NeurOn Neural Network Design Language and Compiler

1.0.0 Alpha



User Manual

Contents

* About neurOn
* Command Line Arguments
* Compilation Chain
* Neuron and Connectome files
* CNS files
* Syntax
  + Expressions
  + Groups
  + Lists
  + Assignment
* Operators
* Variables
* Macros
* Expressions
* Functions
* Inline API
* Dependencies and Imports
* Makefile compilation
* Structure of a connectome file

About neurOn

Biological neural systems are extraordinarily complex. In a SAPA system, the intention is to create a sort of hybrid logical system that behaves similarly to a biological model but built to function on modern hardware. The intent is to create programs capable of complex learning.

With rapid development in mind, it can be extremely tedious and time consuming to constantly write, re-write and edit lower level code. Thus, neurOn offers a higher level way of designing systems without worry about the underlying overhead.

The principle is similar to a hardware design language such as VHDL or Verilog. Circuits may be designed via written expressions and compiled into a final product. However, there are several major differences.

First of all, SAPA systems are dynamic. The program written and compiled will only yield an optimized program and a starting point. From there, a system is free to evolve and learn as time progresses. Secondly, the systems intended to be created are significantly more complex than an average electrical circuit. Thus, expressions are designed with extreme complexity in mind and make heavy use of loops and regional generations.

The intent of this part of the SAPA toolset makes the development of learning machines quick and (relatively) simple. The remainder of this manual contains definitions, explanations and examples to begin creating unique neural systems.

Command Line Arguments

The table below lists all accepted command line arguments. Note that options must be set in the form of -[n] or --[setting]=*value* with no spaces in an expression.

|  |  |
| --- | --- |
| CMD Option | Description |
| -d | Enable Debugging |
| -b | Enable Build (Automatic if no makefile specified) |
| -m | Searches source directory for makefile (implicitly enables build) |
| -c | Enables connectome engine compilation |
| -r | Enables runtime engine compilation |
| -v | Prints version |
| -s | Prints symbolic signature |
| -h | Prints help information |

Compilation Chain

The compilation process in neurOn is different from many other languages in the regard that it actually produces two conjugate outputs. The first is a runtime engine, and the other a ‘connectome file’, or simulation file.

The runtime engine is an executable binary with pre-built types and functionality. On startup, a built runtime engine acting a simulation environment loads a set bytecode file called a connectome, or \*.ctm, file. From here, all saved-state information from when the simulation was last interrupted can be resumed. In addition, this allows an optimized environment to load compatible networks designs.

Neuron and Connectome Files

CNS files

A CNS file is the project file used by the compiler to generate connectomes and runtime engines. Here, compilation settings can be set, and the required neurOn source files are identified.

CNS files are written in blocks that identify what sort of information is to follow. The first block is the *config* block which sets project information and compilation preferences. The second is the *source* block which lists all files to be included in compilation, and allows attributes to be assigned to each. An example of a CNS is described below:

**$config**

**bitmode=64**

**project\_title=”demoTitle”**

**version=”1.0.0”**

**$source**

**demo1.nrn ? {master, attribute2}**

**demo2.nrn**

**demo3.nrn**

Syntax

Operators

Variables

Macros

Expressions

Functions

Inline API

Dependencies and Imports

Makefile Compilation

Structure of a Connectome File