libLISSOM Quick Reference: a quick tutorial

Giacomo Spigler

July 9, 2008

Basic usage tutorial.

1) A **Retina** object has to be created.
This class is used as an input layer for further processing LISSOM layers, and can be used both with real images loaded into it and with synthetic elongated-gaussians patterns.

```
Retina *retina = new Retina(RetinaN, RetinaN);
```

This object can then be used with 3 methods:

```
Retina::setinput(unsigned char *im, int widthstep=0);
Retina::randomGaussian(int centered=0, int number=0, float a2=56.25, float b2=2.25, int x=-1, int y=-1, int angledeg=-1);
Retina::getoutput(unsigned char *im, int widthstep=0);
```

setinput requires a ‘float *‘ image in range [0, 1], passed as ‘unsigned char *‘. ‘widthstep‘ is the pitch.
randomgaussian generates an elongated gaussian with random orientation (used for orientation tuning in primary visual cortex V1 simulations).
getoutput acts inversely to setinput: it retrieves the current retinal activation from the corresponding GPU buffer.

2) Now we need a **LISSOM** object.
Instantiating a **LISSOM** object can be really difficult, because of the high number of parameters needed.
LISSOM also supports 2 operating modes: either scaling area or cortical density.
Scaling area implies using a bigger retina, hence simulating a bigger field of view.
Scaling cortical density keeps a small retina but makes use of a bigger LISSOM

layer, with longer lateral connections, hence simulating a better model of the visual cortex (but being much more computationally expensive).
In the following

LISSOM *layer = new LISSOM(width, height, size of afferent receptive field, retina (for input layer size), 1, scaleAreaOrDensity [0 for area, 1 for density; default is 0]);

The **LISSOM** class features many methods to modify network's parameters (set*, ...), to debug (getoutput, getweight, ...), which won't be described in this short reference.

3) Now we have to connect the two layers:

layer - > ConnectAfferent(retina, 0);

This just connects the two layers.
WARNING: forgetting doing this operation will crash the simulation.

4) Training/Testing phase.
Training can be a bit complicated, because some parameters have to be adjusted after n learning steps to make self-organization occur (the most important thing is that **LISSOM::AdjustWeights()** uses the last simulation data obtained to adjust weights with Hebbian learning like mechanisms).
We will look at the Testing methods.
A simulation is made up by the following steps:

LISSOM::FirstStep();
LISSOM::Step();

Basically, **FirstStep** gets the first neural activation, and **Step** computes something about 10 iterations of self-organizing lateral connection simulations.

For complete examples look into the **example** directory of liblissom_cuda.