

Bernstein Tutorial @ CNS*09

Large-Scale Neuronal Network Models

Principles and Practice

Hans Ekkehard Plesser

Norwegian University of Life Sciences
Simula Research Laboratory

UPDATED VERSION 12 AUGUST 2009



Important update information

- ▶ NEST versions 1.9.r8375 and later include the Topology module in a more natural way in PyNEST.
- ▶ The topology module is now initialised by

```
import nest
import nest.topology as topo
topo.CreateLayer(...)
```

- ▶ The following functions are imported from `nest.topography` and thus must be prefixed with `topo` (or the alias you choose): `CreateLayer`, `ConnectLayer`, `LayerGidPositionMap`, `GetRelativeDistance`, `GetPosition`, `GetLayer`, `GetElement`, `PrintLayerConnections`
- ▶ Several bugs are fixed in the `ht.py` implementation of the simplified Hill-Tononi model. They mostly were caused by not copying dictionaries correctly.





Outline

Resources

Download material

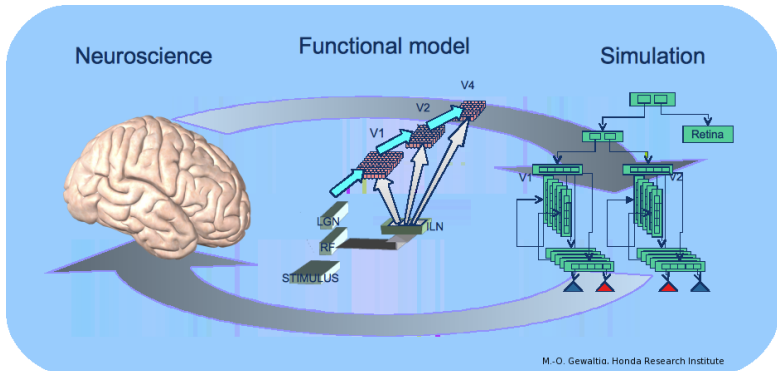
- ▶ Available from `http://www.nest-initiative.org/index.php/Software:Documentation`
- ▶ `CNS09_Tutorial_Updated.pdf`: **these slides**
- ▶ `CNS09_Tutorial_Updated_Material.tbz2`: **archive containing**
 - ▶ `scripts`: **scripts discussed here**
 - ▶ `htmodel`: **simplified Hill-Tononi model**
 - ▶ `ht.py` **Python script with embedded comments**
 - ▶ `ht.pdf` **Result from `pyreport -l ht.py`**





Principles

Computational neuroscience



The goal of neural modeling is to relate, in nervous systems, function to structure on the basis of operation.

— MacGregor & Lewis (1977)

What makes science science?

Refutable hypotheses

Hypotheses must be stated with sufficient detail and precision so that one can devise meaningful tests or counterexamples.

Reproducible experiments

Experiments must be described and performed so carefully, that others can *reproduce* them. Genuine failure to reproduce results invalidates original findings.

Accumulation of knowledge

Accumulation of knowledge through exchange, evolution and (sometimes) revolution of ideas.



Computational science?

Donoho et al (2009)

The vast body of results being generated by current computational science practice suffer a large and growing credibility gap: **it is impossible to verify most of the computational results shown in conferences and papers.**

... **[C]urrent computational science practice does not generate routinely verifiable knowledge.**

... Almost no time is devoted to explaining to the audience why one should believe that errors have been found and eliminated.

The core of the presentation is not about the struggle to root out error—as it would be in mature fields—it is rather a **sales pitch[.]**

... How dare we imagine that computational science, as routinely practiced, is reliable! Many researchers using scientific computing **are not even trying** to follow a systematic, rigorous discipline that would in principle allow others to verify the claims they make.

Computing in Science & Engineering 11:8–18 (2009), doi: 10.1109/MCSE.2009.15



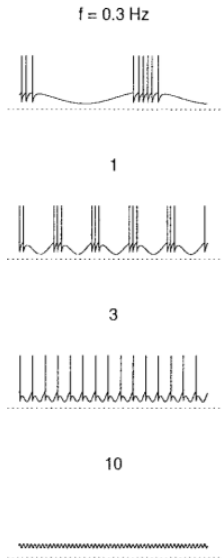


And computational neuroscience?

- ▶ Many researchers can tell you about modeling papers they could not reproduce.
- ▶ Practically no systematic comparison of modeling papers.
- ▶ Few disagree that “computational neuroscience simulation papers are generally not reproducible”.
- ▶ No foul play, “just” sloppiness.
- ▶ Let us look at some examples . . .

Example 1

- ▶ Single neuron model
- ▶ Generally well presented
- ▶ Paper-and-pencil analysis shows that row “3” should have no spikes
- ▶ Could not be resolved in collaboration with author
- ▶ Probably figure mix-up
- ▶ No qualitative consequences



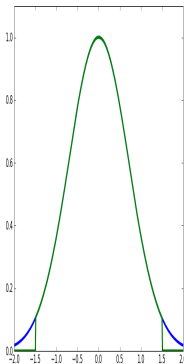
Example 2

- ▶ Well-known integrate-and-fire network model
- ▶ Chosen as benchmark for simulator comparison
- ▶ Author of paper unable to reproduce figures from his own paper with his own simulator
- ▶ Differences probably due to “minor” changes in simulator code
- ▶ Never resolved
- ▶ No qualitative consequences



Example 3

- ▶ Well-known paper on plasticity
- ▶ Neuronal connections based on Gaussian profile
- ▶ Reproduction failed qualitatively
- ▶ Inspection of original C-code revealed Gaussian with cut-off
- ▶ Were original authors aware of role of cut-off?

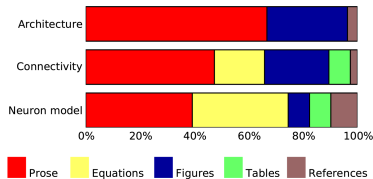
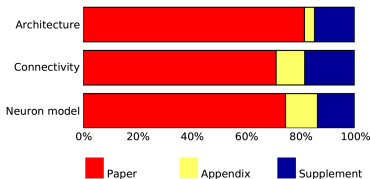


Network descriptions today

- ▶ Each author uses their own style
- ▶ Descriptions largely in prose
- ▶ Imprecise, difficult to check
- ▶ Are description and code in agreement?

If code and comment disagree, probably both are wrong.

— Donald E. Knuth

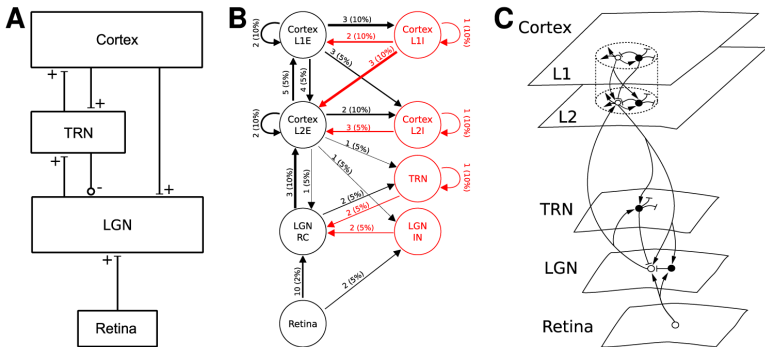


Nordlie, Gewaltig, Plesser (2009)



Network diagrams

- ▶ Each author uses different style
- ▶ Not even agreement on symbols for elementary aspects such as excitatory and inhibitory connections.



Good description practice

- ▶ Clear rules for describing models
- ▶ Templates for tables for model descriptions
- ▶ Top-down description, starting with basic “Nutrition Facts” box
- ▶ Nordlie, Gewaltig, Plesser.

Towards reproducible descriptions of neuronal network models.

PLoS Computational Biology
 5(8):e1000456. <http://doi.dx.org/10.1371/journal.pcbi.1000456>.

A		Model Summary
Populations	Three: excitatory, inhibitory, external input	
Topology	—	
Connectivity	Random convergent connections	
Neuron model	Leaky integrate-and-fire, fixed voltage threshold, fixed absolute refractory time (voltage clamp)	
Channel models	—	
Synapse model	δ -current inputs (discontinuous voltage jumps)	
Plasticity	—	
Input	Independent fixed-rate Poisson spike trains to all neurons	
Measurements	Spike activity	

B		Populations
Name	Elements	Size
E	laf neuron	$N_E = 4N$
I	laf neuron	N_I
Ext	Poisson generator	$C_E(N_E + N_I)$

C				Connectivity
Name	Source	Target	Pattern	
EE	E	E	Random convergent $C_E \rightarrow 1$, weight J , delay D	
IE	E	I	Random convergent $C_C \rightarrow 1$, weight J , delay D	
EI	I	E	Random convergent $C_I \rightarrow 1$, weight $-gJ$, delay D	
E	I	I	Random convergent $C_C \rightarrow 1$, weight $-gJ$, delay D	
Ext	Ext	E ∪ I	Non-overlapping $C_E \rightarrow 1$, weight J , delay D	

D		Neuron and Synapse Model
Name	laf neuron	
Type	Leaky integrate and fire, δ -current input	
Subthreshold dynamics	$\tau \dot{V}(t) = -V(t) + RI(t) \quad \text{if } t > t^* + \tau_p$ $V(t) = V_C \quad \text{else}$	
Spiking	$R[V(t) - C] \geq \theta \Rightarrow V(t+) \geq \theta$ <ol style="list-style-type: none"> set $t^* = t$ emit spike with time-stamp t^* 	

E		Input
Type	Description	
Poisson generators	Fixed rate λ_{Ext} , C_C generators per neuron, each generator projects to one neuron	

F		Measurements
Spike activity as raster plots, rates and “global frequencies”, no details given		



Literate programming

- ▶ Keep code and comment in one file
- ▶ Execute, create documentation, or mix both
- ▶ Tool: [pyreport](http://pyreport.gael-varoquaux.info/computers/pyreport)
(gael-varoquaux.info/computers/pyreport)

Pyreport example

```
#! This is a Pyreport example  
#! _____  
#! ::author:: Hans Ekkehard Plesser  
import pylab  
  
#! define x-axis data  
x = pylab.arange(-5.0, 5.0, 0.1)  
  
#! Plot the function  
#$ \begin{equation*}y = x * \sin(x)\end{equation*}  
pylab.plot(x, x * pylab.sin(x), 'r-')  
pylab.show()
```



Pyreport: example

Run example as `pyreport -l pyreport_ex.py`

`/Users/plesser/Talks/CNS09_Tutorial/figs/pyreport_ex.py`

July 14, 2009

1

This is a Pyreport example

Author: Hans Ekkehard Plesser

```
6 import pylab
```

Define x-axis data

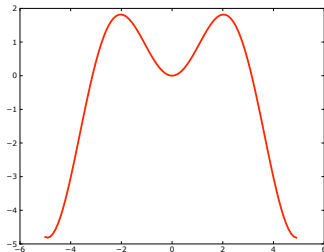
```
9 x = pylab.arange(-5.0, 5.0, 0.1)
```

Plot the function

$$y = x \sin(x)$$

```
12 pylab.plot(x, x * pylab.sin(x), 'r-', linewidth=3)
```

```
13 pylab.show()
```



Version control

Where did those data come from ... ?

How to know, a year post publication, which simulator version and which scripts you used to generate and plot your data?

- ▶ Version control helps you keep track of your files
- ▶ Allows reconstruction of files to past state
- ▶ Aid collaboration with co-authors
- ▶ Also useful for manuscripts
- ▶ Wide range of software: SVN, Mercurial, Bazaar, Git,
...



Version control: Example

```
# All svn interaction should be done via Python's svn module
import subprocess, os, sys

datadir = "/Users/Plessner/Papers/GamblersRuin/data/rw_data"
codedir = "/Users/Plessner/Papers/GamblersRuin/code"
prog = os.path.join(codedir, "grsim")

# — snip —

# get script name
script = os.path.join(sys.path[0], os.path.split(sys.argv[0])[-1])

# open log file name
logfile = open(os.path.join(datadir, "log.txt"), 'w')

logfile.write("Log_file_for_%s\n\n" % script)

# record script and code status
logfile.write("SCRIPT_STATUS\n-----\n")
logfile.write(subprocess.Popen(["svn", "status", "-v", script],
                                stdout=subprocess.PIPE).communicate(None)[0])

logfile.write("\n\n")
logfile.write("CODE_STATUS\n-----\n")
logfile.write(subprocess.Popen(["svn", "status", "-v", codedir],
                                stdout=subprocess.PIPE).communicate(None)[0])

logfile.write("\n\n")

# — snip —

# Simulation -----
subprocess.Popen([prog, "SRRS", tp, datadir, str(N), str(K), str(seed)]).wait()

# Check in results
```



Resulting log file

Log file for /Users/plesser/Papers/GamblersRuin/scripts/rw_data.py

SCRIPT STATUS

```
-----  
          18          18 plesser      /Users/plesser/Papers/GamblersRuin/scripts/rw
```

CODE STATUS

```
-----  
          8          8 plesser      /Users/Plesser/Papers/GamblersRuin/code  
          8          7 plesser      /Users/Plesser/Papers/GamblersRuin/code/mt199  
          8          7 plesser      /Users/Plesser/Papers/GamblersRuin/code/mt199  
          8          7 plesser      /Users/Plesser/Papers/GamblersRuin/code/mt199  
          8          7 plesser      /Users/Plesser/Papers/GamblersRuin/code/mt199
```

- snip -

DATA LOGIN

```
-----  
A          /Users/Plesser/Papers/GamblersRuin/data/rw_data/log.txt  
A          /Users/Plesser/Papers/GamblersRuin/data/rw_data/SRRS  
A          /Users/Plesser/Papers/GamblersRuin/data/rw_data/SRRS/A2  
A          /Users/Plesser/Papers/GamblersRuin/data/rw_data/SRRS/A2/sc_l1000_r2000_s100  
A          /Users/Plesser/Papers/GamblersRuin/data/rw_data/SRRS/A2/sc_l1000_r2000_s100  
A          /Users/Plesser/Papers/GamblersRuin/data/rw_data/SRRS/A2/sc_l1000_r2000_s200  
Adding      Plesser/Papers/GamblersRuin/data/rw_data/log.txt
```

- snip -

Transmitting file data
Committed revision 19.

DATA STATUS

```
-----  
          9          9 plesser      /Users/Plesser/Papers/GamblersRuin/data/rw_da  
M          19          19 plesser      /Users/Plesser/Papers/GamblersRuin/data/rw_da  
          19          19 plesser      /Users/Plesser/Papers/GamblersRuin/data/rw_da
```





Break



NEST Topology Module 1



Prerequisites

- ▶ Some experience with NEST / PyNEST
- ▶ PyNEST installed on your computer

PyNEST: A simple example

```
import nest
import nest.voltage_trace
nest.ResetKernel()

neuron=nest.Create("iaf_neuron")
noise =nest.Create("poisson_generator",2)
sine =nest.Create("ac_generator")
voltmeter= nest.Create("voltmeter")

nest.SetStatus(noise, ["rate":75000.0, "rate":20000.0])
nest.SetStatus(sine, ["amplitude":100.0, "frequency":2.0])

nest.SetStatus(voltmeter, ["withgid": True, "withtime": True])

nest.ConvergentConnect(noise,neuron,weight=[1.,-1.],delay=1.0)
nest.Connect(voltmeter,neuron)
nest.Connect(sine,neuron)

nest.Simulate(1000.0)
nest.voltage_trace.from_device(voltmeter)
```



About PyNEST

- ▶ Python wrapper around NEST
- ▶ Translates Python commands to NEST's native scripting language SLI
- ▶ For more information, see
 - ▶ Eppler et al.
PyNEST: a convenient interface to the NEST simulator.
Frontiers in Neuroinformatics (2009).
Doi 10.3389/neuro.11.012.2008
 - ▶ Workshop 12: Python in Neuroscience



Overview

- ▶ Idea: High-level support for layered networks
- ▶ Implementation: Kittel Austvoll
- ▶ Describe network as collection of layers
- ▶ Elements of a layer can be
 - ▶ Individual neurons
 - ▶ Groups of neurons (e.g. Microcolumn)
 - ▶ Placed on a fixed grid
 - ▶ Placed arbitrarily in space
- ▶ Connections described by
 - ▶ Masks: no connections outside mask
 - ▶ Kernels: distance-dependent connection probability



Getting started

Documentation

- ▶ Topology User Manual: usually in `/usr/local/share/doc/nest`
- ▶ Online help: usually available through `file:///usr/local/share/doc/nest/index.html`
- ▶ See [SLI command index by name](#), search for `topology::`
- ▶ Installation location will differ from `/usr/local` if you configured with `--prefix`

Activating Topology (MODIFIED!)

```
import nest
import nest.topology as topo
nest.ResetKernel()
```



Populations

- ▶ Called **Layers**
 - ▶ two-dimensional
 - ▶ internally: NEST `subnet` with information about spatial structure
- ⇒ Neurons do not know where they are, only the layer knows.
- ▶ Properties
 - columns, rows** x-, y-size of grid-based layer
 - positions** neuron locations in non-grid layer
 - extent** size of layer in e.g., visual space
 - center** location of layer midpoint in, e.g., visual space
 - elements** neurons or subnets at each layer position
 - edge_wrap** whether to apply periodic boundary conditions (True/False)



Simple population on grid

Create layer

```
l1 = topo.CreateLayer({'columns': 4, 'rows': 3,  
                      'extent': [1.5, 1.0],  
                      'elements': 'iaf_neuron'})
```

Inspect layer

```
nest.PrintNetwork()  
+-[0] root dim=[1 12]  
  |  
  +-[1] layer dim=[12]
```

```
nest.PrintNetwork(2)  
+-[0] root dim=[1 12]  
  |  
  +-[1] layer dim=[12]  
    |  
    +-[1]...[12] iaf_neuron
```

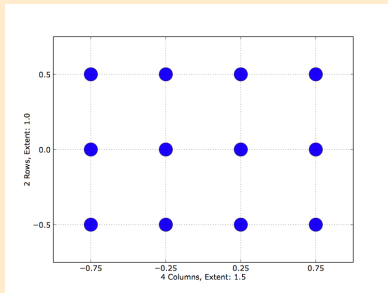


Visualize network

A minimal visualization script

```
# extract position information
npos = zip(*[topo.GetPosition([n])
             for n in nest.GetLeaves(l1)[0]])

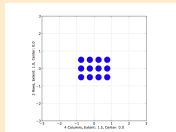
# plot
pylab.plot(npos[0], npos[1], 'o', markersize=20)
```



Extent and Center

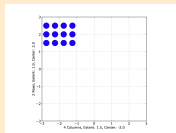
Centered

```
topo.CreateLayer(  
    {'columns': 4, 'rows': 3,  
     'extent': [1.5, 1.0],  
     'center': [0.0, 0.0],  
     'elements': 'iaf_neuron'})
```



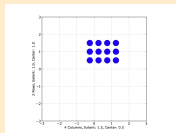
Top-left

```
topo.CreateLayer(  
    {'columns': 4, 'rows': 3,  
     'extent': [1.5, 1.0],  
     'center': [-2.0, 2.0],  
     'elements': 'iaf_neuron'})
```



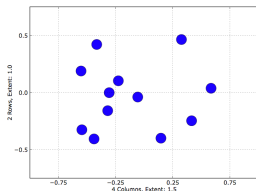
Top-right

```
topo.CreateLayer(  
    {'columns': 4, 'rows': 3,  
     'extent': [1.5, 1.0],  
     'center': [0.5, 1.0],  
     'elements': 'iaf_neuron'})
```



Freely placed nodes

- ▶ Grids can cause artifacts
- ▶ Can place nodes anywhere in space
- ▶ Must supply positions as (x,y)-pairs
- ▶ positions must be inside extent
- ▶ center meaningless



```
pos = [ [random.uniform(-0.75,0.75),  
        random.uniform(-0.5,0.5) ]  
        for j in range(12) ]
```

```
topo.CreateLayer('extent': [1.5, 1.0],  
                'positions': pos,  
                'elements': 'iaf_neuron')
```

Composite layer elements

- ▶ Layer elements can be composed of several neurons
- ▶ Tip: use `CopyModel()` to create specific model types even if models are identical

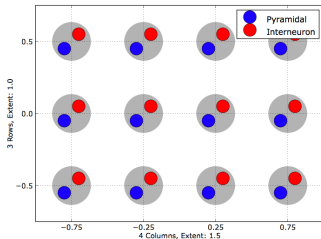
```
nest.CopyModel('iaf_neuron', 'pyr')
nest.CopyModel('iaf_neuron', 'in')
ctx = topo.CreateLayer({'columns': 4, 'rows': 3,
                       'extent': [1.5, 1.0],
                       'elements': ['pyr', 'in']})
```

```
nest.PrintNetwork(0, ctx)    nest.PrintNetwork(2, ctx)
+--[1] layer dim=[12 2]      +--[1] layer dim=[12 2]
                             |
                             +--[1] subnet dim=[2]
                             | |
                             | +--[1] pyr
                             | +--[2] in
                             |
                             +--[2] subnet dim=[2]
                             | |
                             | +--[1] pyr
                             | +--[2] in
```



Visualizing composite layer

- ▶ More challenging
- ▶ Solution below
 suboptimal: will overlook
 non pyr/in nodes



```
# extract position information
ppyr = pylab.array(zip(*[topo.GetPosition([n]) for n in nest.GetLeaves(ctx)[0]
                        if nest.GetStatus([n], 'model')[0]=='pyr']))
pin = pylab.array(zip(*[topo.GetPosition([n]) for n in nest.GetLeaves(ctx)[0]
                        if nest.GetStatus([n], 'model')[0]=='in']))

# plot
pylab.plot(ppyr[0]-0.05, ppyr[1]-0.05, 'bo', markersize=20,
           label='Pyramidal', zorder=2)
pylab.plot(pin [0]+0.05, pin [1]+0.05, 'ro', markersize=20,
           label='Interneuron', zorder=2)

pylab.plot(ppyr[0], ppyr[1], 'o', markerfacecolor=(0.7,0.7,0.7),
           markersize=60, markeredgewidth=0, zorder=1, label='_nolegend_')
```



Concepts: Masks & Kernels

- ▶ Connect layers to layers: from source to target
 - Convergent:** for each neuron in target, choose sources.
 - Divergent:** for each neuron in source, choose targets.
- ▶ Following discussion assumes divergent connections
- ▶ Mask: choose targets only from inside the mask
- ▶ Kernel: probability to create a connection
- ▶ Mask and Kernel are distance-dependent functions
- ▶ Weights and delays can be randomized



Example

```
a = topo.CreateLayer({'columns': 31, 'rows': 31,  
                    'extent': [3.0, 3.0],  
                    'elements': 'iaf_neuron'})  
b = topo.CreateLayer({'columns': 31, 'rows': 31,  
                    'extent': [3.0, 3.0],  
                    'elements': 'iaf_neuron'})  
topo.ConnectLayer(a, b,  
                 {'connection_type': 'divergent',  
                 'mask': {'circular':  
                         {'radius': 0.5}},  
                 'kernel': 0.5,  
                 'weights': {'uniform':  
                             {'min': 0.5, 'max': 2.0}},  
                 'delays': 1.0})
```



Visualization of connections

```
# plot targets of neurons in different grid locations
for ctr in [[15,15], [0,0]]:

    # plot connections
    ctr_id = topo.GetElement(a, ctr)
    conn = nest.GetConnections(ctr_id, 'static_synapse')
    tpos = zip(*[topo.GetPosition([n])
                 for n in conn[0]['targets']])
    pylab.scatter(tpos[0], tpos[1],
                  30 * pylab.array(conn[0]['weights']), zorder = 1

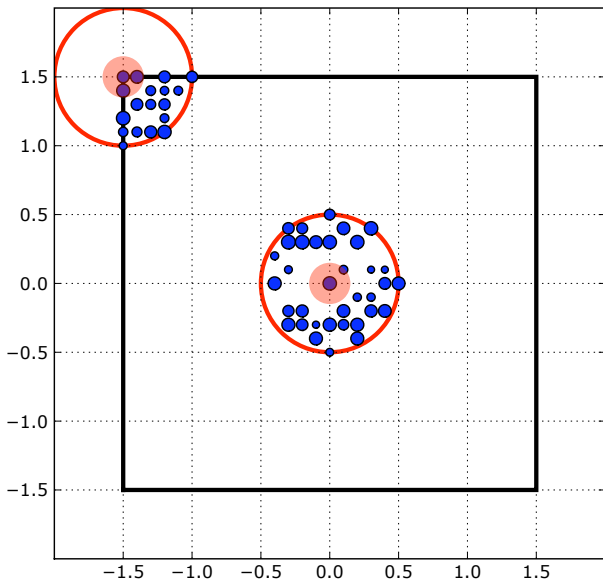
    # mark sender position
    ctrpos = topo.GetPosition(ctr_id)
    pylab.gca().add_patch(pylab.Circle(ctrpos, radius=0.15, zorder=
                                     fc = 'r', alpha = 0.4, ec =

    # mark mask position
    pylab.gca().add_patch(pylab.Circle(ctrpos, radius=0.5, zorder=
                                     fc = 'none', ec = 'r', lw=3

    # mark layer edge
    pylab.gca().add_patch(pylab.Rectangle((-1.5,-1.5), 3.0, 3.0, zorder=
                                     fc = 'none', ec = 'k', lw=
```

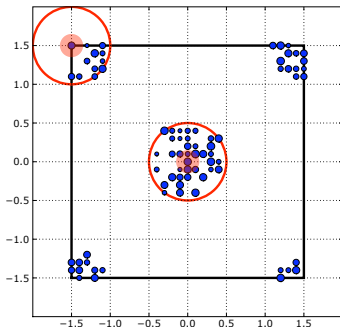


Visualization: result



Effect of 'edge_wrap'

```
b = topo.CreateLayer({'columns': 31, 'rows': 31,  
                    'extent': [3.0, 3.0],  
                    'elements': 'iaf_neuron',  
                    'edge_wrap': True})
```



Mask types

Circular `{'circular':
{'radius': 1.5}}`

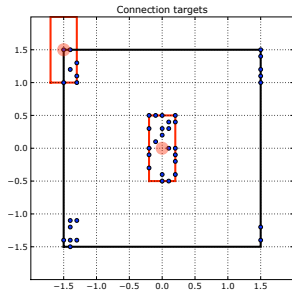
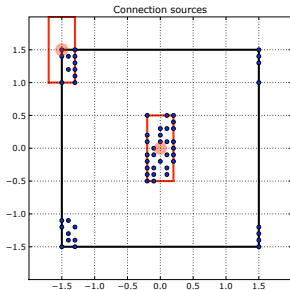
Rectangular `{'rectangular':
{'lower_left': [-0.5, -1.5],
'upper_right': [0.5, 1.5]}}`

Doughnut `{'doughnut':
{'inner_radius': 1.5,
'outer_radius': 2.5}}`



Convergent connections

```
topo.ConnectLayer(a, b, {'connection_type': 'convergent',  
    'mask': {'rectangular': {'lower_left': [-0.2, -0.5], 'upper_right': [0.2, 0.5]}},  
    for ctr in [[15,15], [0,0]]:  
        ctr_id = topo.GetElement(b, ctr)  
        spos=zip(*[topo.GetPosition([n]) for n in nest.GetLeaves(a)[0]  
            if nest.GetConnections([n], 'static_synapse')[0]['targets']  
                .count(ctr_id[0]) > 0])  
  
        pylab.scatter(spos[0], spos[1], 20, zorder = 10)
```



Kernels: Connection probabilities

gaussian $p = c + p_{\text{center}} e^{-(d-\mu)^2/2\sigma^2}$

gaussian2D similar, but not circular

linear $p = ad + c$

exponential $p = c + ae^{-d/\tau}$

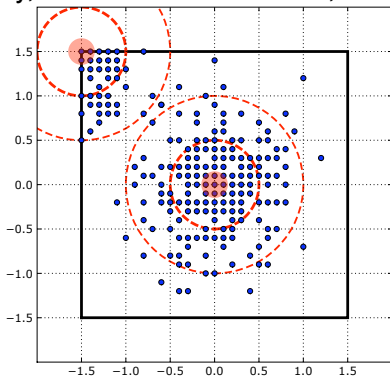
number if kernel is a number, it is used as p



Example

```
topo.ConnectLayer(a, b, {'connection_type': 'divergent',  
  'mask': {'circular': {'radius': 3.0}},  
  'kernel': {'gaussian': {'p_center': 1.0, 'sigma': 0.5}},  
  'weights': 1.0, 'delays': 1.0})
```

Visualize density, dashed circles mark σ , 2σ



Connections & composites

- ▶ If source or target layer has composite nodes, all possible pairs of nodes are considered.
- ▶ Specific nodes can be selected by model

```
a = topo.CreateLayer({'columns': 31, 'rows': 31, 'extent': [3.0,  
                    'elements': ['pyr', 'in']})  
b = topo.CreateLayer({'columns': 31, 'rows': 31, 'extent': [3.0,  
                    'elements': ['pyr', 'in']})
```

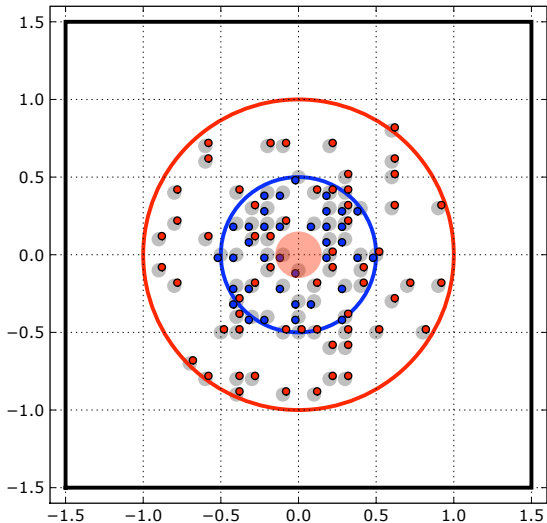
```
topo.ConnectLayer(a, b, {'connection_type': 'divergent',  
                        'sources': {'model': 'pyr'},  
                        'targets': {'model': 'pyr'},  
                        'mask': {'circular': {'radius': 0.5}},  
                        'kernel': 0.5,  
                        'weights': 1.0,  
                        'delays': 1.0})
```

```
topo.ConnectLayer(a, b, {'connection_type': 'divergent',  
                        'sources': {'model': 'pyr'},  
                        'targets': {'model': 'in'},  
                        'mask': {'circular': {'radius': 1.0}},  
                        'kernel': 0.2,  
                        'weights': 1.0,  
                        'delays': 1.0})
```





Resulting pattern





Break



Visual Pathway Model



Hill-Tononi Model

We go through the `ht.py` script as far as we get.



Concluding advice

Using NEST Topology

- ▶ Write to `nest_user@nest-initiative.org` if you encounter problems!
- ▶ Send feature requests/suggestions to `nest_user@nest-initiative.org`!
- ▶ NEST Topology will most likely be improved this fall.
- ▶ Please cite NEST (Gewaltig & Diesmann, Scholarpedia) and the Topology module (Plesser & Austvoll, CNS*09)!
- ▶ Please let us know about your successes!



Other Tools & Initiatives

- ▶ PyNN: Wrapper for NEST, Neuron, ...
 - ▶ similar *Populations* and *Projections* concept as NEST Topology
 - ▶ network construction slow at present
 - ▶ <http://neuralensemble.org/trac/PyNN>
- ▶ neuroConstruct/NetworkML
 - ▶ GUI tool & XML-based description language
 - ▶ NetworkML not supported by NEST at present
 - ▶ network construction slow at present
 - ▶ <http://neuroml.org>
- ▶ Contributions at CNS*09
 - ▶ Crook, Silver, & Gleeson: Describing and exchanging models of neurons and neuronal networks with NeuroML (F1).
 - ▶ Ansorg & Schwabe. Declarative model of code generation for hybrid individual- and population-based simulations of the early visual system (P57).



General advice

- ▶ Use standard simulation tools (Neuron, NEST, Genesis, Moose, Brian, ...)!
- ▶ Check, check, and check again that your network is built as you intended!
- ▶ Use revision control to manage your network definition and simulation scripts!
- ▶ Document your models following a *Good model description practice*, e.g., Nordlie, Gewaltig, Plesser, *Towards Reproducible Descriptions of Neuronal Network Models*, PLoS Computational Biology 5(8):e1000456. <http://doi.doi.org/10.1371/journal.pcbi.1000456>!

