Technical description of the programs used for generating neural models

Brain Imaging and Modeling Section, NIDCD
National Institutes of Health, Bethesda MD 20892

August 15, 2003

1 Introduction

This documentation for implementing large-scale neural models (LSNM) is based on computer code developed by Husain and Horwitz (2002) (auditory model) and Tagamets and Horwitz (1998) (visual model). The applications are explained first in general terms, followed by their application to the implementation of Husain and Horwitz (2002)'s auditory model.

2 Programs

Figure 1 schematizes the process of network generation and simulation for a neural network model. $au*.out$ and all of the other output files for each module of the network are generated by executing the command **sim** $< au*.s >$, where $< au*.s >$ contains pointers to the files $au*.rs$ and $ausimweightlist.txt$. The file $au*.rs$ contains a script of the experimental timeline for one trial. The file $ausimweightlist.txt$ contains a list of the files containing the connection weights of the network. The timeline script specifies the stimuli by using the $*.inp$ files, which contains a description of the stimulus. The list of files with the network connection weights are in the format $*.w$. This format is generated by the **netgen** $< *.ws >$ command, where $< *.ws >$ is a file created by hand (or backprop in the visual model) containing weight values. The script **makeweights** can be used here in order to execute **netgen** for each one of the $< *.ws >$ files.

3 Commands

Two unix commands are used. $netgen$, to generate the network, and sim, to execute a given simulation.

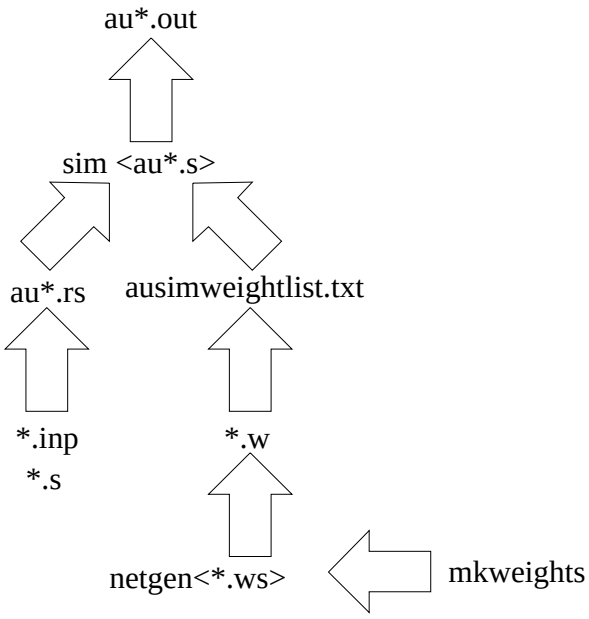

Figure 1: Process for generating and executing a network model

3.1 netgen

SYNOPSIS netgen $[*.ws]$

DESCRIPTION For the given $*.ws$ file, it generates a weights file $*.w$. The script shells can be used here to execute netgen1 on several files. The script shell is called $mkweights$.

3.2 sim

SYNOPSIS sim $[*.s]$

DESCRIPTION Given an experimental trial script, it runs a simulation and generates output files.

4 Generation of network structure with netgen

Programs used here are in C++, and they are:

unixmain.cc Main netgen interface

generate.cc Generates the network

The following headers are necessary to run the cc programs:

win.h Definitions specific to unix

defs.h Definitions of command options

structs.h Defines the network structure

decls.h

prototyp.h

A makefile can be used to generate the executable **netgen**:

makefile

5 Executing the simulation with sim

Programs used here are in C++, and they are:

unixmain.cc Main **sim** interface

win.cc Functions specific to unix

init.cc Initializes simulation parameters

parse.cc Interprets the commands for simulation

actfuncs.cc Neuron activation functions

messages.cc Defines error messages

outfuncs.cc Functions for computing output of a neuron

parsecxn.cc Parses connection commands

parseset.cc Defines set structure

siminit.cc Initializes data structures

simlearn.cc Learns weights with hebbian rule

simulate.cc Executes network simulation

writeout.cc Writes out output data in MATLAB format, including PET calculations.

The following headers are necessary to run the cc programs:

win.h Definitions specific to unix

simdefs.h

classdef.h

externs.h

macros.h

simproto.h

simdecls.h

A makefile can be used to generate the executable **sim**:

makefile

6 Definition of a network

6.1 Commands

The following commands are available to design experimental trials:

SET Creates a new network module.

#INCLUDE Reads specifications from external file.

RUN Runs simulation for specific number of iterations.

CONNECT Connects one module to the other by using the specified pattern of connections.

6.1.1 Object properties for the command SET

ACTRULE *ActRule*. Possible values are: Clamp, clamp activations to set values; SigAct, non-differential style sigmoid rule in which a new activation is just a sigmoid of the difference between input and threshold; DiffSig, differential equation-style sigmoid activation which add sigmoid of the input-threshold difference to old activation and subtracts from it the product of decay and old activation; Lin, linear activation rule, Noise, noisy clamp; Sigm, sigmoidal activation; Shift, shifting activation rule which shifts the clamped activation by dx every $delta$ iterations (useful for learning invariance of position of an object).

INPUTRULE In case of input rules. Not implemented.

LEARNRULE *LearnRule*. Possible values are: Off, no learning; Aff, afferent hebbian; Eff, efferent hebbian.

OUTPUTRULE *OutputRule*. Possible values are: C; SumOut.

NODEACT *Node Activation*. Initialize activation values. Possible values: ALL, all nodes;

BINACT Initial activation values, set by input matrix

PHASE Initial phase values. Possible values: ALL.

PARAMETER *Parameters*. Set parameter values. Possible values: ALL.

LEARNPARAM . Learning rule parameter values. Possible values: ALL, learn all parameters.

SHIFT Explicitly shift activations

TOPOLOGY . *Topology*. Specifies the network topology. Possible values: LINEAR, linear topology; RECT, rectangular topology.

WRITEOUT *Write*. Set file output to every i iterations

WRITEWEIGHTS *WW*. Set weight writeout output to every i iterations

6.1.2 Object properties for the command #INCLUDE

The only parameter here is the name of the file.

6.1.3 Object properties for the command RUN

The only parameter here is the number of iterations.

6.1.4 Object properties for the command CONNECT

FROM *From*.

6.2 Data structures

The global data structure used in the programming of LSNM is illustrated in Fig. 2. NodeSet is a set of nodes that can be called a module. Initially, there is reserved space for up to *max* number of modules. Every module has its properties plus a pointer to the actual array of nodes that compose that structure. This nodes are of type NodeStruct and can be of different lengths (although in the current models this number is the same for all modules). Each node of the module contains its own specification plus pointers to the weights to which is connected, thereby affecting other nodes in any location of the network. WeightStruct is the data structure for a connection weight between two modules. Every connection weight contains its own set of specifications plus a pointer to the node to which it affects.

Note: only the relevant functions are defined below. Figure 3 illustrates the structure of a module in the network. The pointer values that make up this structure are as follows:

Sfs Points to a file

Wtfs Points to a weight file

Params

LrnParams

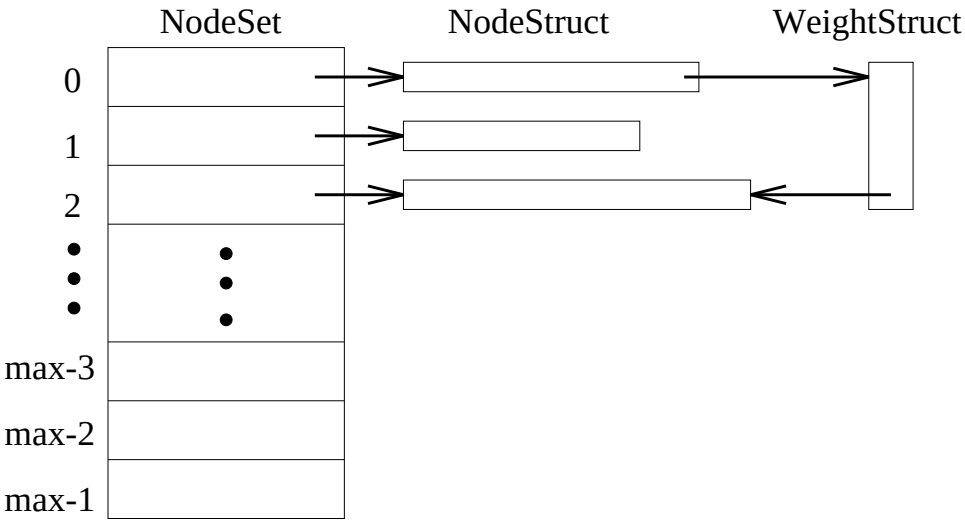

Figure 2: Global structure of an LSNM model.

NodeSet structure

SetName	* → Sfs
SetIndex	* → Wtfs
WriteOut	* → Params
WriteWts	* → LrnParams
Topology	* → Vars
XDim	* → InitSet
YDim	* → OutputRule
NSetParams	* → InputRule
NLrnParams	* → ActRule
NSetVars	* → Update
N_Nodes	* → LearnRule
	* → Node

Figure 3: Data structure for a network module. Pointer values are located at the right and non-pointer values are located at the left.

Vars

InitSet

OutputRule

InputRule

ActRule

Update

LearnRule

Node Points to the first of the actual nodes of the current structure

The non-pointer values that make up this structure are as follows:

SetName Character string containing structure name

SetIndex

WriteOut How frequency the state of the structure are output to a file

WriteWts How frequency the weights of the structure are output to a file

Topology Network topology

XDim Number of elements along the x axis of matrix

YDim Number of elements along the y axis of matrix

NSetParams

NLrnParams

NSetVars

N_Nodes Number of nodes in this structure

Every node of the structure described above is a structure of the type *NodeStruct*. Figure 4 illustrates the structure of a node in each module of the network. The pointer values that make up this structure are as follows:

InputPtr

AltInputPtr

InWt Points to a weight class

OutWt

AltWt

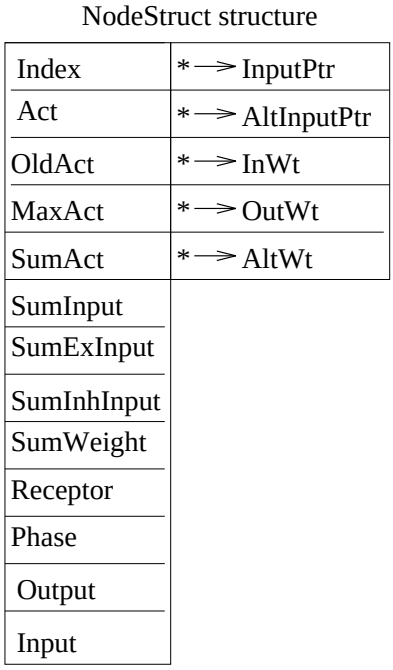

Figure 4: Data structure for a single node. Pointer values are located at the right and non-pointer values are located at the left.

The non-pointer values that make up this structure are as follows:

Index Index of the current node

Act

OldAct

MaxAct

SumAct

SumInput

SumExInput

SumInhInput

SumWeight

Receptor

Phase

Output

Input

WeightStruct structure

LearnOn	* → Set
Value	* → DestNode
WtVar	

Figure 5: Data structure for a single weight. Pointer values are located at the right and non-pointer values are located at the left.

Figure 5 illustrates the structure of each of the weights interconnecting the modules of the network. The pointer values that make up this structure are as follows:

Set

DestNode

The non-pointer values that make up this structure are as follows:

LearnOn

Value

WtVar

6.3 How to define a network

The following example illustrates the use of the command SET. In this example, the MGN module was defined for the auditory model (taken from *auseq.s*, a script file for the sequences memory model):

```
set(MGNs,81) {
  Write 5
  Topology: Rect(1,81)
  ActRule: Clamp
  OutputRule: SumOut
  Node Activation { ALL 0.01 }
}
```

The command *set* creates a new structure called *MGNs*, consisting of 81 nodes, and initializes all of the structures' elements to zero. The instruction *Write 5* indicates that the state of each of the 81 nodes will be recorded in an output file every 5 iterations. The instruction *Topology: Rect(1,81)* sets the topology of the network to a vector of one column by 81 rows.

The following example illustrates how to use the commands #INCLUDE and RUN (taken from *auseq.rs*, a script file for the sequences memory model):

```
#include noinp_au.inp
Run 200  % <---------- 1000 ms
```

The following example shows the first few lines of a network connection specifications (taken from *mgnsea1d.w*, a file containing connection weights between MGN and the excitatory part of down-selective Ai of the sequences memory model):

```
Connect(mgns, ea1d)  {
  From:  (1, 1)  {
    ([ 1,81]  0.047928)     ([ 1, 1]  0.099059)     |                |
  }
```

This example file of specification of connection weights was generated by the command **netgen**. The command **netgen** takes a **.ws* file and generates the connection code. The original file of corresponding to the connections above is *mgnsea1d.ws*, which contains the following lines:

```
mgns ea1d SV I(1 81) O(1 81) F(1 3) 0 0.0 Offset: 0 0
0.05:0.003 0.10:0.002 0.00:0.002
```

Note: strict observation of syntax in these files is important since a simple syntax mistake can make the weight generation process to reverberate indefinitely. The first two strings in this file, *mgns* and *ea1d*, represent the two modules between which the weights are being defined, that is, MGN and Ai excitatory down-selective module. *S* indicates that the fanout weights will be specified in the current weights file. Other options are *R* and *A*. The R option allows to specify that random weights will be generated. The A option allows to specify that absolute weights positions will be used. The symbol *V* after the *S* indicates a verbose output. The expressions $I(181)$, $O(181)$, and $F(13)$ specify the input set size, output set size, and fanout size, respectively. The three groups of number in the second row of the weights file indicates each of the weights used for each of the connections. In the above example, a fanout size of 1 to 3 was specified, which means that each unit of MGNs will be connected to 3 units in EAid. The connections from the MGN module will made from the current unit to the corresponding unit in the EAid module and to the 2 nearest neighbors of that unit. The weights of those connections are the numbers in the second row of the file. Accordingly, $0.05 : 0.003$ specifies that the first of those weights will equal 0.05 ± 0.003. The second weight, $0.10 : 0.002$, represents a connection weight of 0.10 ± 0.002. The third weight, $0.00 : 0.002$, represents connection weights of 0.00 ± 0.002. The small variations around the specified weight values are made by addition of bounded random noise.

7 Simulating an experimental trial

7.1 Steps to run the simulation

sim $< au* >$, where $< au* >$ is a file with extension *.s, and contains all the specifications needed to execute one simulation of the network.

7.2 To plot raster files of the simulations

run plotoutput.m in MatLab

7.3 To execute the decision module of a simulation

run decision.m in matlab

Note that this module can be executed independently of the others. The code counts the number of neurons in R that responded. If this number exceeds a given threshold, say 5, then it signals a match. A mismatch is signaled otherwise.

References

Husain, F. and Horwitz, B. (2002). Understanding central auditory processing via functional brain imaging and neural modeling studies. Manuscript in preparation.

Tagamets, M.-A. and Horwitz, B. (1998). Integrating electrophysiological and anatomical experimental data to create a large-scale model that simulates a delayed match-to-sample human brain imaging study. *Cerebral Cortex*, 8:310–320.

A The extended auditory model

The diagram of the auditory model before modifications is shown in Fig. 6. This network

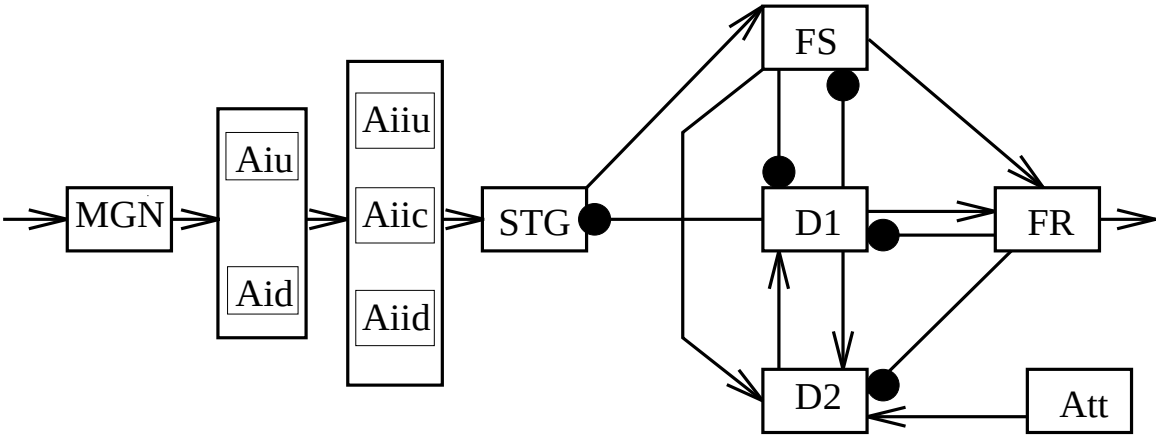

Figure 6: Model of prefrontal working memory of auditory stimuli. For simplicity, only main connections are depicted. Each module is composed of both an excitatory and an inhibitory submodules.

was extended in order to perform pattern recognition of long sequences of tonal patterns. The extended network includes three modules for each prefrontal module in the old network. That is, the modules FS_a and FS_b were added to FS, $D1_a$ and $D1_b$ were added to D1, $D2_a$ and $D2_b$ were added to D2, and FR_a and FR_b were added to FR. In order to make this modifications, a set of **set** instructions were added in the file *auseq.s* (see description of **set** above).

After adding the new modules in the script file *auseq.s*, connection weights were also added by creating **.ws* files in the *weights* directory. The connection weights among the

new modules and between each of the new modules and STG were identical to the connection weights among the old modules and the old modules and STG (see below subsections on connection weights between modules and submodules). As an example, the names of the new connection weights files, **estgexfs_a.w** and **estgexfs_b.w**, were added to the script files containing connections weights, namely, the following lines were added to **ausimweightlist.txt**:

```
#include weights/estgexfs_a.w
#include weights/estgexfs_b.w
```

The connections internal to the FS module were also added:

```
#include weights/exfs_aexfs_a.w
#include weights/exfs_ainfs_a.w
#include weights/infs_aexfs_a.w
#include weights/exfs_bexfs_b.w
#include weights/exfs_binfs_b.w
#include weights/infs_bexfs_b.w
```

A similar procedure was followed in order to add the connection weights of the new buffers for modules D1, D2, and R.

Next, the ***.w** files were regenerated by using the shell script **mkweights**, after which the simulation was executed. Remember that the shell script **mkweights** executes **netgen** for each of the ***.ws** files:

```
#! /bin/csh

foreach file (*.ws)
  ../bin/netgen $file
end
```

It is assumed that a directory WEIGHTS already exists. The WEIGHTS directory contains all the connection weights between regions. These weights are contained in *.w files. The *.w files were generated by executing the command $netgen1 < *.ws >$ for each of the *.ws files. The shell script $mkweights$ can be used to execute netgen1 for each of those files. The *.ws files can be modified by hand, unlike most of the other input files for network generation and simulation.

The original *.ws files are named in the following way:

A.1 New brain regions

A gating mechanism was added, by adding the following structure in the file *auseq.s*:

```
set(Gate,1) {
  Write 5
  ActRule: Clamp
  OutputRule: SumOut
  Node Activation { ALL 1.0 }
}
```

This command creates a module called **Gate**, writes its state to an output file every 5 iterations, clamps its activity to the set activation, and intializes the structure's units to 1.

This gating mechanism was connected to the FS module in PFC, by adding a weights file *gateinfs.ws* to the weights directory. The corresponding *gateinfs.w* file was generated by **netgen**, and the following input was added to the file *ausimweightlist.txt*:

```
#include weights/gateinfs.w
```

Similar entries were created for the gating signals of the other two elements of the buffer.

A.2 Input files

auseq.s	Contains all the necessary information to execute an experimental trial
ausimweightlist.txt	Description of connections among regions
auseq.rs	Timeline of events in an exp. trial
noinp_au.inp	Useful for a no input stream
pethiattn.s	Something related to attention
auseq$< n >$**.inp**	Contains the time-varying input for a sound sequence
resetall_au.inp	Useful to reset all the nodes

A.3 Output files

au*.out	General output of the simulation
debug.txt	Usually empty
mgns.out	MGN units
ea1u.out	Ai up-selective units
ea1d.out	Ai down-selective units
ea2u.out	Aii up-selective units
ea2c.out	Aii contour-selective units
ea2d.out	Aii down-selective units
estg.out	STG units
efd1.out	D1 units
efd2.out	D2 units
exfr.out	R units
exfs.out	C units
spec_pet.m	MatLab specification of PET

A.4 Connection weights within MGN

None

A.5 Connection weights from MGN to Ai

mgnsea1u.ws MGN $\rightarrow$ excitatory Ai up-selective

mgnsea1d.ws MGN → excitatory Ai down-selective

A.6 Connection weights within Ai

ea1uea1u.ws Excitatory Ai up-selective → excitatory Ai up-selective

ia1uea1u.ws Inhibitory Ai up-selective → excitatory Ai up-selective

ea1uia1u.ws Excitatory Ai up-selective → inhibitory Ai up-selective

ea1dea1d.ws Excitatory Ai down-selective → excitatory Ai down-selective

ea1dia1d.ws Excitatory Ai down-selective → inhibitory Ai down-selective

ia1dea1d.ws Inhibitory Ai down-selective → excitatory Ai down-selective

A.7 Connection weights from Ai to Aii

ea1uea2u.ws Excitatory Ai up-selective → excitatory Aii up-selective

ea1uea2c.ws Excitatory Ai up-selective → excitatory Aii contour-selective

ea1dea2c.ws Excitatory Ai down-selective → excitatory Aii contour-selective

ea1dea2d.ws Excitatory Ai down-selective → excitatory Aii down-selective

A.8 Connection weights within Aii

ea2uea2u.ws Excitatory Aii up-selective → excitatory Aii up-selective

ea2uia2u.ws Excitatory Aii up-selective → inhibitory Aii up-selective

ia2uea2u.ws Inhibitory Aii up-selective → excitatory Aii up-selective

ea2cea2c.ws Excitatory Aii contour-selective → excitatory Aii contour-selective

ea2cia2c.ws Excitatory Aii contour-selective → inhibitory Aii contour-selective

ia2cea2c.ws Inhibitory Aii contour-selective → excitatory Aii contour-selective

ea2dea2d.ws Excitatory Aii down-selective → excitatory Aii down-selective

ea2dia2d.ws Excitatory Aii down-selective → inhibitory Aii down-selective

ia2dea2d.ws Inhibitory Aii down-selective → excitatory Aii down-selective

A.9 Connection weights from Aii to Ai

The following feedback has been deprecated, and is not currently used in the simulations.

ea2uea1u.ws Excitatory Aii up-selective → excitatory Ai up-selective

ea2dea1d.ws Excitatory Aii down-selective → excitatory Aii down-selective

Note that there is not feedback from Aii contour-selective units to Ai.

A.10 Connection weights from Aii to STG

ea2uestg.ws Excitatory Aii up-selective → excitatory STG

ea2cestg.ws Excitatory Aii contour-selective → excitatory STG

ea2destg.ws Excitatory Aii down-selective → excitatory STG

A.11 Connection weights within STG

estgestg.ws Excitatory STG → excitatory STG

estgistg.ws Excitatory STG → inhibitory STG

istgestg.ws Inhibitory STG → excitatory STG

A.12 Connection weights from STG to Aii

estgea2u.ws Excitatory STG → excitatory Aii up-selective

estgea2c.ws Excitatory STG → excitatory Aii contour-selective

estgea2d.ws Excitatory STG → excitatory down-selective

A.13 Connection weights from STG to PFC

estgexfs.ws Excitatory STG → Excitatory cue-selective (first in buffer).

estgexfs_a.ws Excitatory STG → Excitatory cue-selective (second in buffer).

estgexfs_b.ws Excitatory STG → Excitatory cue-selective (third in buffer).

A.14 Connection weights within PFC

A.14.1 Connection weights between FS and D1

exfsifd1.ws Excitatory cue-selective → inhibitory delay-selective (1st in buffer)

efd1infs.ws Excitatory delay-selective → inhibitory cue-selective (1st in buffer)

exfs_aifd1_a.ws Excitatory cue-selective → inhibitory delay-selective (2nd in buffer)

efd1_ainfs_a.ws Excitatory delay-selective → inhibitory cue-selective (2nd in buffer)

exfs_bifd1_b.ws Excitatory cue-selective → inhibitory delay-selective (3rd in buffer)

efd1_binfs_b.ws Excitatory delay-selective → inhibitory cue-selective (3rd in buffer)

A.14.2 Connection weights between FS and D2

exfsefd2.ws Excitatory cue-selective → excitatory cue-and-delay-selective (1st in buffer)

exfs_aefd2_a.ws Excitatory cue-selective → excitatory cue-and-delay-selective (2nd in buffer)

exfs_befd2_b.ws Excitatory cue-selective → excitatory cue-and-delay-selective (3rd in buffer)

A.14.3 Connection weights between FS and R

exfsexfr.ws Excitatory cue-selective → excitatory response (1st in buffer)

exfs_aexfr_a.ws Excitatory cue-selective → excitatory response (2nd in buffer)

exfs_bexfr_b.ws Excitatory cue-selective → excitatory response (3rd in buffer)

A.14.4 Connection weights within FS

exfsexfs.ws Excitatory cue-selective → excitatory cue-selective (1st in buffer)

exfsinfs.ws Excitatory cue-selective → inhibitory cue-selective(1st in buffer)

infsexfs.ws Inhibitory cue-selective → excitatory cue-selective(1st in buffer)

exfs_aexfs_a.ws Excitatory cue-selective → excitatory cue-selective (2nd in buffer)

exfs_ainfs_a.ws Excitatory cue-selective → inhibitory cue-selective (2nd in buffer)

infs_aexfs_a.ws Inhibitory cue-selective → excitatory cue-selective (2nd in buffer)

exfs_bexfs_b.ws Excitatory cue-selective → excitatory cue-selective (3rd in buffer)

exfs_binfs_b.ws Excitatory cue-selective → inhibitory cue-selective (3rd in buffer)

infs_bexfs_b.ws Inhibitory cue-selective → excitatory cue-selective (3rd in buffer)

A.14.5 Connection weights between D1 and D2

efd1efd2.ws Excitatory delay-selective → excitatory cue-and-delay-selective (1st in buffer)

efd2efd1.ws Excitatory cue-and-delay-selective → excitatory delay-selective (1st in buffer)

efd1_aefd2_a.ws Excitatory delay-selective → excitatory cue-and-delay-selective (2nd in buffer)

efd2_aefd1_a.ws Excitatory cue-and-delay-selective → excitatory delay-selective (2nd in buffer)

efd1_befd2_b.ws Excitatory delay-selective → excitatory cue-and-delay-selective (3rd in buffer)

efd2_befd1_b.ws Excitatory cue-and-delay-selective → excitatory delay-selective (3rd in buffer)

A.14.6 Connection weights between D1 and R

efd1exfr.ws Excitatory delay-selective → excitatory response

exfrifd1.ws Excitatory response → inhibitory delay-selective

efd1_aexfr_a.ws Excitatory delay-selective → excitatory response

exfr_aifd1_a.ws Excitatory response → inhibitory delay-selective

efd1_bexfr_b.ws Excitatory delay-selective → excitatory response

exfr_bifd1_b.ws Excitatory response → inhibitory delay-selective

A.14.7 Connection weights within D1

efd1efd1.ws Excitatory delay-selective → excitatory delay-selective

efd1ifd1.ws Excitatory delay-selective → inhibitory delay-selective

ifd1efd1.ws Inhibitory delay-selective → excitatory delay-selective

efd1_aefd1_a.ws Excitatory delay-selective → excitatory delay-selective

efd1_aifd1_a.ws Excitatory delay-selective → inhibitory delay-selective

ifd1_aefd1_a.ws Inhibitory delay-selective → excitatory delay-selective

efd1_befd1_b.ws Excitatory delay-selective → excitatory delay-selective

efd1_bifd1_b.ws Excitatory delay-selective → inhibitory delay-selective

ifd1_befd1_b.ws Inhibitory delay-selective → excitatory delay-selective

A.14.8 Connection weights between D2 and R

exfrifd2.ws Excitatory response → inhibitory cue-and-delay-selective

exfr_aifd2_a.ws Excitatory response → inhibitory cue-and-delay-selective

exfr_bifd2_b.ws Excitatory response → inhibitory cue-and-delay-selective

A.14.9 Connection weights within D2

efd2efd2.ws Excitatory cue-and-delay-selective → excitatory cue-and-delay-selective

efd2ifd2.ws Excitatory cue-and-delay-selective → inhibitory cue-and-delay-selective

ifd2efd2.ws Inhibitory cue-and-delay-selective → excitatory cue-and-delay-selective

efd2_aefd2_a.ws Excitatory cue-and-delay-selective → excitatory cue-and-delay-selective

efd2_aifd2_a.ws Excitatory cue-and-delay-selective → inhibitory cue-and-delay-selective

ifd2_aefd2_a.ws Inhibitory cue-and-delay-selective → excitatory cue-and-delay-selective

efd2_befd2_b.ws Excitatory cue-and-delay-selective → excitatory cue-and-delay-selective

efd2_bifd2_b.ws Excitatory cue-and-delay-selective → inhibitory cue-and-delay-selective

ifd2_befd2_b.ws Inhibitory cue-and-delay-selective → excitatory cue-and-delay-selective

A.14.10 Connection weights within R

exfrexfr.ws Excitatory response → excitatory response

exfrinfr.ws Excitatory response → inhibitory response

infrexfr.ws Inhibitory response → excitatory response

exfr_aexfr_a.ws Excitatory response → excitatory response

exfr_ainfr_a.ws Excitatory response → inhibitory response

infr_aexfr_a.ws Inhibitory response → excitatory response

exfr_bexfr_b.ws Excitatory response → excitatory response

exfr_binfr_b.ws Excitatory response → inhibitory response

infr_bexfr_b.ws Inhibitory response → excitatory response

A.15 Connection weights from PFC to STG

efd1istg.ws Excitatory delay-selective → inhibitory STG

efd2estg.ws Excitatory cue-and-delay-selective → excitatory STG

efd1_aistg.ws Excitatory delay-selective → inhibitory STG

efd2_aestg.ws Excitatory cue-and-delay-selective → excitatory STG

efd1_bistg.ws Excitatory delay-selective → inhibitory STG

efd2_bestg.ws Excitatory cue-and-delay-selective → excitatory STG

A.16 Connection weights from PFC to Aii

efd2ea2u.ws Excitatory cue-and-delay-selective → excitatory Aii up-selective

efd2ea2c.ws Excitatory cue-and-delay-selective → excitatory Aii contour-selective

efd2ea2d.ws Excitatory cue-and-delay-selective → excitatory Aii down-selective

efd2_aea2u.ws Excitatory cue-and-delay-selective → excitatory Aii up-selective

efd2_aea2c.ws Excitatory cue-and-delay-selective → excitatory Aii contour-selective

efd2_aea2d.ws Excitatory cue-and-delay-selective → excitatory Aii down-selective

efd2_bea2u.ws Excitatory cue-and-delay-selective → excitatory Aii up-selective

efd2_bea2c.ws Excitatory cue-and-delay-selective → excitatory Aii contour-selective

efd2_bea2d.ws Excitatory cue-and-delay-selective → excitatory Aii down-selective

A.17 Connection weights from Attention module to PFC

attsefd2.ws Attention → excitatory cue-and-delay-selective

attsefd2_a.ws Attention → excitatory cue-and-delay-selective

attsefd2_b.ws Attention → excitatory cue-and-delay-selective

attvatts.ws Attention → attention

A.18 Connection weights in control region

This region is independent of all the other regions, and is useful as a control area in order to normalize the activity in the network.

ectlectl.ws Excitatory control → excitatory control

ectlictl.ws Excitatory control → inhibitory control

ictlectl.ws Inhibitory control → excitatory control

ictlictl.ws Inhibitory control → inhibitory control