

# Classification of Iris data set

Mentor:  
Assist. prof. Primož Potočnik  
Student:  
Vitaly Borovinskiy

Ljubljana, 2009

## 1. Problem statement

Fisher's Iris data base (Fisher, 1936) is perhaps the best known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other two; the latter are not linearly separable from each other.

The data base contains the following attributes:

- 1). sepal length in cm
- 2). sepal width in cm
- 3). petal length in cm
- 4). petal width in cm
- 5). class:
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

Fisher's Iris data base is available in Matlab (load fisheriris) and in Internet (for example, on <http://archive.ics.uci.edu/ml/datasets/Iris>).

The goal of the seminar is to demonstrate the process of building a neural network based classifier that solves the classification problem. During the seminar various neural network based approaches will be shown, the process of building various neural network architectures will be demonstrated, and finally classification results will be presented.

## 2. Theoretical part

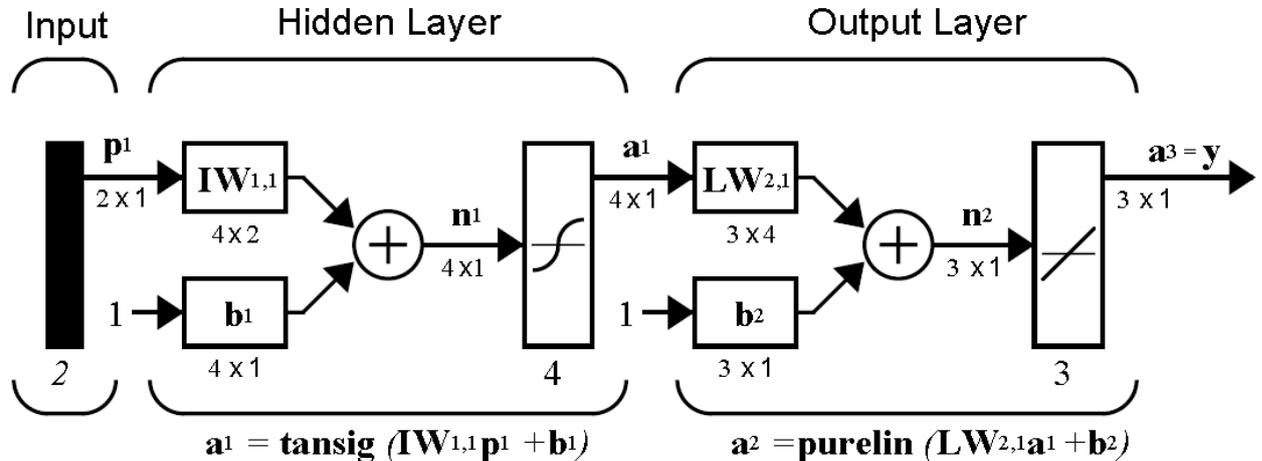
In this seminar classification problem is solved by 3 types of neural networks:

- 1) multilayer perceptron;
- 2) radial basis function network;
- 3) probabilistic neural network.

These network types are shortly described in this seminar. Each of these networks has adjustable parameters that affect its performance.

### 2.1 Multilayer perceptron

Multilayer perceptron is a multilayer feedforward network.



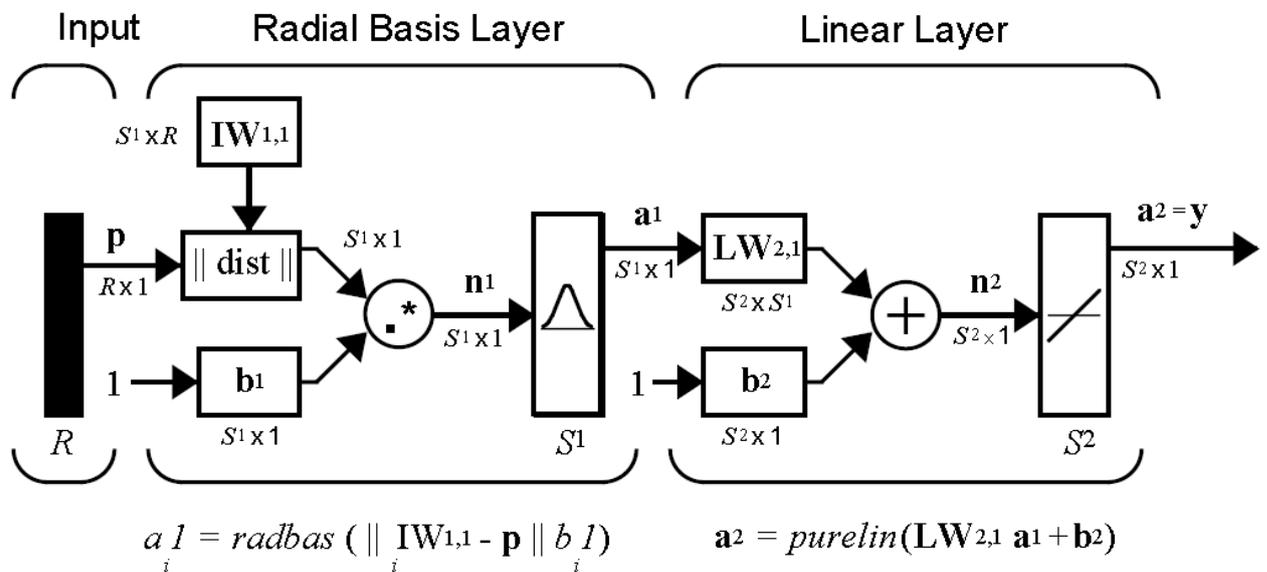
Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors. The linear output layer lets the network produce values outside the range -1 to +1.

In this seminar the transfer functions of hidden layers are hyperbolic tangent sigmoid functions. Network architecture is determined by the number of hidden layers and by the number of neurons in each hidden layer.

The network is trained by the backpropagation learning rule.

## 2.2 Radial basis function network

Radial basis function network is a feedforward network.



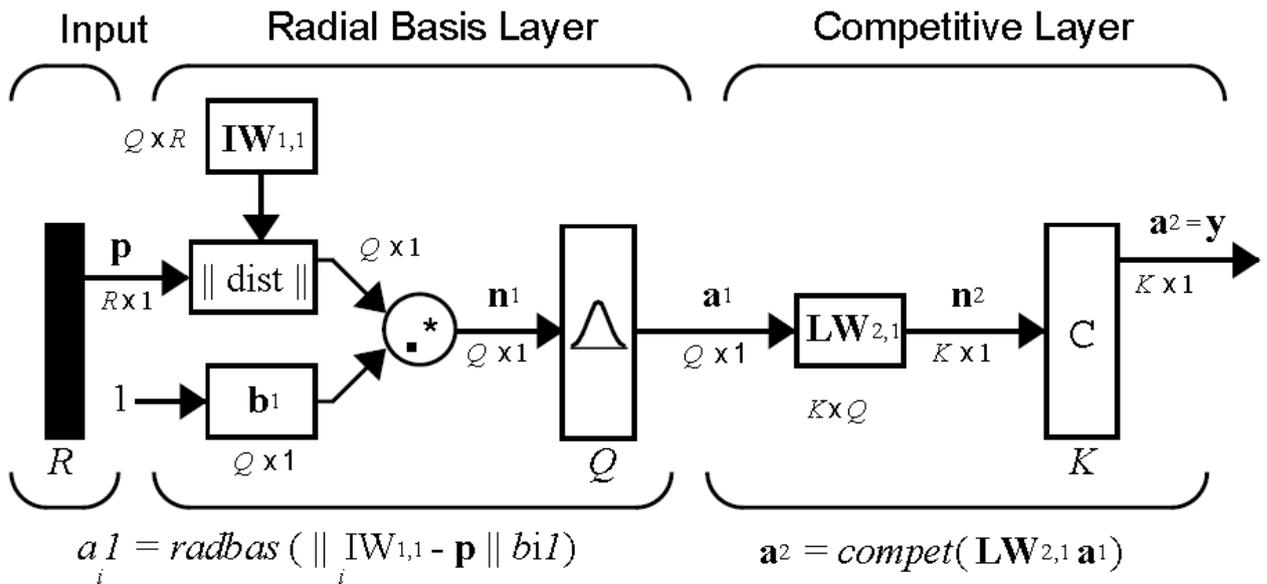
Radial basis function networks consist of two layers: a hidden radial basis layer of  $S_1$  neurons, and an output linear layer of  $S_2$  neurons. Each radial basis layer neuron's weighted input is the distance between the input vector and its weight vector. Each radial basis layer neuron's net input is the element-by-element product of its weighted input with its bias. Each neuron's output is its net input passed through radial basis transfer function.

Radial basis function network is created iteratively one neuron at a time. Neurons are added to the network until the sum-squared error falls beneath an error goal or a maximum number of neurons has been reached.

Design parameter of radial basis function network is spread of radial basis transfer function.

### 2.3 Probabilistic neural network

Probabilistic neural network is a feedforward network. It is specialized to classification.



When an input is presented, the first layer computes distances from the input vector to the training input vectors and produces a vector whose elements indicate how close the input is to a training input. The second layer sums these contributions for each class of inputs to produce as its net output a vector of probabilities. Finally, a competitive output layer picks the maximum of these probabilities, and produces a 1 for that class and a 0 for the other classes.

Design parameter of probabilistic neural network is spread of radial basis transfer function.

Little or no training is required for probabilistic neural network (except spread optimization).

### **3. Practical part**

#### **3.1 Cross-validation**

In this seminar a cross-validation procedure is applied to provide better generalization of neural network classifiers. To perform the cross-validation procedure input data is partitioned into 3 sets:

- 1) training set;
- 2) validation set;
- 3) test set.

The training set is used to train the network. The validation set is used to validate the network, to adjust network design parameters. The test set is used to test the generalization performance of the selected design of neural network.

To ensure a correct comparison of different types of neural networks the division of input data into training, validation and test sets is performed by independent part of code (see Appendix) and the division result is stored.

The partitioning of input data is performed randomly with a certain ratio of input entities to be stored as training set, validation set and test set (0.7, 0.15 and 0.15 respectively).

#### **3.2 Multilayer perceptron**

As soon as the architecture and the performance of multilayer perceptron are determined by the number of hidden layers and by the number of neurons in each hidden layer these are the network design parameters that are adjusted. The correct classification function is introduced as the ratio of number of correctly classified inputs to the whole number of inputs.

Multilayer perceptrons with 1 and 2 hidden layers are investigated. The procedure of adjusting the number of neurons in hidden layers is organized as a grid search (see Appendix). With each combination of numbers of neurons in the hidden layers the multilayer perceptron is trained on the train set, the value of correct classification function for the train set is stored. The validation set is used for standard early stopping procedure, the value of correct classification function for the validation set is stored as well.

The values of the correct classification function are plotted versus the corresponding number of neurons in the hidden layer.

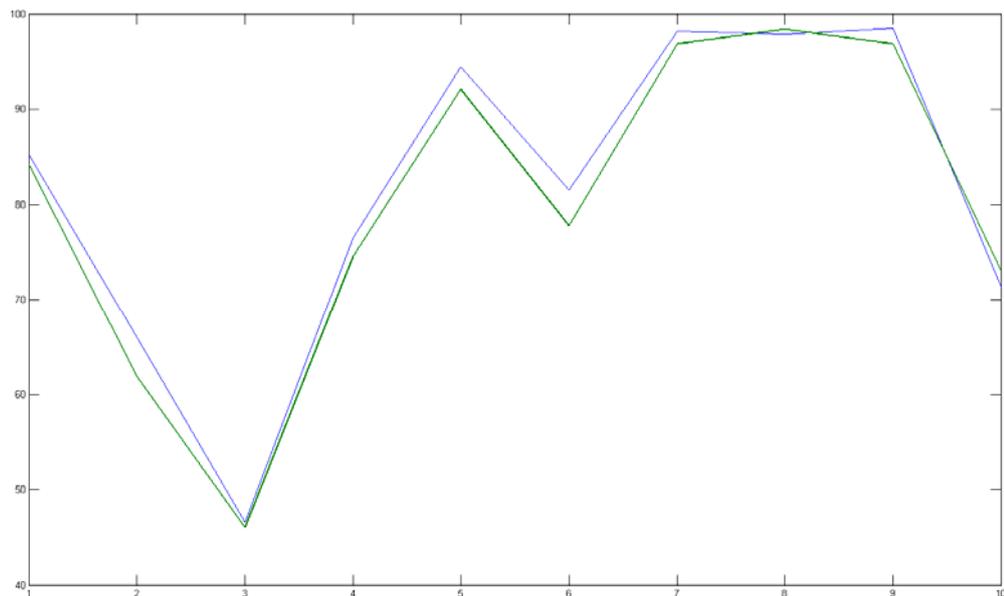


Fig. 1. Correct classification function for multilayer perceptron with 1 hidden layer. Blue line – training set; green line – validation set

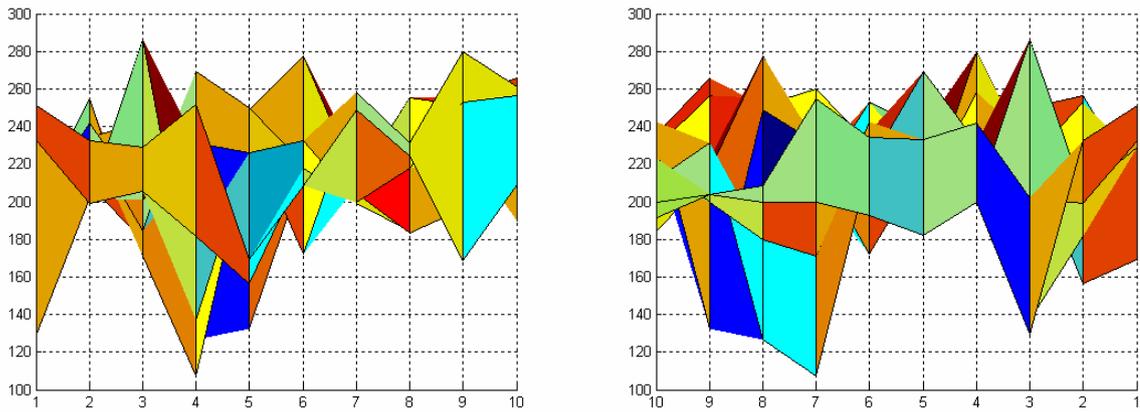


Fig. 2. Correct classification function for multilayer perceptron with 2 hidden layers (2 orthogonal projections of surface). Training set

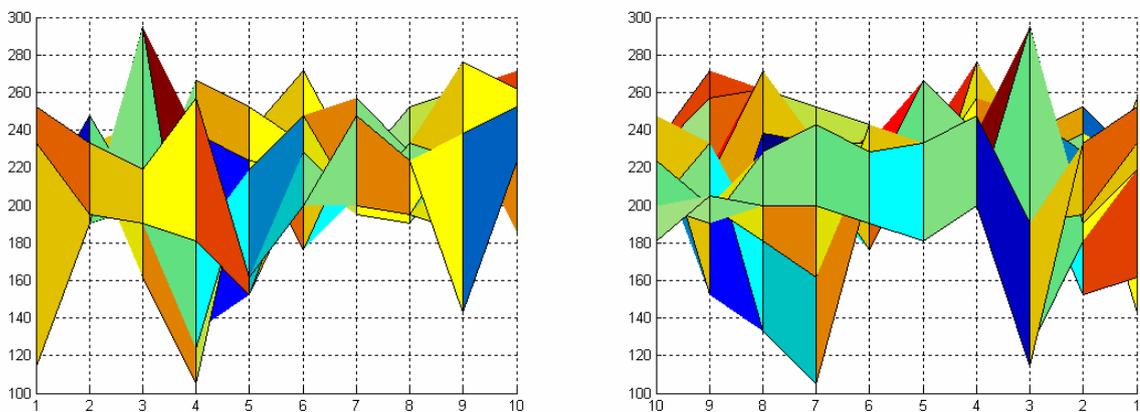


Fig. 3. Correct classification function for multilayer perceptron with 2 hidden layers (2 orthogonal projections of surface). Validation set

The number of neurons that ensures the best generalization is chosen. The training and simulation of the chosen model of multilayer perceptron is performed on joined training and validation sets, the value of correct classification function is calculated.

Finally, the generalization performance of the network is simulated on the test set. The corresponding value of correct classification function is calculated.

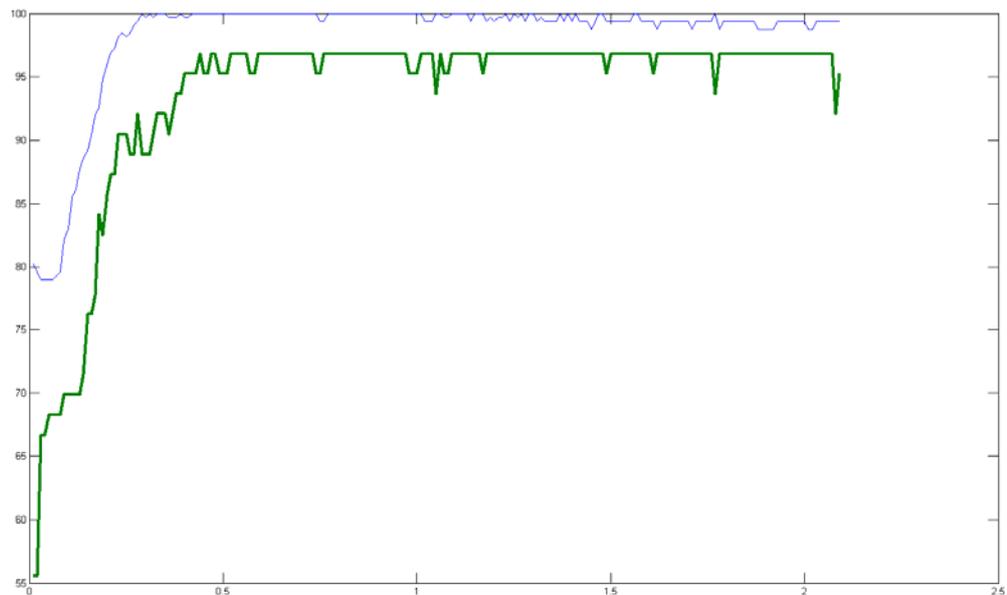
For the training of the multilayer perceptron BFGS algorithm is used, as it is known that for small networks quasi-Newton algorithms are preferable.

### 3.3 Radial basis function network

To obtain the optimal performance of the radial basis function network spread parameter is adjusted. The procedure of adjusting the spread is organized as a search (see Appendix). The search is performed in 2 iterations with different range of varying parameter and different search step.

For each value of spread the radial basis function network is simulated for the train and validation sets, the values of correct classification function for the train set and for the validation set are stored.

The values of the correct classification function are plotted versus the spread.



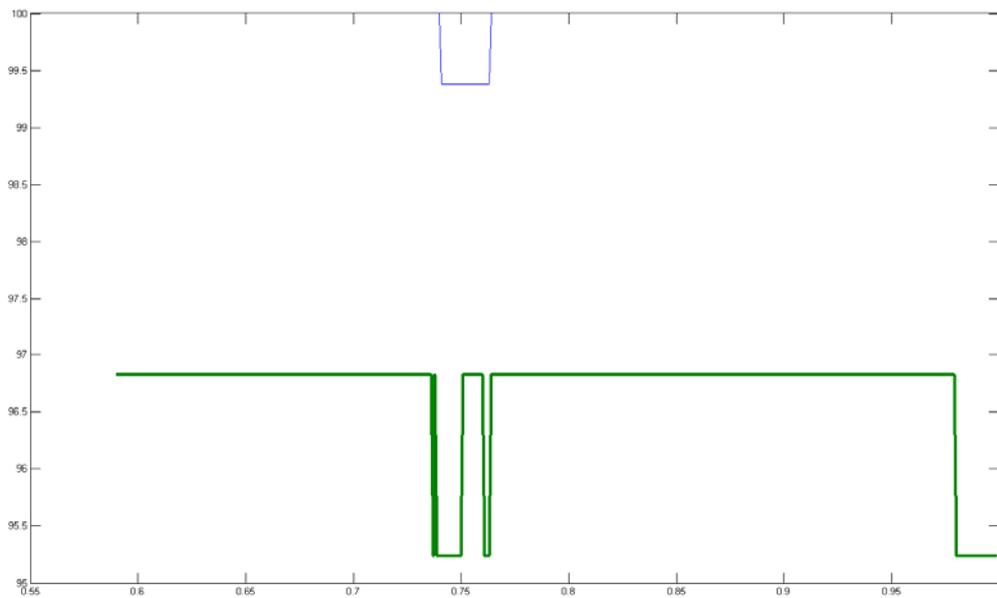


Fig. 4. Correct classification function for radial basis function network. Blue line – training set; green line – validation set

The value of spread parameter that ensures the best generalization is chosen. The radial basis function network is built and simulated on joined training and validation sets, the value of correct classification function is calculated.

Finally, the generalization performance of the network is tested on the test set. The corresponding value of correct classification function is calculated.

### 3.4 Probabilistic neural network

To obtain the optimal performance of the probabilistic neural network spread parameter is adjusted. The procedure of adjusting the spread is organized as a search (see Appendix). The search is performed in 2 iterations with different range of varying parameter and different search step.

For each value of spread the probabilistic neural network is simulated for the train and validation sets, the values of correct

classification function for the train set and for the validation set are stored.

The values of the correct classification function are plotted versus the spread.

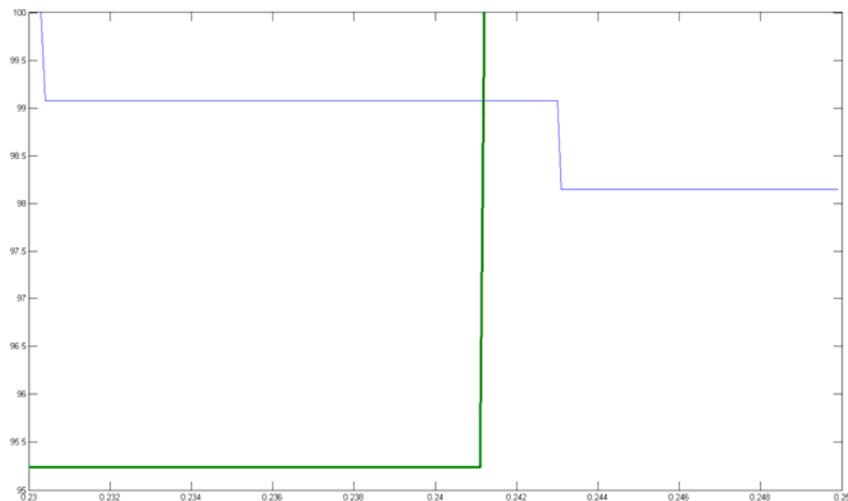
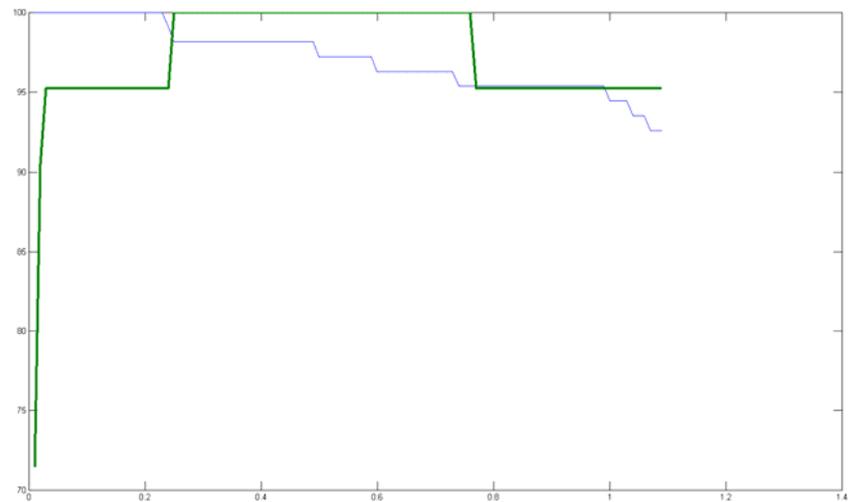


Fig. 5. Correct classification function for radial basis function network. Blue line – training set; green line – validation set

The value of spread parameter that ensures the best generalization is chosen. The probabilistic neural network is built and simulated on joined training and validation sets, the value of correct classification function is calculated.

Finally, the generalization performance of the network is tested on the test set. The corresponding value of correct classification function is calculated.

### 3.5 Results comparison

In the following table the values of correct classification function obtained by supplying different sets of input data into the chosen models of neural networks and processing their outputs are presented.

Table 1

Sets of inputs	Neural networks		
	Multilayer perceptron	Radial basis function network	Probabilistic neural network
training + validation	99.483%	99.225%	98.450%
test	96.825%	100%	95.238%

#### 4. Possible ways to improve the performance of discussed neural networks

From working on my seminar and from discussions with my mentor I see the following ways to improve the performance of neural networks investigated:

##### 1. Input data pre-processing

1) Partitioning of the input data for the cross-validation procedure has effect on the neural network performance. Indeed, even when the same values of division ratios are kept (0.7/0.15/0.15) and the whole data set is partitioned randomly again, the values of the correct classification function change:

Table 2

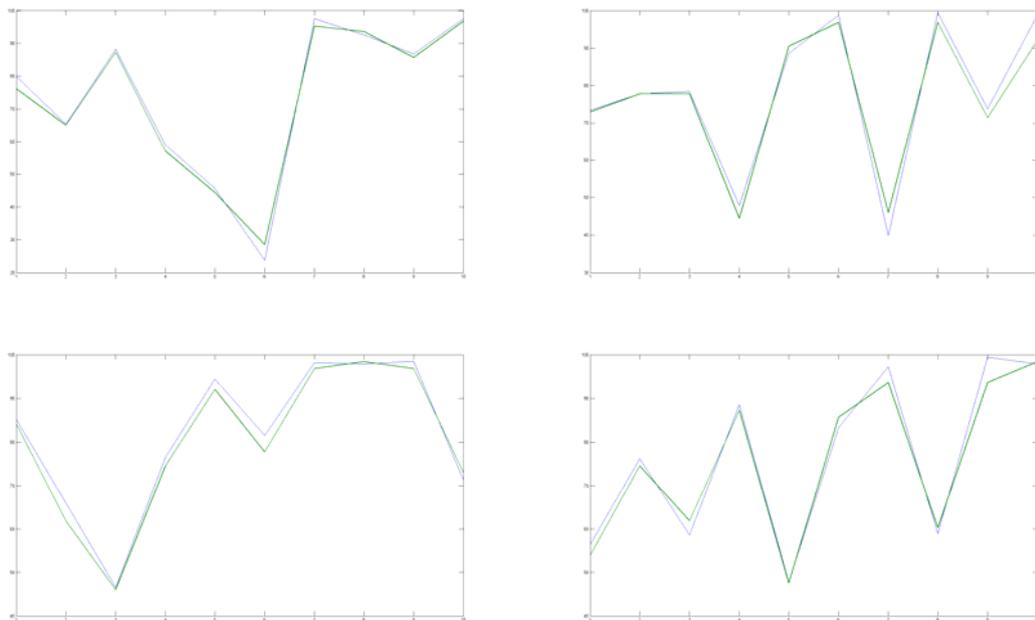
Sets of inputs	Neural networks		
	Multilayer perceptron	Radial basis function network	Probabilistic neural network
training + validation	100%	99.483%	100%
test	96.825%	96.825%	95.238%

This probably happens because the number of inputs is very small and the performance of the network is very sensitive to the way the original set is partitioned. The partitioning of original set can also be optimized. The possible way is to divide the original data set into a number of small sets and to search through them for the one that ensures the best generalization being used as validation set.

2) The principal component analysis can be applied to the original data set to reduce its dimensionality.

## 2. Multilayer perceptron adjustment

While adjusting the number of neurons in hidden layers of multilayer perceptron the results of grid search appear not unique. For example, for a single hidden layer the plots of the correct classification function versus number of neurons are different each time the search is performed:



This probably happens because on every search run the training is finished in a different local minima of network performance function.

Applying a different network training function doesn't solve the problem. Still, the BFGS algorithm is used for training instead of standard Levenberg-Marquardt algorithm, because the training is performed slightly faster.

## 5. Conclusions

1. Classification performance of all 3 investigated types of neural networks is acceptable.
2. Radial basis function network exhibits better generalization performance than multilayer perceptron and probabilistic neural network.
3. Small number of inputs effect crucially on the generalization performance of neural network classifier.

# Appendix

## 1. Multilayer perceptron Matlab code

```
close all; clear; clc
%% load divided input data set
load divinp.mat
% coding (+1/-1) of 3 classes
a = [-1 -1 +1]';
b = [-1 +1 -1]';
c = [+1 -1 -1]';
% define training inputs
trainInp = [trainSeto trainVers trainVirg];
% define targets
T = [repmat(a,1,length(trainSeto)) repmat(b,1,length(trainVers))
repmat(c,1,length(trainVirg))];
%% network training
trainCor = zeros(10,10);
valCor = zeros(10,10);
Xn = zeros(1,10);
Yn = zeros(1,10);
for k = 1:10,
Yn(1,k) = k;
for n = 1:10,
Xn(1,n) = n;
net = newff(trainInp,T,[k n],{'trainbfg'});
net = init(net);
net.divideParam.trainRatio = 1;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0;
%net.trainParam.show = NaN;
net.trainParam.max_fail = 2;

valInp = [valSeto valVers valVirg];
VV.P = valInp;
valT = [repmat(a,1,length(valSeto)) repmat(b,1,length(valVers))
repmat(c,1,length(valVirg))];

net = train(net,trainInp,T,[],[],VV);%,TV);

Y = sim(net,trainInp);

[Yval,Pfval,Afval,Eval,perfval] = sim(net,valInp,[],[],valT);

% calculate [%] of correct classifications
trainCor(k,n) = 100 * length(find(T.*Y > 0)) / length(T);
valCor(k,n) = 100 * length(find(valT.*Yval > 0)) / length(valT);
end
end
figure
surf(Xn,Yn,trainCor/3);
view(2)
figure
surf(Xn,Yn,valCor/3);
view(2)
%% final training
k = 3;
```

```

n = 3;

fintrain = [trainInp valInp];
finT = [T valT];

net = newff(fintrain,finT,[k n],{},'trainbfg');
net.divideParam.trainRatio = 1;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0;

net = train(net,fintrain,finT);

finY = sim(net,fintrain);

finCor = 100 * length(find(finT.*finY > 0)) / length(finT);
fprintf('Num of neurons in 1st layer = %d\n',net.layers{1}.size)
fprintf('Num of neurons in 2nd layer = %d\n',net.layers{2}.size)
fprintf('Correct class = %.3f %%\n',finCor/3)
%% Testing
% define test set
testInp = [testSeto testVers testVirg];
testT = [repmat(a,1,length(testSeto)) repmat(b,1,length(testVers))
repmat(c,1,length(testVirg))];

testOut = sim(net,testInp);
testCor = 100 * length(find(testT.*testOut > 0)) / length(testT);
fprintf('Correct class = %.3f %%\n',testCor/3)

% plot targets and network response
figure;
plot(testT')
xlim([1 21])
ylim([0 2])
set(gca,'ytick',[1 2 3])
hold on
grid on
plot(testOut','r')
legend('Targets','Network response')
xlabel('Sample No.')

```

## 2. Radial basis function network Matlab code

```

close all; clear; clc
%% load divided input data set
load divinp.mat
% coding (+1/-1) of 3 classes
a = [-1 -1 +1]';
b = [-1 +1 -1]';
c = [+1 -1 -1]';
% define training inputs
trainInp = [trainSeto trainVers trainVirg];
% define targets
T = [repmat(a,1,length(trainSeto)) repmat(b,1,length(trainVers))
repmat(c,1,length(trainVirg))];
%% choose a spread constant (1st step)
spread = 2.1;
Cor = zeros(2,209);
Sp = zeros(1,209);
Sp(1,1) = spread;
for i = 1:209,
spread = spread - 0.01;

```

```

Sp(1,i) = spread;
% choose max number of neurons
K = 40;
% performance goal (SSE)
goal = 0;
% number of neurons to add between displays
Ki = 5;

% create a neural network
net = newrb(trainInp,T,goal,spread,K,Ki);

% simulate RBFN on training data
Y = sim(net,trainInp);

% define validation vector
valInp = [valSeto valVers valVirg];
valT = [repmat(a,1,length(valSeto)) repmat(b,1,length(valVers))
repmat(c,1,length(valVirg))];
[Yval,Pf,Af,E,perf] = sim(net,valInp,[],[],valT);

% calculate [%] of correct classifications
Cor(1,i) = 100 * length(find(T.*Y > 0)) / length(T);
Cor(2,i) = 100 * length(find(valT.*Yval > 0)) / length(valT);
end
figure
pl = plot(Sp,Cor/3);
set(pl,{'linewidth'},{1,3}');
%% choose a spread constant (2nd step)
spread = 1.0;
Cor = zeros(2,410);
Sp = zeros(1,410);
Sp(1,1) = spread;
for i = 1:410,
spread = spread - 0.001;
Sp(1,i) = spread;
% choose max number of neurons
K = 40;
% performance goal (SSE)
goal = 0;
% number of neurons to add between displays
Ki = 5;

% create a neural network
net = newrb(trainInp,T,goal,spread,K,Ki);

% simulate RBFN on training data
Y = sim(net,trainInp);

% define validation vector
valInp = [valSeto valVers valVirg];
valT = [repmat(a,1,length(valSeto)) repmat(b,1,length(valVers))
repmat(c,1,length(valVirg))];
[Yval,Pf,Af,E,perf] = sim(net,valInp,[],[],valT);

% calculate [%] of correct classifications
Cor(1,i) = 100 * length(find(T.*Y > 0)) / length(T);
Cor(2,i) = 100 * length(find(valT.*Yval > 0)) / length(valT);
end
figure
pl = plot(Sp,Cor/3);
set(pl,{'linewidth'},{1,3}');
%% final training

```

```

spr = 0.8;
fintrain = [trainInp valInp];
finT = [T valT];
[net,tr] = newrb(fintrain,finT,goal,spr,K,Ki);

% simulate RBFN on training data
finY = sim(net,fintrain);

% calculate [%] of correct classifications
finCor = 100 * length(find(finT.*finY > 0)) / length(finT);
fprintf('\nSpread          = %.3f\n',spr)
fprintf('Num of neurons   = %d\n',net.layers{1}.size)
fprintf('Correct class    = %.3f %%\n',finCor/3)
% plot targets and network response
figure;
plot(T')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
hold on
grid on
plot(Y','r')
legend('Targets','Network response')
xlabel('Sample No.')
%% Testing
% define test set
testInp = [testSeto testVers testVirg];
testT = [repmat(a,1,length(testSeto)) repmat(b,1,length(testVers))
repmat(c,1,length(testVirg))];

testOut = sim(net,testInp);
testCor = 100 * length(find(testT.*testOut > 0)) / length(testT);
fprintf('\nSpread          = %.3f\n',spr)
fprintf('Num of neurons   = %d\n',net.layers{1}.size)
fprintf('Correct class    = %.3f %%\n',testCor/3)

% plot targets and network response
figure;
plot(testT')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
hold on
grid on
plot(testOut','r')
legend('Targets','Network response')
xlabel('Sample No.')

```

### 3. Probabilistic neural network Matlab code

```

close all; clear; clc
%% load divided input data set
load divinp.mat
% coding the classes
a = 1;
b = 2;
c = 3;
% define training inputs
trainInp = [trainSeto trainVers trainVirg];
% define targets
T = [repmat(a,1,length(trainSeto)) repmat(b,1,length(trainVers))
repmat(c,1,length(trainVirg))];
%% choose a spread constant (1st step)

```

```

spread = 1.1;
Cor = zeros(2,109);
Sp = zeros(1,109);
Sp(1,1) = spread;
for i = 1:109,
spread = spread - 0.01;
Sp(1,i) = spread;
% create a neural network
net = newpnn(trainInp,ind2vec(T),spread);

% simulate PNN on training data
Y = sim(net,trainInp);
% convert PNN outputs
Y = vec2ind(Y);

% define validation vector
valInp = [valSeto valVers valVirg];
valT = [repmat(a,1,length(valSeto)) repmat(b,1,length(valVers))
repmat(c,1,length(valVirg))];
Yval = sim(net,valInp,[],[],ind2vec(valT));
Yval = vec2ind(Yval);

% calculate [%] of correct classifications
Cor(1,i) = 100 * length(find(T==Y)) / length(T);
Cor(2,i) = 100 * length(find(valT==Yval)) / length(valT);
end
figure
pl = plot(Sp,Cor);
set(pl,{'linewidth'},{1,3}');
%% choose a spread constant (2nd step)
spread = 0.25;
Cor1 = zeros(2,200);
Sp1 = zeros(1,200);
Sp1(1,1) = spread;
for i = 1:200,
spread = spread - 0.0001;
Sp1(1,i) = spread;
% create a neural network
net = newpnn(trainInp,ind2vec(T),spread);

% simulate PNN on training data
Y = sim(net,trainInp);
% convert PNN outputs
Y = vec2ind(Y);

Yval = sim(net,valInp,[],[],ind2vec(valT));
Yval = vec2ind(Yval);

% calculate [%] of correct classifications
Cor1(1,i) = 100 * length(find(T==Y)) / length(T);
Cor1(2,i) = 100 * length(find(valT==Yval)) / length(valT);
end
figure
p11 = plot(Sp1,Cor1);
set(p11,{'linewidth'},{1,3}');
%% final training
spr = 0.242;
fintrain = [trainInp valInp];
finT = [T valT];
net = newpnn(fintrain,ind2vec(finT),spr);

% simulate PNN on training data

```

```

finY = sim(net,fintrain);
% convert PNN outputs
finY = vec2ind(finY);

% calculate [%] of correct classifications
finCor = 100 * length(find(finT==finY)) / length(finT);
fprintf('\nSpread          = %.3f\n',spr)
fprintf('Num of neurons   = %d\n',net.layers{1}.size)
fprintf('Correct class    = %.3f %%\n',finCor)
% plot targets and network response
figure;
plot(T')
ylim([0 4])
set(gca,'ytick',[1 2 3])
hold on
grid on
plot(Y','r')
legend('Targets','Network response')
xlabel('Sample No.')
%% Testing
% define test set
testInp = [testSeto testVers testVirg];
testT = [repmat(a,1,length(testSeto)) repmat(b,1,length(testVers))
repmat(c,1,length(testVirg))];

testOut = sim(net,testInp);
testOut = vec2ind(testOut);
testCor = 100 * length(find(testT==testOut)) / length(testT);
fprintf('\nSpread          = %.3f\n',spr)
fprintf('Num of neurons   = %d\n',net.layers{1}.size)
fprintf('Correct class    = %.3f %%\n',testCor)

% plot targets and network response
figure;
plot(testT')
ylim([0 4])
set(gca,'ytick',[1 2 3])
hold on
grid on
plot(testOut','r')
legend('Targets','Network response')
xlabel('Sample No.')

```