    scoped-neuron

indices:
    [*index-list*]

index-list:
    index-expression
    index-expression , *index-list*

index-expression:
    index-num
    :
    index-num-1 : *index-num-2*
    index-num-1 : *const-int-expr* : *index-num-2*

One-based indexing is used. A single colon (ex. [:]) references all of the elements in the respective dimension. *index-num-1* is the first number in the range when expanded and *index-num-2* is the last. If specified, the middle number, *const-int-expr*, specifies the increment, otherwise each consecutive number is included in the range.

index-num:
    const-int-expr
    Id
    begin
    end

Id in this case must be an index defined by a for macro.

const-int-expr:
    Int
    (*const-int-expr*)
    const-int-expr + *const-int-expr*
    const-int-expr - *const-int-expr*
    const-int-expr * *const-int-expr*
    - *const-int-expr*
    size-macro §7.1

The operator . and subscripting group left to right.

5.2 Convolution operator

5.2.1 *expression* ** *kernel-call*

The binary operator ** indicates convolution. The expression to the left must evaluate to a matrix of fixed size. On the right, a kernel function is referenced and a matrix is generated that matches the dimension of the expression on the left. A convolution performs a pointwise multiplication on the matrices and the last convolution of a chain of convolutions takes the sum of the elements of the matrix.

5.3 Unary operator

5.3.1 - *expression*

The unary operator - negates the expression and has the same type. If the expression is a matrix, then every element is negated.

5.4 Exponential operator

5.4.1 *expression-1* ^ *expression-2*

The binary operator ^ indicates expression-1 being raised to the power of expression-2. expression-1 must be a float, an int, or a matrix. If it is a matrix, then each element in expression-1 is raised to the power of expression-2. expression-2 must be a float or an int. The result is either a float or a matrix of float.

5.5 Multiplicative operators

5.5.1 *expression-1* * *expression-2*

The binary operator * indicates pointwise multiplication. If both operands are matrices, then the element-by-element product is returned. In this case, both operands must have the equal dimensions. Otherwise, at least one of the expressions is a scalar. If either of the operands is a matrix, then the result is a matrix; else if either of the operands is a float, then the result is a float; otherwise both of the operands is an int and an int is returned.

5.5.2 *expression-1* / *expression-2*

The binary operator / indicates pointwise division. The same size considerations apply as for multiplication. If either operand is a matrix the result is a matrix; otherwise the result is a float. Integer division does not exist in Synapse. An expression that contains division must not be used when constant integers are required, as when defining the size of a matrix.

5.6 Additive operators

5.6.1 *expression-1* + *expression-2*

The binary operator * indicates pointwise addition. The same size and type considerations apply as for multiplication.

5.6.2 *expression-1* - *expression-2*

The binary operator - indicates pointwise subtraction. The same size and type considerations apply as for multiplication.

6 Function Calls

6.1 Built-in functions

6.1.1 sin

sin (x) calculates the sine of x in radians.

6.1.2 cos

cos (x) calculates the cosine of x in radians.

6.1.3 exp

exp (x) calculates e^x.

6.2 User-defined functions

Kernel functions may be referenced directly after a convolution operator.
kernel-call:
Id ( )
Id (*fparam-list*) §4.3.1

Activation functions may be called anywhere 5 can be used.
activation-call:
Id ( )
Id (*fparam-list*) §4.3.1

7 Macros

7.1 size macro

The size macro returns the size of a module or macro in the specified dimension.

size-macro:
size (*indexable-expression* , *Int*) §5.1

7.2 for macro

The for-macro makes it easier to connect a large number of modules, matrices of modules, and matrices of neurons.

for-list:
for-expression
for-expression for-list

for-expression:
Id = *index-num-1* : [*index-expression*] §5.1

Ranges are specified with the above colon syntax. *index-num-1* is the first number in the range when expanded and *index-num-2* is the last. If specified, the middle number, *const-int-expr*, specifies the increment, otherwise each consecutive number is included in the range.

Inside a for-expression, **begin** and **end** specify the smallest and largest number that are well defined in the expression that the for-macro is being applied respectively. Used as an index to a module or neuron, it stands the smallest (ie. 1) and largest number that are well defined in that module or neuron respectively.

8 Scope

The scope of neurons are local to the current module. The neurons may be specified in any order. Neurons within or between modules may have circular or recurrent connections.

When connecting input and output neurons between modules, the module for which the neuron belongs must be specified.
neuron-scoped:
Id-1 $indices_{opt}$. *Id-2* §5.1

Id-1 specifies the module or matrix of modules and *Id-2* specifies the neuron contained in *Id-1*. *indices* is used if and only if *Id-1* is a matrix of modules. While modules may be nested, only local neurons may be input or output neurons. Hence, only a single module is ever needed to reference a neuron.

All functions and module definitions have global scope. A function may only use functions defined before it. Module definitions may only contain nested instances of modules defined before it. Neither functions nor modules may be recursive.

Synaptic connections can connect modules and neurons regardless of location.

9 Parallelism

Unlike traditional programming languages, all of the values at the synaptic connections are calculated in parallel. Each synapse connection takes a single time step and at time t only the values from time $t-1$ are used for the calculations.

Since it takes a while for the values from the input to be propagated throughout the program, the initial values are to be defined by the compiler runtime options. The compiler or interpreter must support the option of initialization of the neurons to zero. Other options, such as initialization by use of random distributions may also be supported. When the program starts writing output is also compiler or interpreter defined. It must at minimum support the option to start writing output as soon as it starts running, which means the initial output will be garbage.

While various inputs are supported, sequences of images or videos are well suited for reading and writing a large number of values.

10 Future Additions

The following additions are planned for Synapse. Some of the features may be included if it seems essential for the language to be useful.

At any time $t+1$, the current version of Synapse only allows values from time t to be referenced. Future versions will allow any t or older neuron values to be referenced. This will be support by making t a keyword that can be used in arrays. For example, x would be the same thing as writing $x[t]$, $y[t, 1:10]$ would be the same thing as $y[1:10]$, and $x[t-1]$ and $y[t-2, 1:10]$ would refer to previous versions.

Some form of inline switch statements will be allowed in functions.

A dimension macro will be added that could be referenced in weight definitions. A way of automatically normalizing the dynamic kernels will be added. For example, Z may become a macro that stands for the sum of the weights of the current kernel.

Matrix constants can be defined in the form of: [[1, 2, 3, 4; 5, 6, 7, 8]].

Support for spike trains will be added by adding support for booleans and by adding support for Poisson spike generators.

11 Examples

11.1 Image Sharpening

The following program creates a network that models a network of on-center surround cells. It sharpens the video that is passed.

```
kernel sharpen( x,y; pos_sigma=1.0, neg_sigma=2.0 ) =
1 / 2 * pi * pos_sigma ^ 2
* exp( -.5 * ( x^2/pos_sigma^2 + y^2/pos_sigma^2 - 2*x*y / pos_sigma^2 ) )
- 1 / 2 * pi * neg_sigma ^ 2
* exp( -.5 * ( x^2/neg_sigma^2 + y^2/neg_sigma^2 - 2*x*y / neg_sigma^2 ) );

module OnCenterSurround in[7,7] >> out
{
out << in ** sharpen();
}
```

```
OnCenterSurround on[ (size($1,1)-6)/2, (size($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;
```

11.2 Image Mirror

The following one-liner mirrors the video.

```
$2[ y,size($2,1)-x+1] << $1[y,x] for x=begin:end, y=begin:end;
```