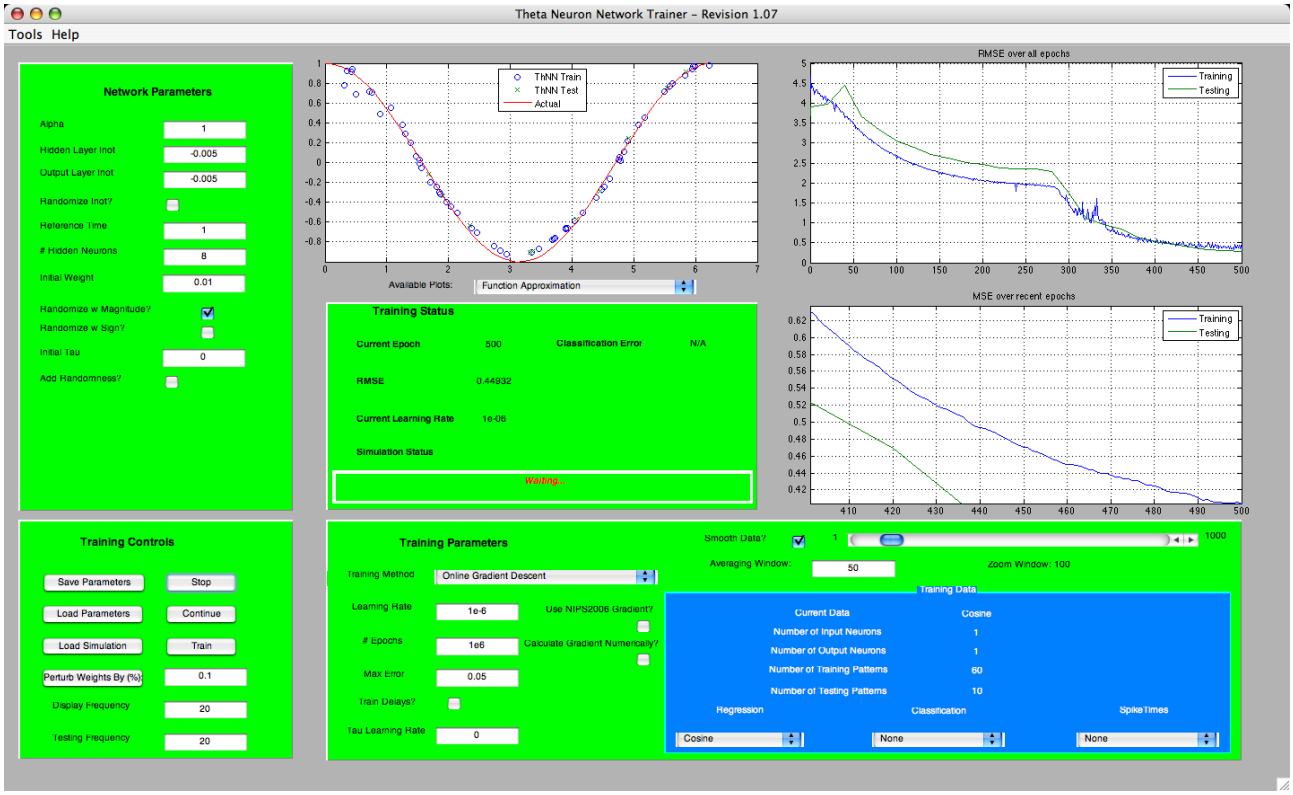
Theta Neuron Network Trainer
Release 1.07

Documentation

The Theta Neuron Network Trainer (ThNNT) is a Matlab-based GUI (graphical user interface) for training and running multi-layer network of theta neurons. It includes a gradient tester tool for gradient calculation verification under different simulation conditions. To run the ThNNT GUI, within Matlab, navigate to the directory where the current version of ThNNT is unzipped to (TNNT_1_07 for example). Next type "start_ThNN" from the Matlab command line. The GUI should start automatically. A screen capture of the main ThNNT screen follows:




**To Generate a Data Set:**

Create an m-file in the Regression, Classification or SpikeTimes folders within the DataSets folder. The information in the m-file will depend on the type of input data. Regression refers generally to a continuous function that the ThNN should attempt to approximate. Classification refers to real-valued input data that is assigned into membership classes. The ThNN is trained to learn and generalize the classes. SpikeTimes refers to giving the network spike time values directly. Generally, the first two types are SISO, and the last can handle single or multiple input and output spikes per synapse. The examples below will clarify what the data generation file should contain. After the the data generation is run, a binary mat file will be created that ThNNT can use directly and which should appear automatically in the Training Data window when ThNNT is next started.

**Regression Example:**

```
%Cosine Data Generation
clear all;

%To make sure input are different each time this is run
```

```
rand('state',sum(100*clock));

EncodeMethod='Linear';
DecodeMethod='Linear';
InputSpikeRange=[2 8];
OutputSpikeRange=[16 28];
InputRange=[0 2*pi];
OutputRange=[-1 1];
Inputs=2*pi*rand(60,1);
Outputs=cos(Inputs);
TestFlag=1;
TestingInputs=2*pi*rand(10,1);
TestingOutputs=cos(TestingInputs);
FH=@cos;

save('Cosine');
```

**Classification Example:**

```
%Fisher Iris Data Generation
% Fisher, R. A. (1936). "The Use of Multiple Measurements in
Axonomic Problems." Annals of Eugenics 7, 179-188.
clear all;

%PW PL  SW  SL  Type
IrisData = [
     2    14    33    50     0
     ...
    24    51    28    58     1
     2    15    37    53     0];

EncodeMethod='Linear';
DecodeMethod='Linear';
InputSpikeRange=[2 8];
OutputSpikeRange=[20 30];
InputRange=[min(min(IrisData(:,1:4)))...
max(max(IrisData(:,1:4)))];
OutputRange=[min(IrisData(:,5)) max(IrisData(:,5))];
Inputs=IrisData(1:100,1:4);
Outputs=IrisData(1:100,5);
TestFlag=1;
TestingInputs=IrisData(101:end,1:4);
TestingOutputs=IrisData(101:end,5);

clear IrisData;
save('Fisher_Iris');
```

**SpikeTimes Example:**

```
%Basic MIMO Data
clear all;

EncodeMethod='Spikes';
DecodeMethod='Spikes';
InputSpikeTimes={[1 2; 1 4] [1 1.5]}
DesiredSpikeTimes={[1 28] [1 25; 1 27]}
TestFlag=0;
```

```
save('BasicMIMO');
```

**How to Run a Training Simulation:**

*Network Parameters*
These parameters determine the type of ThNN to be used in a simulation. They cannot be changed mid-simulation. Generally with MIMO (more than one spike per input and output synapse) data sets, we are using a positive Inot value in the output layer and a negative Inot value in the hidden layer. The randomize Inot checkbox currently indicates that the sign (but not the magnitude) of the hidden layer Inot values will be randomized (i.e. about half the hidden neurons will have a positive Inot and half the hidden neurons will have a negative Inot). This randomization may be helpful in breaking symmetry in the dynamics that slows or prevents training in certain cases. All the default values here should be fine, except perhaps increasing the number of hidden neurons may increase the training precision (and decrease the training speed as a tradeoff).

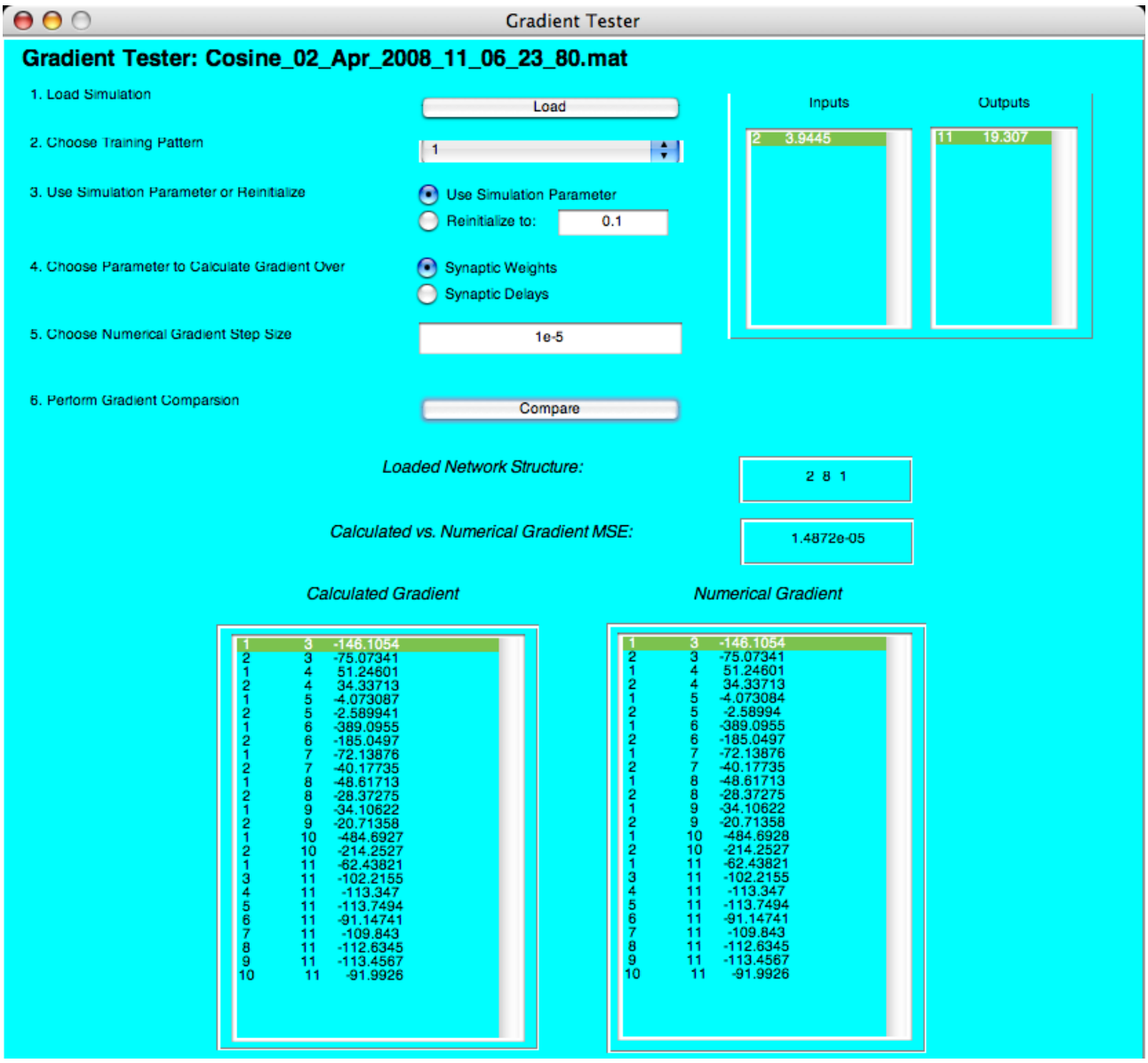
*Training Parameters*
These parameters may be changed during a simulation. Online training will proceed quicker than batch in general, but may also be less stable. The learning rate is the most important parameter. Some data sets prefer a higher or lower value. If the initial learning rate is too high, it may cause all the hidden neurons to not fire, which in effect means the network has gone unstable. The training should be restarted with a lower value of the learning rate.

*Training Status Panel and Axes*
These axes and the panel display current results at an interval determined by Display and Testing frequency. The zoom window sliding bar may be used to change to the zoom scale on the bottom mse plot. Likewise, the Smooth Data checkbox and the Averaging Window edit box allow for the user to smooth the local data to flush out undesirable high frequency oscillations.

*Training Controls*
Display and Testing frequency determine how often testing data is generated, how often results are displayed on the panels and axes, and how often data is saved to an output file. Parameters may be saved and loaded using the training controls. Load simulation allows the results (or partial results) from a previous simulation to be loaded, including the parameters with which the simulation was conducted. Training can then be Continued. The Train button starts a simulation from scratch. The Stop button stops a simulation. The Continue button continues the simulation from where it was last stopped.

**Gradient Tester**

Gradient Tester is a separate GUI that can be accessed through the tools menu of ThNNT. The Gradient Tester GUI lists five simple steps that allow the user to choose previously generated simulation results and generate calculated and numerical gradients for comparison. Gradient Tester has been designed to work with SISO (single spike per input and output synapse) and MIMO data sets. A screen capture of the Gradient Tester Window follows:




**TBD in Future Releases**

- Improve this help file to include in part the equations from upcoming publications once they are submitted
- Add ability to load a network and run sample data through it to see if after the fact, the network has generalized in a particular way
- Likewise, add an animation for displaying network running for complex recursive networks
- Add more menu bar functionality
- Add interactive graphical method for adding datasets
- Reintroduce two-input regression functions
- Add the ability to compare to SpikeProp
- Generate more plots similar to QwikNet ©
- There is possibly a small gradient calculation error when all the weights are the same (I assume that some spike time sorting somewhere has inconsistent behavior)

**Known Bugs**

Please let me know when you find them (email to: Samuel.McKennoch@loria.fr)