Artificial Neural Networks – Lab 4
Multi-Layer Feedforward Neural Networks

Purpose

To study multi-layer feedforward (MLFF) neural networks by using Matlab's neural network toolbox

Presentation

The results from the exercise should be presented individually to one of the lab organizers. If a question or sub-task appears with a box 1. then your respective answer **must** be presented. Students must be present at the next lab to answer questions on their work (the deadline for completion is the start of the next lab). If two students work together in a pair, then *both* students must be present and able to demonstrate the software that they have written and explain the answers.

Data and Matlab functions

Data and m-files can be downloaded from the course homepage (`http://www.aass.oru.se/~tdt/ann`). The files you need for this lab are

`dis.mat`	`rate.m`	`getcls.m`
`gen2dv.m`	`genzmesh.m`	

Note: Read each task carefully before starting to solve it.

Preparation

Read chapter 2 of the course textbook. Then read each task carefully before starting to solve it. You can also try some of the others demonstrations in Matlab if you want to improve your knowledge of ANNs.

Task 1, Pattern recognition with MLFF networks

In this task, you have to carry out some experiments to investigate the behaviour of a MLFF network. Here we will try to solve the hardest pattern recognition problem from Lab 2 — remember that the minimum distance classifier and Bayes optimal classifier didn't work very well on this data.

Experiment 1

Load `dis.mat` into the workspace. Create a matrix `p` with input vectors from the classes `dis3_1` and `dis3_2`; this matrix will then be of size 2×240. Create also a matrix with target vectors `t` using the 1-of-m coding. For example, if vector 7 in `p` belongs to class 2 then column 7 of `t` should have a 1 in row 2 and a zero in row 1.

Then use the function `newff` to create a new MLFF network (this is similar to the function `newp` used in the last lab). Initialize a feedforward network with one hidden layer using

```
>> net = newff(minmax(p), [5,2], {'tansig','logsig'}, 'traingdm', 'learngdm', 'mse');
```

The function call `minmax(p)` gives the minimum and maximum values of the inputs. `[5,2]` says that the hidden layer should have five neurons and the output layer should have two neurons. `{'tansig','logsig'}`

says that the hidden layer should have `tansig` as the activation function and that the output layer should have `logsig` as the activation function. `'traingdm'` and `'learngdm'` says that standard backpropagation training with momentum should be used. `'mse'` says use the mean squared error function.

Then set the learning rate and the number of epochs as follows:

```
>> net.trainParam.lr = 0.3;
>> net.trainParam.epochs = 100;
```

Then you can train the network as follows:

```
>> net = train(net, p, t);
```

where `net` is the initialized network, `p` are the training vectors and `t` are the corresponding target vectors. During training, information will be written on the screen about the current epoch and mean squared error (MSE). A graph will also be drawn.

After training, the network may be simulated using the function `sim`. The command below uses the trained network to classify all training vectors.

```
>> y = sim(net, p);
```

Measurement of the rate of correct classified vectors can be done as follows

```
>> rate(getcls(y), t);
```

Try to classify all training vectors (using `sim`) and record the classification result (using `getcls` and `rate`) and the mean squared error (MSE). Do 10 calls to `train`. After each call, classify the training vectors and record the result. The network should perform better and better.

1. Make a table where you record the number of epochs, the mean squared error (MSE) and the classification result (using `rate`), like this:

Epoch	MSE	Rate
100		
200		
300		
400		
500		
600		
700		
800		
900		
1000		

2. Comment on the result (e.g., does the MSE/Rate go up or down? sometimes or usually or always??) Note that sometimes the training might get stuck in a local minimum — if this happens, you have to reinitialize the network and start the training again.

3. Repeat the experiment. Comment on the consistency of the results.

Experiment 2

Now you have to illustrate graphically how the MLFF network learns to solve this problem. Write an m-function that performs the following steps:

- Initialize a new MLFF network, as described above, using the same data.
- Create test vectors in a rectangular area that covers the the two classes. You can do this with the following commands:

```
% Create test vectors over the whole range of input values.
mm = minmax(p);
x1 = linspace(mm(1,1),mm(1,2),50);
x2 = linspace(mm(2,1),mm(2,2),50);
v=gen2dv(x1,x2);
```

  The above code will create a matrix `v` containing the test vectors. Run this code manually, then have a look at `v` and make sure you understand what it contains.

- Initialize a new figure as follows:

```
figure,
myfigure = gcf;
```

- The network should then be trained for 50 rounds where each round is 25 epochs (a simple loop will do this for you). Use the learning rate 0.5.
- After each round the test vectors should be classified using `sim`.
- After classifying the test vectors, you should plot the results on screen so that you can see the decision boundary learnt by the network. You can do this with the following code:

```
z = genzmesh(x1,x2,1.0*hardlim(y(1,:)-y(2,:)));
figure(myfigure),
colormap(gray);
pcolor(x1,x2,z);
hold on,
drawnow,
```

- Plot the training vectors in the same graph.

In the above code, we are using the function `hardlim` again, but this time we are using it to look at the difference between the two outputs of the neural network. The difference will be positive if the first output is bigger than the second, and negative if the second output is bigger than the first. (Just like the decision functions in the work on classical pattern recognition :-)

4. Run your function a few times and explain what happens. Compare with the result from Lab 2 and comment.

5. Try removing the hardlimitting function, then repeat the experiment and see what happens — comment on the results.

If you want you can also create a movie by using the commands `moviein`, `getframe` and `movie` (see the online help) and look at it in real time after the training.

Experiment 3

Test the final trained MLFF network on the original test data for Set 3. Create input vectors and target vectors for `dis3_1t` and `dis3_2t`. Use the `sim` command for testing, and compute the percentage of correctly classified vectors.

6. Compare your results for the MLFF classifier with the results you obtained for the minimum distance classifier and Bayes optimal classifier in Lab 2. Comment.

Task 2, Function approximation

In this task you will study the generalization and extrapolation capabilities of a MLFF network.

Generalization

Investigate how the number of neurons affects the ability of a MLFF to generalize (that is, to classify unseen pattern vectors). The task of the network is to approximate the following function

$$y = x + 0.3\sin(2\pi x)$$

Create training data as follows:

```
>> x = 0:0.02:1;                % x-values for the function
>> y = x + 0.3*sin(2*pi*x);     % Exact function values for each x
>> p = x;                       % x-values used for training of the network
>> t = y + 0.1*randn(size(y));  % Noisy measurements of the function values for p
>> xt = 0:0.01:1;               % x-values to use for testing
>> yt = xt + 0.3*sin(2*pi*xt);  % Correct function values for xt
```

Try to work out what the above code actually does. (Hint: Have a look at the online help for the function `randn`.)
Plot `y` as a function of `x` (e.g., green line) and `t` as a function of `p` (e.g., blue dots) in the same graph so that you can see what the real function and the training data look like.

Create a feedforward network with 1 neuron in the hidden layer with a sigmoid activation function and 1 output with the identity activation function using

```
>> net = newff(minmax(p), [1,1], {'logsig', 'purelin'});
```

Train the network with

```
>> net = train(net, p, t);
```

Note: (MATLAB stuff...) Before using `train`, you will need to save the current figure using

```
>> hold on;
>> myfigure = gcf;
```

then after using `train`, you will need to restore the saved figure using

```
>> figure(myfigure);
```

(Hint: put all the code into a single m-function!)

Then test the network as follows:

```
>> z = sim(net, xt);
```

1. Plot the result in the same graph that you have used for plotting the function and training data, i.e., plot `z` as a function of `xt` (e.g., red line). Compute also the sum-squared error on the test set using `sumsqr(z-yt)`. Is the approximation good? Comment.

2. Do the same initialization, training and testing procedure as above but for different number of neurons in the hidden layer: 2,3,4,5,6,7,8,9,10 and 50. The graph of the approximation together with the real function and training data should be presented in the report. Discuss the generalization performance of the network in each case — e.g., does it underfit or overfit the data?

3. Which network gave the best results? You should also show a table or graph showing the SSE against the number of hidden neurons.

Extrapolation

Is it possible for a trained feedforward neural network to extrapolate? Create data that is not in the interval of the training data above as follows:

```
>> xp = -1:0.01:2;
```

Then answer the following questions:

4. Get the exact function values for `xp` and simulate `xp` on the network that gave the best generalization in the previous exercise. Plot the two graphs in the same figure and comment on the result.

5. Repeat the experiment a few times and see what happens.

Namn: _____ Personnummer: _____

Godkänd: _____ Datum: _____