MemBrain Examples

Inhaltsverzeichnis

Remarks to the Help File

Remarks to the Help File

Some pictures in this help file have been taken from older versions of MemBrain and thus may look slightly different than they would do with the current one.
Also the coloring of the neurons may be a bit different in some of the pictures compared with the newest version of MemBrain.

The Examples

Included Examples

This help file gives some information on the Examples currently provided with MemBrain.

Please note that this help file does not provide instructions for first steps with MemBrain.

Before to work with the examples you should have worked through the 'Short Beginner's Tutorial' that is included in MemBrain's help file (Press F1 in MemBrain and you will find your way to get there).

Currently there are the following examples available for download on the MemBrain homepage:

1. An example of a time invariant feed forward net using Backpropagation (supervised training). This example demonstrates detection of numbers in a neuron input matrix.

2. An example of a time variant net, also trained with Backpropagation. The example demonstrates time series prediction on the often used benchmark 'Mackey-Glass' chaotic time series. This example actually is built up as some kind of **advanced tutorial**.

3. An example for demonstration of how SOMs (Self Organized Maps) are handled in MemBrain (unsupervised learning).

4. A net that demonstates the dynamic simulation capabilities of MemBrain's neuron and link model (spiking neurons).

Supervised Learning

Supervised Learning

Time invariant feed forward net

Time invariant feed forward

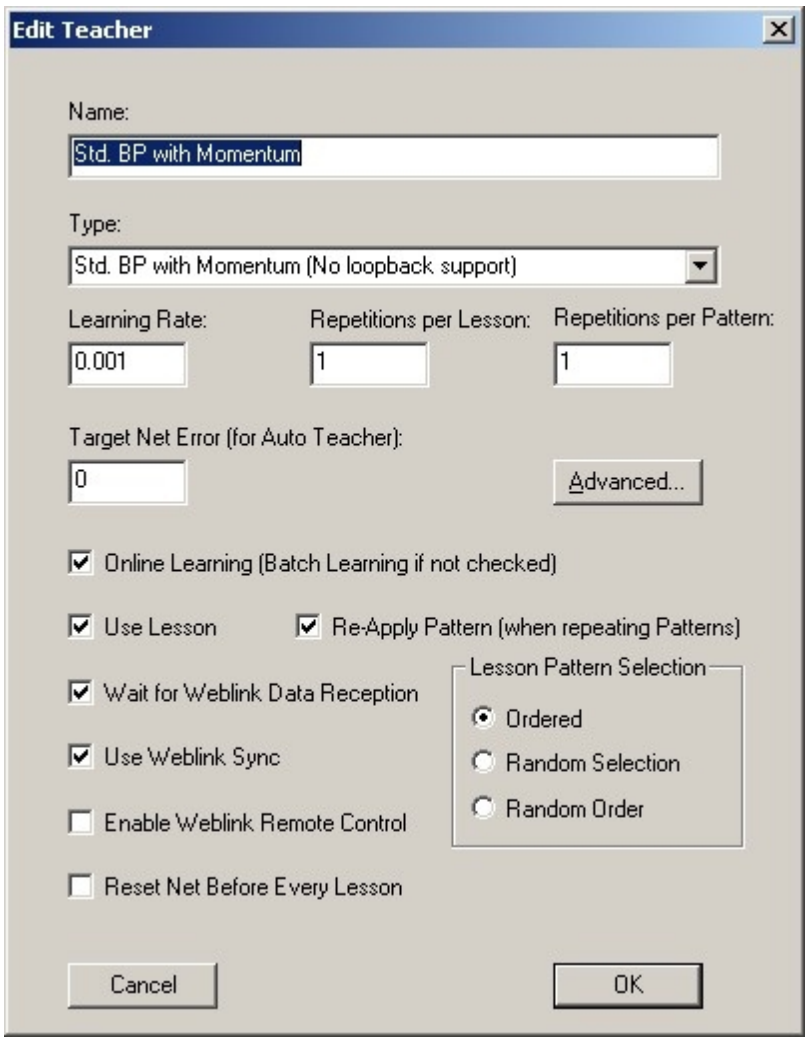
File names:
NumberDetection.mbn and NumberDetection.mbl

The example shows a time invariant Feed Forward net that is intended to detect numbers between 0 and 9 in a matrix of input neurons.
There is also a lesson included in the examples that has the same name as the net. The net is already trained to reproduce this lesson, still it is not really good at interpreting new input patterns. You might want to play around with that net using MemBrain's "Paint Brush Selection" feature and also add more patterns to the Lesson by doing so. If you don't know about the 'Paint Brush Selection' method available in MemBrain then search the MemBrain help file for the corresponding entry.

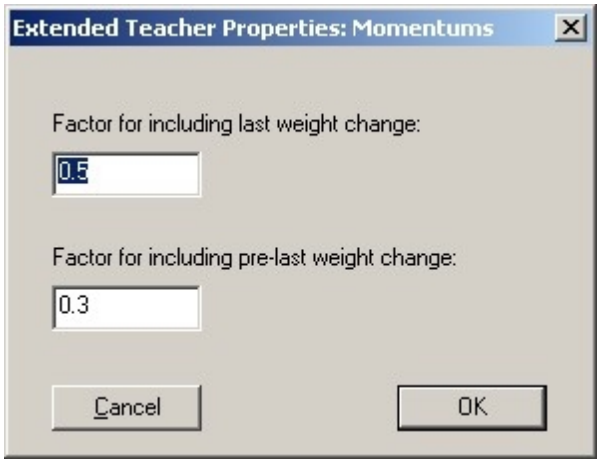
Also you could modify the net architecture and see if you can get better generalization results.
Note: This is a quite big net and performance is significantly improved if you **deselect** the options **<View><Show Links>** and **<View><Show Activation Spikes on Links>.**

The net has been trained using the 'Std BP with Momentum' teacher. However, since this example has been created MemBrain has been equipped with **more advanced learning algorithms**, e.g. the '**RPROP**' teacher which can be used for this net without changing any of the teacher's default parameters.

However, if you still want to use the 'Std BP with Momentum' teacher you can use the following settings for this teacher:



If you click on the button 'Advanced':



Note that training will take significant time because of the size of the Lesson and the large number of links that have to be trained!
Be sure to adjust the setting <Teach><Set Teach Speed> to a value of '0' as this will result in the fastest possible teaching speed.

Time series prediction (Advanced Tutorial)

Time variant feed forward

Advanced Tutorial: Time Series Prediction

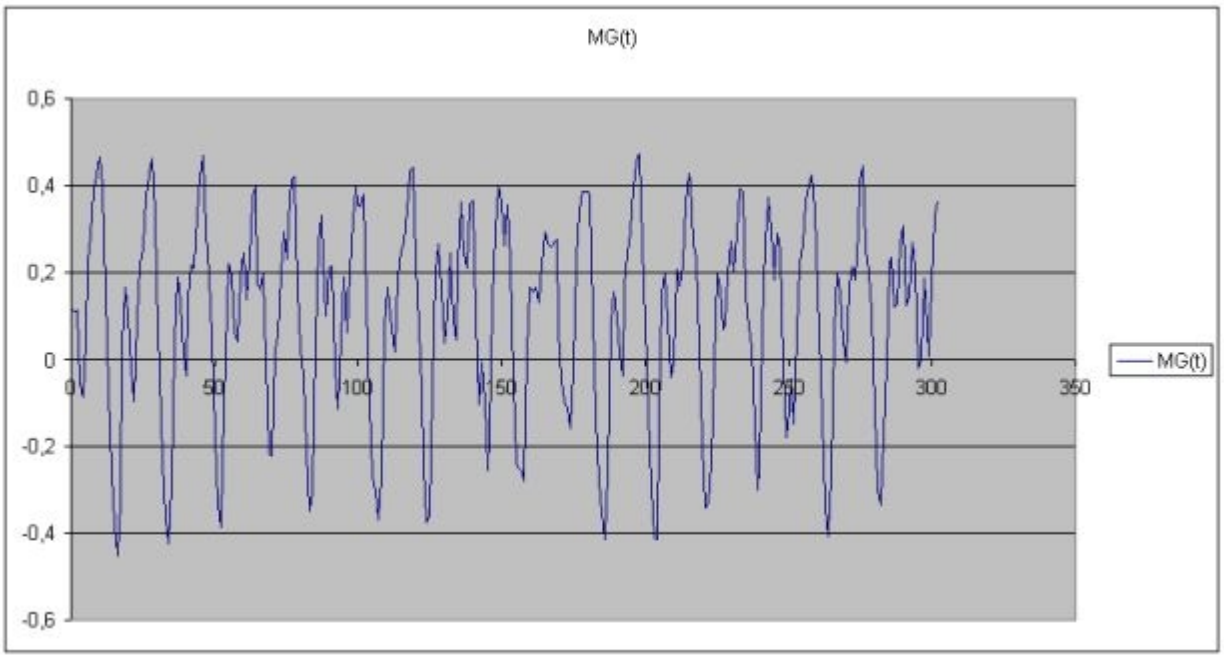
This tutorial is intended for users who already worked through the Short Beginner's Tutorial and who want to get in touch with some more advanced features of MemBrain on a time series prediction example.

Note that this tutorial does not always fully describe every single step that has to be performed since it assumes that you already have basic knowledge of handling neural nets and lesson data in MemBrain.

In this tutorial the goal is to build up and train a neural net to **predict** the so called **Mackey-Glass time series** three time steps ahead.

The Mackey-Glass is a mathematical series that is chaotic in its long term progression but which is predictable in the short term.

The following picture shows the Mackey-Glass series for the time values 0 through 302, i.e. the first 303 values of the series.



For the **training** of the net we will use the **values** for **t = 0 through t = 198** of the Mackey-Glass series where
the input data is the Mackey-Glass time series MG(t) itself and the output is the Mackey-Glass time series value
three times later in time, i.e. MG(t+3).

For **validation** we use a lesson with the Mackey-Glass time series values for t = 199 to t = 299 as input, again
the output being validated against MG(t+3).

The corresponding lessons can be found in the files

MackeyGlassTrain.mbl and **MackeyGlassValidate.mbl**

Go to beginning of the tutorial

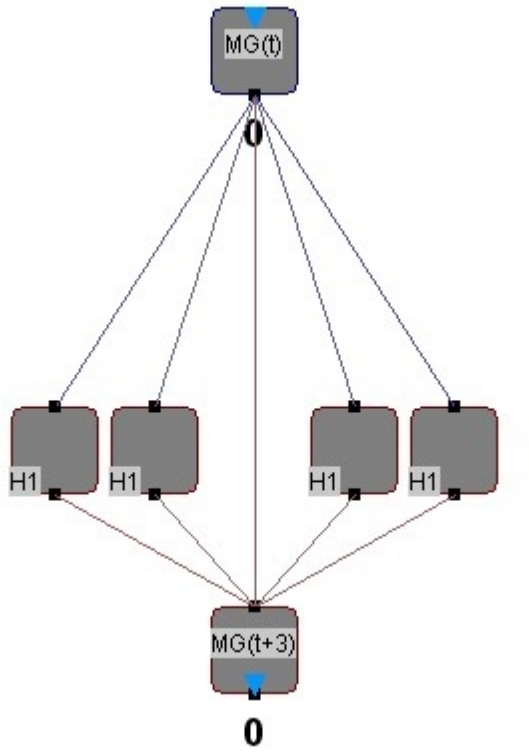
Test with time invariant net

Results when using a time invariant net

As a first attempt we want to see if a **time invariant net** can do the time series prediction job. Certainly we do **not expect this to work** since a time invariant net does not store any information of past patterns internally, i.e. the output solely depends on the current input.

Nevertheless we want to give it a try to verify the just made assumption and to be able to compare the results to those from time variant nets later on.

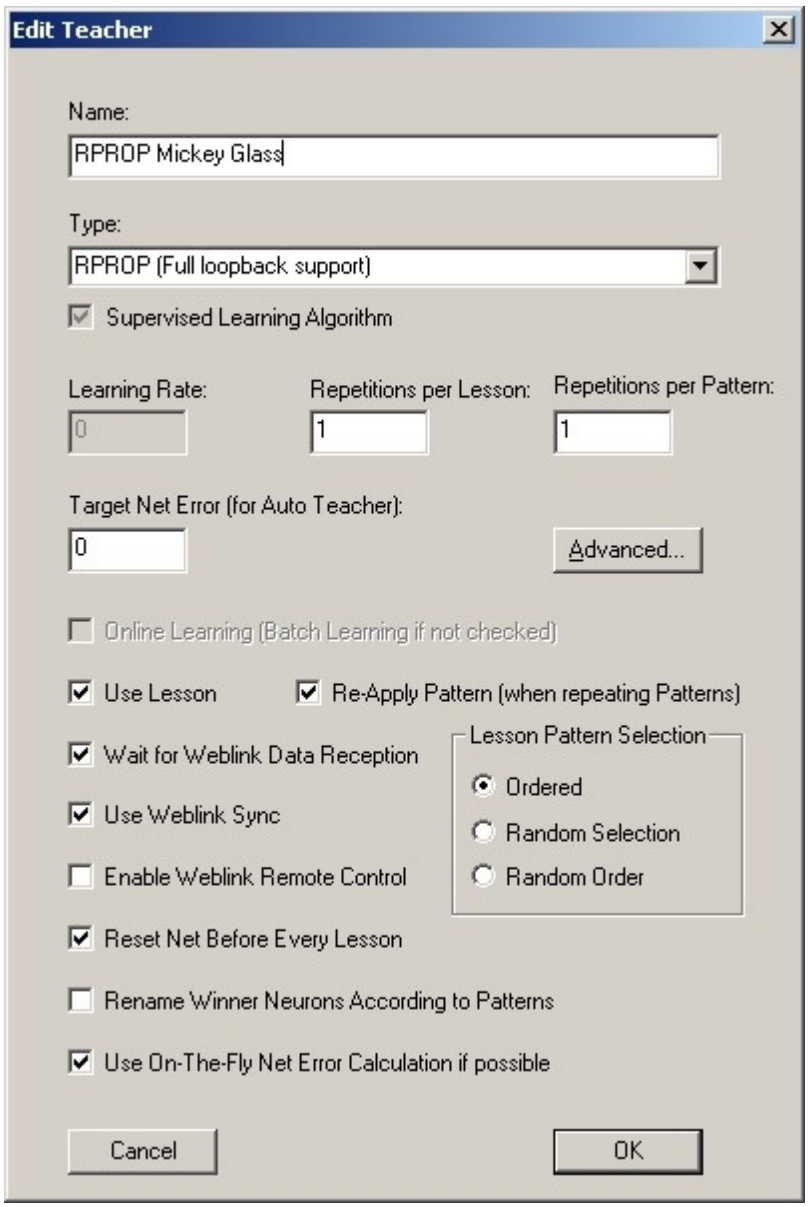
We build up the following net.



The **input neuron** has the activation function **IDENTICAL** while for **all other neurons** we select the activation function **TAN HYP**.
For the **output** neuron we set the data **normalization range to -0.6 .. 0.6**

Note:
If you do not want to draw the net yourself you can also load the example net from file:
MackeyGlassTimeInvariant.mbn

As a teacher we select the 'RPROP' teacher. The only settings we change with respect to the defaults are the 'Lesson Pattern Selection' which we adjust to 'Ordered' and the option 'Reset Net Before Every Lesson' which we additionally select as shown in the following screen shot.



The option 'Ordered' advices the teacher to present the patterns to the net in the order of the lesson. This is certainly important if we want the net to learn rules about the order of the patterns. Learning order rules between patterns actually is the core goal when training nets to predict time series.

The check mark in the box 'Reset Net Before Every Lesson' instructs the teacher to reset all activations stored in the neurons and links right before every lesson start during training.
This is to achieve a defined state before every new lesson run.

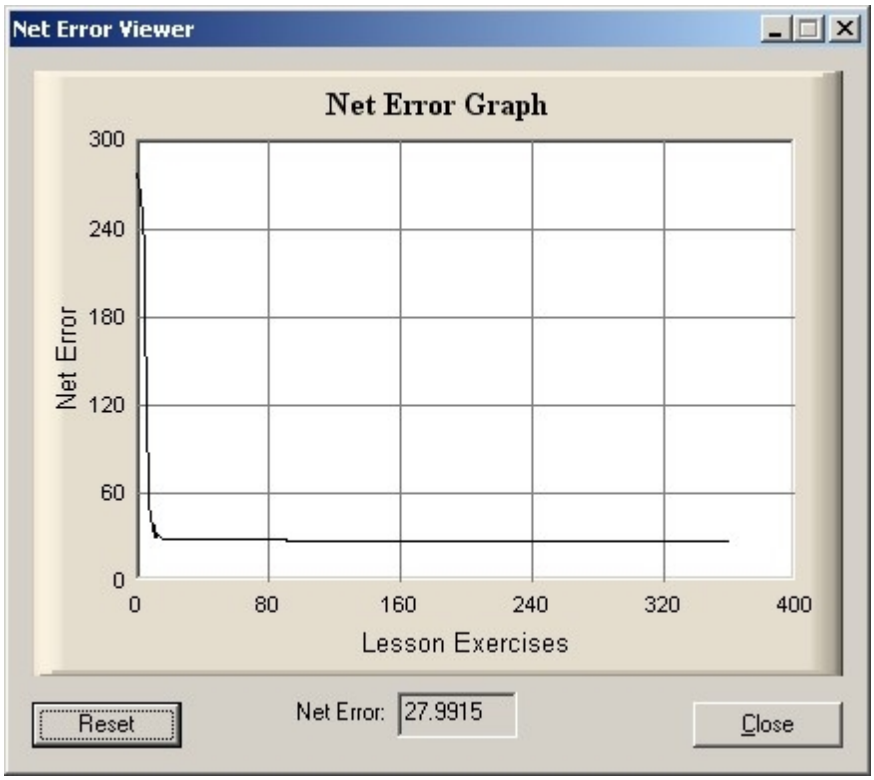
Note: Both mentioned settings are actually not required for time invariant nets. However, we want to also train time variant nets later on during this example and there we will need these adjustments to be in place.

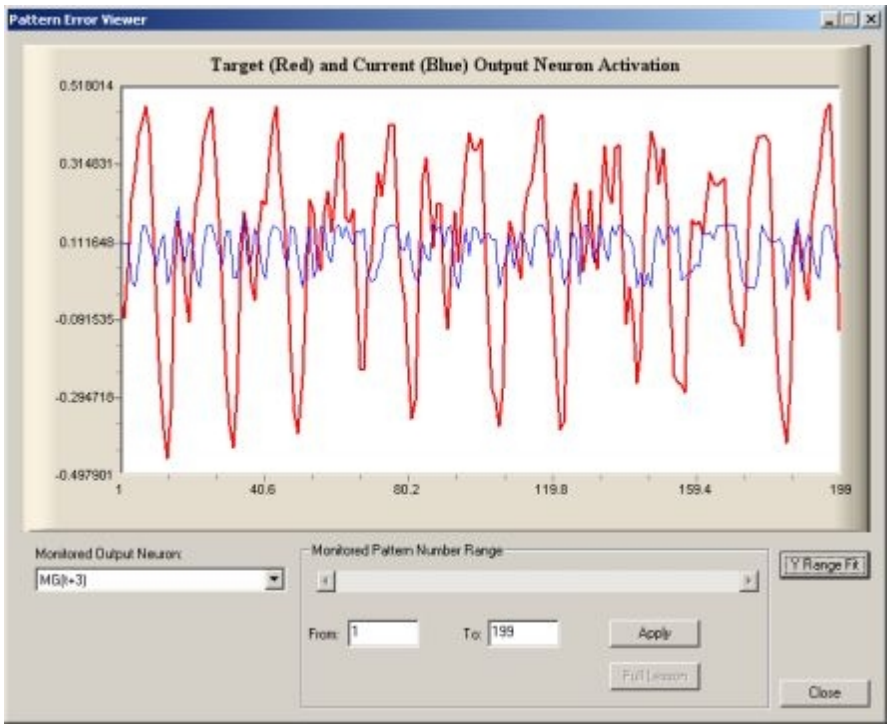
Now we do the following.

- load the training lesson into the lesson #1 using <Lesson Files><Load Current Lesson...> from the Lesson Editor's menu bar.
- increase the number of lessons in the Lesson Editor to 2 using the arrows beside 'Number of Lessons'.
- adjust the currently edited lesson to #2 using the arrows beside 'Currently Edited (Training) Lesson'
- Load the validation lesson using the menu again.
- Adjust the currently edited lesson to #1 again so that this will be our training lesson.

Open the Pattern Error Viewer and the Lesson Error Viewer and start the training. When asked to randomize
the net choose 'Yes'.

You should see something similar to the following on the error viewer and the Pattern Error Viewer.





What we see here is the output of the net during the training phase. We can see that the net is not even able to approximate the **training** data set. Thus its no use to check the reaction on the validation data set.

We will first have to improve our net which in this case means to turn it into a time variant net.

Continue with time variant net version #1

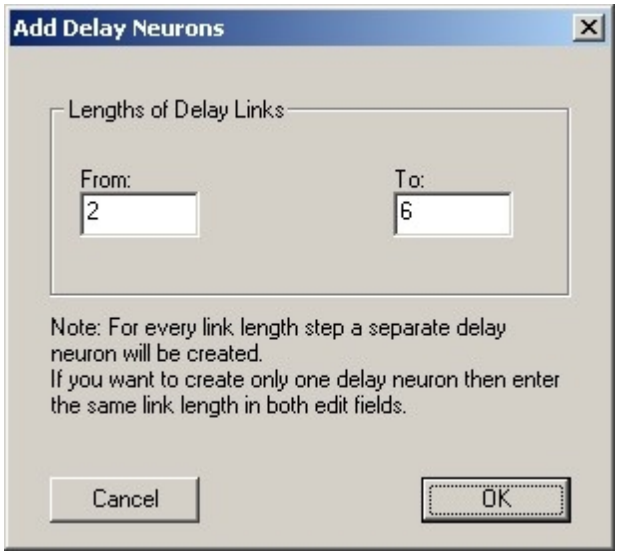
Time variant net: Version #1

Time variant net: Version #1

Mackey-Glass time variant net #1

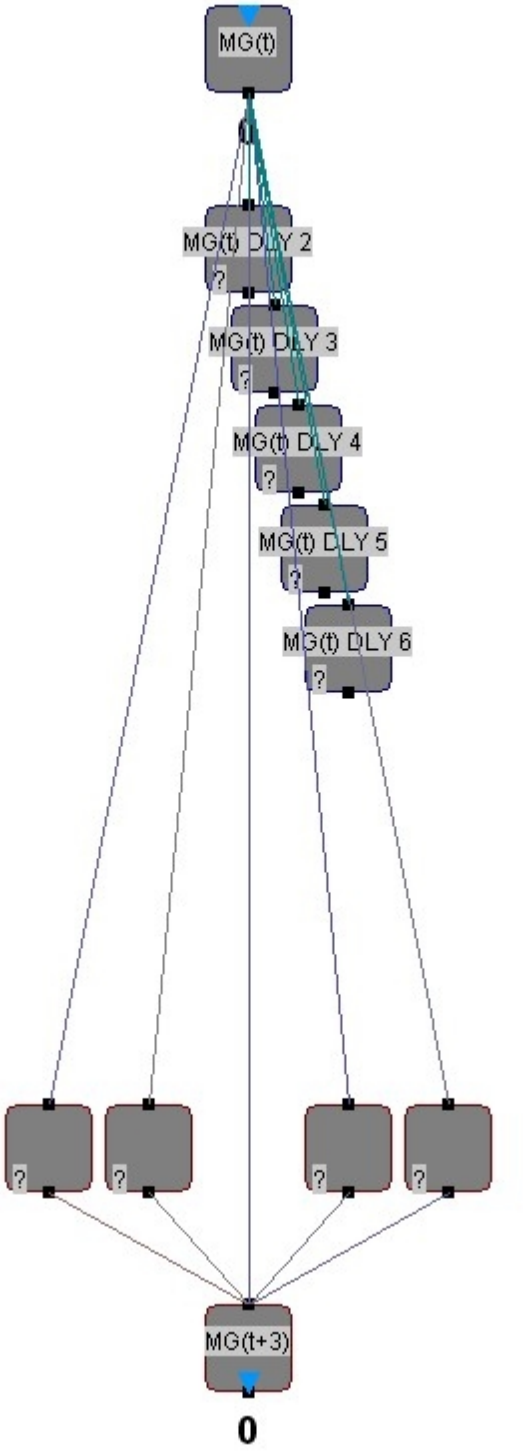
As a first approach to make our net time variant we will add some delay neurons to the input. For more information about delay neurons see the MemBrain help file in section 'Neurons in MemBrain' - 'Adding Delay Neurons'.

- Ensure there is enough space between the input neuron and the first hidden layer (use the zoom function to get more free space in the drawing area).
- Select the input neuron. Then right click on the selected input neuron and select <Add Delay Neurons...>.
  The following dialog will appear.



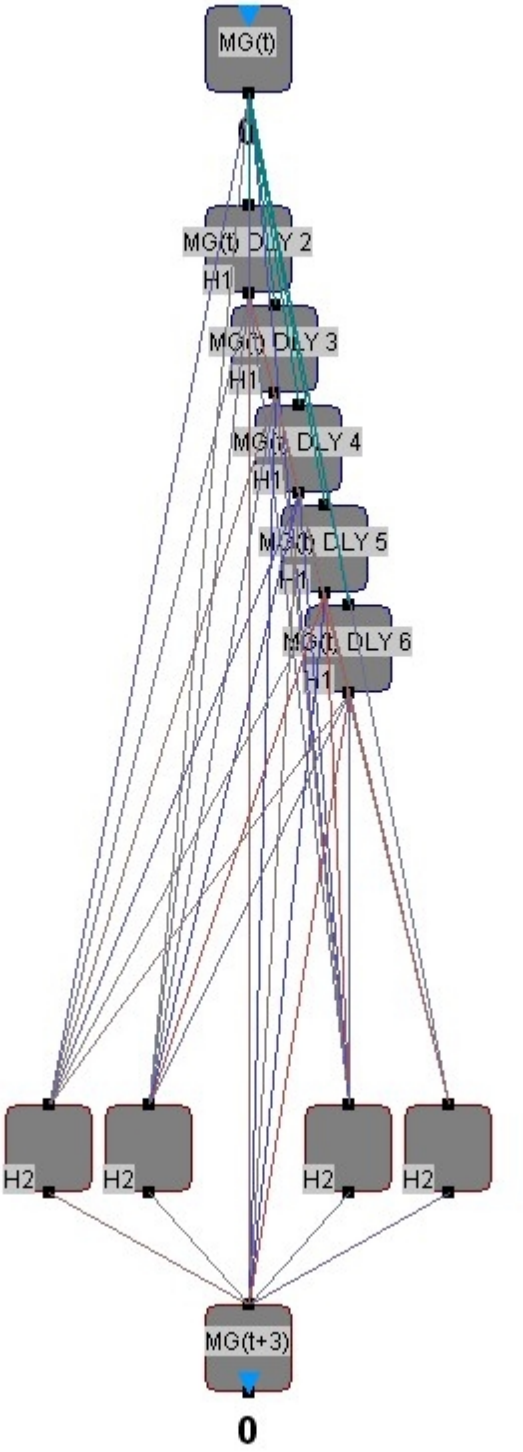
- Click on OK

The net now looks as following.



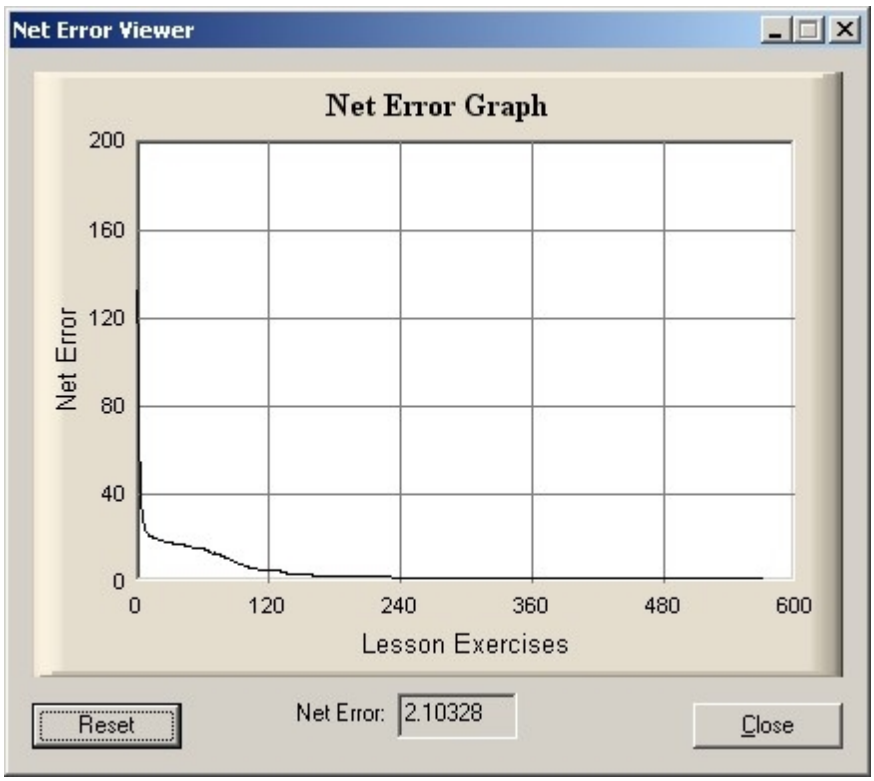
Five delay neurons have been added to the input. These reflect the input signal at t = -1 through t = -5.

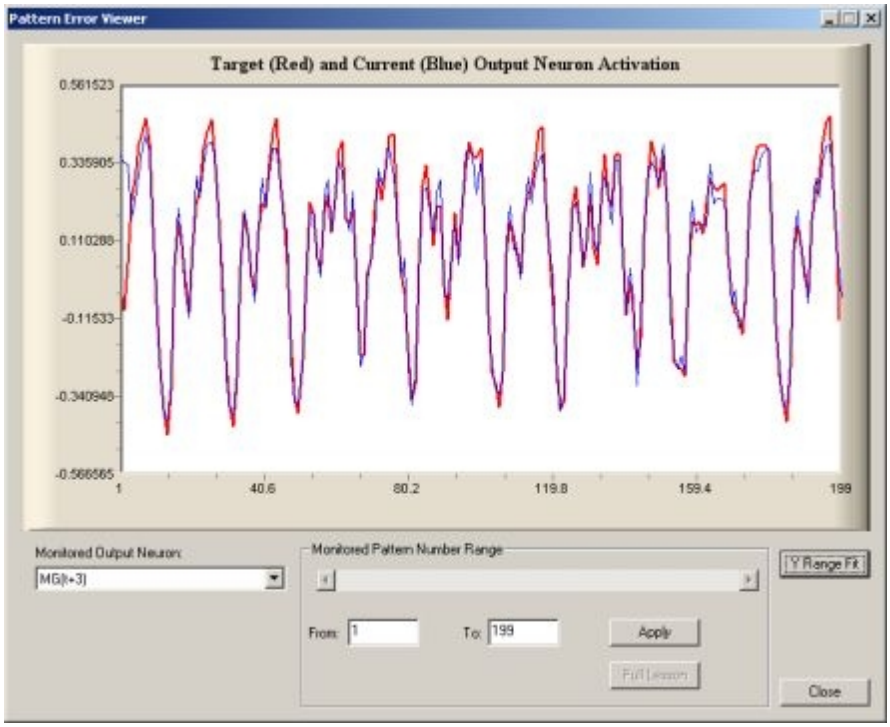
We will now connect these delay neurons to the hidden neurons and the output neuron to build the following net.



Note:
If you do not want to do the net edits yourself you can also load the example net from file:
MackeyGlassTimeVariant1.mbn

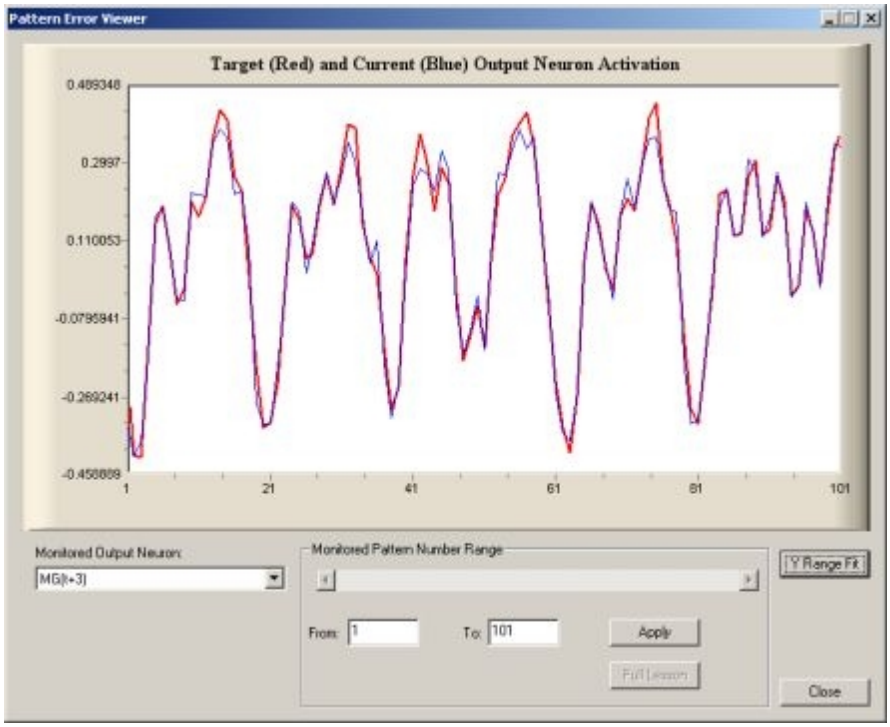
We now randomize the net and repeat the training. After some stabilization time
we get the following results:





As we can see things have become significantly better now! The training data set can be learned quite well already.

Try to **switch to the validation lesson during training**: Change the number of the 'Net Error Lesson' on the Lesson Editor to #2 using the arrows beside the corresponding text. You will have to **check the box** <**Set Manually**> before you can change the Net Error Lesson.
The Net Error Viewer and the Pattern Error Viewer will immediately update to reflect the validation lesson result:



This now is the reaction of the net to the **untrained validation data**. As we can see the net already generalizes very well.

Note that you can see the validation result during training already! You can switch back and forth from training result to validation result using the Lesson Editor's setting 'Net Error Lesson'. The training result is not influenced by this action at all, i.e. the net does not get trained on your validation data. It is only trained on the lesson that is selected as the 'Currently Edited (Training) Lesson' in the Lesson Editor which is Lesson #1, i.e. our training lesson.

We will now try to further improve our net using decay neurons in addition to the already used delay neurons.

Continue with time variant net version #2

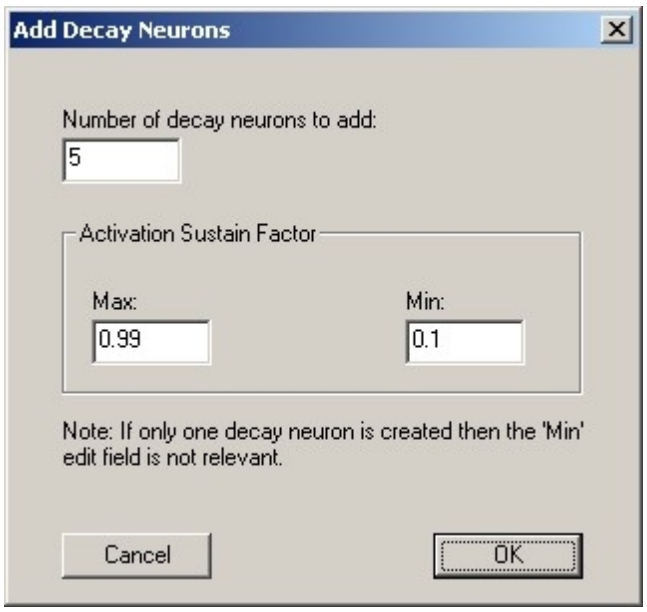
Time variant net: Version #2

Time variant net: Version #2

Mackey-Glass time variant net #2

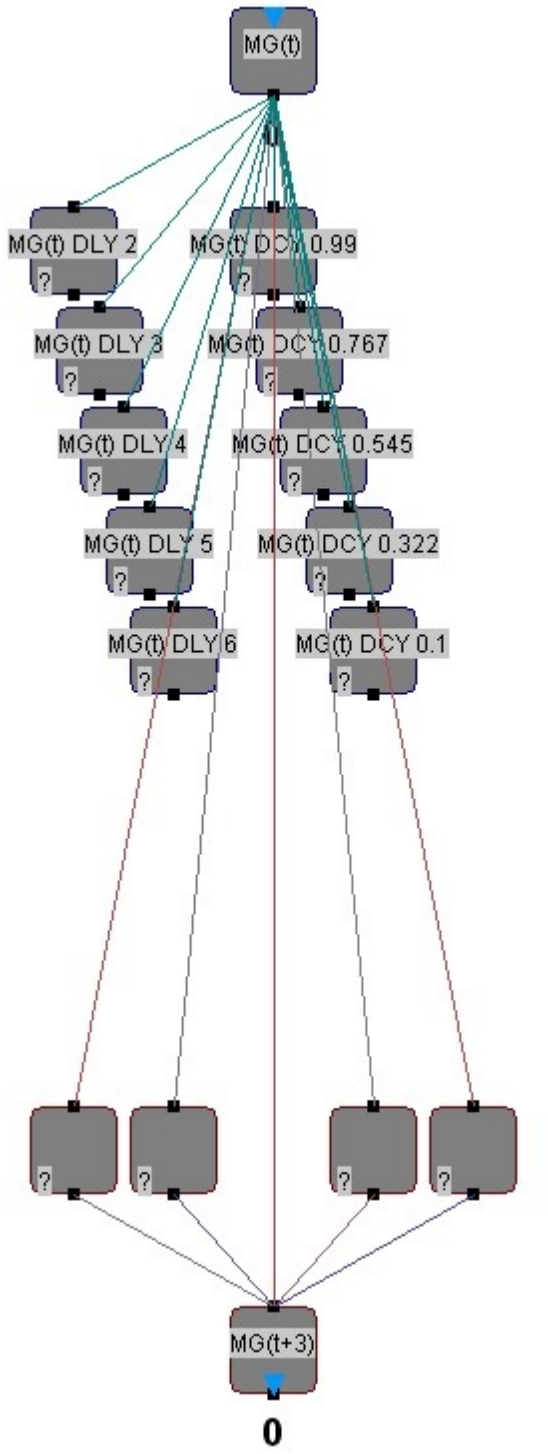
We are now going to add so called decay neurons to our net. For more information about decay neurons see the MemBrain help file in section 'Neurons in MemBrain' - 'Adding Decay Neurons'.
To add the decay neurons do the following.

- Move the already added delay neurons one or two locations to the left since the decay neurons will be generated at the same positions as the already present decay neurons. Thus, if you don't move them away they will be placed over each other which is not practical.
- Select the input neuron. Then right click on the selected input neuron and select <Add Decay Neurons...>.
  The following dialog will appear.



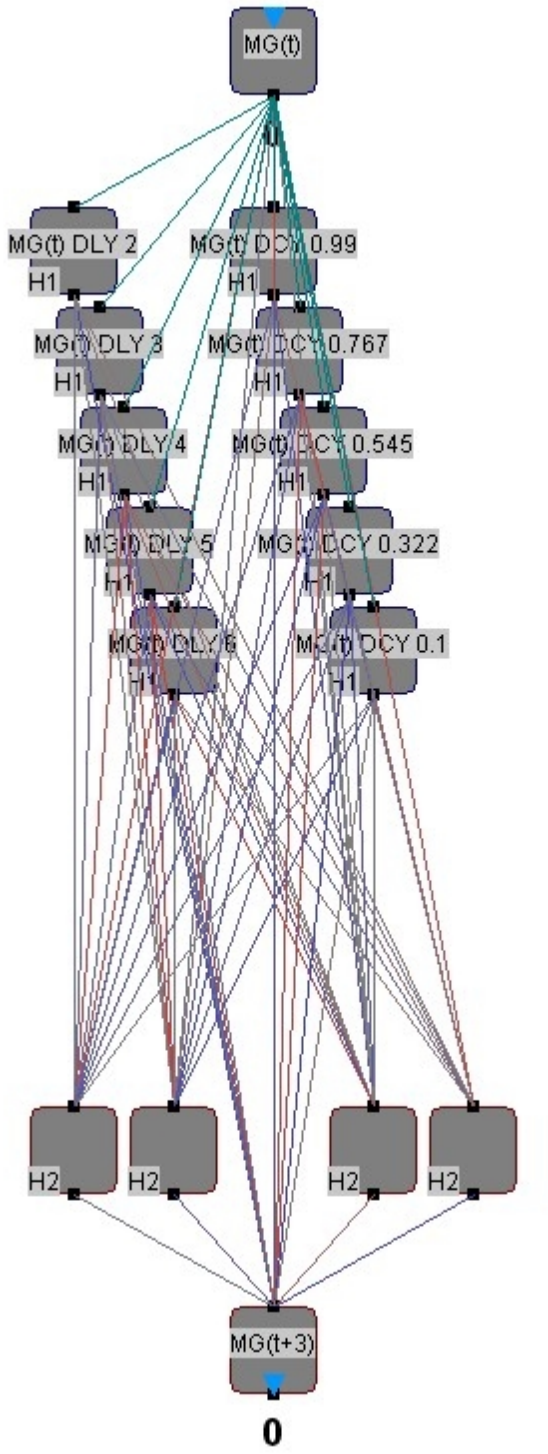
- Click on OK

The net now looks as following.



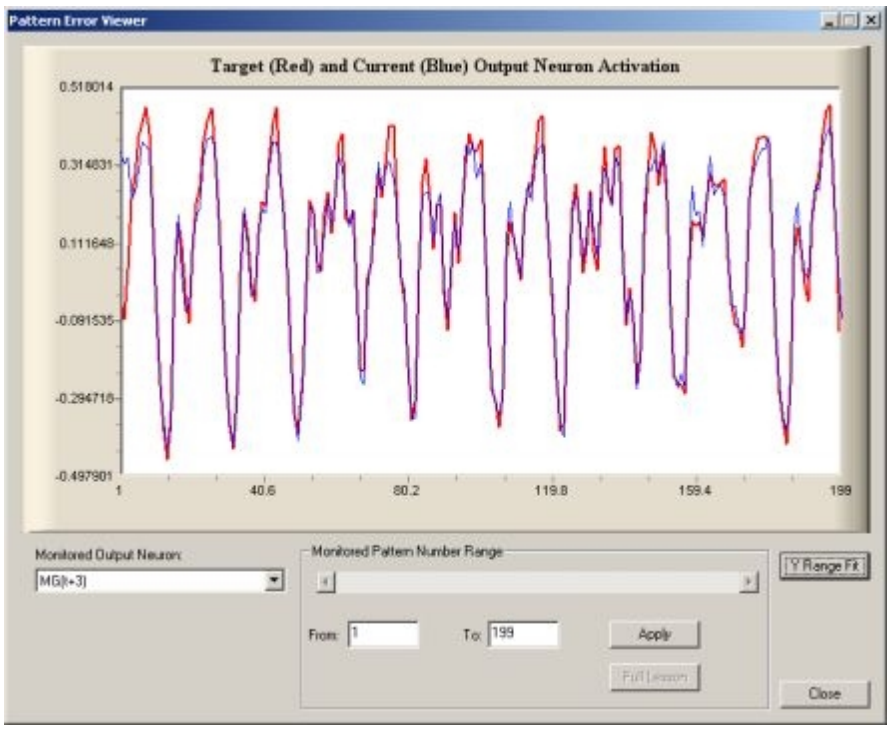
Five decay neurons have been added to the input. These reflect the input signal with 'Activation Sustain Factors ranging from 0.99 to 0.1.

We will now connect these delay neurons to the hidden neurons and the output neuron to build the following net.

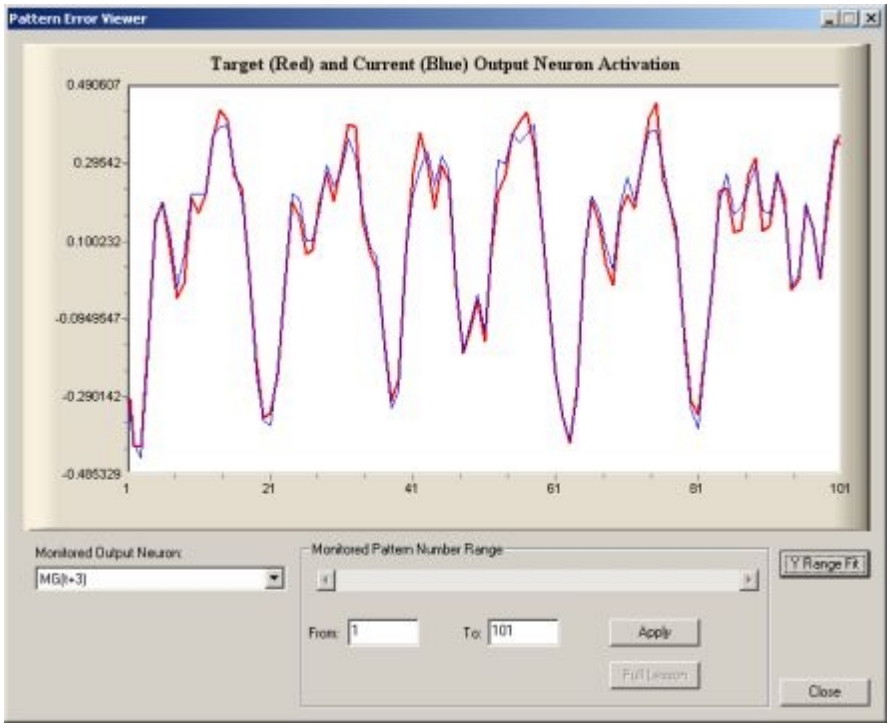


Note:
If you do not want to do the net edits yourself you can also load the example net from file:
MackeyGlassTimeVariant2.mbn

We now randomize the net and repeat the training. After some stabilization time
we get the following results:



Switching to the validation lesson during training shows the following graph.



As we can see the results did not significantly improve anymore. It looks like the delay information already was enough for the net to learn the rules.

As a cross check we now want to delete the delay neurons from the net and keep just the decay neurons:

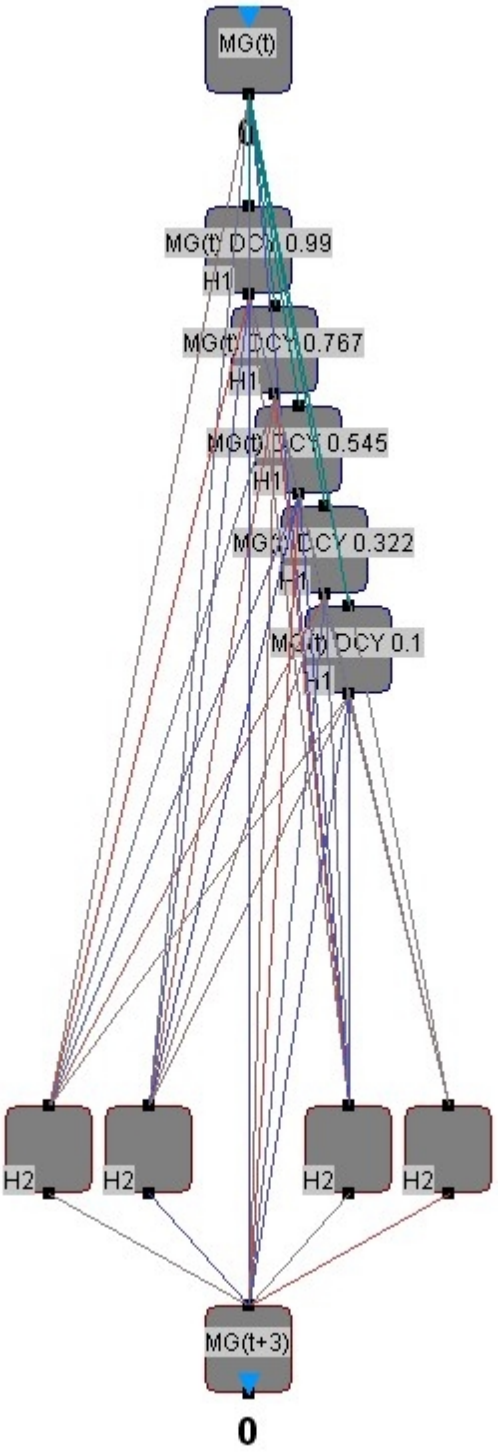
Continue with time variant net version #3

Time variant net: Version #3

Time variant net: Version #3

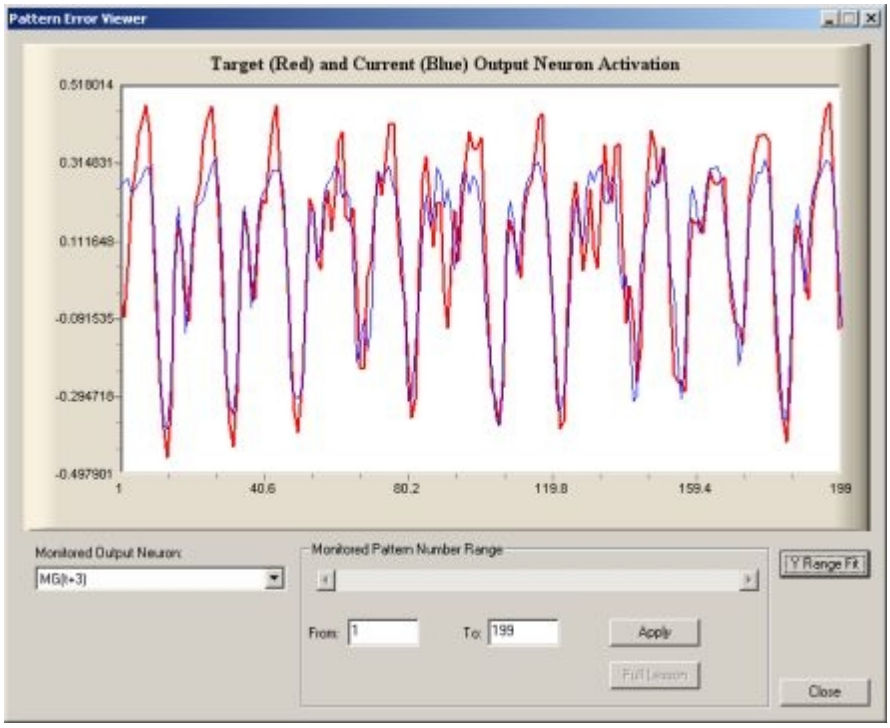
Mackey-Glass time variant net #3

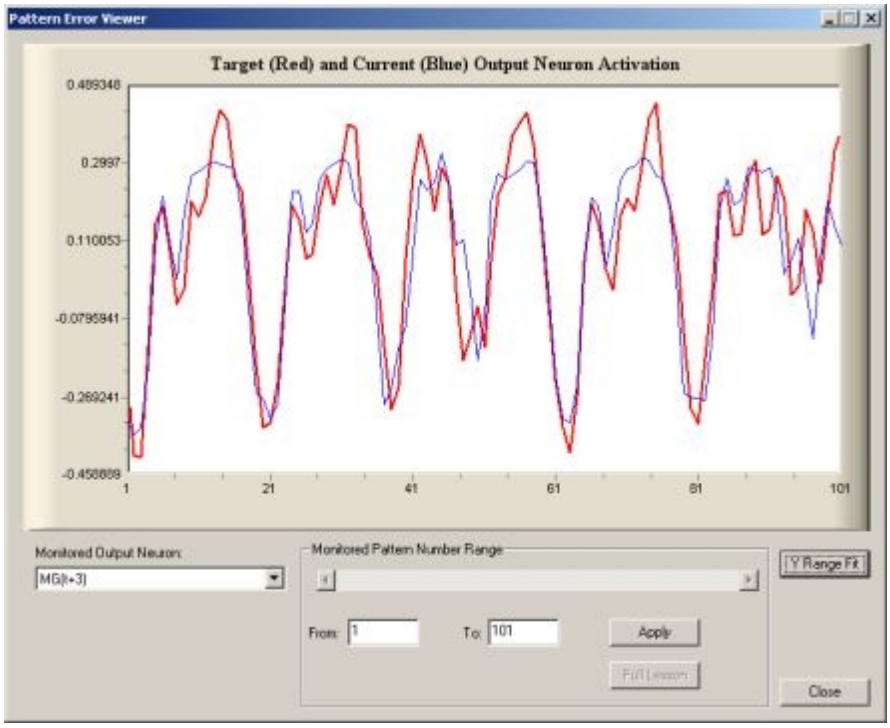
As a cross check we delete the delay neurons from the net and keep just the decay neurons:



Note:
If you do not want to do the net edits yourself you can also load the example net from file:
MackeyGlassTimeVariant3.mbn

This leads to the following results for training and validation:





As we can see the decay neurons significantly improve the training results compared to the time invariant net. However, the delay neurons seem to be far more important. This is due to the characteristics of the Mickey-Glass time series: The recent values provided by the delay neurons are of much higher interest for the prediction of the future values than the low pass filtered past value that the decay neurons can provide.
In other words the future value mostly depends on the few recent time values and not so much on the gross time course of the function during the past.

We now want to see if we can further improve the version #1 net (the one with the delay neurons).

Continue with time variant net version #4

Time variant net: Version #4

Time variant net: Version #4

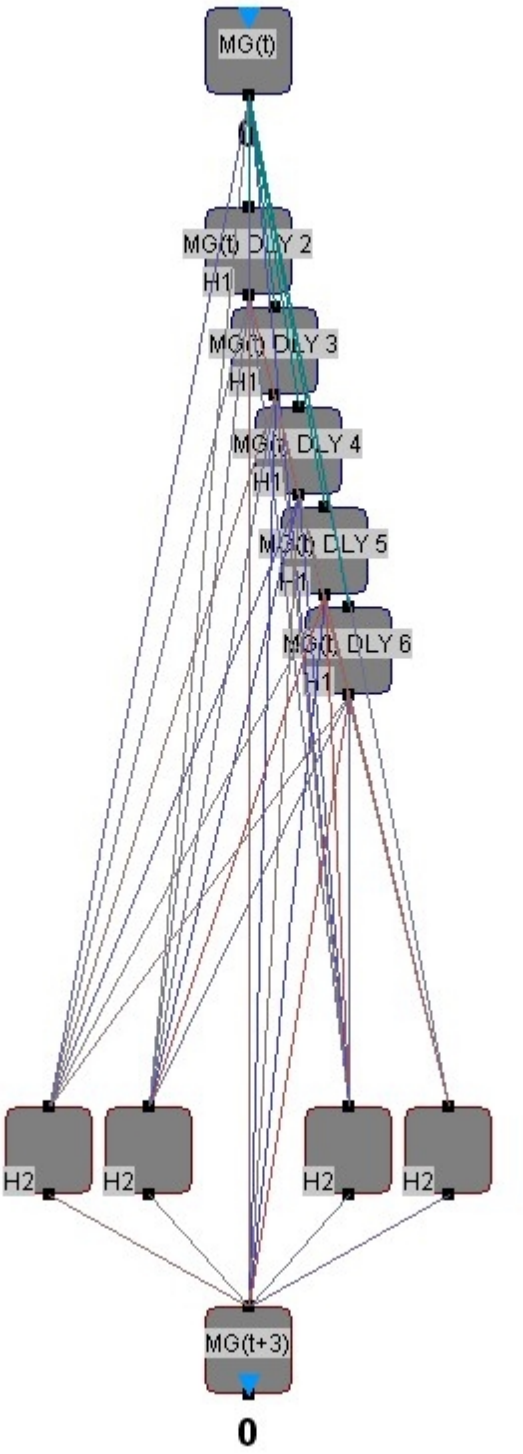
Mackey-Glass time variant net #4

We will now try to improve the time variant net #1 by adding differential neurons. These neurons build the difference between two source neurons. If the two source neurons are an input and one of its delayed signals then the differential neuron will output a signal that is proportional to the change ratio of the input neuron. I.e. the neuron will output the differential signal of the time series.

See the MemBrain help file for more information about differential neurons.

In order to check if this approach helps with the Mackey-Glass problem we modify the net version #1 to incorporate a differential neuron.

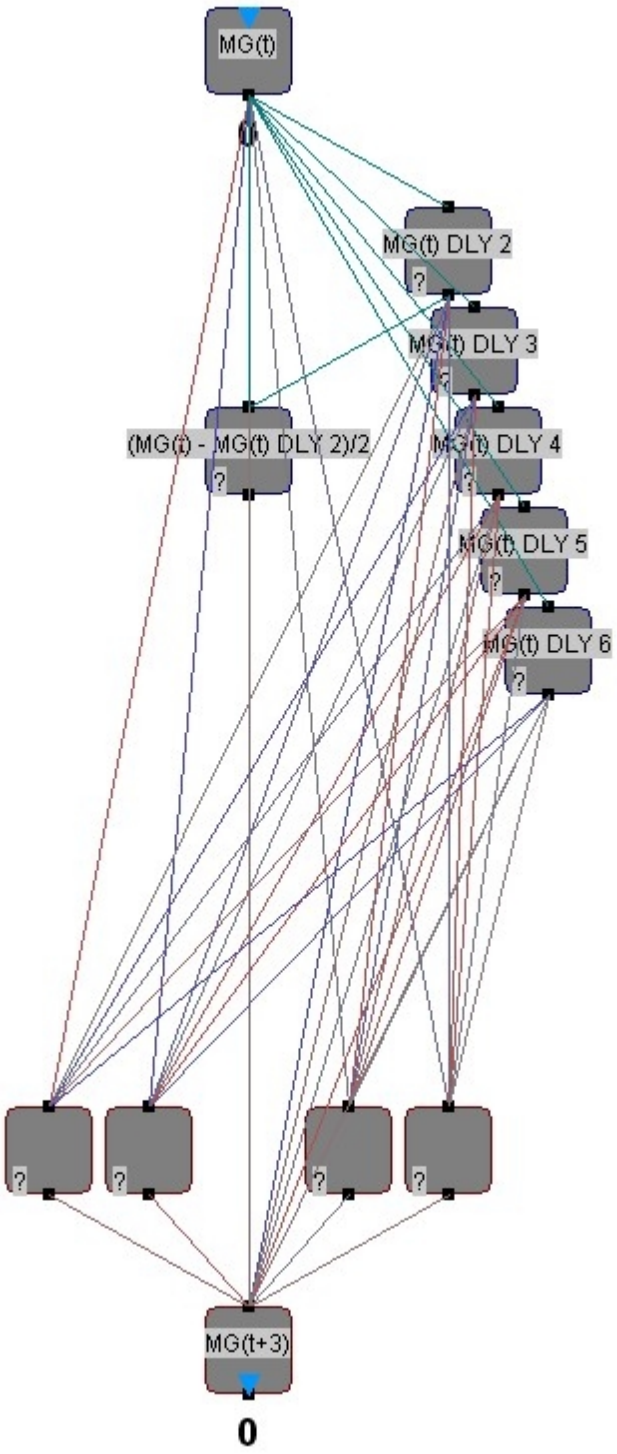
The following picture shows the time variant net #1 again.



Now we do the following.

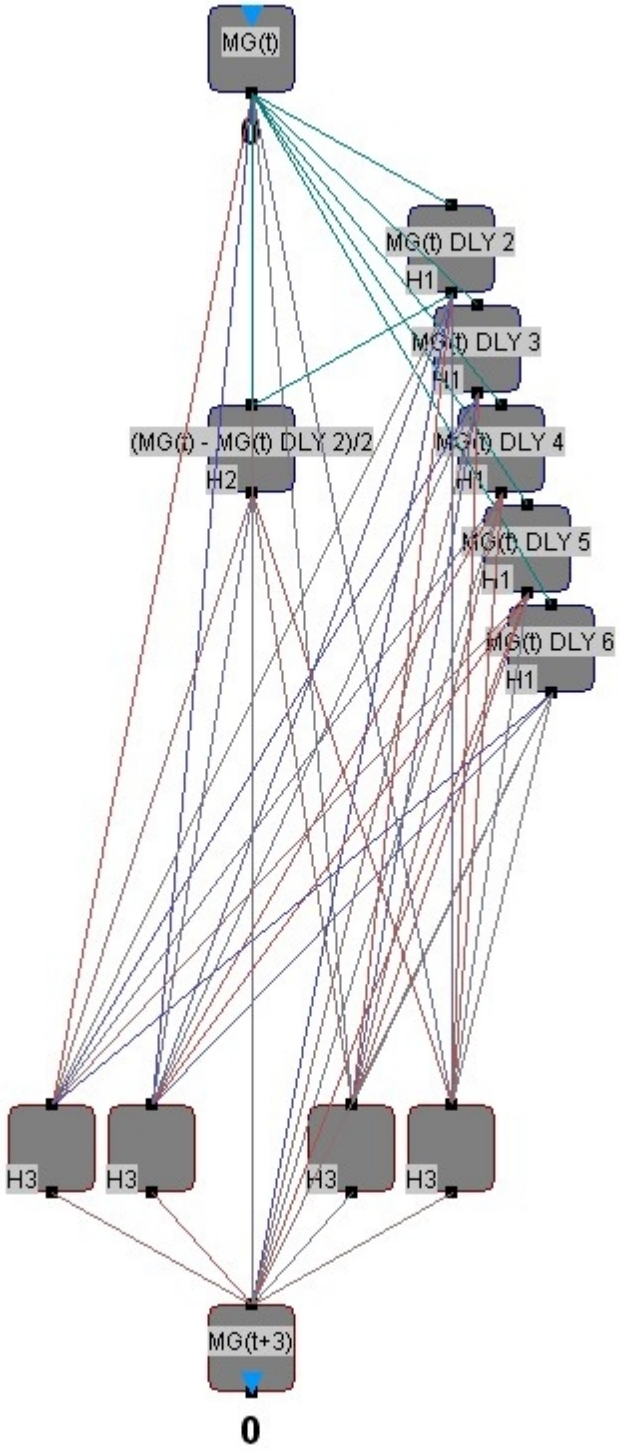
- Move the delay neurons a bit aside to gain space beneath the input neuron.
- Select the **input neuron and the first delay neuron** (called 'MG(t) DLY 2').
- Right click on one of the neurons and select <Add Differential Neuron> from the context menu.

The net now looks as follows.

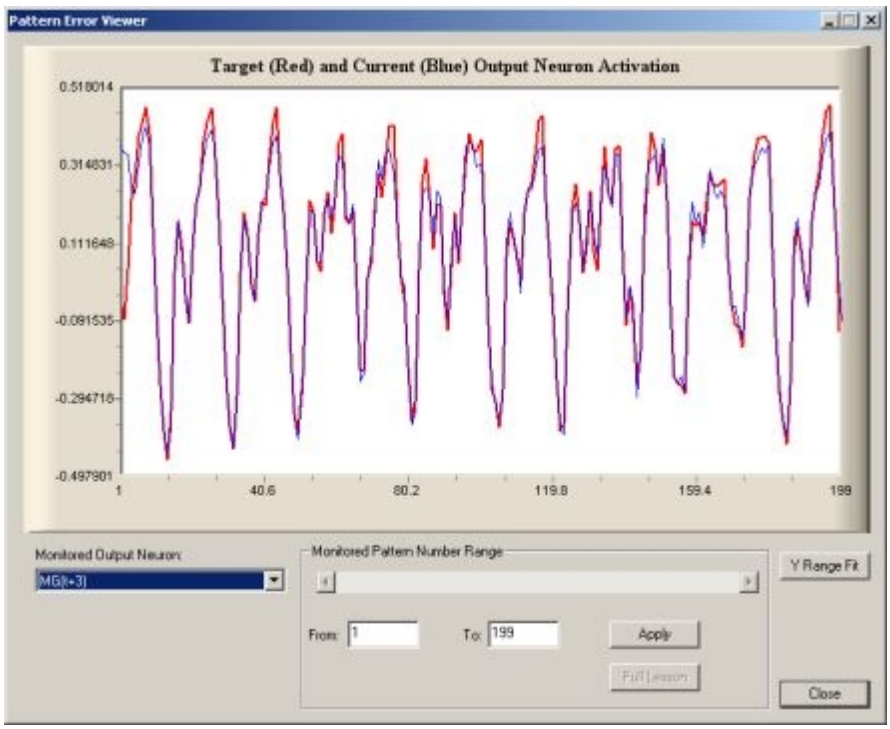


Note the newly created differential neuron with the name '**(MG(t) - MG(t) DLY 2)/2**'. It represents the difference between neuron MG(t) and its first delay neuron divided by 2.

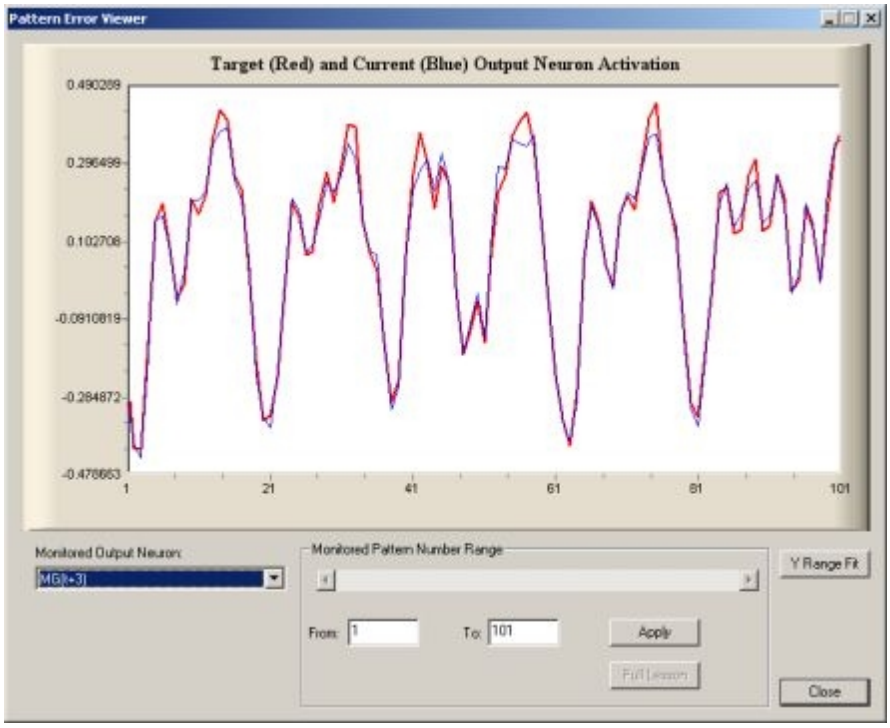
We now connect this new differential neuron to the hidden and the output layer of the net. After randomization the net looks like following.



We start the training and look at the results for training...



... and for validation:



Again it is difficult to tell if this has improved the results or not. At least it doesn't seem to have significant influence.

Finally, we can conclude that the Mackey-Glass time series can be best predicted with a neural net that incorporates the last few time values of the series derived from delay neurons.
This example has not been optimized so far. If you like you can try to further optimize the net, e.g. by adding more delay neurons that reach further into the past or by adding more neurons in the hidden layer, more layers etc.

<End of tutorial>

Autoencoders (Advanced Tutorial)

Autoencoders

Advanced Tutorial: Autoencoder for recognition of handwritten digits

This tutorial is intended for users who already worked through the Short Beginner's Tutorial
and who want to get in touch with some more advanced features of MemBrain on an autoencoder network example.

Note that this tutorial does not always fully describe every single step that has to be performed since it assumes that you already have basic knowledge of handling neural nets and lesson data in MemBrain.

In this tutorial the goal is to train an autoencoder network on basis of handwritten digits taken from the MNIST database.
After training the autoencoder, the outputs of the network are trained to predict the digits 0 .. 9 from the graphical digit pixel input.

Go to beginning of the tutorial

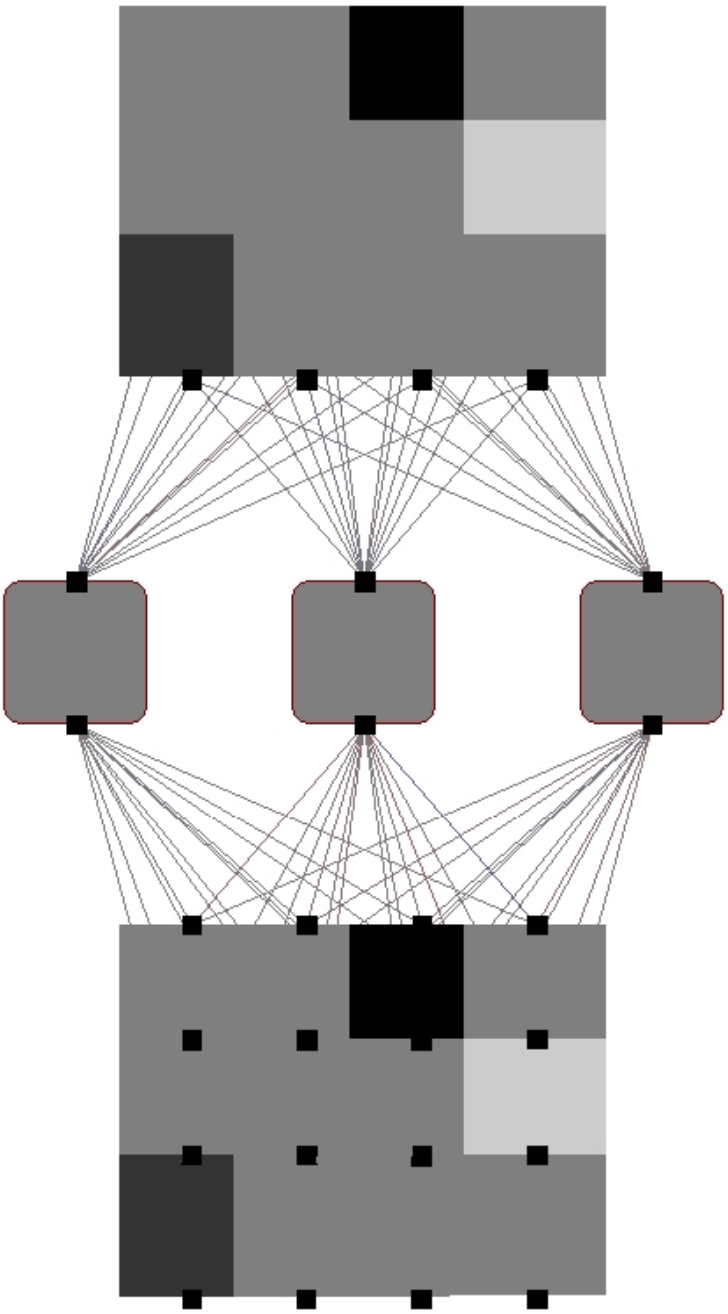
What is an Autoencoder?

What is an Autoencoder?

Architecture of an Autoencoder network

An Autoencoder net is a neural net that learns to reproduce the activations of a set of input neurons at its output neurons.
The input neurons typically represent pixels of an image. I.e., the set of input neurons typically forms an image.
Since the inputs shall be reproduced at the outputs of the Autoencoder, the output neurons are arranged in the same way as the inputs.
One or more hidden layers interconnect the inputs and the outputs of the Autoencoder net.

The following example shows a small Autoencoder net. It features an input and an output matrix of 12 neurons/pixels each and a single hidden layer with three neurons:



Continue with tutorial

What are Autoencoders used for?

What are Autoencoders used for?

Compressed information is available in hidden layer of Autoencoder

A standalone Autoencoder typically is of limited use: Once trained, it moree or less reproduces the input activations at its outputs. This functionality can be helpful for filtering data: Since the Autoencoder must route all information through its smaller hidden layer, it is forced to generalize from its inputs and find a compressed representation of the input matrix within its hidden layer. Loss of data is the intended consequence here. The output of the Autoencoder network is "smoother" than its input since it does not provide all data of each single input pixel anymore.
Of much higher interest is the hidden layer itself, however: Since it contains a compressed representation of

the input data it can be used to train further neural network layers from in order to acually classify the input data.
This is what we will do in this tutorial: We will train an Autoencoder to reproduce images of handwritten digits (taken from the popular MNIST database). After having trained the Autoencoder we use the trained hidden layer of the Autoencoder network part to train another set of layers, including an oputput layer which represents a digit between 0 and 9. The final network then is able to detect and classify handwritten digits.

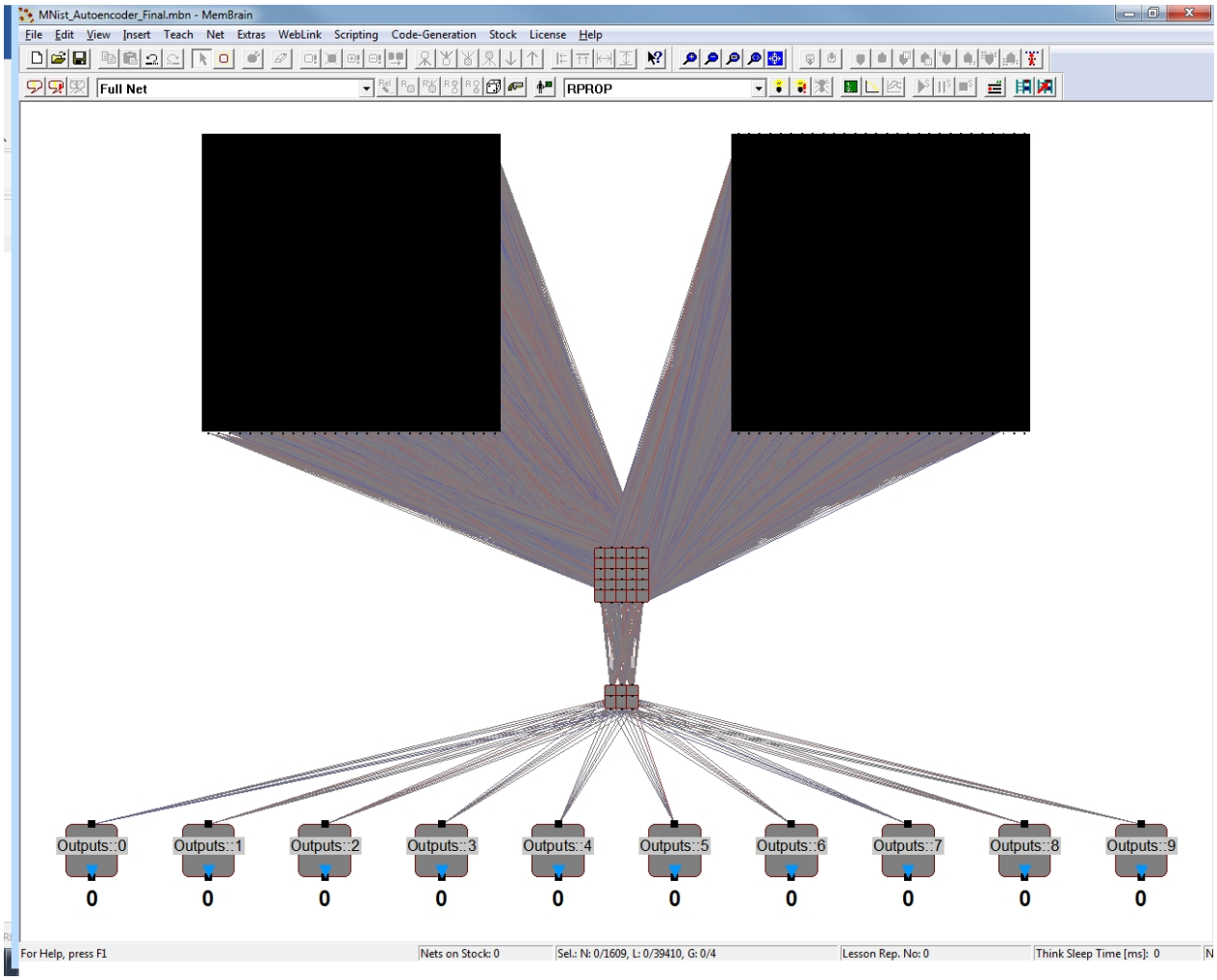
Continue with tutorial

Autoencoder net: Overview

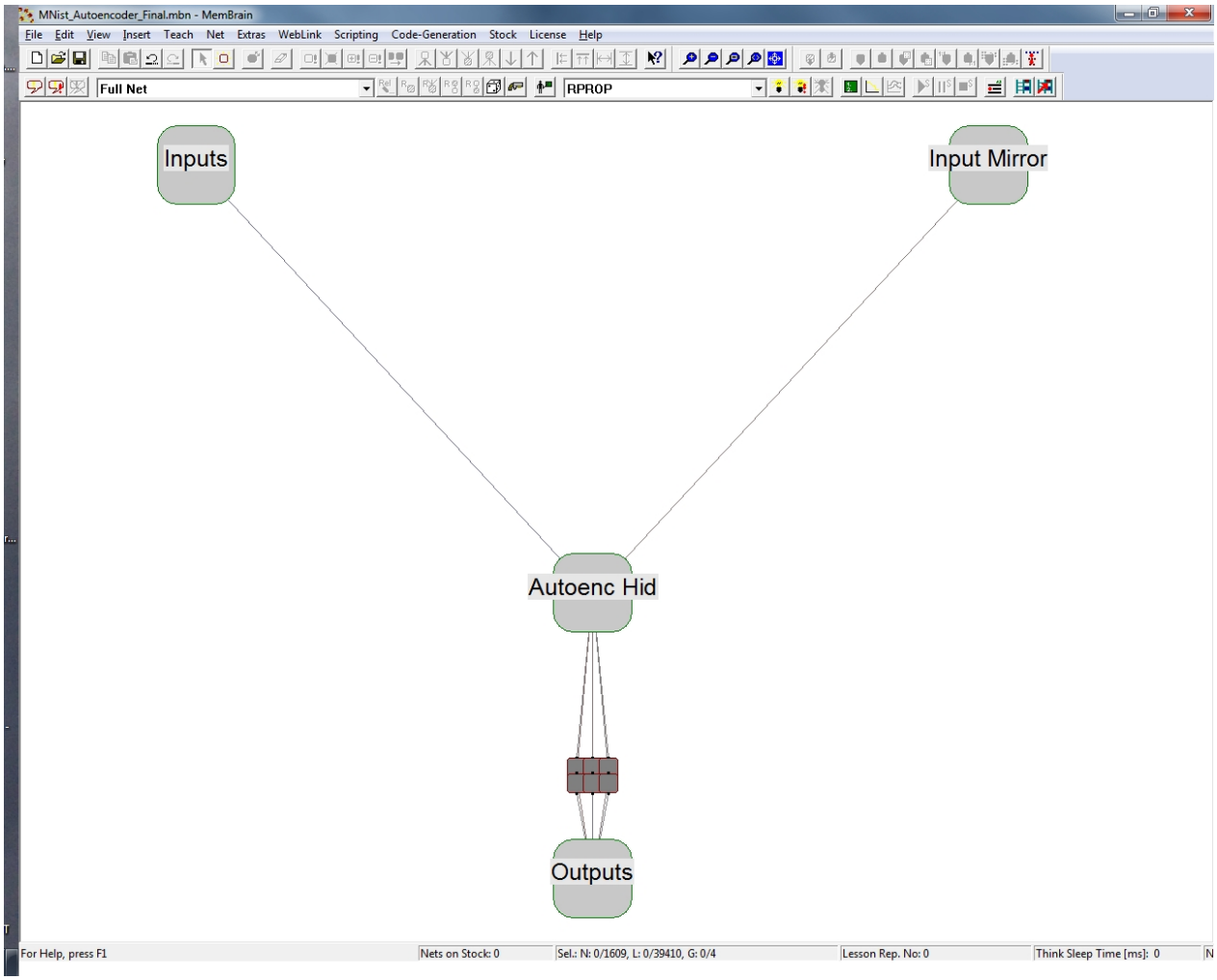
Autoencoder net: Overview

Full Autoencoder net

First, take a look at the net we are going to build and train during this tutorial.

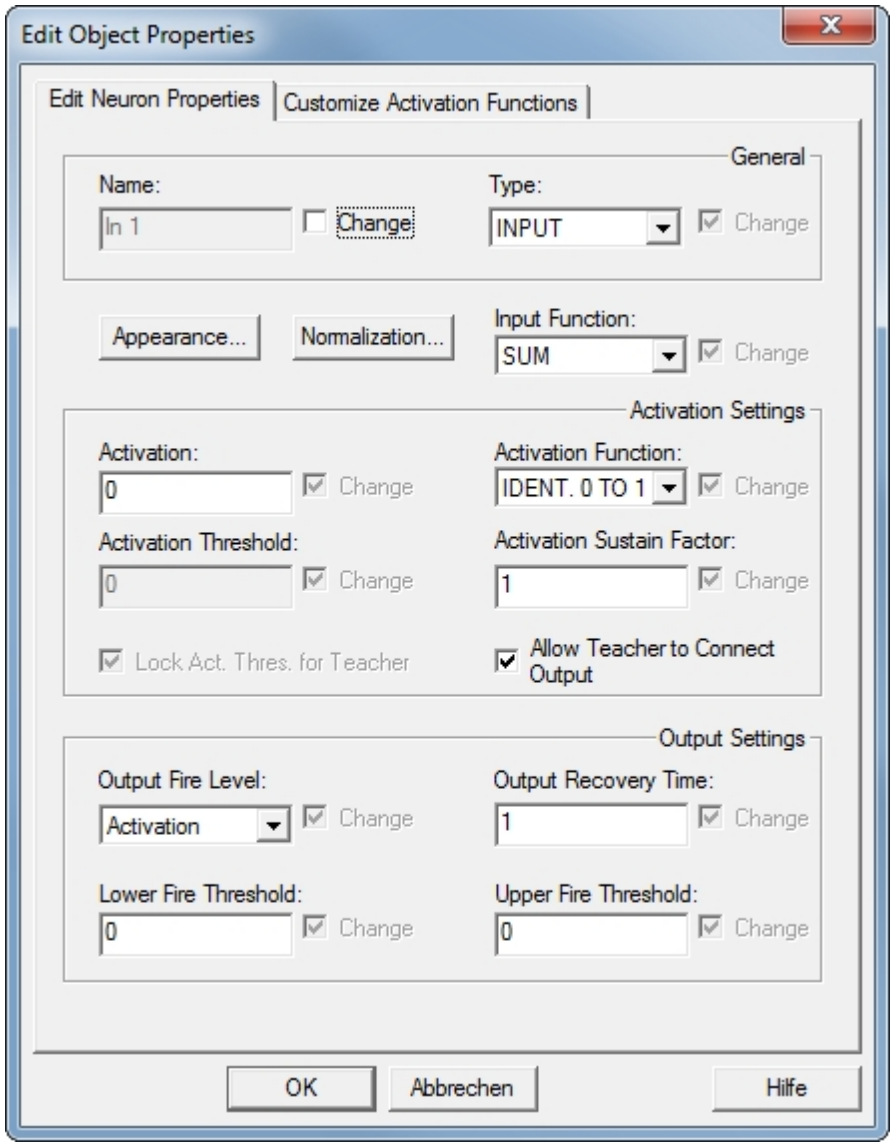


The net containes four groups of neurons. In the following picture all groups are collapsed so that their names can be identified:



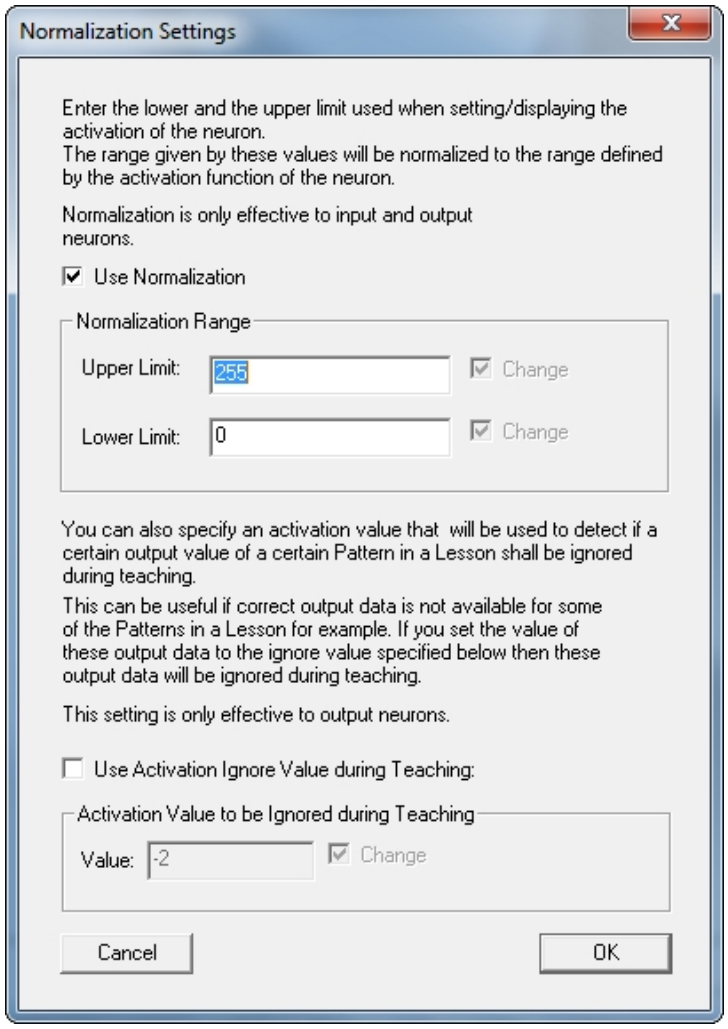
The goups have the following meanings:

1. Group "**Inputs**"
   Contains the input pixel matrix with a dimension of 28 x 28 = 784 gray scale pixels
   The following properties are applied to these input neurons:

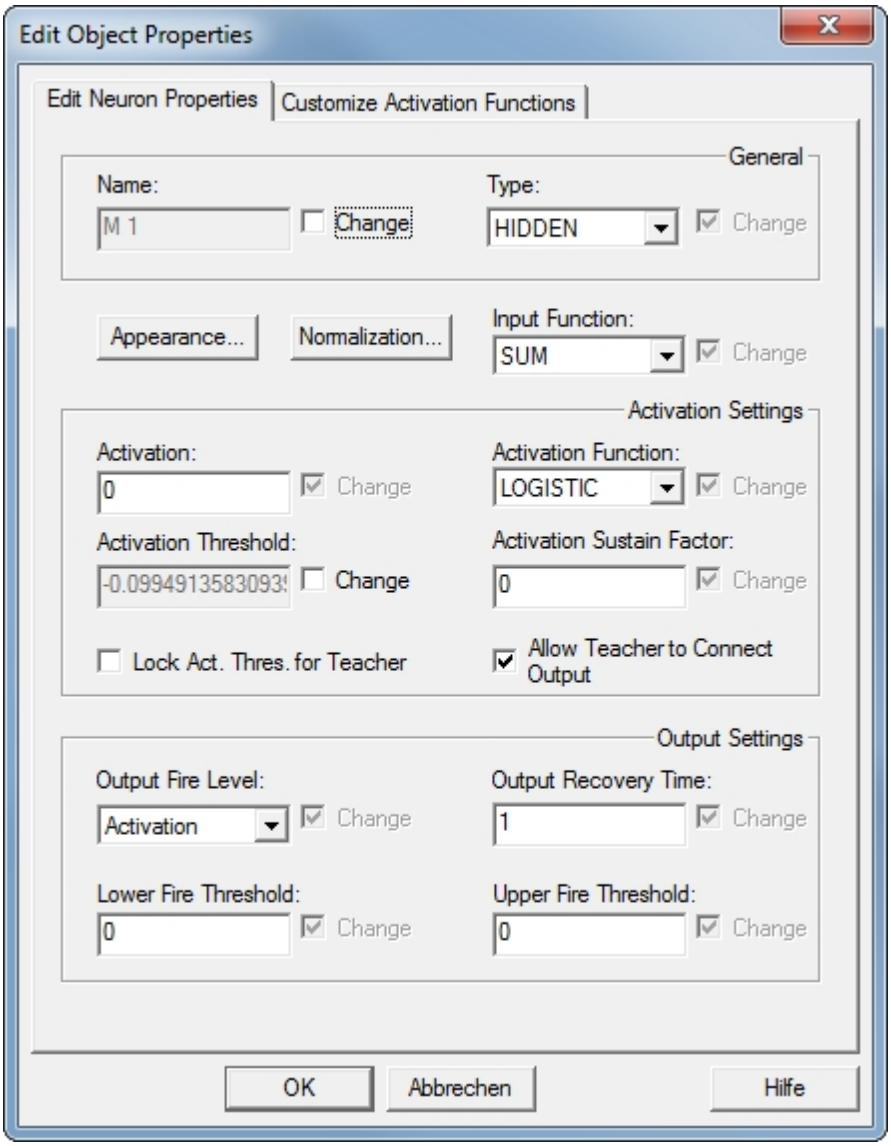


The Appearance properties are set as following:

Normalization is as following:

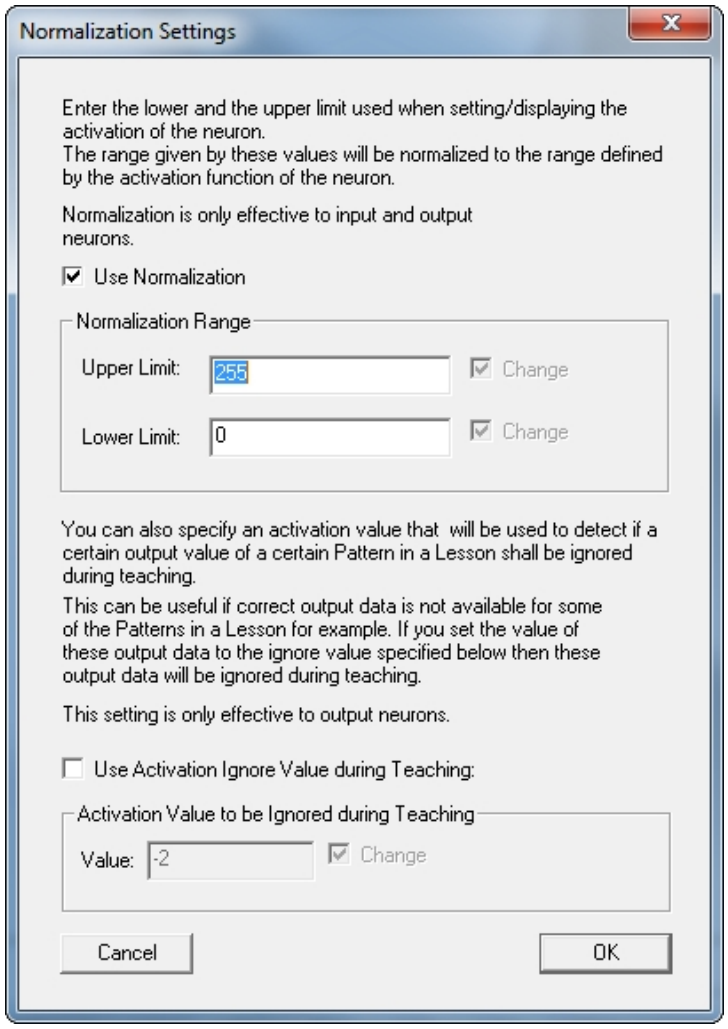


2. Group "**Input Mirror**"
   A group of hidden neurons shaped identically to group "Inputs". The following properties apply:

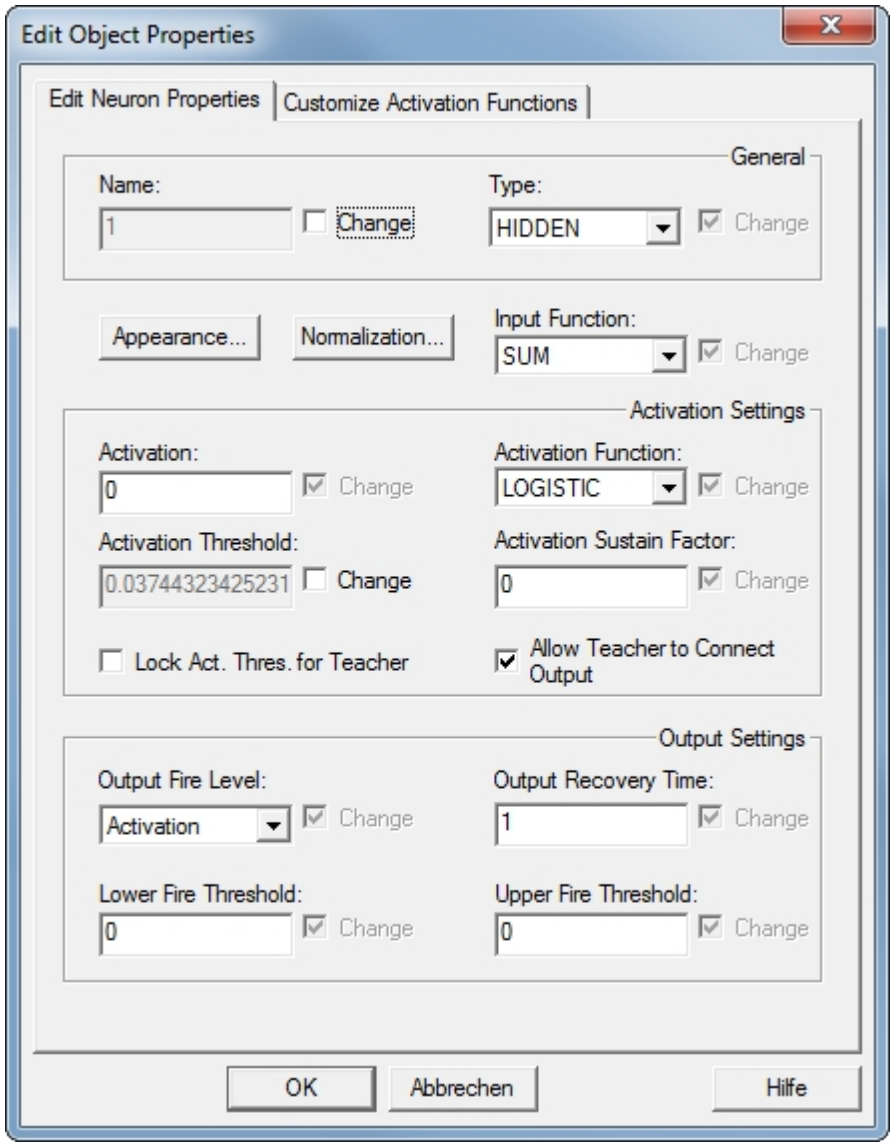


3. The Appearance properties are set as following:

Normalization is as following:

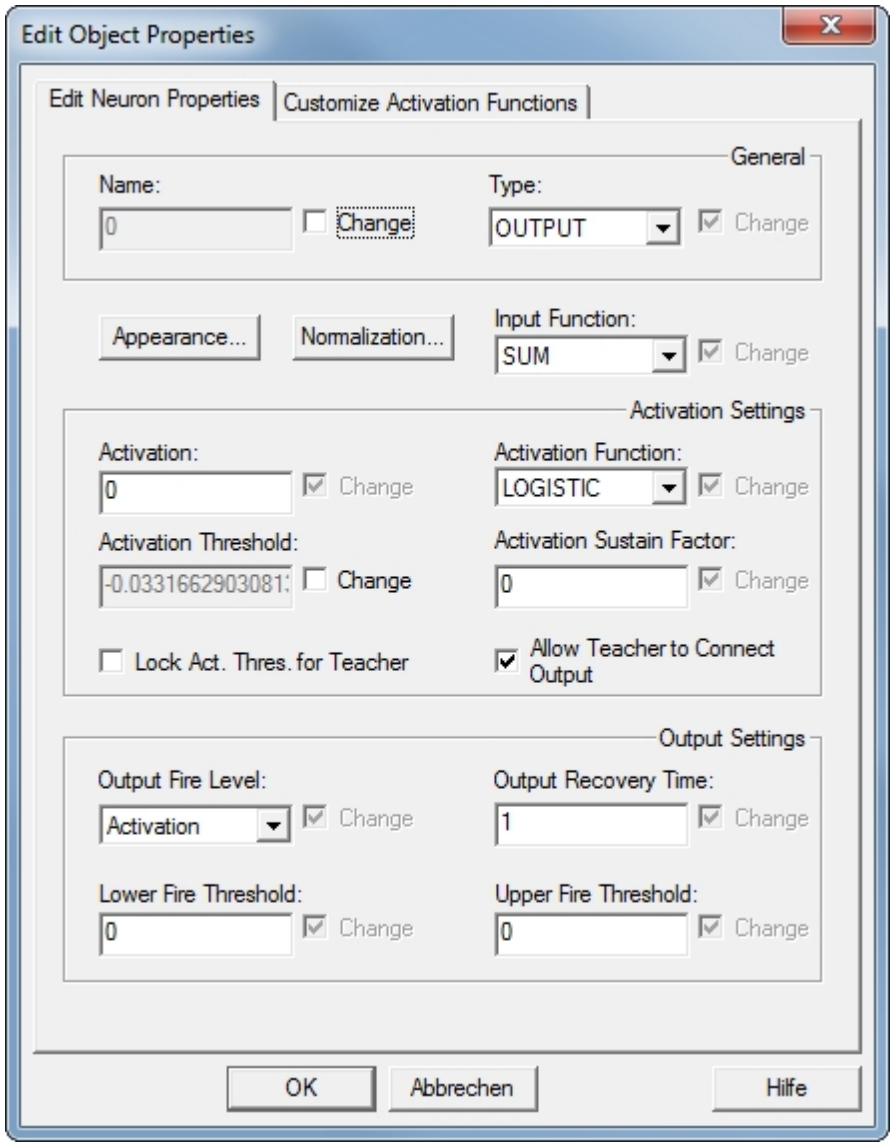


4. Group "**Autoenc Hid**"
   A hidden layer of 25 neurons with following properties.

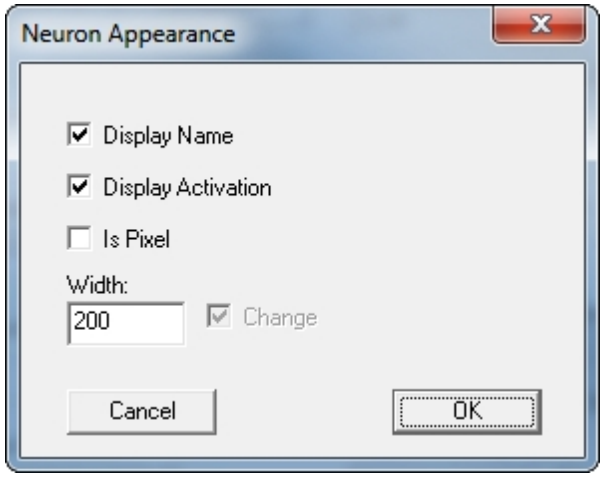


Normalization is not used here.

5. Group "**Outputs**"
   Contains all output neurons with the following properties:

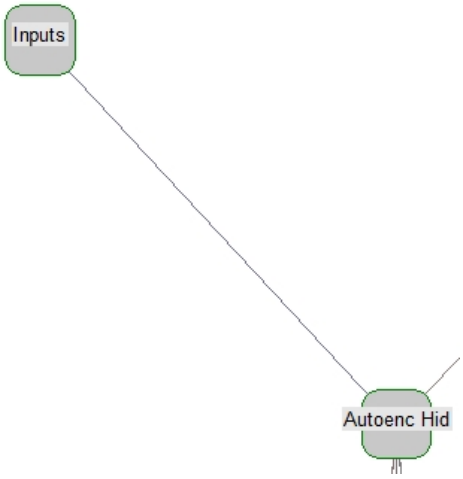


Normalization is not used here. Appearance is set as following:

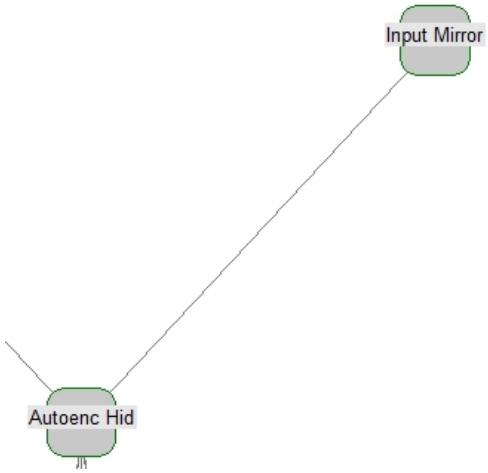


The neuron layers are fully connected as following:

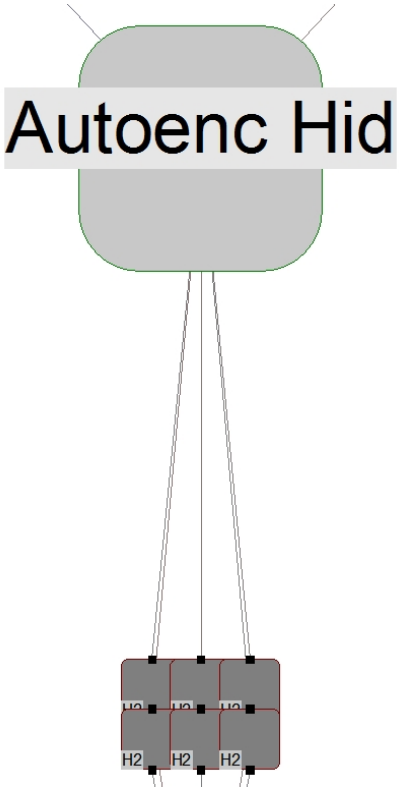
1. Inputs --> Autoenc Hid



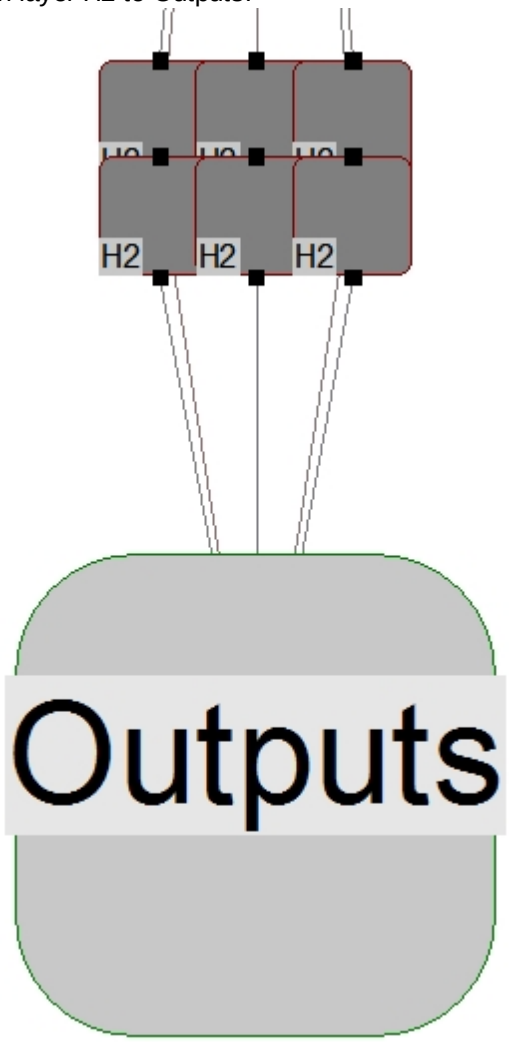
2. Autoenc Hid --> Input Mirror



3. Autoenc Hid --> 6 ungrouped neurons forming hidden layer H2 (enable <View><Show Layer Info> to display the layer information):



4. Hidden layer H2 to Outputs:



The grid width has been adjusted to 40 via the menu option <View><Set Grid Width>. The black background option has been disabled <View><Black Background>)

The final version of the autoencoder net built during this example can be found in file MNist_Autoencoder_Final.mbn.
If you want to skip the steps of building the net and defining the required group relations in MemBrain you can also load the net directly from file and jump to the section where the net is actually trained.

Continue with tutorial

Autoencoder net: Build the net

Autoencoder net: Build the net

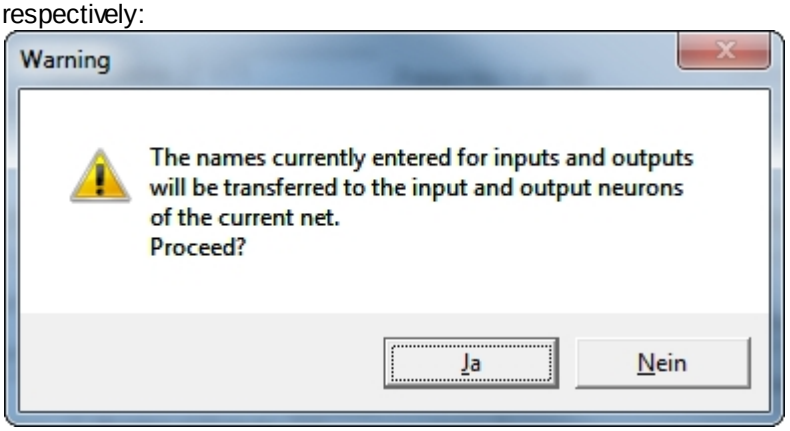
Neurons and interconnections

Building the net is traight forward: Place first neurons, then use copy/paste to quickly form the different areays of the net.
Select whole areas, press <ENTER> to access the properties. Adjust the neuron properties as provided in the overview section for this tutorial.

Once the neurons are placed, interconnect them using Selection in combination with *Extra Selection*. Establishing all interconnections is a metter of a few seconds with this approach. See overview section for more information on the required connection scheme.

Loading the lesson and adjusting the input and output names

Open the Lesson Editor and load the lesson file *MNIST_100_Patterns.mbl*
On the Lesson Editor click button <Names to Net> and confirm the following dialog with <Yes> or <Ja>, respectively:



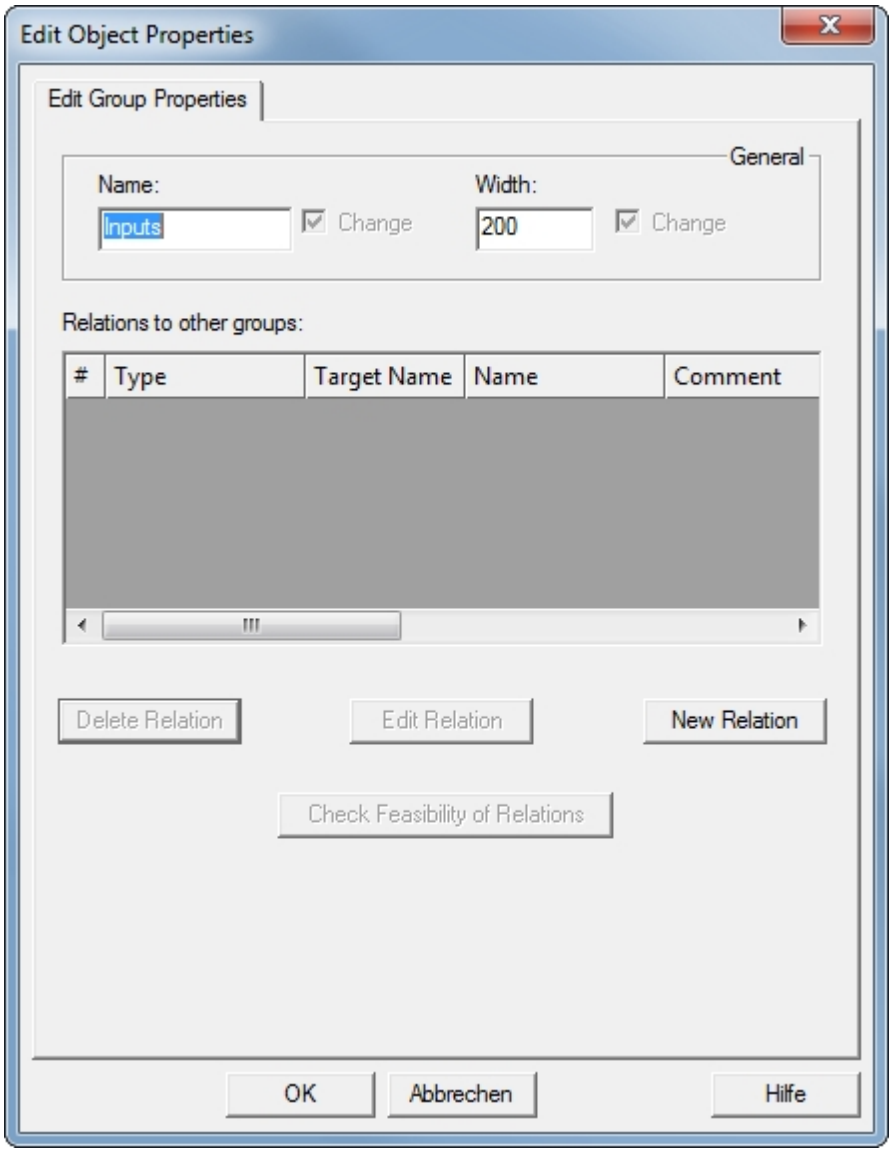
This will automatically adjust all input and output neuron names in sync with the just loaded lesson.

Grouping

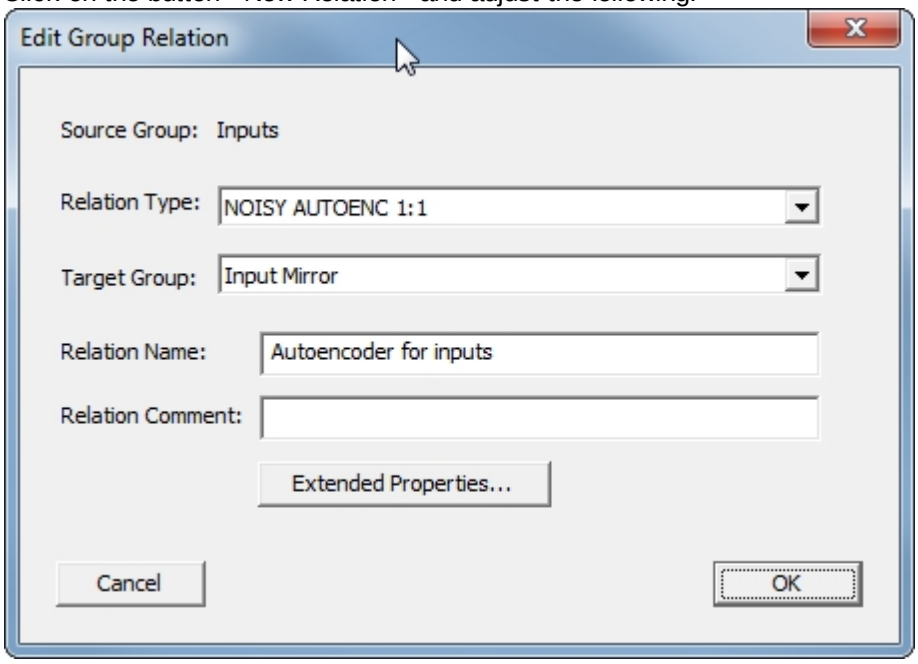
In order to define the Autoencoder portion of the net we need to establish the neuron groups as indicated in the overview section. In order to establish a certain group select all neurons for the group, then right-click on one of the selected neurons and select <Group Element(s)> from the context menu. All selected elements will be collapsed into a group. Double-click on the group and edit its name. See overview section for the names to enter for each group.
Note that you can also set the width for displaying each group when collapsed. This is a parameter of the group's properties. A suitable value is 200 for the width of the groups in this network.

Defining the Autoencoder relation

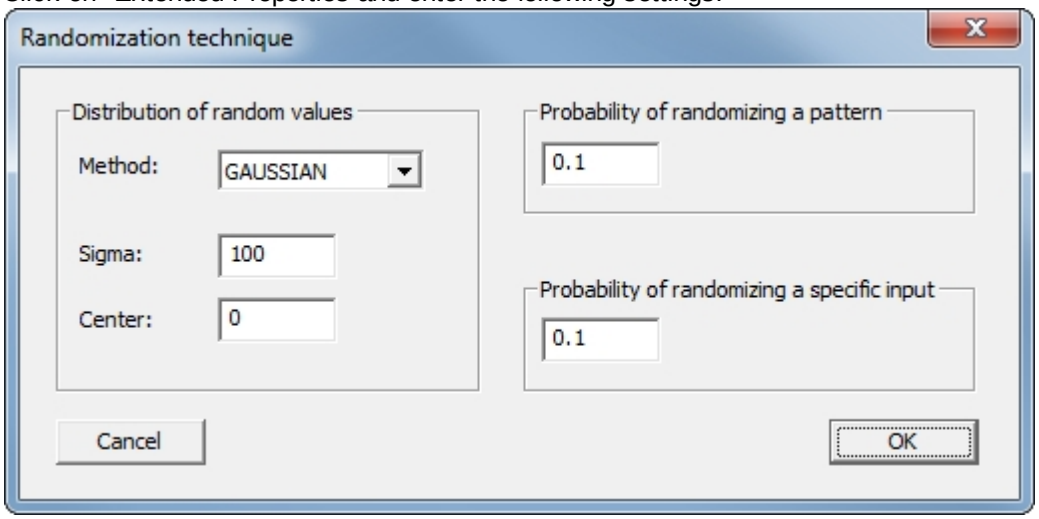
In order to correctly train the Autoencoder part of the net you need to tell MemBrain which portion of the net shall form the input and which portion shall be the output of the Autoencoder. This is done via group relations. A group relation logically ties two groups together in a specific context:
Edit the properties of the group "Inputs" (double-click on the group when it is collapsed or right-click on one of its neurons and select <Edit Owning Group(s)...> from the context menu.
The following dialog will appear:



Click on the button <New Relation> and adjust the following:



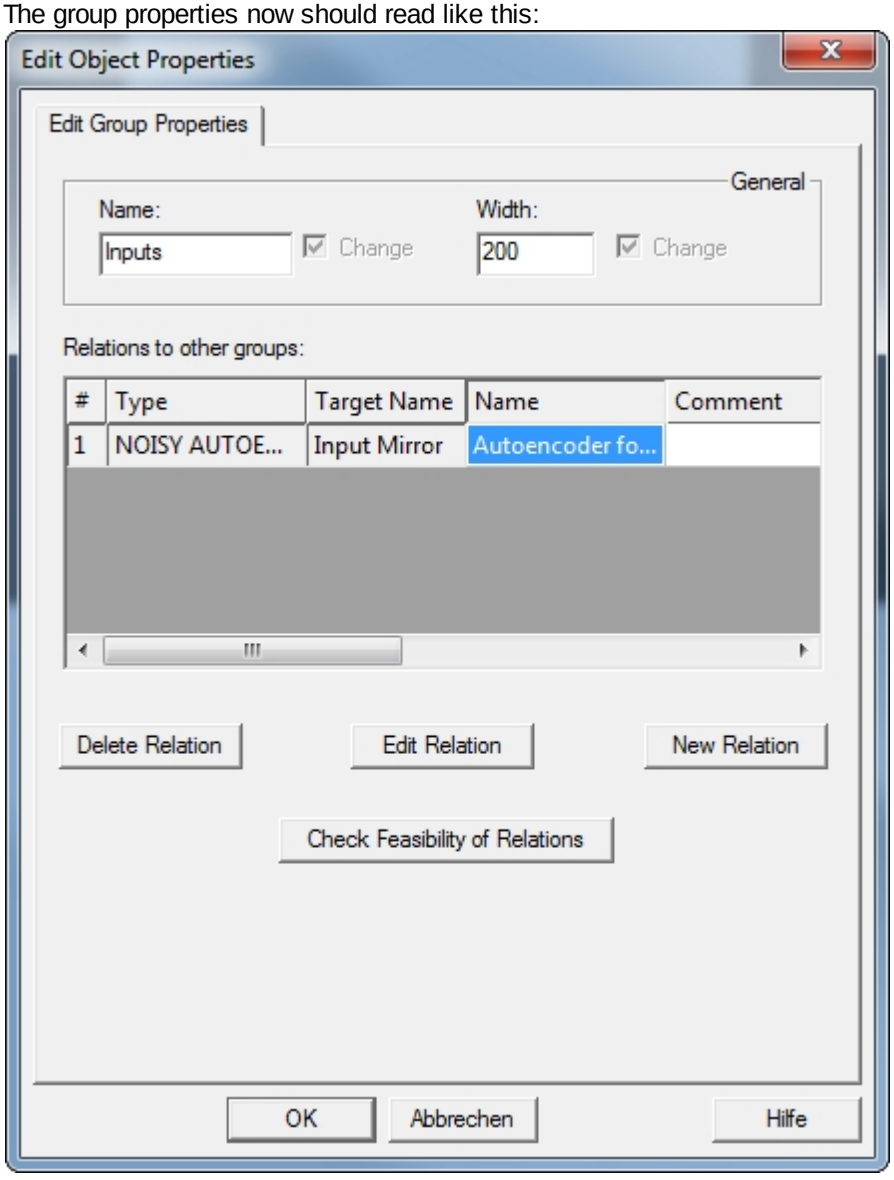
Click on "Extended Properties and enter the following settings:



Note: This will add random noise to the input data during training which generally leads to more robust training results.

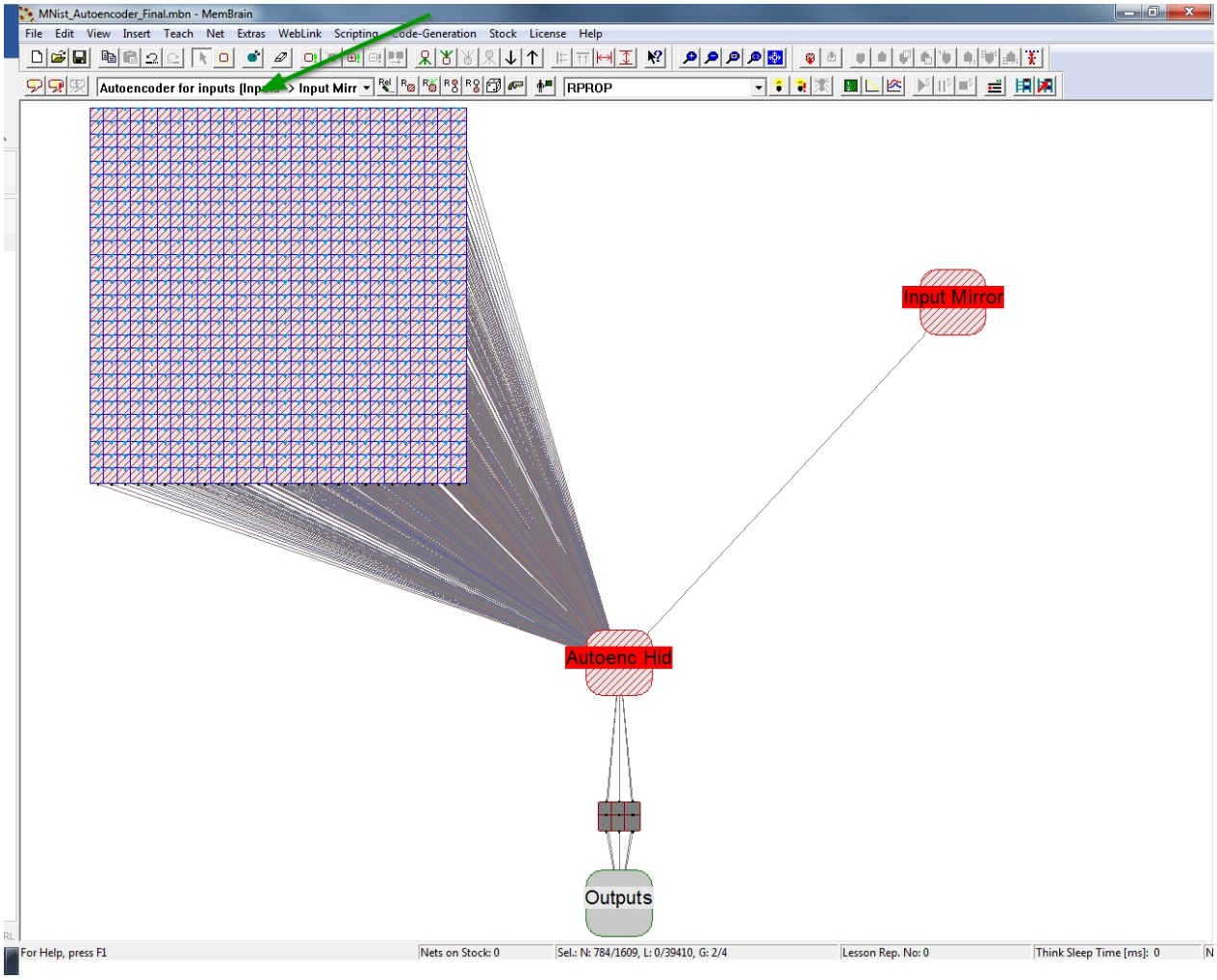
Click <OK> twice.

The group properties now should read like this:



Click <OK> again.

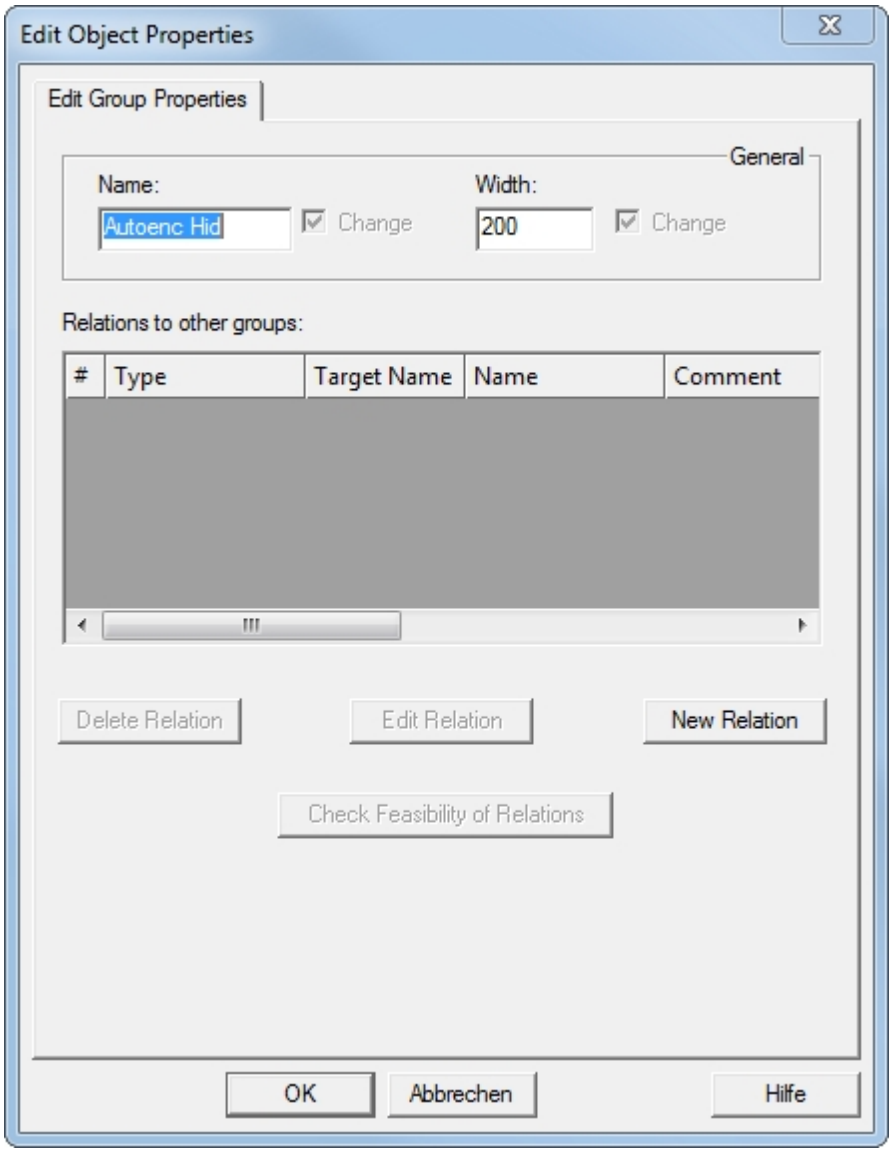
In the toolbar, select the newly created relation in the drop down menu. MemBrain now selects the elements which belong to the Autoencoder sub net defined by the relation:



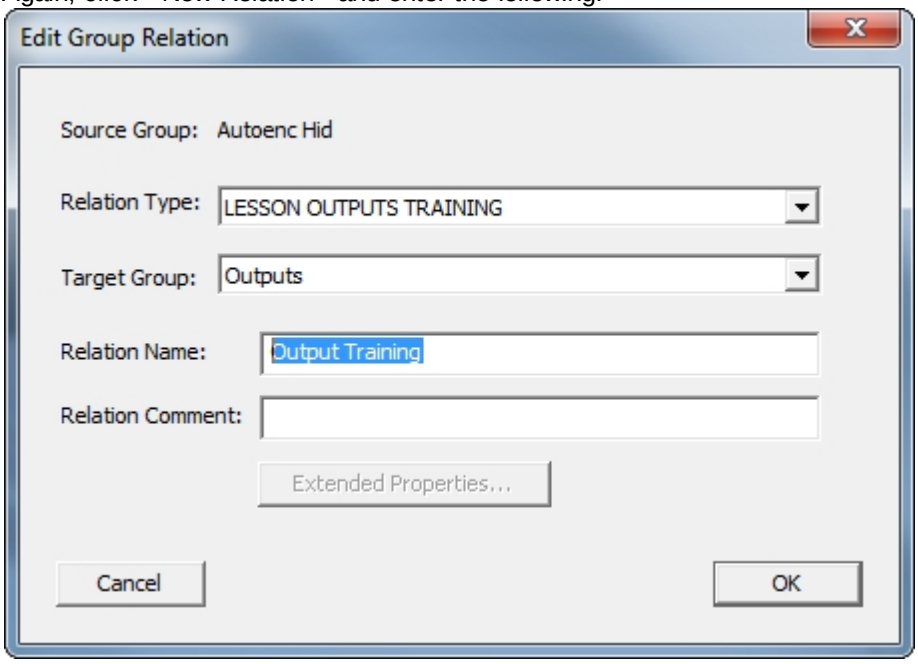
Note that your screen may deviate from the above dependiong on what groups are currently collapsed/uncollapsed in your setting. Also note that MemBrain automatically selects the group "Autoenc Hid" (or all neurons of the group, if uncollapsed), although the defined relation only gows from the group "Inputs" to the group "Input Mirror". This is because MemBrain automatically detects that the connection paths inside the net include the neurons in group "Autoenc Hid".

Defining the output training relation

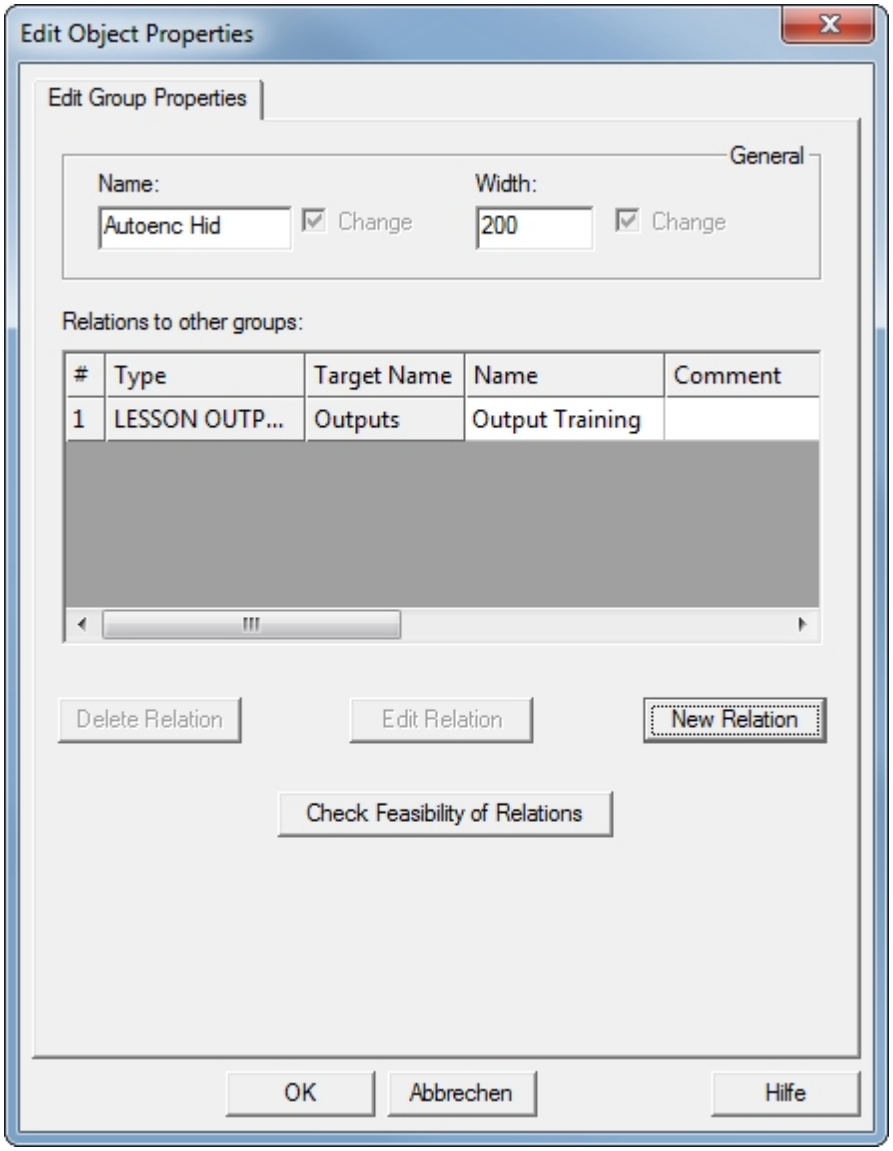
In order to make use of our newly created Autoencoder we later on want to train the outputs of the net from the hidden layer of the Autoencoder, which is contained in the group "Autoenc Hid".
Thus, we edit the group "Autoenc Hid" now:



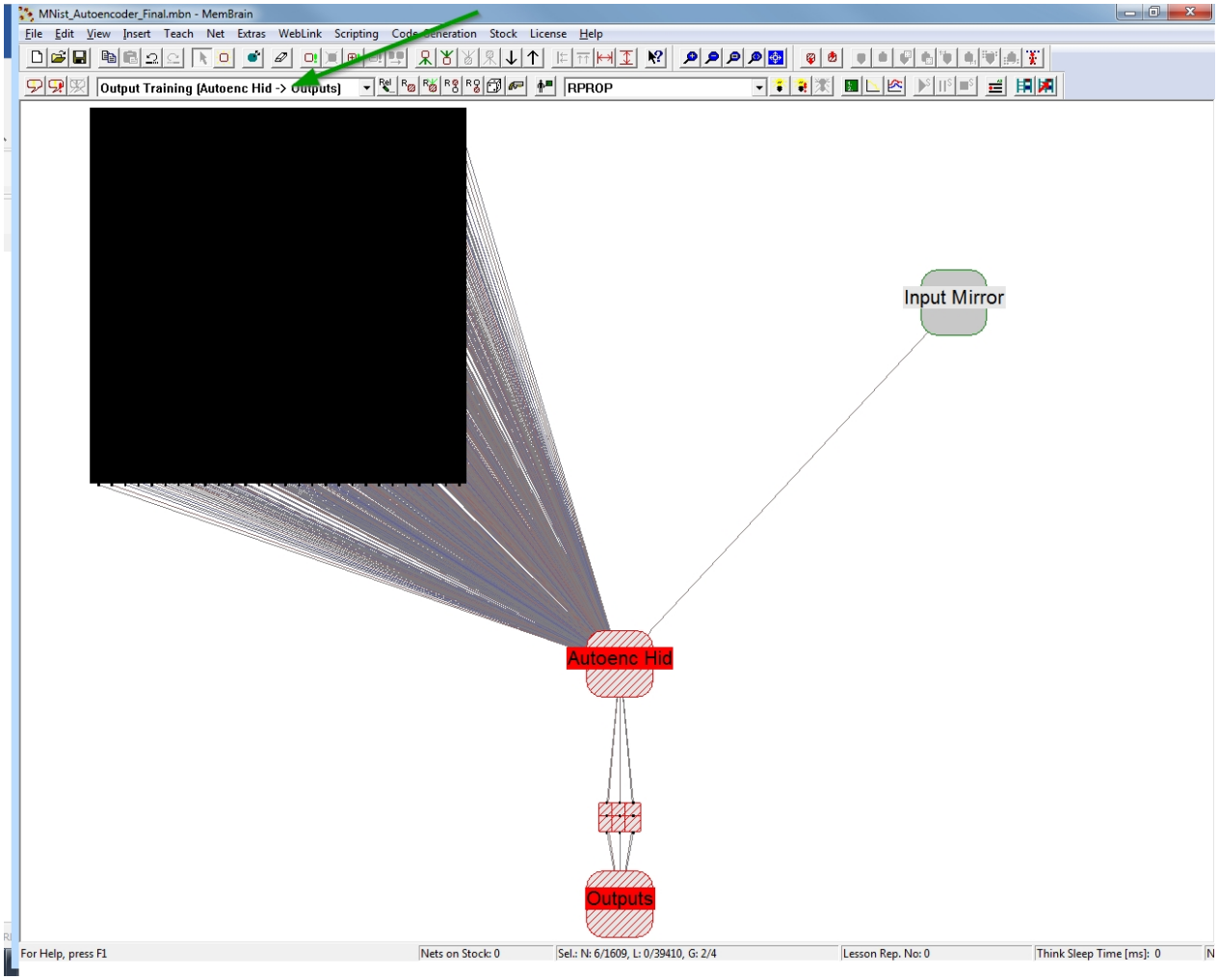
Again, click <New Relation> and enter the following:



Click <OK>:



Click <OK>.
Select the newly created relation in the drop down menu of the toolbar:



As you can see MemBrain now selects the sub net created by the just defined output training relation. This relation defines that the output section of the net can be trained from the information represented in the group "Autoenc Hid".

Now it's time to **pre-train the Autoencoder** sub net.

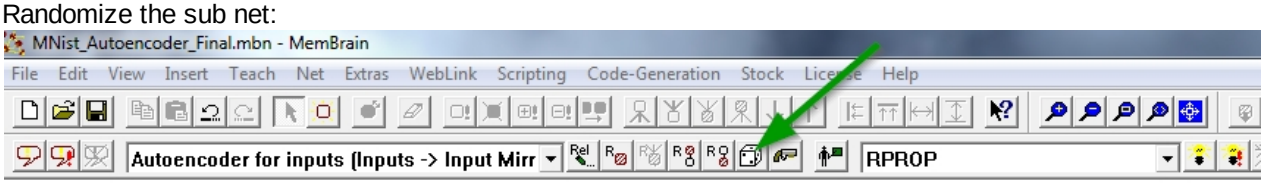
Autoencoder net: Pretraining

Autoencoder net: Section template

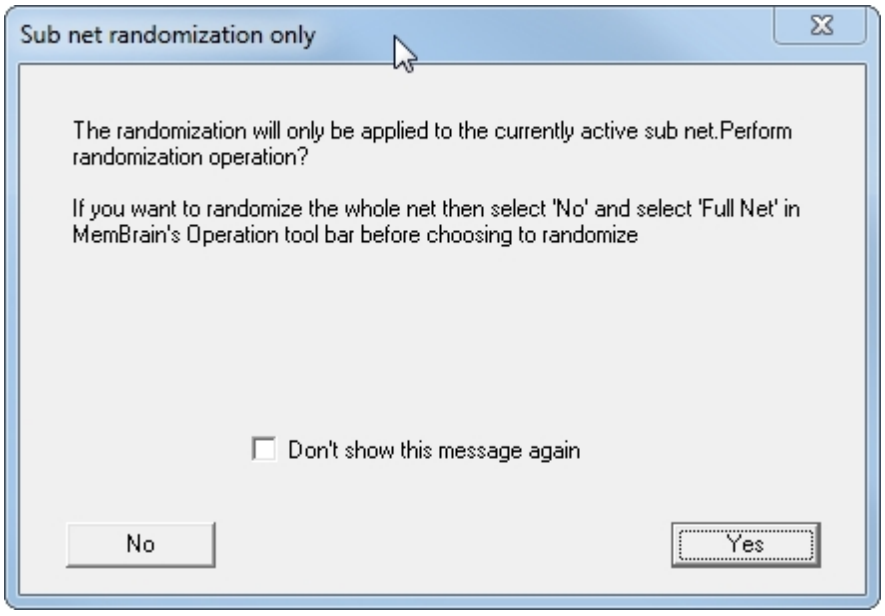
Autoencoder Pretraining

If not yet happened, open the Lesson Editor and load the lesson file *MNIST_100_Patterns.mbl.*
Select the Autoencoder sub net in MemBrain's toolbar. Uncollapse all groups.
Deselect the option <View><Show Links> in order to increase performance for the now following Autoencoder pretraining.

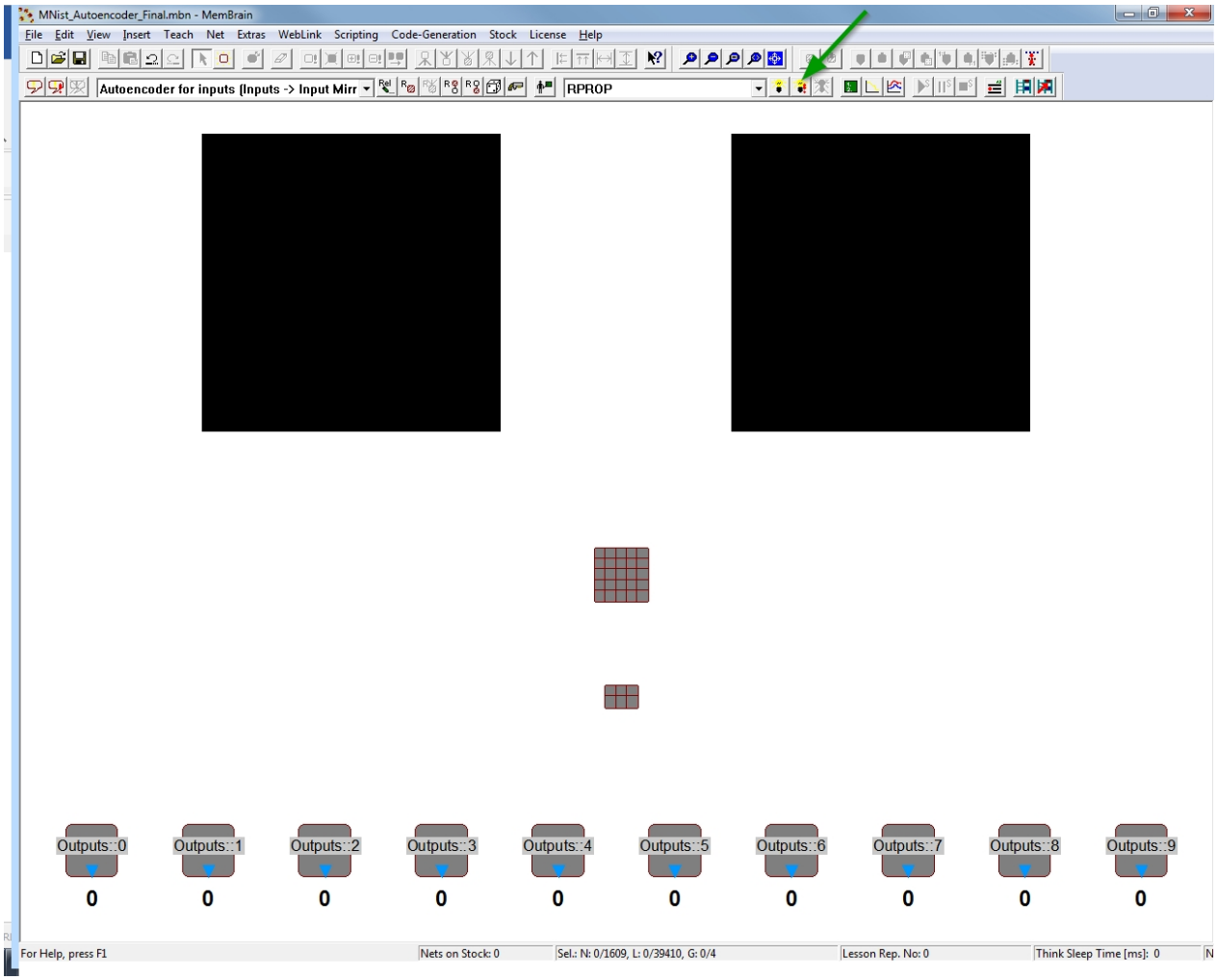
Randomize the sub net:



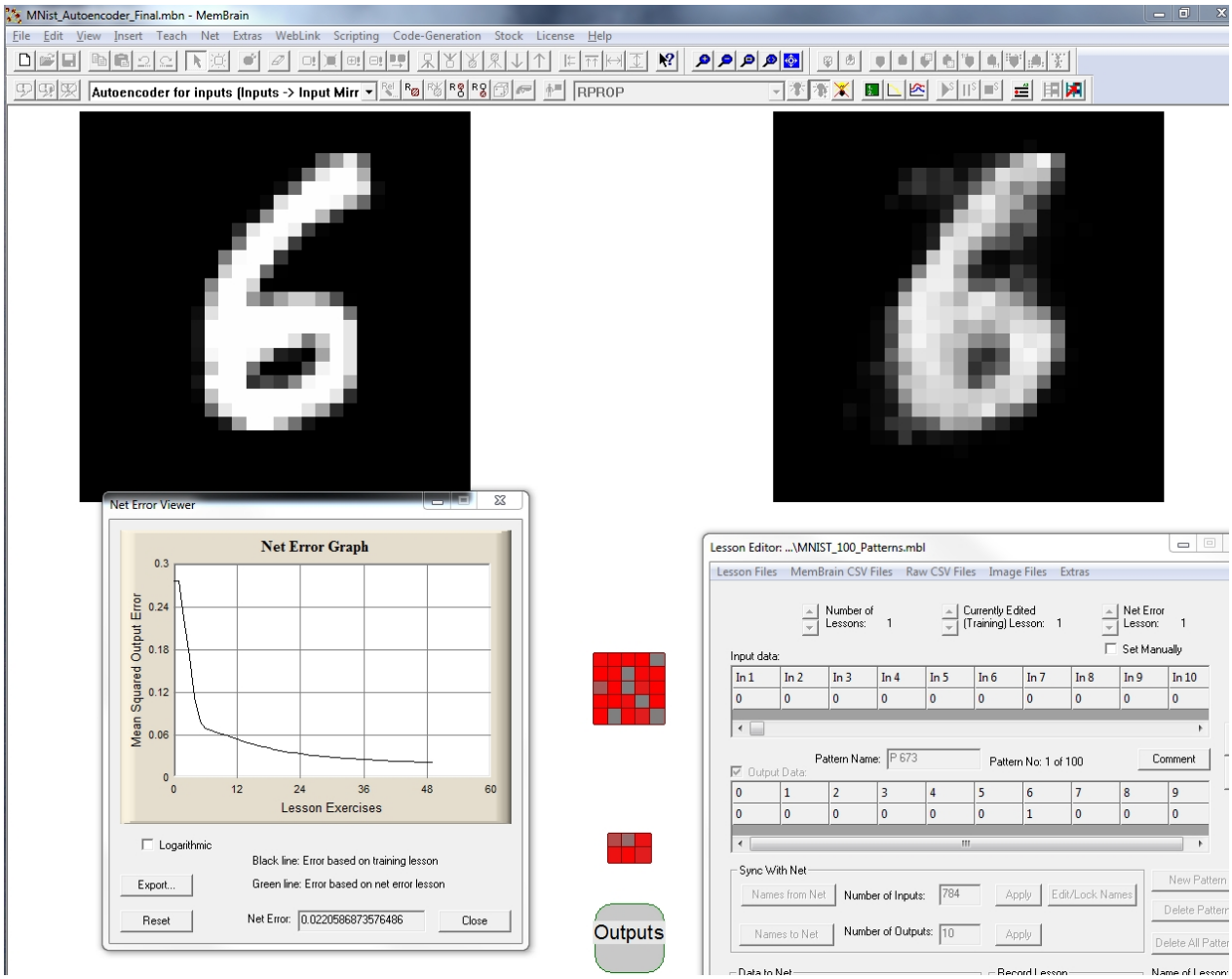
Depending on your settings the following dialog may pop up. Confirm with <Yes>:



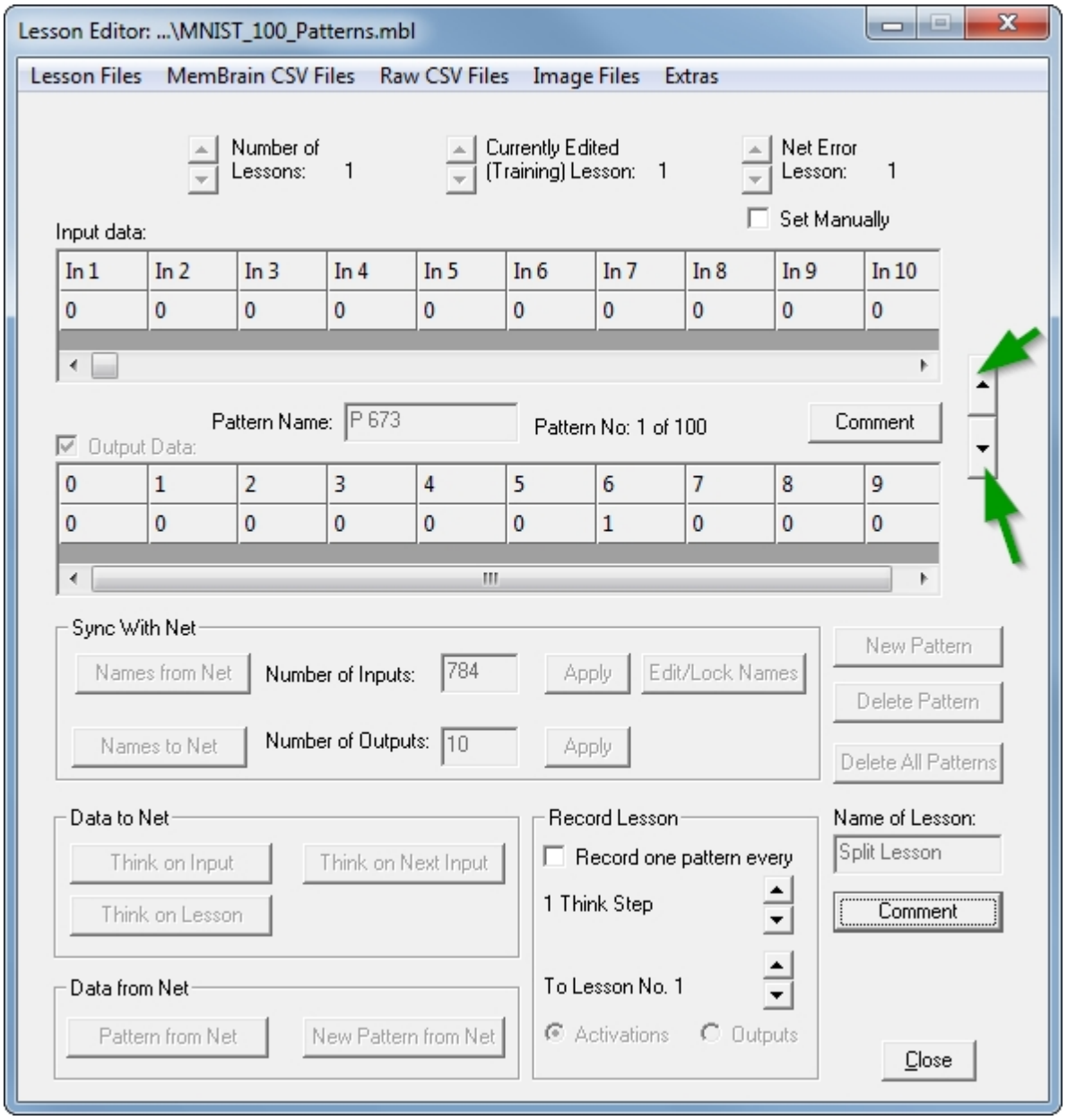
Your screen should now look like this. Ensure that the training algorithm ("Teacher") *RPROP* is selected. Start the pretraining of the selected sub net by clicking on the icon indicated by the green arrow below. You may also decide to display the Net Error Viewer to observe the Net Error during training.



The Autoencode pretraining has started, the net error decreases and the neuron group "Input Mirror" starts to form images like the ones in the group "Inputs":



Note that you can switch the visible pattern during training using the Up/Down buttons on the Lesson Editor:



Stop the training once you feel that the net error doesn't decrease anymore. Save your net.

Note1:
During training you will see some noise in the input pixel data every now and then. This is intended and is related to the Noisy Autoencoder type we chose during setting up the net.

Note2:
Although the loaded lesson has output data defined (digits 0 .. 9 as binary flags) this would not be necessary for the actual Autoencoder pre-training. If you like to try this out you may delete all output data columns in the Lesson Editor (i.e. set the number of outputs to 0). The pre-training still will work.
This is s big advantage of autorencoders: They can work with unlabelled data, i.e. data where no target output values have been defined. This allows to use a large number of data sets for Autoencoder pretraining and only use a smaller amount of data sets for the lesson output training where labelled data are required.

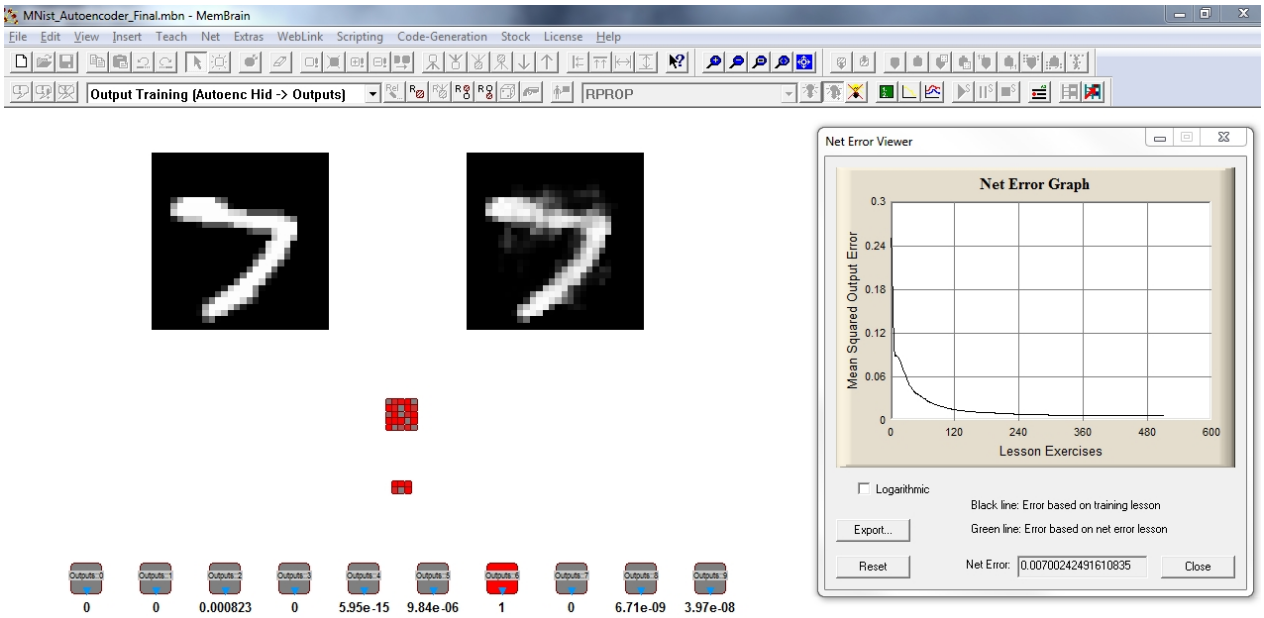
Next step: **Lesson output training**.

Autoencoder net: Lesson output training

Autoencoder net: Lesson output training

Train the net outputs on the output data of the lesson

Now that the Autoencoder portion of the net is pretrained we can use the compressed information held in the Autoencoder's hidden layer to train the output neurons of the net:
Select the "Output Training" relation in MemBrains toolbar, randomize the sub net and start the training.
You should see something like this:



Note that the images in the corresponding neuron groups won't update during the training since MemBrain uses an optimized approach here to increase performance:
All patterns from the lesson are applied only once to the input neurons and the resulting pattern in the Autoencoder's hidden layer neurons is captured. During all following lesson exercises MemBrain applies the captured activations to the hidden layer of the Autoencoder directly and then only calculates the activations of the neurons that influence the output layer downwards from the hidden Autoencoder layer. This massively increases training speed during lesson output training as you probably have noticed.

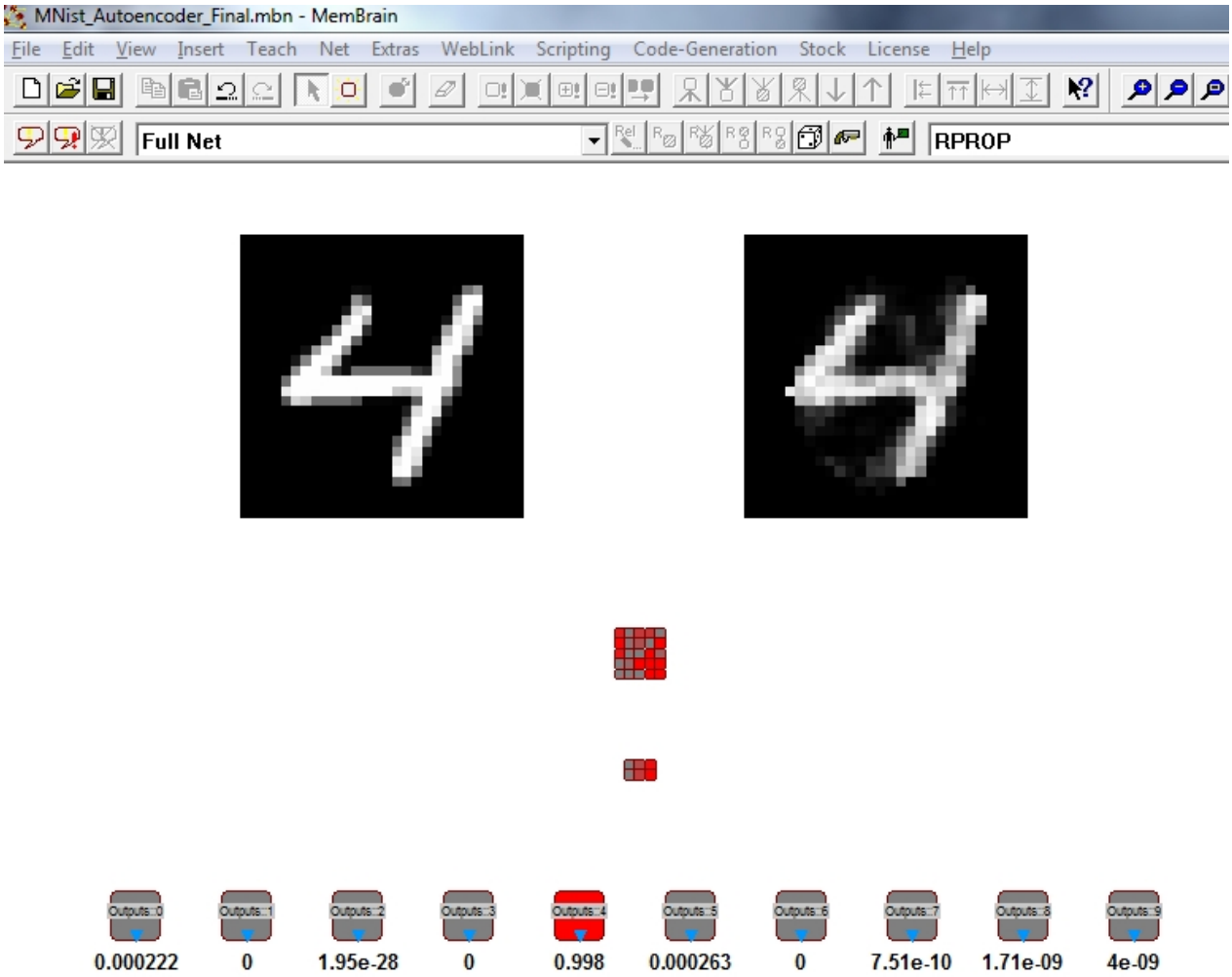
See what it has learned

Autoencoder net: See what it has learned

Autoencoder net: See what it has learned

Apply patterns to the full net, updating all neurons

Both the Autoencode sub net and the lesson output data sub net have been trained. Now it's time to see what the overall performance of the net looks like:
Select the entry "Full Net" in MemBrain's drop down menu.
On the Lesson Editor click on the button <Think on Next Input> over and over. With every click the next pattern from the lesson is applied to the net ans all resulting neuron activations are calculated, including the output. A detected digit is represented by an acvtivation close to the value "1" of the corresponding output neuron. The following screen shot shows the reaction on a digit "4". As you can see, the output neuron named "4" is activated.



<End of tutorial>

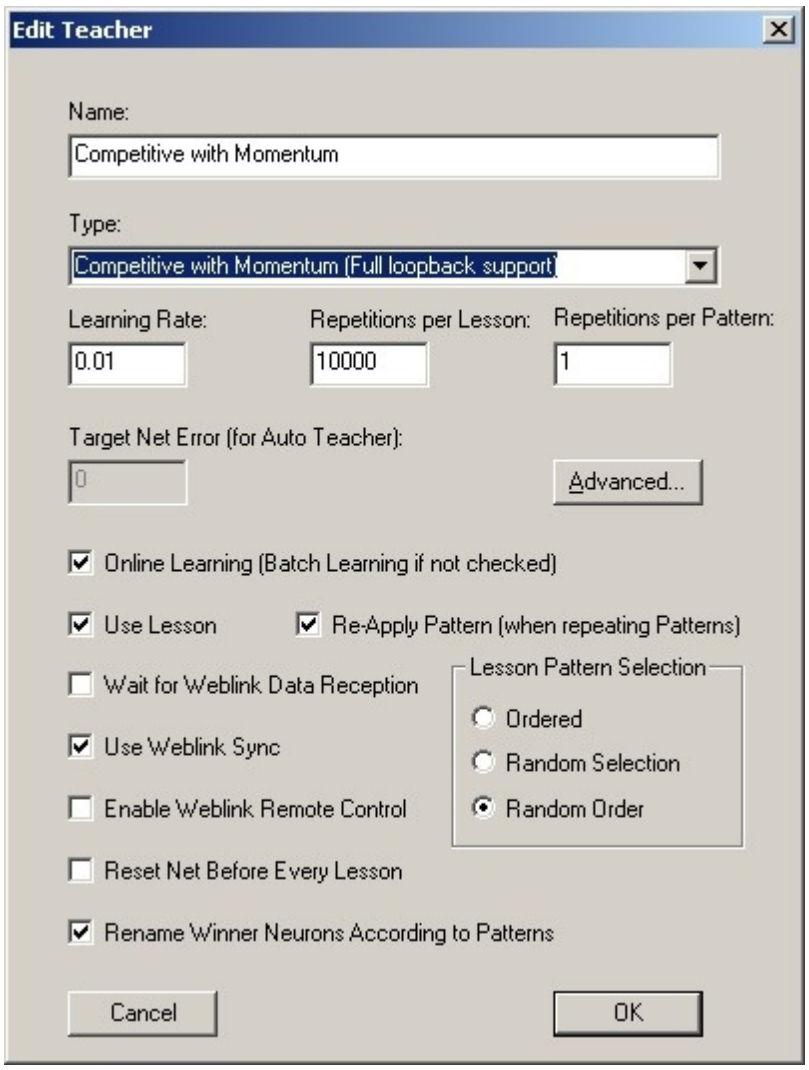
Unsupervised Learning

Unsupervised Learning

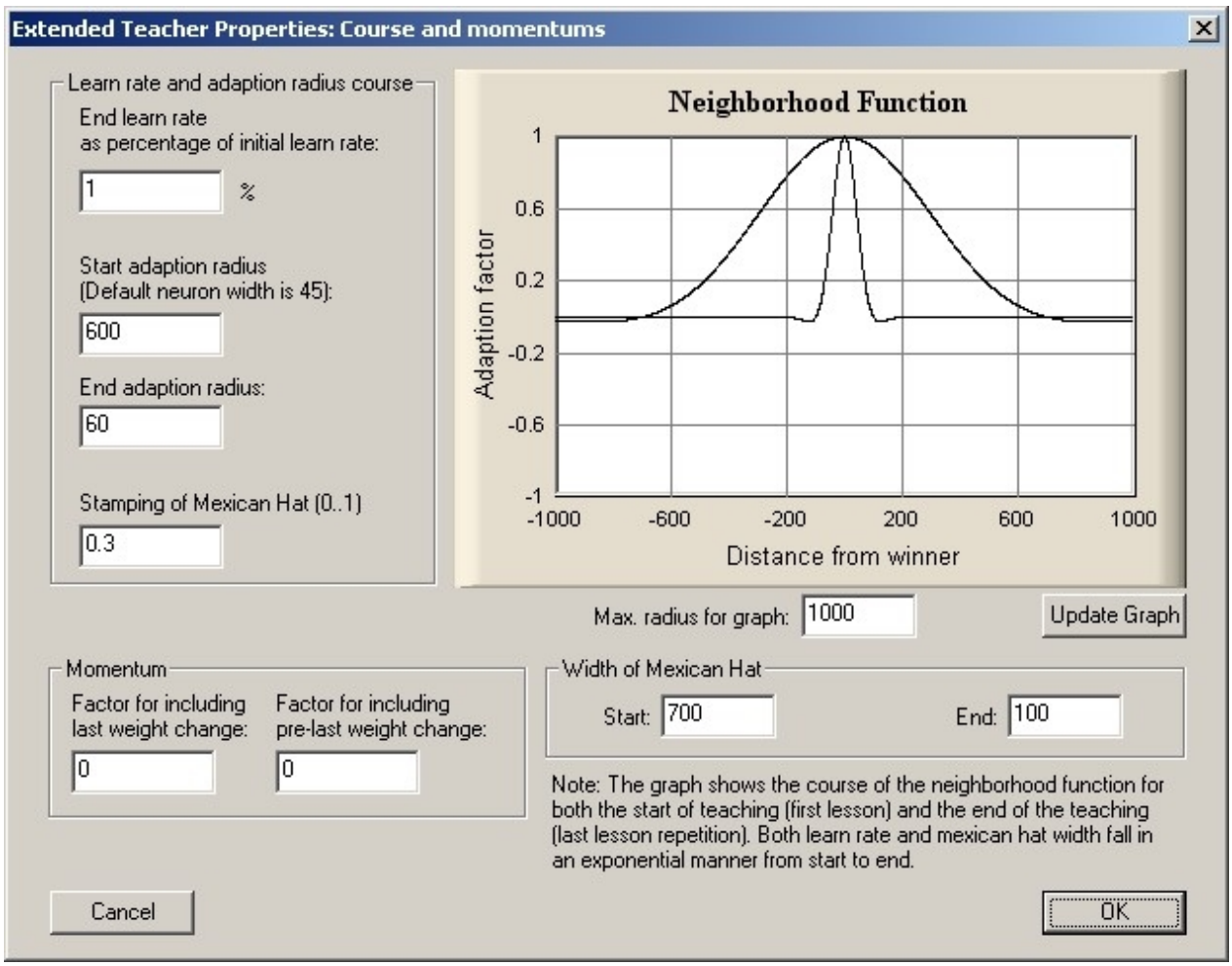
File names:
SOM.mbn and SOM.mbl

This example demonstrates a Self Organizing Map (SOM) of 10 x 10 output neurons and 2 input neurons.

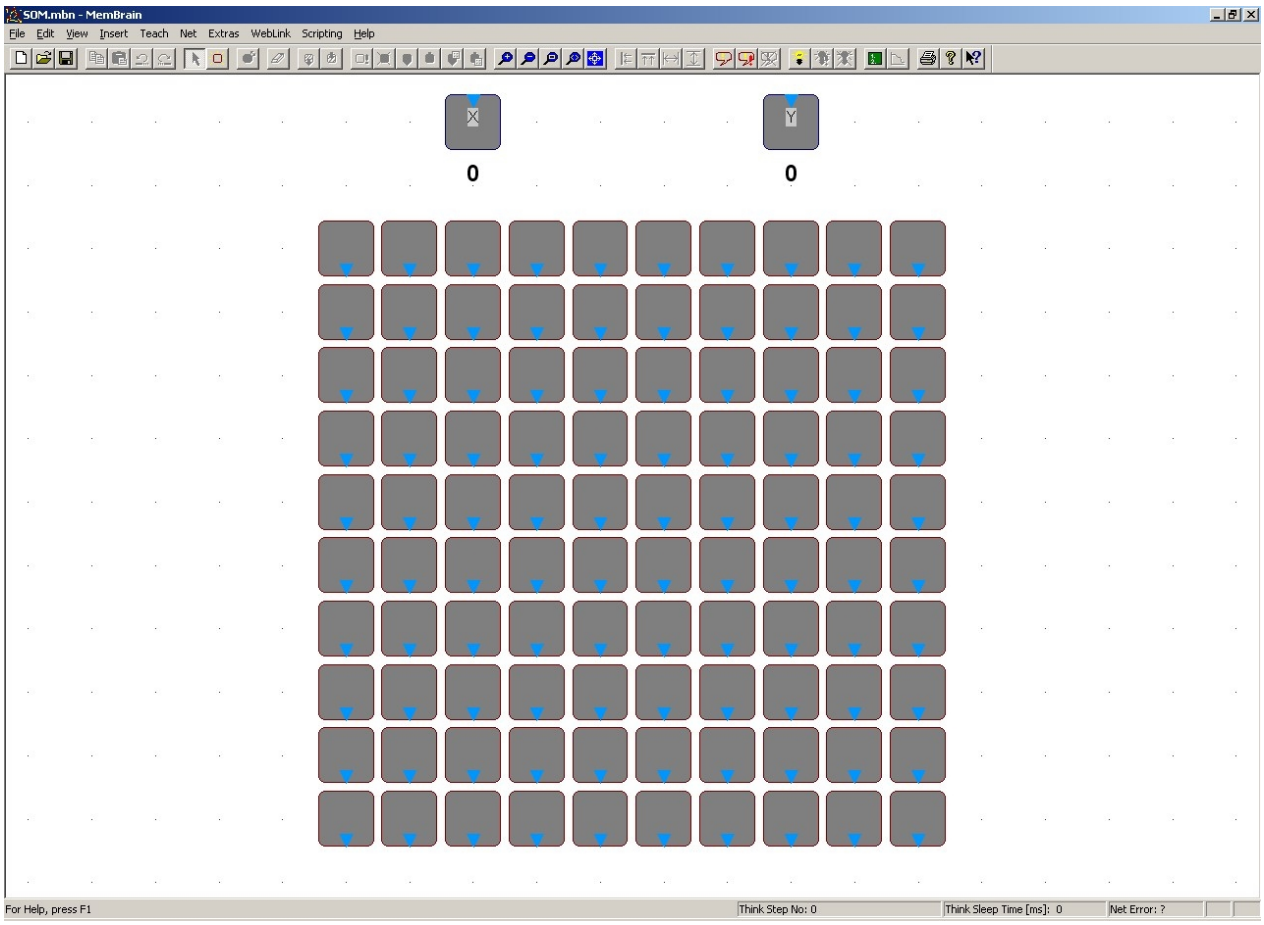
For unsupervised training the teacher 'Competitive with Momentum' is used. The following teacher settings can be used for the example:



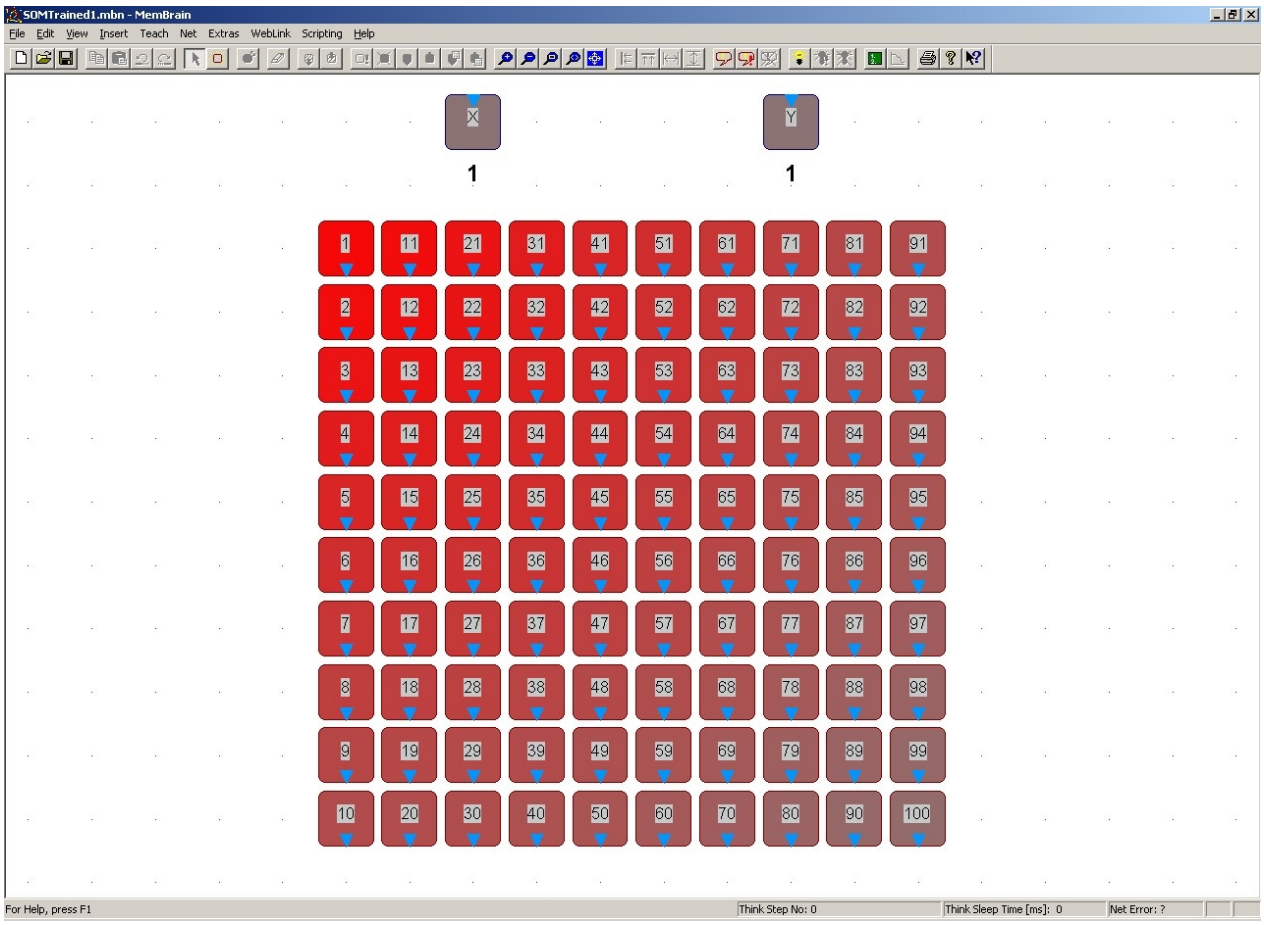
If you click on the button 'Advanced':



If you load the SOM example (SOM.mbn) it will look something like this:



Load the lesson SOM.mbl into the Lesson Editor and start the teacher. Ensure that the setting <View><Update
View during Teach> is activated.
You will then see how the patterns of the current lesson (named from "1" to "100") arrange on the SOM and find
locations in a way so that similar patterns locate close together in the SOM. Finally after the training the SOM
should look something like this:



Note that the ouput neurons of the SOM have been named according to the pattern for which they are the winner
neuron, i.e. the top left neuron has been named as '1' because this is the winner neuron for the pattern with name
'1' in the Lesson.
The lesson consists of patterns with X- and Y- input values ranging from 1 to 10. This correlates to rows and columns
in the SOM. The patterns are named in the Lesson from 1 to 100.

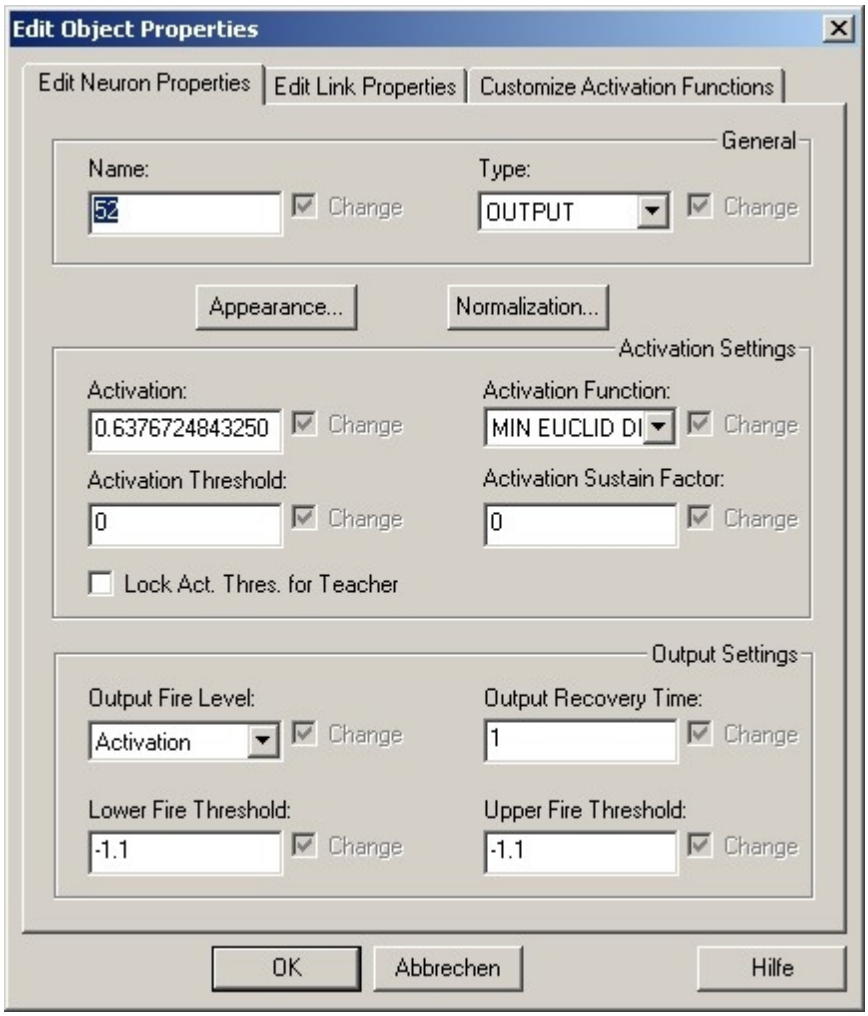
Since similar patterns are grouped together in a SOM after the training the pattern names with same X- resp. X-coordinate
value are located in the same row or column. Depending on the training run rows and columns may be flipped i.e. differ
from the above picture. This actually depends on the randomization start values for the weights of the links and also on
the random order of the patterns chosen during training.

Now check the menu option <View><Show Winner Neuron>, open the lesson editor and click on the button <Think on Next Input> several times. You will see that the winner neuron of the corresponding input pattern is visualized in the SOM by a blue cross:

You can use this function if you want to quickly identify the winner neuron of a SOM when applying new (untrained)
patterns. The location of the winner neuron in relation to the winner neurons named through the training will

indicate
similarity between the new input pattern and the already trained patterns.

Note that the activation function for the SOM output neurons is 'MIN_EUCLID_DIST' which means 'Minimum Euclidian
Distance'. You can see this if you double click on one of the output neurons:



If you want to learns more about this activation function then check out MemBrain's help file.
What you should keep in mind is that you will need this activation function for the output neurons
of a SOM in order to obtain proper results.

Experimental (Chaotic Net)

Experimental (Chaotic Net)

File:
ActivationSpikesDemo.mbn

The example 'ActivationSpikesDemo.mbn' is of no direct use but it demonstrates
the dynamic neuron model used by MemBrain and looks quite fancy.
In order to have this net displayed the best you should adjust MemBrain in
the following way:

Menu <View>:

<Show Links> : Off (unchecked)

<Show Activation Spikes On Links>: On (checked)
<Show Fire Indicators>: On
<Draw Links in Foreground>: Off
<Show Layer Info>: Off
<Use Display Cache>: On
<Black Background (...)>: Off
<Show Grid>: Off

Menü <Net>

<Set Simulation Speed...>: Adjust 1ms

Then choose **<Net><Start Thinking (Auto)>** or click on the toolbar symbol

to start the simulation.

Note that all options to navigate in the drawing area can still be performed while the simulation is running. You can even move and edit neurons during the simulation.

Using <CTRL> + <R> you can reduce the simulation speed, <CTRL> + <I> increases the speed.

Feedback and Contact

Feedback and Contact

If you have bug findings, questions, suggestions for new releases or any other feedback with regard to MemBrain or the provided examples then please feel free to visit the MemBrain homepage at

www.membrain-nn.de

or contact me directly at

Thomas.Jetter@membrain-nn.de

Any feedback is highly appreciated, whether in english or in german language!

Please use the word "MemBrain" in the subject of your mail.

Thanks in advance to all who help improving MemBrain this way!