Classification of faces with an artificial neural network

Prerequisites:

What do you need?

- **MemBrain** (Windows-only, free at membrain-nn.de)
- Text editor like **Notepad++** (to create simple text files)
- **Gimp** (open-source at gimp.org)
- **Jaffe-dataset** (kasrl.org/jaffe.html)

The following presentation is limited to the modeling of a neural network for face classification. It makes sense to introduce the topic beforehand and to have already worked with MemBrain.

Nevertheless, it is possible to work through this teacher manual step by step without any previous knowledge.

In German a video explains this tutorial:
https://youtu.be/KzdVPi4RNMg

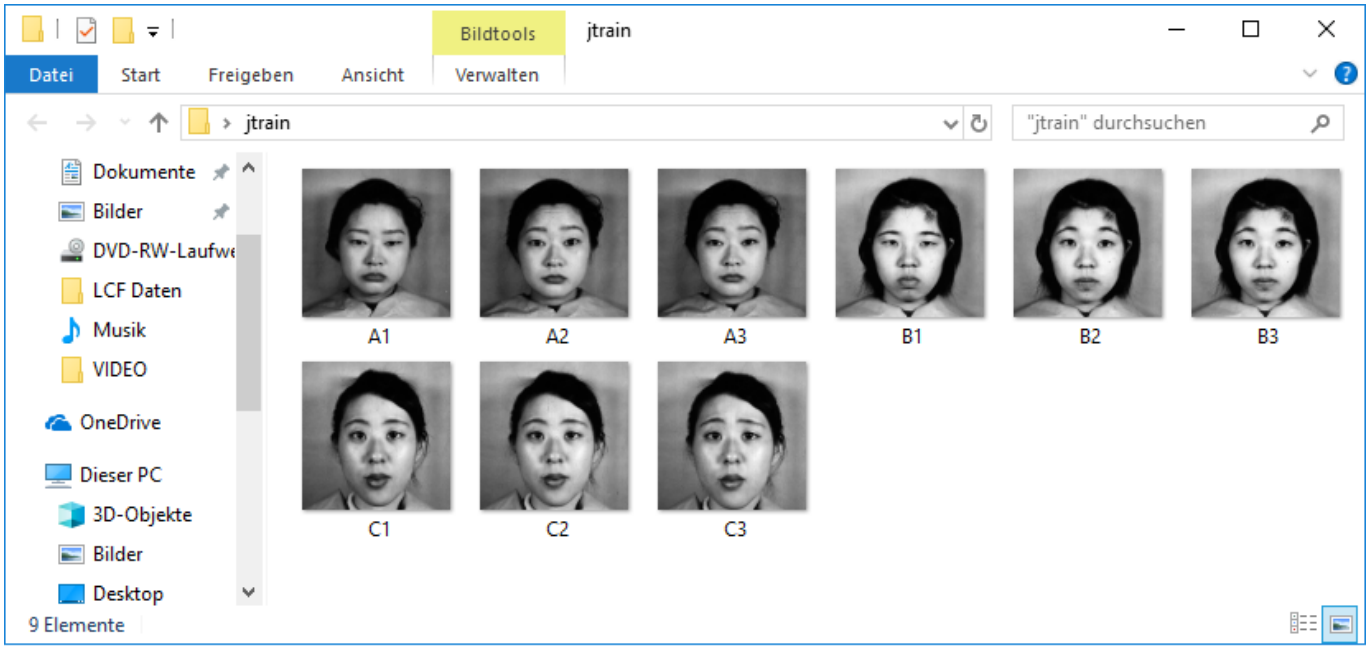
Face recognition:

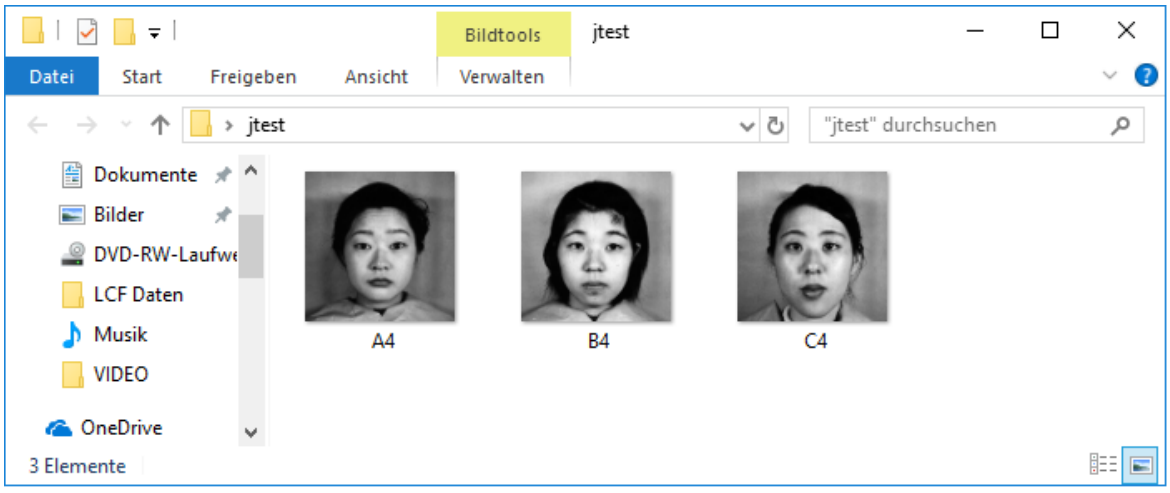
Summary:

- **Teacher distributes pictures to student groups**
- **Students develop face model**
- **Students collect the face data for the CSV training file with GIMP**
- **Students create in MemBrain the Artificial Neural Network**
- **Students load the CSV training file and train the network**
- **Students collect the facial data for the CSV test file with GIMP**
- **Students load the CSV test file and test the network**

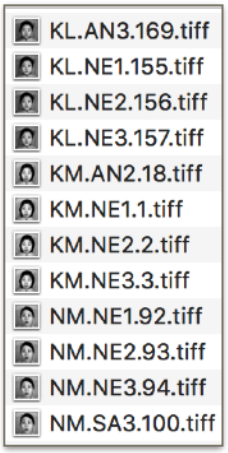
Preparation

The teacher distributes one folder with training data and one folder with test data from the JAFFE data-set per student group. For example, the training data could consist of three pictures of three persons each, i.e. a total of nine pictures. The test data could, for example, consist of one picture each of three people, i.e. a total of three pictures. The following illustrations visualize this for the Training Data folder and the Test Data folder:





Tip: For the gain of knowledge this limitation to a few pictures is probably sufficient. The pictures of the adjoining list led to good results. However, you can also increase the amount of data without any problems by giving each group its own (i.e. other 12) pictures and creating a common training and test file from as many different people as possible in addition to the group's own training and test tables. Here, however, one has to be careful later when coding the output, so that one really works with e.g. 8 classes for 8 different persons. An approach to this is presented below as an alternative.

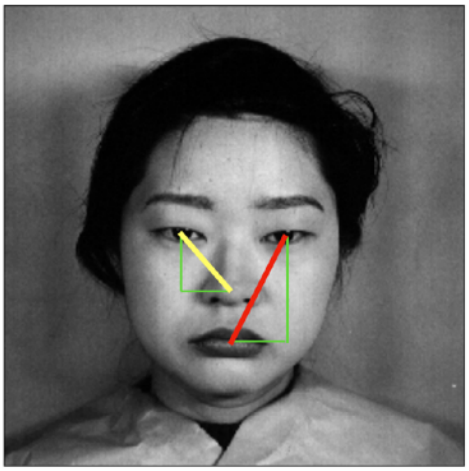


Face model

For face recognition, a face model must first be developed. For this purpose, the students are to develop their own face model (with only four features) on the basis of the photos from the JAFFE data set. The photos are normalized to the interpupillary distance and all taken from the same perspective, which facilitates comparability. In principle, the SuS can also use their own photos, but it is recommended to use the above data set.

For example, a possible facial model could be to choose the following four characteristics:

- distance right pupil to tip of nose (in pixels)
- Angle right pupil to tip of nose (in °)
- distance left pupil to middle of mouth (in pixels)
- Angle left pupil to middle of mouth (in °)



Tip: It does not make sense to take the interpupillary distance, since all images of the data set are normalized to it.

Here, too, you could work with more features like in 'real' applications for face classification, the limitation to four features only serves to save time. The principle remains the same.

Data collection in GIMP

Students open GIMP and proceed as follows for all photos from their training folder:

They open a photo with GIMP (other programs may also be suitable) by dragging and dropping it from the folder into the program or by opening it via the File menu. Then they zoom into the image so that it can be seen as large as possible on the screen. Now they can easily measure distances (in pixels) and angles (in °) with the tool Scale Tape. Select the tape measure tool and click on the two points to be tracked. GIMP displays the desired information in a small info field, which the students can then simply write into a text file. How this works most reliably is described below.

Creation of a CSV file suitable for MemBrain with the training data

It is possible to write the values that the students collect in GIMP directly into an Excel spreadsheet and save them later as a CSV file. The best way, however, is to choose the most crisis-proof way and let the students create a pure text file with the program Notepad or Notepad++ or similar, which is saved under the file extension .csv (Important: not .txt). The students now write exactly one line into this file for each processed image, so that the result is a file that is structured like the following example:

```
M1;M2;M3;M4;Y
0.66;0.64;0.64;0.63;0
0.69;0.63;0.68;0.66;0
0.68;0.63;0.67;0.65;0
0.72;0.66;0.72;0.63;0.5
0.73;0.68;0.75;0.64;0.5
0.75;0.67;0.75;0.66;0.5
0.80;0.69;0.81;0.70;1
0.82;0.68;0.83;0.70;1
0.83;0.69;0.84;0.71;1
```

Tip: For practice purposes, copy the values shown and save them in a file, e.g. under the name train.csv.

The four characteristics are shown here (here with two decimal places), each followed by a semicolon. On the far right, the number of the person to whom the photo belongs is entered. At this point, however, three important agreements must be observed:

1. **The first line of the CSV file must contain the names of the neurons created later in MemBrain as column headers. It is recommended to simply name them M1, M2, M3 and M4. As 'last column header' the name of the output neuron must not be missing. It is best to use the name Y here.**

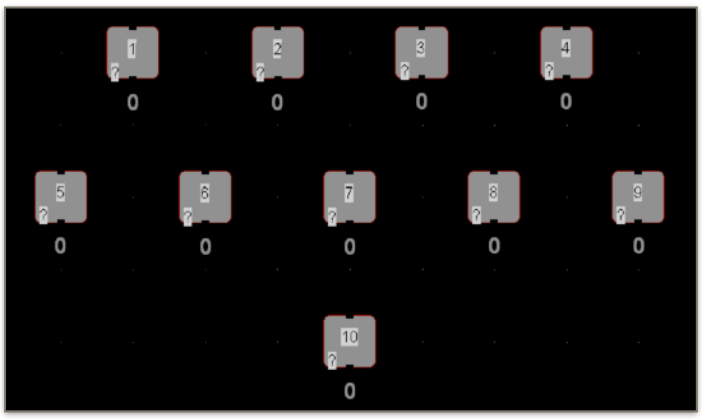
2. **The input range must be in the range [0..1]. Raw values must be normalized to the value range of the activation functions before processing with neural networks. In order not to have to deal with normalization methods for didactic reduction reasons, it is sufficient that the students simply write a 0 followed by a point in front of the value. If they have measured an 81 as value in GIMP, 0.81 is entered into the CSV file. This applies to the first four values in each line.**

3. **The last value in each line contains the class label. You must therefore code here which person the four characteristics are. Since the value range [0..1] must also be selected here, it makes sense to code the first person with 0, the second person with 0.5 and the third person with 1.**

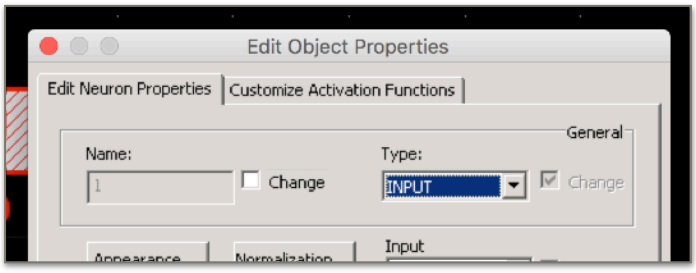
Creating the network in Membrain

In order for the classification to be successful, a number of things must first be prepared. This concerns the modeling of the network, the correct settings and the loading of the training data.

1. At first the modeling shall take place. With the Insert new neurons symbol, neurons can now be distributed on the screen by clicking on them. We need four input neurons and one output neuron, if we want to map four characteristics to a class label Y. In between there can be any number of layers of so-called hidden neurons. For our purposes a small net with three layers (input layer, hidden layer and output layer) and a few additional neurons in the hidden layer is more than enough. The topology (thus quasi the appearance of the net) is a matter of taste and experience and can be different in the different student groups.



2. Since the neurons have different tasks (input, hidden, output), they have to be defined, accordingly. To do this, you can select several neurons in MemBrain with the mouse button pressed down and then click on one with a double-click to set the properties for all selected neurons at the same time. In the upper row the entry INPUT must be selected in the menu under Type.



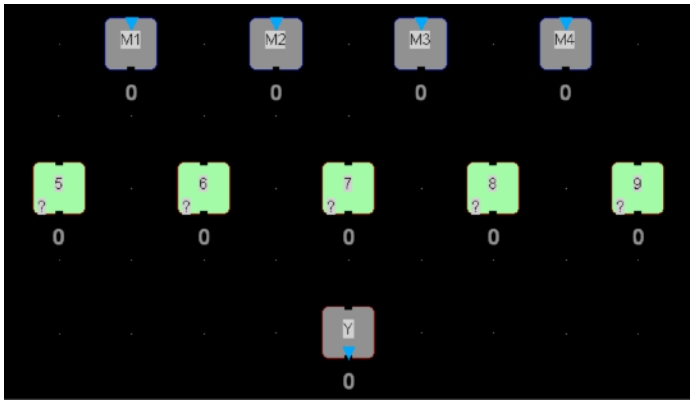
The neurons must then have the same names as specified in the CSV file. To do this, click on each input neuron one after the other and change the name of the neuron under Name accordingly to M1, M2, M3 and M4.

The input layer should then look as follows:

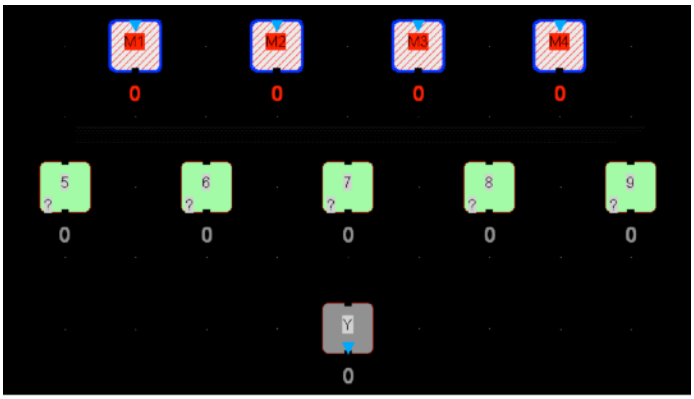
The neurons in the hidden layer do not need to be processed. The default setting can be left.

The individual output neuron is selected by double-click, then the name Y is entered in the Name field and it is declared as OUTPUT under Type. The output neuron now looks like this (the small blue arrows indicate the purpose):

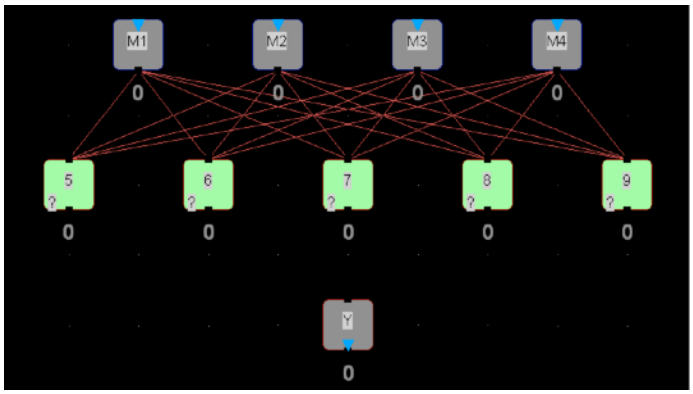
3. To really create a network, the neurons have to be connected to each other. MemBrain offers the concept of extra selection. With the symbol Add to Extra Selection you can add several neurons to the extra selection. It makes sense to mark the neurons of the hidden layer and click on the symbol. The neurons then appear green.



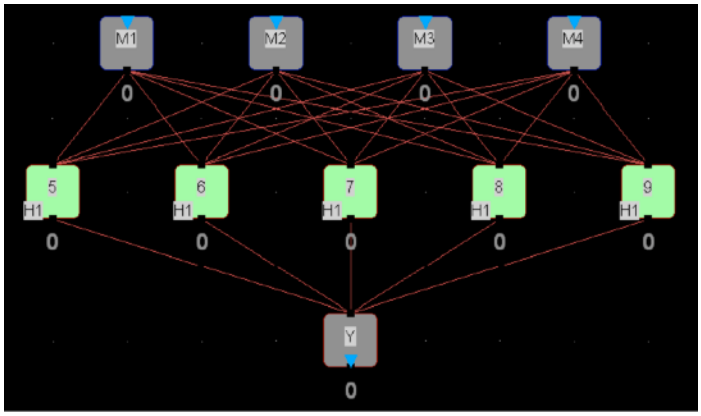
Now it is possible to connect them all at once with the complete upper input layer. To do this, select the input layer (keep the mouse button pressed and select all neurons of the input layer) ...



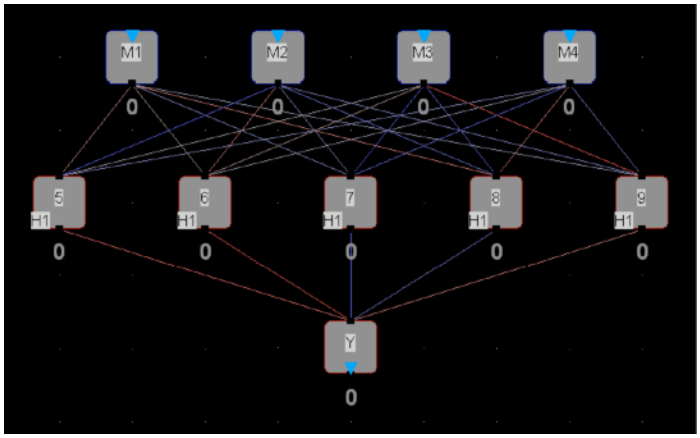
... and then click on the Connect TO Extra Selection icon. Now the two layers are connected automatically and you save processing time. The result will look like this:



Then a click on the output neuron is enough to connect the hidden layer completely with the output neuron by clicking on the Connect FROM Extra Selection icon.



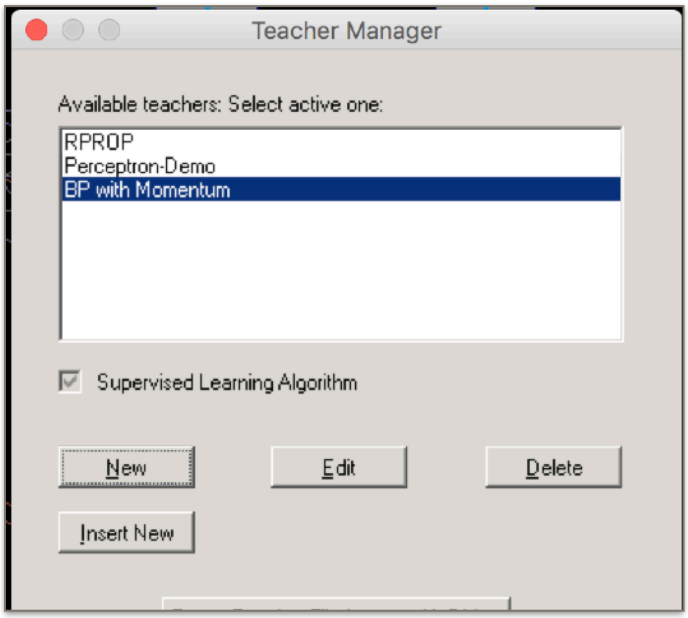
The last step is to undo the extra selection. This can be done by clicking on the Clear Extra Selection icon.



The net is now ready for facial recognition.

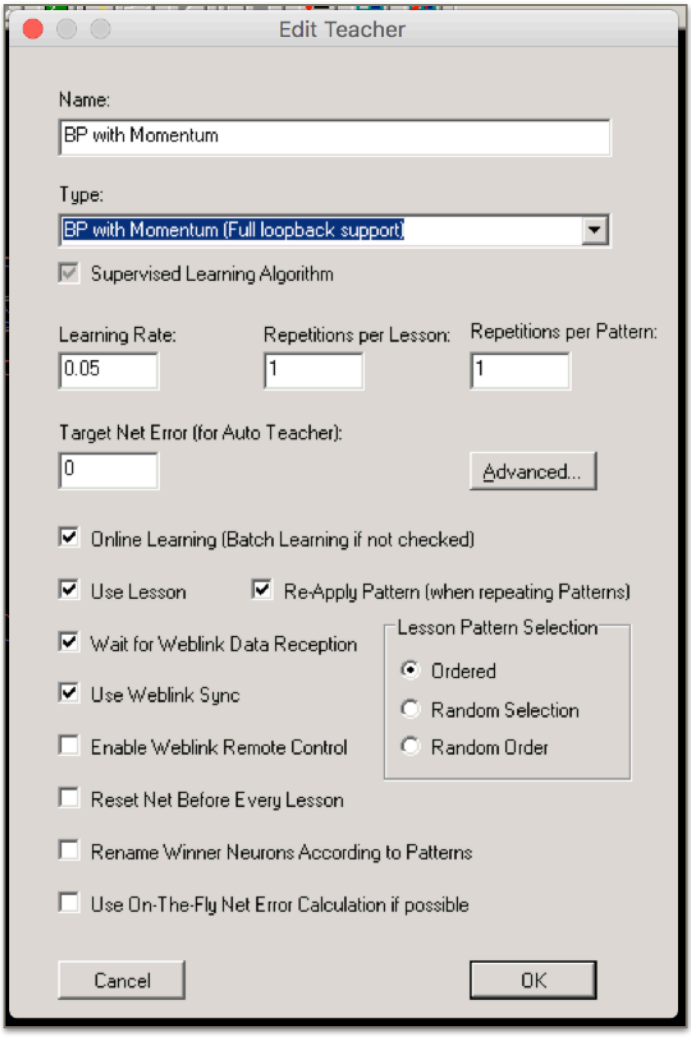
4. The Teacher Manager, which can be found in the Teach menu, is the control center for different learning algorithms. From very simple learning methods to more complicated variants, the right choice must first be made here, since the right learning method is a basic prerequisite for the success of the network. You should follow the following instructions for facial recognition:

Click on the Teacher Manager entry in the Teach menu. The following dialog opens (with possibly fewer entries for a new installation of the program):



Now click New to create a new learning rule. Select BP with Momentum (Full loopback support) from the list at the top of the dialog box.

If necessary, set the parameters as shown in the adjacent picture:



Click OK to return, then select the newly created procedure from the list and confirm the selection again with OK. From now on the network will always use this learning algorithm.

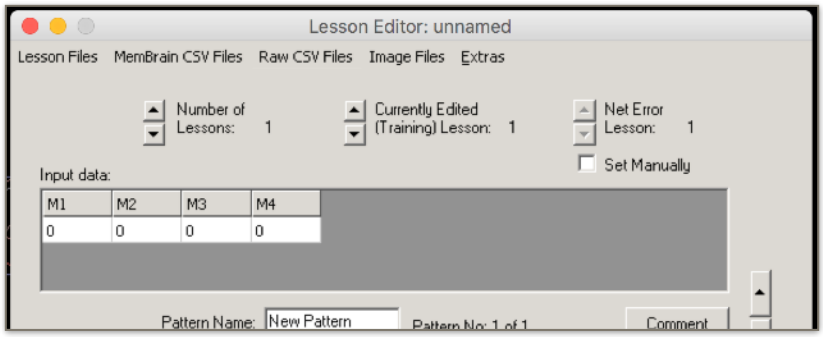
5. Click the Randomize Net icon to initialize the network with random synapse weights, which is an important operation. If a training should lead to very bad results, this may be due to unfavorable initial values in the synapse weights. With randomization, you can try it again with different starting values. Each time the net is to be completely re-trained, it also has to be randomized again.

Note: If you take care not to confuse the model with the original, you can argue here that people also have individual 'starting values' and learning the same can still work.

The coloring of the connections between the nets also represents a numerical value, as is the case with neurons. The creation is now complete and the net is ready to be trained.

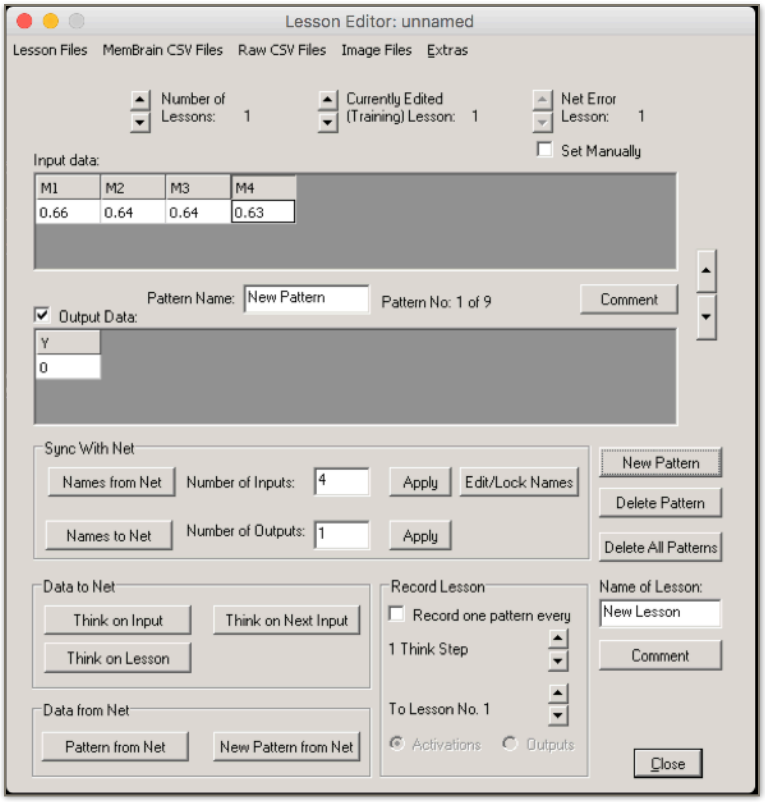
Loading training data in MemBrain

A click on the Show Lession Editor icon opens a dialog window with which you can easily import CSV files, among other things.



In the menu Raw CSV Files under the menu item Import Current Lesson (Raw CSV) ... the functionality to load the training data file is hidden.

After you have selected the correct file, the patterns will be loaded automatically and you should also see that there are now nine patterns available that you can navigate through using the two arrow buttons, possibly also to make a final check. If everything is OK, the dialog can be closed by clicking Close.



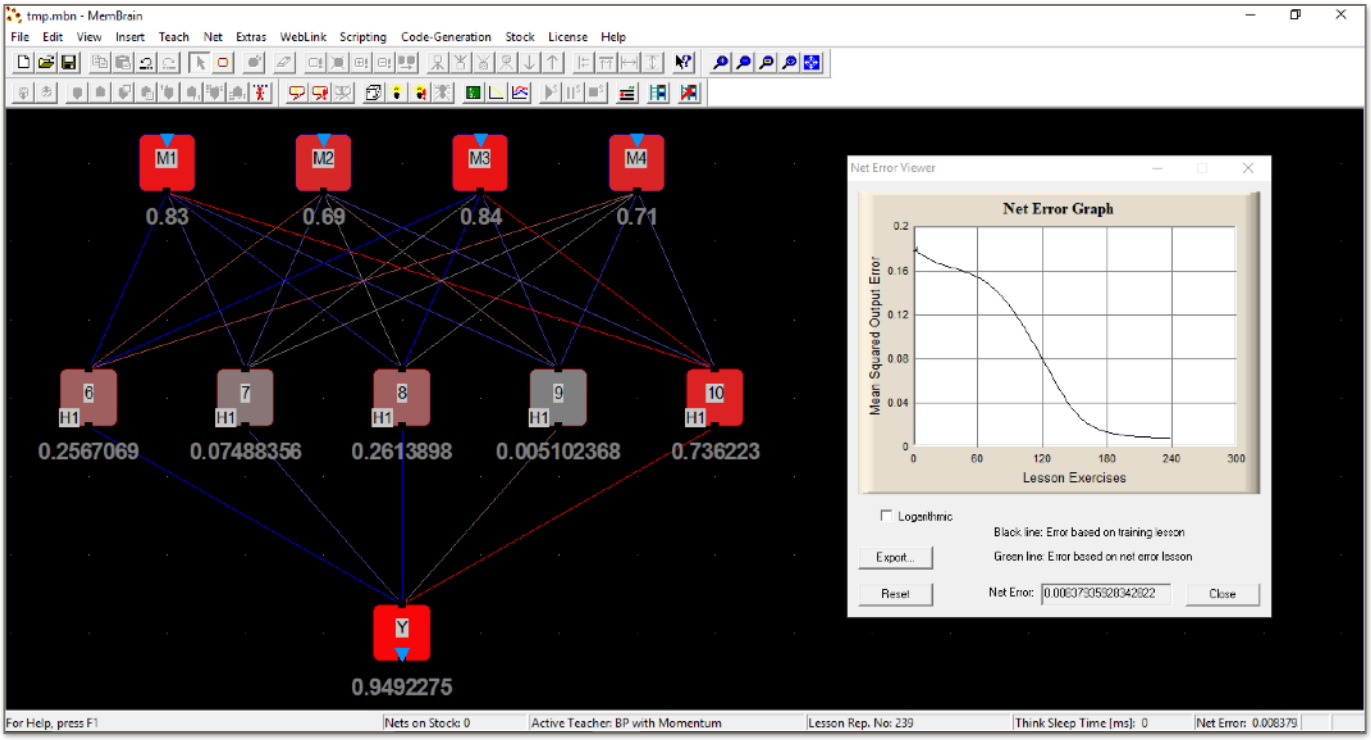
The Training

Before the training starts, it is essential to open the net error window by clicking on the Show Net Error Viewer icon. This window can remain open during the training and shows very nicely how the error of the net during the training is minimized until the net has learned all faces sufficiently well.

With a click on the Start Teacher icon the training can finally be started.

However, with the small amount of data, the error of the network in the Error Viewer will decrease very quickly, which is why a quick click on the Stop Teacher symbol should be made as soon as the error curve reaches its lowest point in a recognizable way.

Note: For demonstration purposes, the amount of data at the network size is OK. For real applications, such as Google's semantic image search, the (much larger and more complex) neural networks there are trained with a number of images far in excess of billions of images. The problem with data sets that are too small is that overtraining occurs - in principle, the network only "memorizes" the few patterns instead of implicitly developing a general 'procedure' for recognition.



The network is now trained and ready for use with the test dataset.

Testing the network

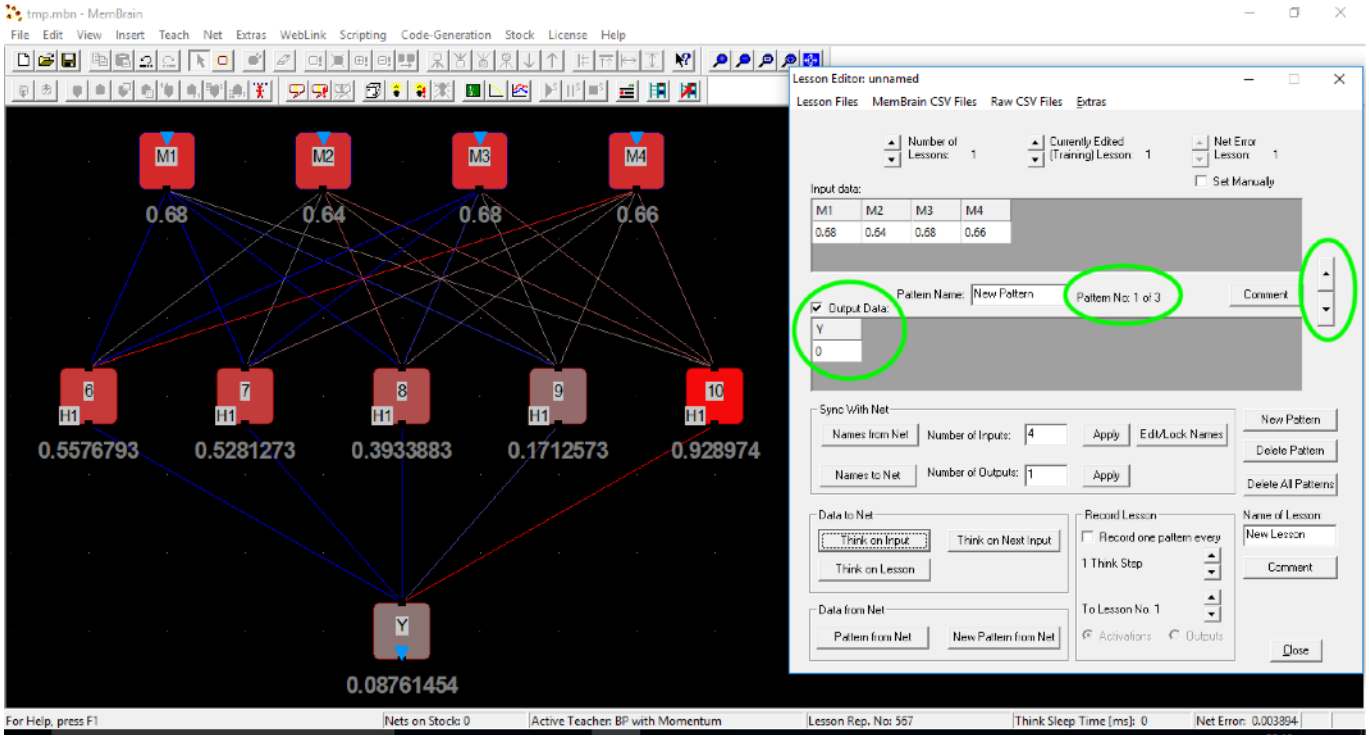
1. To test the network, the students first create a new CSV file, which they save under the name test.csv, for example. This file must have exactly the same structure as the training data file. Again they determine (this time only for three photos) the four exactly same characteristics and enter them then accordingly into the CSV file.

The finished CSV file could look like this:

```
M1;M2;M3;M4;Y
0.68;0.64;0.68;0.66;0
0.74;0.67;0.75;0.65;0.5
0.84;0.69;0.84;0.69;1
```

2. In MemBrain it is now necessary to load the test data into the Lesson Editor. This is opened by clicking on the Show Lession Editor icon. The CSV test file is now loaded in the same way as a training file was imported. It replaces the nine existing patterns with the three patterns in the file. Here it is also a good idea to check again whether all values have been transferred correctly. If this is the case, the test can be started directly in the Lession Editor.

3. It is best to place the dialog window on the screen so that the network is visible next to it. For testing purposes, all parts of the dialog box marked green in the following figure are of importance:



First select the first pattern with the arrow buttons on the right, so that Pattern No. 1 of 3 is in the middle and under Output Data you can also see a 0 as class label. Then a click on the button Think on Input in the area Data to Net causes the current test pattern to be sent through the network and the network in the main window of MemBrain shows directly below the output neuron to which person it assigns the pattern. A value close to 0 therefore means that the person with the class label 0 was also recognized here. A value of 0.39 would indicate that the network behind the current four characteristics is more likely to assume the person with the label 0.5 than the person with the label 0. Which person it is (for comparison) can also be seen in the green area under Output Data (the 0 in this case is simply ignored for the network and is only used for its own control).

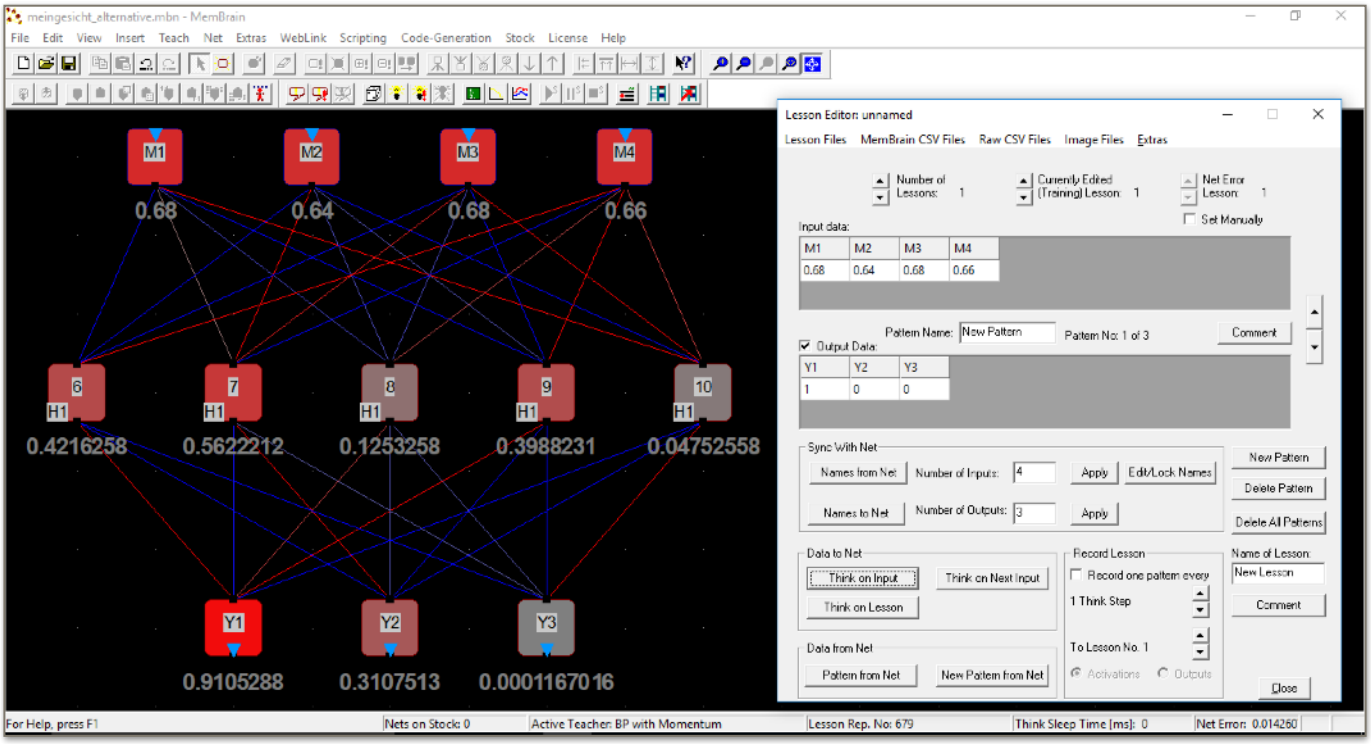
If now the right person is recognized, the students note this and change with the arrow buttons to the next pattern.

4. The recognition rate can then be determined quite easily, which should tend to be 100% depending on the network topology and selected characteristics.

Tip: Enlarging the data set will reduce the recognition rate, which can be very fruitful for the progress of the discussion on machine learning.

Alternatives & thoughts to the facial recognition presented here:

Alternative modeling (e.g. for several classes)



Another way to encode the class label can be to take exactly the number of output neurons that corresponds to the number of people. These are renamed Y1, Y2 and Y3. The training and test data must then be adjusted accordingly:

```
M1;M2;M3;M4;Y1;Y2;Y3
0.66;0.64;0.64;0.63;1;0;0
0.69;0.63;0.68;0.66;1;0;0
0.68;0.63;0.67;0.65;1;0;0
0.72;0.66;0.72;0.63;0;1;0
0.73;0.68;0.75;0.64;0;1;0
0.75;0.67;0.75;0.66;0;1;0
0.80;0.69;0.81;0.70;0;0;1
0.82;0.68;0.83;0.70;0;0;1
0.83;0.69;0.84;0.71;0;0;1
```

```
M1;M2;M3;M4;Y1;Y2;Y3
0.68;0.64;0.68;0.66;1;0;0
0.74;0.67;0.75;0.65;0;1;0
0.84;0.69;0.84;0.69;0;0;1
```

Thus the pattern belongs to person 1 if the 'output vector' is corresponding (1,0,0), to person 2 if it is (0,1,0), and to person 3 if it is (0,0,1).

More thoughts:

- More characteristics may lead to better classification performance.
- Using another activation function may increase the performance of the network (e.g. hyperbolic tangent with value range [-1..1] instead of logistic function with value range [0..1]).
- For such simple applications, three layers are sufficient; more layers do not necessarily improve performance.
- Didactically it might be interesting to delete a neuron once. In many cases, the network can still recognize who the person is.
- Enlarging the test data set provides somewhat more valid rates.
- Due to the small size of the data set, overtraining takes place here, which also explains why recognition almost always works so well. Real facial recognition applications are created with more features, larger meshes, and much more data.
- It should also become clear that artificial neural networks can handle real data so well, because they are more error-tolerant than the hard algorithm, but therefore can make errors themselves, so it would be good for a subsequent discussion if not all groups had 100% recognition. This can be provoked by using more images or by creating a common data set from different groups with different faces per group.
- A shock14ing video about face recognition can be found at autonomousweapons.org, an initiative against autonomous weapons (see also https://www.youtube.com/watch?v=TlO2gcs1YvM).
- In all this, the critical eye should never be lost. In the area of information technology, people and society, great discussions can arise about the questions we will have to deal with in the future.