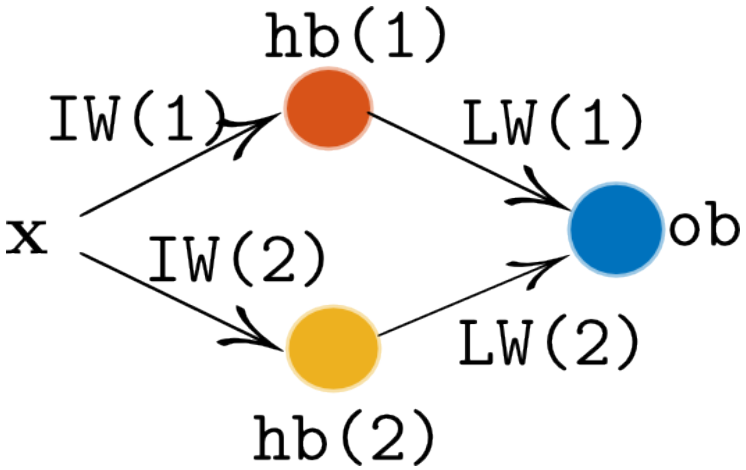
KU LEUVEN

FACULTY OF ENGINEERING

Data Mining and Neural Networks Report



Moritz Wolter

Contents

Session 1

Introduction

1.1 Approximation of noiseless function data

In this first exercise noiseless functions of various complexities will be approximated, as shown in figures 1.1,1.2 and 1.3. Figure 1.1 depicts the *under-fitting case*, with termination at a local optimum or after the maximum number if iterations is reached. In 1.2 where the amount of hidden neurons is increased to two, the solver does sometimes reach a state close to the global optimum, as shown on the left. In other cases, when the random initial condition is not as favorable, the optimization process terminates in a local optimum as shown on the right. If the number of hidden neurons is increased, the process will terminate somewhere close to the global solution more frequently. This is due to the fact that for harder nonlinear problems more hidden neurons enable the network to capture the nonlinearities better. On the other hand additional neurons also increase the chance of *over-fitting* as shown in figure 1.3. Here a neural network with 9 hidden layers is trained to fit to an almost linear function. In comparison to the estimation results with only one hidden neuron, the additional neurons cause strong oscillations. An overview of the feel of network performance based on the authors perception is given in table 1.1.

		Number of hidden neurons								
		1	2	3	4	5	6	7	8	9
Function difficulty	1	good	fair	fair	fair	bad	bad	bad	bad	bad
	2	bad	good	fair	fair	fair	fair	bad	bad	bad
	3	bad	fair	good	fair	fair	fair	fair	bad	bad
	4	bad	bad	fair	good	fair	fair	fair	fair	bad
	5	bad	bad	bad	fair	good	good	good	good	fair
	6	bad	bad	bad	fair	fair	good	good	good	good
	7	bad	bad	bad	bad	fair	fair	good	good	good
	8	bad	bad	bad	bad	bad	fair	fair	good	good
	9	bad	bad	bad	bad	bad	bad	fair	fair	good

Table 1.1: Approximation quality as based on the subjective judgment of the author when using `nnd11gn`.

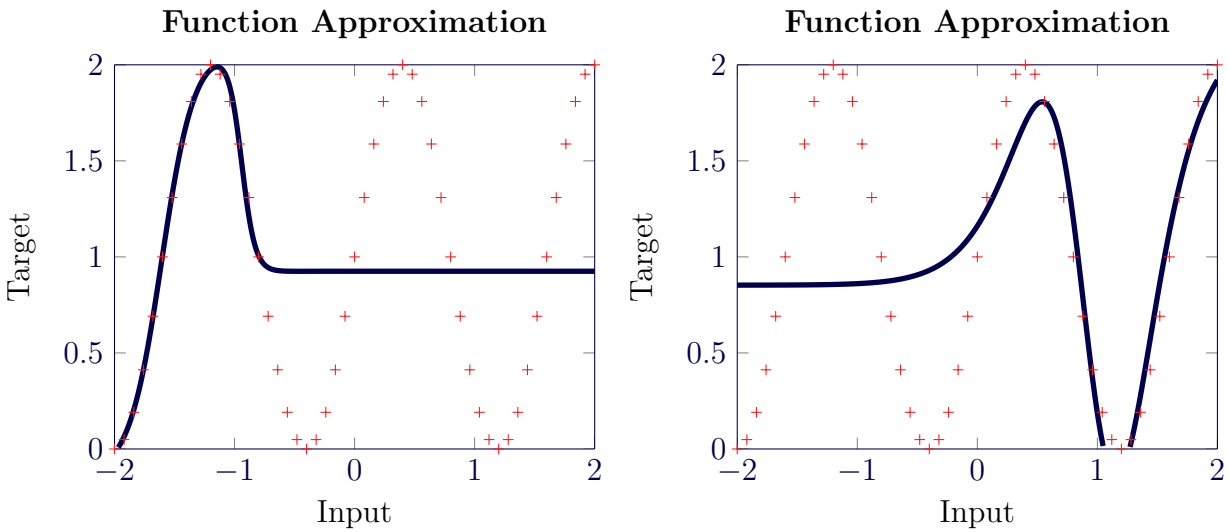


Figure 1.1: Difficulty level 5 noiseless function approximation using two hidden neurons, optimization terminates in local optimum (left), and after the iteration maximum is reached (right).

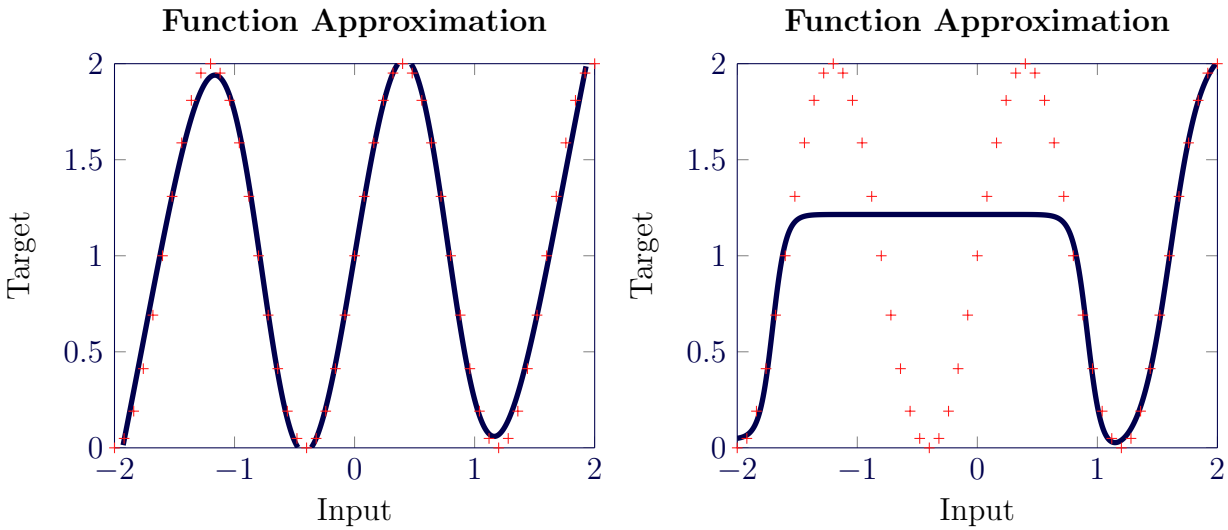


Figure 1.2: Difficulty level 5 approximation using three hidden neurons, optimization terminates in global optimum (left), and in a local optimum (right).

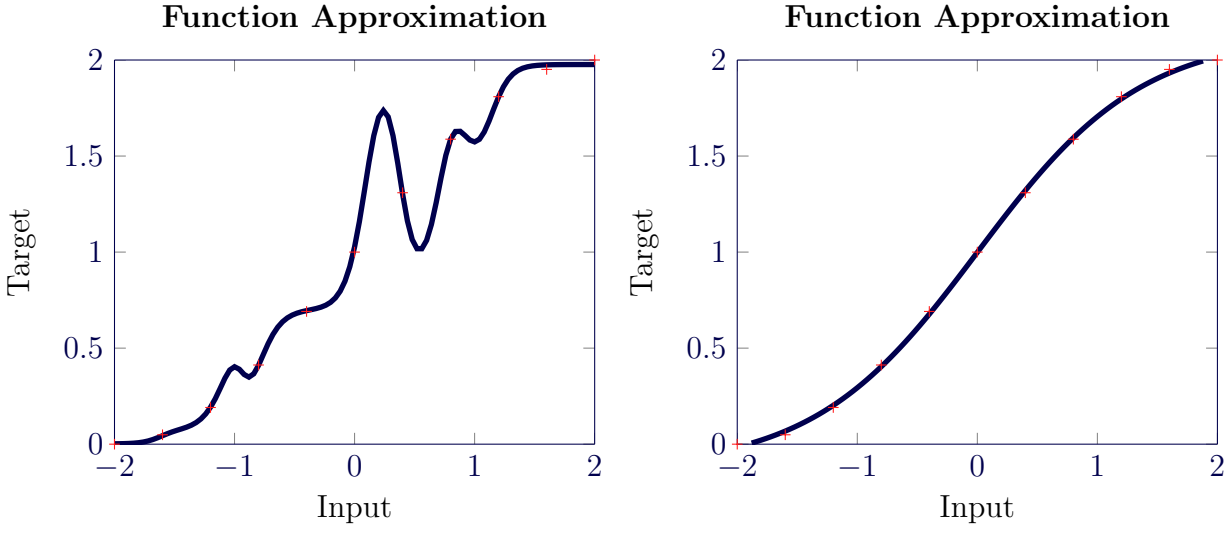


Figure 1.3: Difficulty level 1 function approximation with 9 hidden neurons (left) and one neuron (right).

1.2 The role of the hidden and output layer.

1.2.1 Linear regression

As a second example the use of a neural net to solve a linear regression is problem is considered. The output pattern to be modeled is given by

$$y_p = \omega_1 x_{1,p} + \omega_2 x_{2,p} + \omega_3 x_{3,p} + \cdots + \omega_n x_{n,p} + \beta. \tag{1.1}$$

The parameters $\{\omega_i\}_1^n$ are to be determined from solving the optimization problem

$$\min_{\omega,\beta} \sum_{p=1}^{P} (y_p) - (\omega_1 x_{1,p} + \omega_2 x_{2,p} + \omega_3 x_{3,p} + \cdots + \omega_n x_{n,p} + \beta)^2. \tag{1.2}$$

This may be done from a neural networks point of view using

$$y = \sigma(\omega^T \mathbf{x} + \beta). \tag{1.3}$$

With the activation function sigma just being the linear $\sigma(x) = x$. The corresponding optimization problem, which yields the neuron weights is

$$\min_{\omega,\beta} (y_p - y)^2. \tag{1.4}$$

Which leads back to equation 1.2 as desired. In order to solve this problem a single neuron is sufficient. In this special case no hidden layers exist and the input layer is at the same time the output layer, as shown in the *McCulloch-Pitts model* in the course text on page 21.

1.2.2 Approximating Sine

Next the performance of the network architecture outline above is tested on

$$\mathbf{y_p} = sin(0.7\pi \mathbf{x_p}) \tag{1.5}$$

with $x_p \in [0, 1]$. A plot of relation 1.5 and the output values generated by a linear network trained on this data set is given in figure 1.4. The network is unable to capture the behavior of the sine properly. Generally the sine can only be assumed to be linear for very small angles. When $x \in [0, 1]$ this small angle approximation is no longer valid. Adding a hidden layer with two neurons to the network enables it to deal well with the nonlinearity of the sine.

Figure 1.5 shows the output of the two hidden layer elements, which are colored in red and yellow. The blue curve indicates the network output. The blue neuron produces the corresponding signal. In the arrows are labeled with their weights using the notation used in matlab `Function Fitting Neural Network` objects. Hyperbolic tangents serve as activation functions to the two hidden elements. After training the network and extracting the weights the functions shown in figure 1.5 can be produced by the `matlab` code snippet:

```
h1out = tanh(IW(1)*x + hb(1));
h2out = tanh(IW(2)*x + hb(2));
output = LW(1)*h1out + LW(2)*h2out + ob;
```

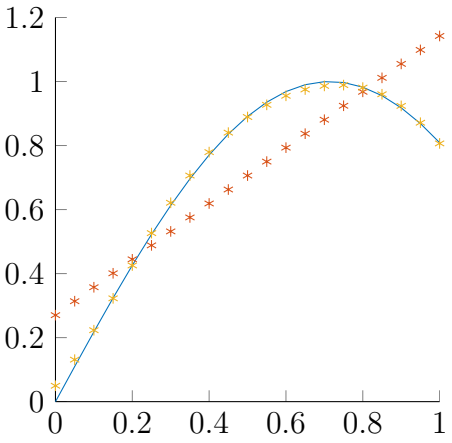


Figure 1.4: Linear net performance on equation 1.5. With the blue line showing the sine function. The output of the linear network is shown by red stars. The output of the net containing a hidden layer with two neurons is shown in yellow.

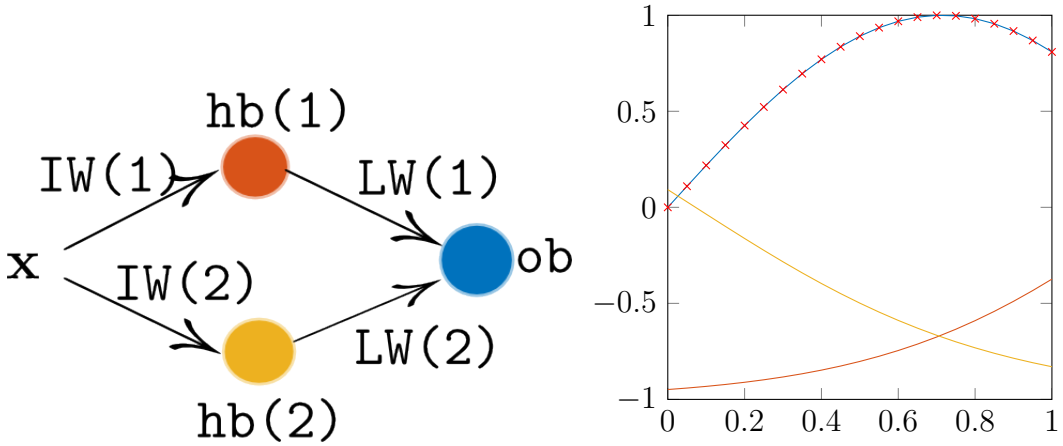


Figure 1.5: Network architecture and corresponding output. The color of the neuron outputs corresponds to their color in the layout. The red crosses indicate target function points.

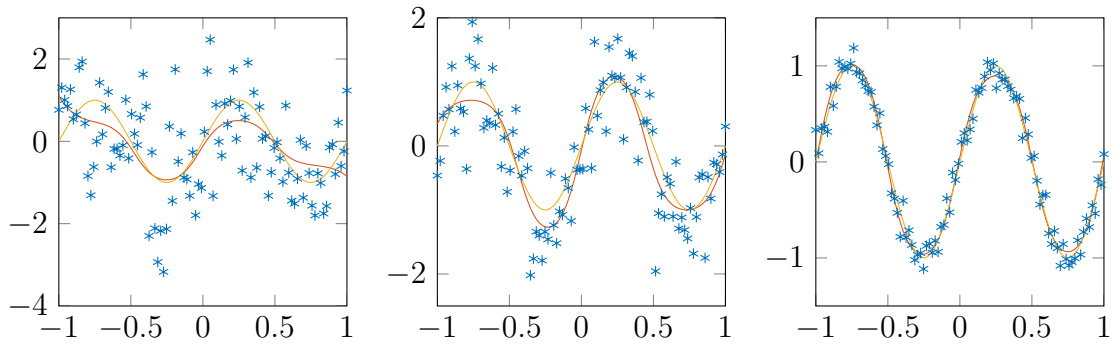


Figure 1.6: Approximation of a noisy sinusoidal signal using 100 data points and 5 hidden neurons. The noise is normal distributed with decreasing variance according to $\mathcal{N}(0, 1^2)$, $\mathcal{N}(0, 0.5^2)$ and $\mathcal{N}(0, 0.1^2)$ from left to right. The blue stars show the noisy training and validation data. The noise free target function is shown in yellow, while the neural network approximation is shown in orange.

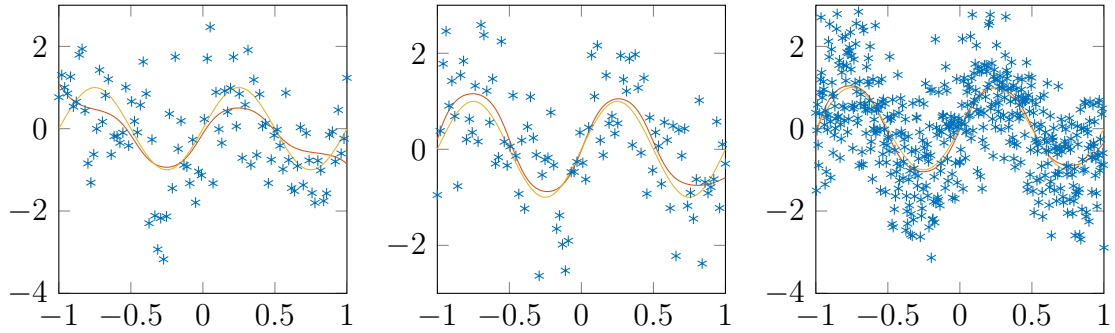


Figure 1.7: Approximation of a noisy sinusoidal signal with noise distribution $\mathcal{N}(0, 1^2)$. The number of input data points has been increased from 100 on the left to 200 in the middle and finally 1000 on the right.

1.3 Function approximation (noisy case)

1.3.1 Variation of the noise variance

Figure 1.6 shows the approximation results of a network with one hidden layer containing five elements. 100 data points have been used. The standard deviation of the normally distributed noise decreased from $\sigma = 1$ to $\sigma = 0.5$ and finally to $\sigma = 0.1$. The figure reveals that the fit improves as the noise variance decreases.

1.3.2 Different sized inputs

In the series of plots shown in figure 1.7, the number of input points has been increased from 100 to 200 and finally to 1000. It can be observed that if the input data is split evenly into training and validation data points, the more data is available the better the network is able to approximate the function. This can be seen as a manifestation of the law of large numbers.

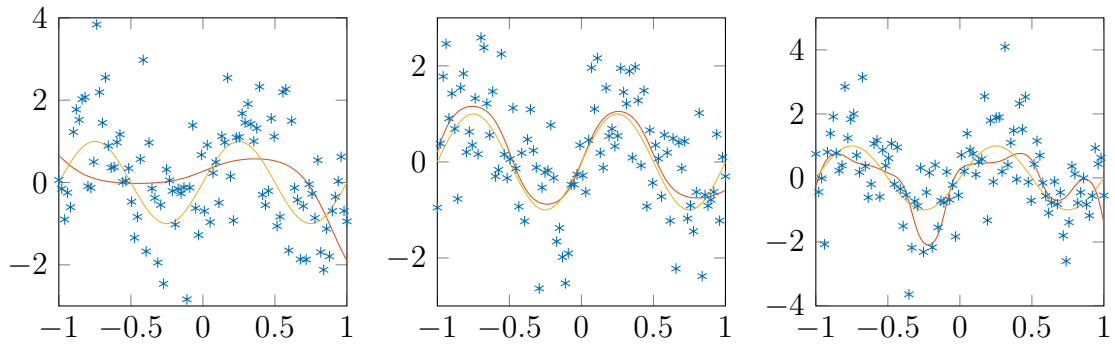


Figure 1.8: Approximation of a noisy sinusoidal signal with noise distribution $\mathcal{N}(0, 1^2)$. In the experiments shown ,the number of neurons has been increased from 3 to 5 and finally to 10 from left to right.

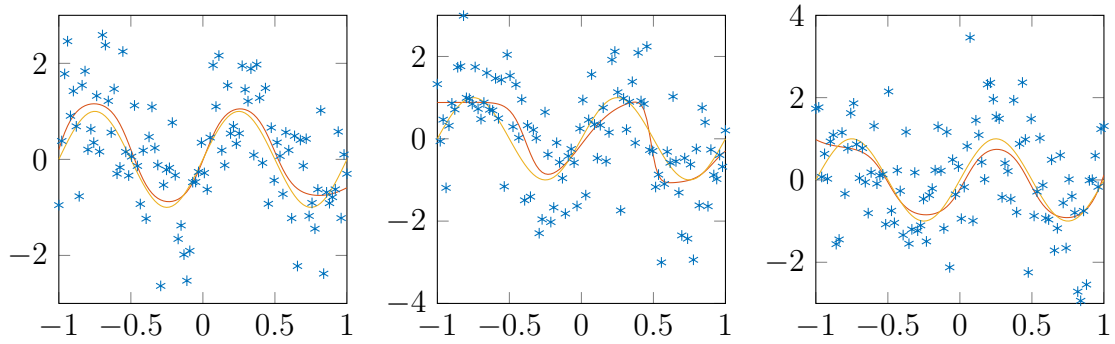


Figure 1.9: Neural network output using different optimization algorithms during the training process. Scaled conjugate gradient (shown left), Levenberg-Marquardt (middle) and BFGS quasi-Newton (right) backpropagation versions have been used to approximate the noisy sinusoid.

1.3.3 The size of the hidden layer

Figure 1.8 shows network outputs to the same input data. The plotted response data has been computed using different sigmoid numbers in the hidden layer. The number of perceptrons has been increased from three to five and finally to 10. For three neurons the network is not able to cope with the complexity of the sinusoid. With 5 neurons it works well. Overfitting can be seen for 10 hidden elements, the effect is particularly pronounced due to the added capacity for noise tracing of the more complex network.

1.3.4 Different training algorithms

In figure 1.9 network outputs from networks trained using different optimization algorithms are shown. Due to the random nature of the sulutions it is hard to draw conclusion about their performance, but it seems that the Levenberg-Marquardt method suffers more from the noise in comparison to the other two methods. In general the scaled gradient descent algorithm is a good choice for large networks. Levenberg-Marquardt and BFGS quasi-Newton methods are suitable for medium sized networks due to their memory

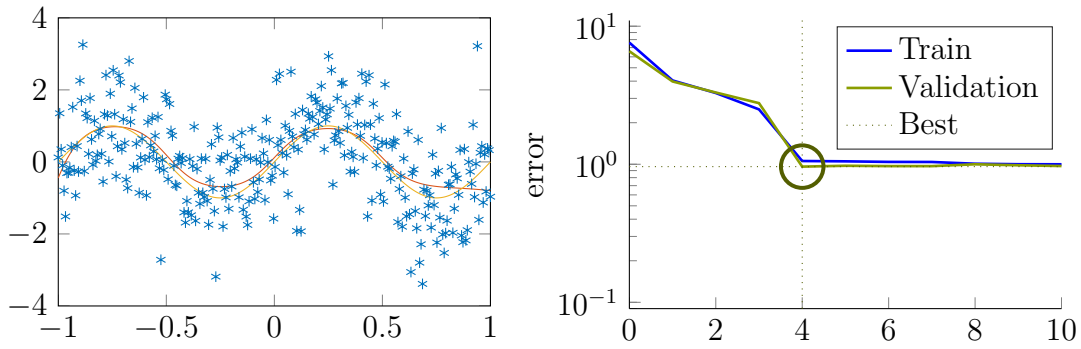


Figure 1.10: The plot on the left shows the optimal fit when using early stopping. On the right the mean squared error is plotted in relation to the optimization iterations or epochs.

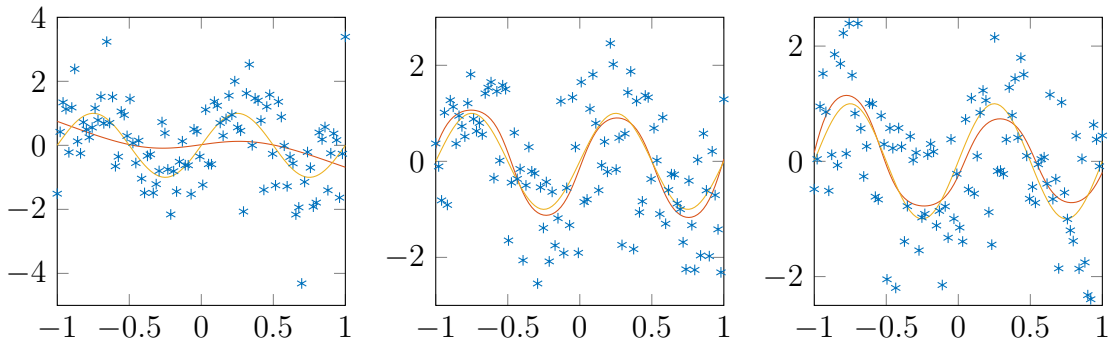


Figure 1.11: Neural network performance using regularization parameters $v = 0.25$, $v = 0.05$ and $v = 0.01$.

requirements and computational overhead. [1]

1.3.5 Early stopping

In order to tackle over-fitting resulting from running the learning process too long, early stopping is used. The process is shown in figure 1.10. Using the validation data the number of iterations is determined after, which the learning process is mainly concerned with memorizing noise contributions. This can be detected by looking at the fit to the validation data set. As soon as the validation error begins to rise, the training is stopped. [2]

1.3.6 Regularization

When thinking about regularization it is helpful to consider the eigenvalue representation of the Hessian $\mathbf{H}\mathbf{u}_j = \lambda_j \mathbf{u}_j$. When the gradient of the regularized problem is formulated

[1] Matlab 2015b documentation

[2] The effect can best be seen in a video like the one at `https://youtu.be/by8eFWhsvlM` where at about 0.75 the network gets drawn to some misleading noisy points.

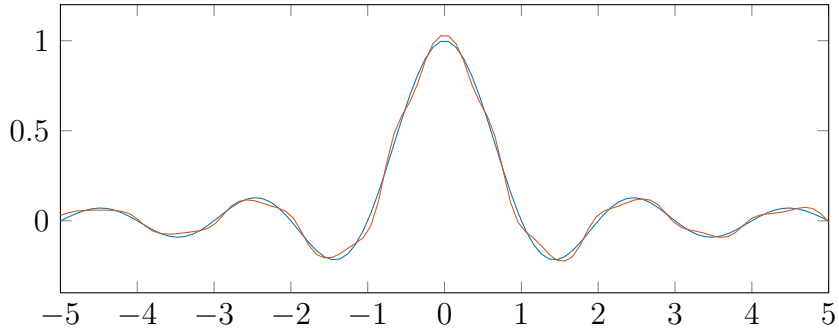


Figure 1.12: Approximation of the noise free function from equation 1.7 in one dimension using a neural net with one hidden layer containing 10 neurons.

and the definition for the regularization parameter is plugged in.[3] The expression

$$\tilde{\omega} = \frac{\lambda_j}{\lambda_j + v}\omega \tag{1.6}$$

is obtained. Which relates the regularized form of the weights $\tilde{\omega}$ and the original parameters of the problem ω. It follows from equation 1.6 that filtering takes place for eigenvalues $\lambda < v$. This fact explains the plots shown in figure 1.11, for $v = 1$ too many eigencomponents are filtered. This leads to the heavily damped network output shown on the left. Optimal damping shown in the middle leads to good tracking of the solution. Where not enough damping is applied in the right, here the network starts to track the noise.

1.3.7 Impact of the initial condition

Depending on the choice of the weights the training process is started with, the optimization algorithms used to teach the network will terminate in different local minima. In order to avoid a particularly bad one it is always a good idea to repeat the training process with several different initial conditions.

1.4 Curse of dimensionality

In this last section of this report. The noise free hat function

$$f(x) = \text{sinc}(\sqrt{\sum_{i=1}^{m} x_i^2}) \tag{1.7}$$

is considered. Its output will be computed in one dimension $m = 1$ in two dimensions $m = 2$ and finally in five dimensions $m = 5$. Function output and network approximation in one dimension is shown in figure 1.12. However in two dimensions the picture changes. Figures 1.13 and 1.14 show the original hat function as well as approximations using different network architectures. Namely 10, 100 perceptrons as well as an MLP with 50 perceptrons in two hidden layers have been trained. As the increase from 10 to 100

[3] Suykens, Data Mining and Neural Networks, Cursustekst page 91.

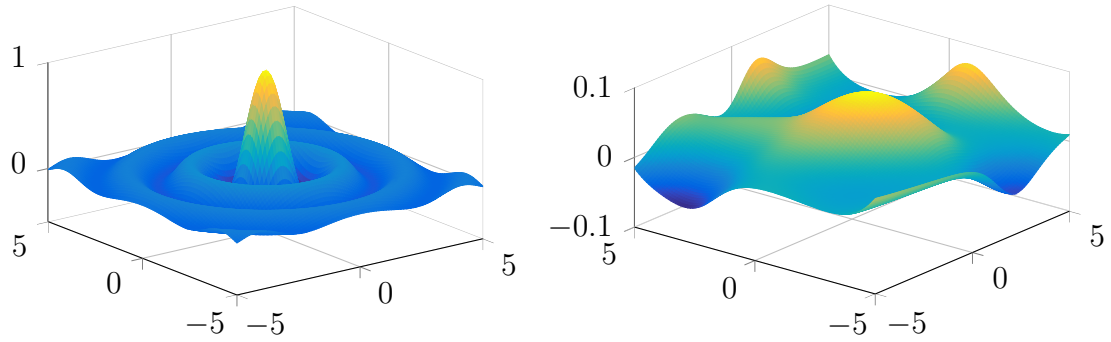


Figure 1.13: Original hat function and neural network approximation using 10 perceptrons in one hidden layer.

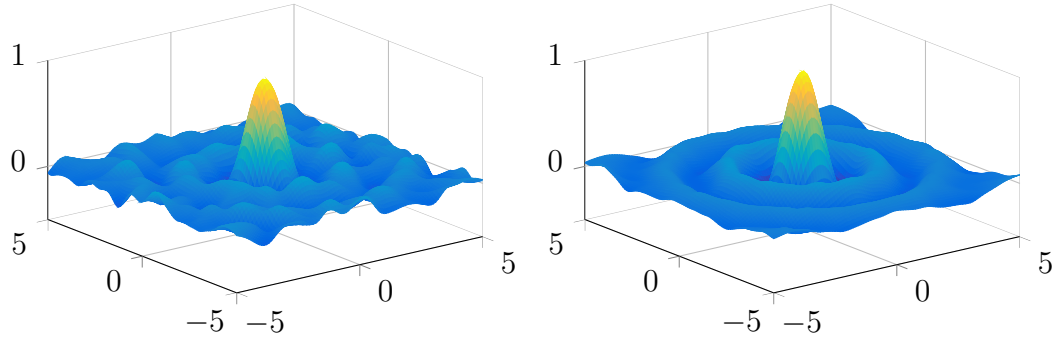


Figure 1.14: Surface plots of two dimensional hat function approximations using 100 elements in one hidden layer and two hidden layers containing 50 neurons each.

neurons increases the qualtiy of the approximations, but even more effective is the split in two hidden layers. This is probably due to the added level of abstraction, that comes with two hidden layers, as the second layer can build upon decisions made by the previous layer. The images above have been computed using a grid ratio of `100` points. In five dimensions this is impossible as this would require `100x100x100x100x100≈74.5GB` of memory, to lay out the complete grid in five dimensions. Of course fewer random points could be considered, but in this report the resolution will be decreased to `20` points, which allows it to train the networks on esat's `vierre64` in all dimensions results are shown in table 1.2. The errors are computed using

$$\frac{\text{norm}(f(x) - N(x))}{\text{norm}(f(x))} \tag{1.8}$$

with $N(x)$ describing the network output. The key to the ability of neural networks to

	`[10]`	`[100]`	`[50 50]`
$\mathbb{R}^1$	1.5425	1.5818	0.6320
$\mathbb{R}^2$	0.9916	0.7167	0.3340
$\mathbb{R}^5$	1.0000	1.0005	1.0005

Table 1.2: Error norm of network approximations using various Architectures of functions of different dimensions.

deal with high dimensional data is their ability to cope with added complexity trough rearrangement of their neurons is more layers, which allows to incorporate information generated in the previous layer. According to Barron the approximation error becomes independent of the dimension of the input space. Which is visible in table 1.2.

Session 2

Santa Fe, characters and diabetes

2.1 Santa-Fee data Set

In this first section time series prediction of Santa-Fee laser data is considered. The data set is obtained from a chaotic laser, which can be described as a nonlinear dynamic system. A training data with 1000 points is available to teach the network. Network performance will then be evaluated using a second prediction data containing 100 measurements. A neural network type suited for this kind of task are nonlinear autoregressive neural networks (`narnet`),

$$\hat{y}_{k+1} = \omega^T \tanh(V[y_k; y_{k-1}; \ldots; y_{k-p}] + \beta) \tag{2.1}$$

Training is done in feedforward mode, which means that the network is in an open loop form. A layout of a `narnet` is shown in figure 2.1. To obtain predictions the network is used in a closed loop form such as the one shown in figure 2.2,

$$\hat{y}_{k+1} = \omega^T \tanh(V[\hat{y}_k; \hat{y}_{k-1}; \ldots; \hat{y}_{k-p}] + \beta) \tag{2.2}$$

which means that the when a new prediction is made the network uses the predictions it has made earlier. However the success of predicting a chaotic system is limited by the Liapunov exponent. Even if the exact equations governing the system are known chaotic systems are extremely sensitive to small errors made during simulation or in the initial conditions. These systems thus become unpredictable after a while. The Liapunov exponents tell something about the speed of which nearby trajectories of these systems diverge.[1]

2.1.1 Variation of the training algorithm and architecture

The available data points are randomly split into three lets training, validation and testing data. 70% are used for learning 15% for validation and finally the last 15% are set aside for testing. To avoid over-fitting early stopping will be used where applicable. In a first series of experiments the network design as well as the training methods will be varied. A network with 25 neurons and a delay of up to 10 using Levenberg-Marquardt has been trained. Fit to the training set and prediction results are shown in figure in figure 2.1.

[1] Strogatz, Nonlinear dynamics and chaos, page 328.

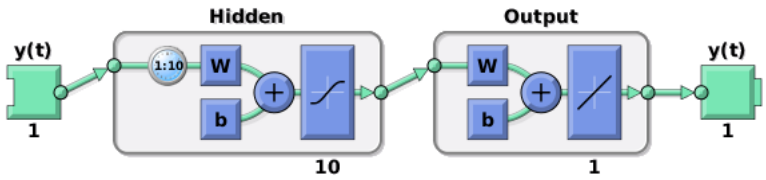


Figure 2.1: Layout of a NAR-Network as used for training purposes. With 10 hidden neurons and a delay of up to ten.

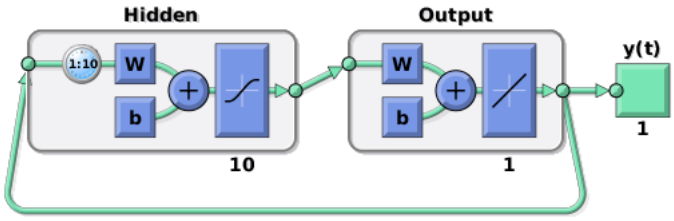


Figure 2.2: Layout of a closed NAR-Network as used for Santa-Fee data prediction.

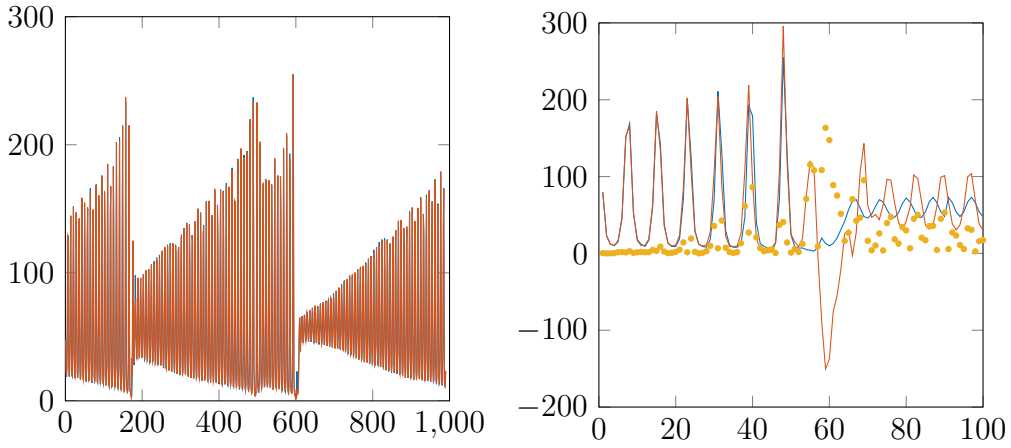


Figure 2.3: Narnet training and prediction results using Levenberg-Marquardt optimization for training. On the teaching set (left) and the prediction set(right). A delay of up to 10 and 25 neurons have been used. The yellow dots in the right plot show the prediction error.

	1:5	1:10	1:15	1:20	1:25
5	42.2161	42.4455	30.0852	39.0152	31.6155
10	4.3969	1.5416	2.8276	1.9532	3.9595
15	4.3253	6.0990	12.0325	5.6509	5.7458
20	21.6387	13.5835	6.1929	7.7123	33.2702
25	2.8752	37.0352	27.9921	21.3609	43.7254

Table 2.1: Mean square error training results using Levenberg-Marquardt backpropagation.

	1:5	1:10	1:15	1:20	1:25
5	65.0282	100.2380	65.7871	57.1775	57.6770
10	20.5488	31.4121	42.0676	69.2126	48.0599
15	209.1161	31.4038	91.4464	109.1161	132.3640
20	216.9201	117.7800	176.0053	156.6302	144.4203
25	136.9215	120.0195	186.6929	191.6889	109.0894

Table 2.2: Mean square error training results using scaled conjugate gradients backpropagation.

	1:5	1:10	1:15	1:20	1:25
5	148.4969	109.3668	90.5823	70.8898	82.2559
10	145.6481	69.4386	53.9265	51.2311	86.7384
15	140.6406	97.8695	86.4476	93.9755	149.2024
20	118.6825	128.1096	140.2982	83.5727	103.7802
25	127.0186	108.6949	116.1468	102.0313	80.5941

Table 2.3: Mean square error training results using resilient backpropagation.

	1:5	1:10	1:15	1:20	1:25
5	0.3351e+05	0.5545e+05	0.0318e+05	0.3218e+05	0.2051e+05
10	0.0455e+05	0.0598e+05	0.5356e+05	0.4324e+05	0.1104e+05
15	0.0480e+05	0.2530e+05	0.0770e+05	0.7009e+05	0.5540e+05
20	1.2492e+05	0.0696e+05	0.0348e+05	0.1640e+05	0.0397e+05
25	0.1298e+05	0.0520e+05	0.2104e+05	0.5399e+05	0.0316e+05

Table 2.4: Mean square error prediction results using Levenberg-Marquardt backpropagation .

	1:5	1:10	1:15	1:20	1:25
5	0.0526e+05	0.1181e+05	0.1035e+05	0.6929e+05	0.4785e+05
10	0.0297e+05	0.0477e+05	1.2537e+05	0.0662e+05	0.2448e+05
15	0.0261e+05	0.0650e+05	0.1148e+05	0.0854e+05	0.0772e+05
20	0.0243e+05	0.0196e+05	0.1296e+05	0.0384e+05	0.0405e+05
25	0.0358e+05	0.0230e+05	0.0556e+05	0.0156e+05	0.0128e+05

Table 2.5: Mean square error prediction results using conjugate gradients backpropagation.

	1:5	1:10	1:15	1:20	1:25
5	0.3940e+04	0.2721+04	0.3880+04	4.6165+04	0.5927+04
10	0.2960e+04	0.3564+04	0.2636+04	0.5647+04	0.2954+04
15	0.9006e+04	0.2586+04	0.5702+04	0.3491+04	0.3793+04
20	0.4404e+04	0.4345+04	0.1902+04	0.3166+04	0.4772+04
25	0.3467e+04	0.3220+04	0.4072+04	0.7259+04	0.2003+04

Table 2.6: Mean square error prediction results using resilient backpropagation.

In this case the network layout was just a wild guess, can it be chosen in a more structured manner? Tables 2.1 to 2.3 show the fit of networks with delays ranging from 5 to 25 and hidden neural numbers increasing from 5 to 25 as well. I both cases steps of 5 are used. Looking at the quality of fit for all training algorithms the most promising values are often found in the center of the table somewhere between a delay of up 10 or 20, with a similar amount of neurons. A problem when attempting to compare network architectures is that the training process ends up in different local minima. As a small remedy for this issue 15 networks have been trained for every delay,hidden layer size and algorithm combination and the best fit to training data has been selected. The more important quantity is the fit of the predictions. The results for different layouts and training strategies are shown in tables 2.4 to 2.6. Unfortunately no obvious link between training data performance and prediction success can be established using the data collected for this report. For example the network 1:25,25 trained using conjugate gradients has the lowest prediction mean squared error, but its fit to the training data was relatively bad. In addition produce the networks trained using Levenberg-Marquardt backpropagation no better prediction results in comparison to the other algorithms, even tough the LM trained networks did fit the training data better in terms of mean square error. In a note of caution it should be added that if the training mse is very large the prediction results will be very bad as well. However such cases have been filtered out by training several networks for each combination and only selecting the good fits.

2.1.2 Committee network prediction

Two networks with prediction mean square error 1.2213e+05 (left) and 4.8843e+03 (right) are shown in figure 2.4. Considering the mse alone one would be inclined to call the right network the better one. But a closer look at their output signal reveals that it got the last peak one which the right network predicted correctly. Additionally the right net does not produce an event after which the peak hight and frequency changes, the right network does but gets it completely wrong. However by not producing the event its output signal is on average closer to the true values. What happens if 100 networks are trained and their output is averaged can be seen in figure 2.5. The mean square error of the average prediction is 19.6963 in this case!

2.1.3 Variation of input vector composition

When looking at the input data the value of data region can be judged by it's autocorrelation. A autocorrelation plot of the `lasertrain` data set is given in figure 2.6 on the left. The plot reveals that values found between sample 150 and 250 are not strongly correlated to the rest of the data. The autocorrelation of the smaller cut dataset is shown on the right. To evaluate the effect of cutting out the data once more 100 networks have been trained on the new smaller data set. Their averaged output is shown in figure 2.7. The result is still a meaningful prediction however significant error appear earlier. The mean square or the error increases to 68.4841 from about 20.

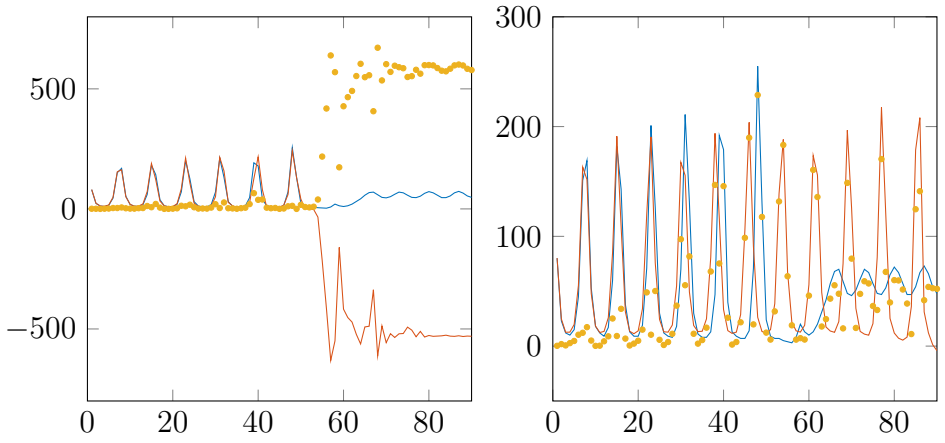


Figure 2.4: Prediction outputs of two different networks, using delays $1 \ldots 10$ and 25 hidden neurons.

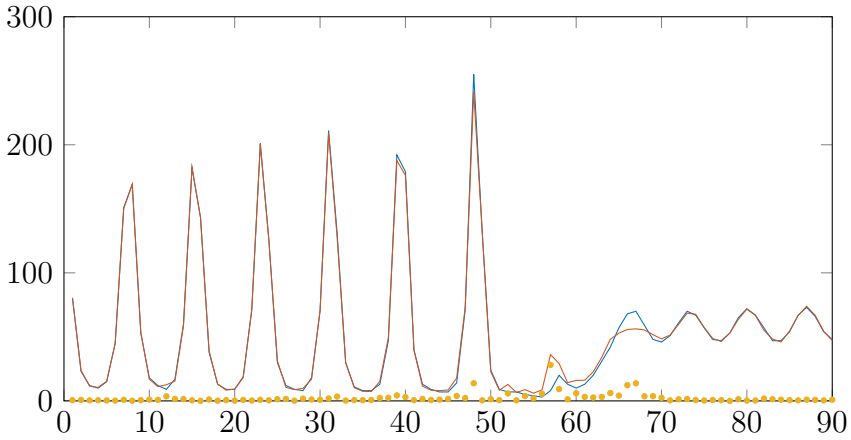


Figure 2.5: Averaged output signal of 100 networks.

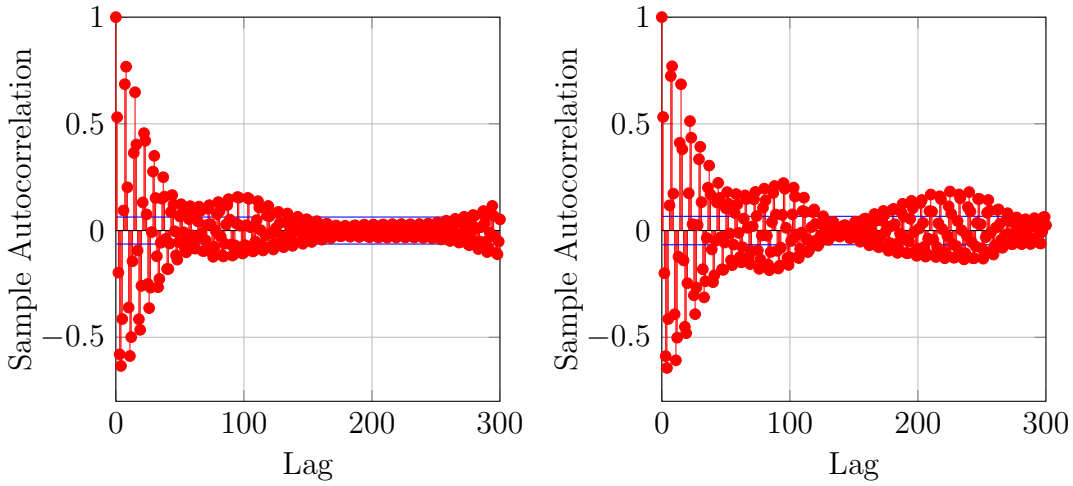


Figure 2.6: Autocorrelation of the `lasertrain` data set (left). On the right the autocorrelation of the shortened version of the signal is shown.

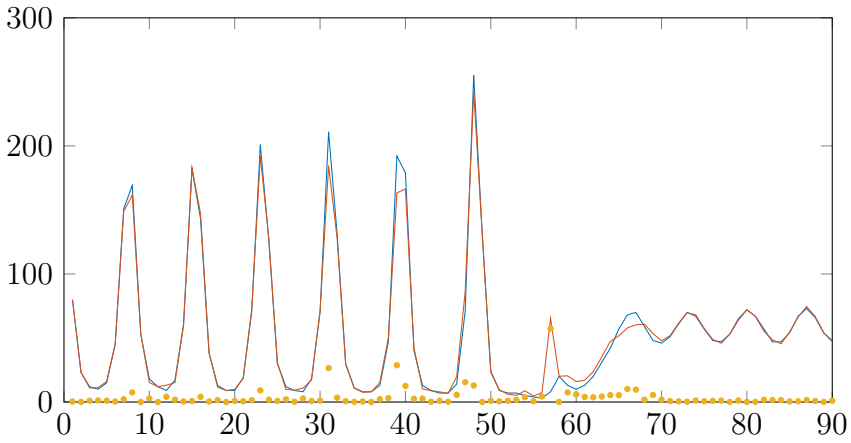


Figure 2.7: Average prediction of 100 networks trained on the shortened data set.

2.1.4 Other variations

Using regularization does not improve matters significantly. If the parameter governing the extend of the regularization is chosen too large it makes matters considerably worse. This is the case, because this is not an identification problem where noise has to be eliminated, so the smooting effect regularization has either does not matter or eliminates useful information. Changing the hidden layer transfer functions to tangent or logarithmic from the default hyperbolic tangent also did not yield any improvements.

2.2 Alphabet recognition

Using data where each letter of the alphabet is a vector of 35 values which can either be 1 or 0. Then Each vector of 35 values defines a 5x7 bitmap of a letter. A target data set contains the actual letter which are contained in the data matrix. In the matlab script `appcr1` two neural networks are trained. Both networks share the architecture shown in figure2.8. It has 35 inputs as each letter is represented as a vector of 35 bitmap values. The 25 hidden neurons are used for recognition. Finally 26 output neurons are used where each corresponds to the network recognizing a particular letter. The first network is only given noise free version of the letters it is supposed to learn to recognize, such as the A shown in figure 2.9 on the left. A second network is trained with noisy versions of the same data, as shown in figure 2.9 on the right. Instead of just one noise free version of each letter the second network is given 30 noisy version of each character. It is trained with more data in the presence of noise. After the training process the performance of both networks is evaluated as shown in figure 2.10. The network trained in the presence of noise does perform better on noisy and noise free data. The performance of the network that did not see noise during the training process has a harder time dealing with it, despite the fact that its architecture is the same. Even tough the second network saw data of poor quality it can perform better, because it had a larger training data set then the first network. For training networks it can thus be concluded that the bigger the data the better. An ideal network would have been trained with noisy and noise free data.

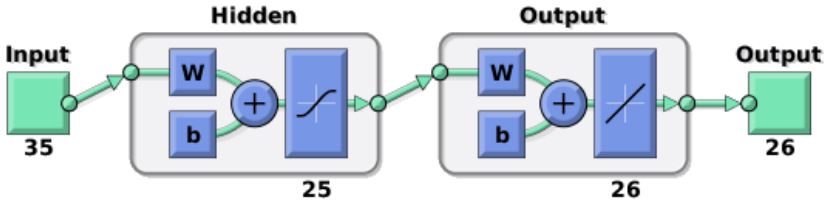


Figure 2.8: Architecture of the two networks trained for the letter recognition this experiment.

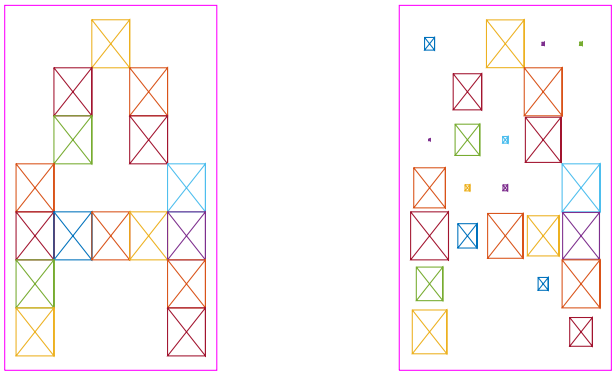


Figure 2.9: A noise free (left) and noisy (right) version of the letter A.

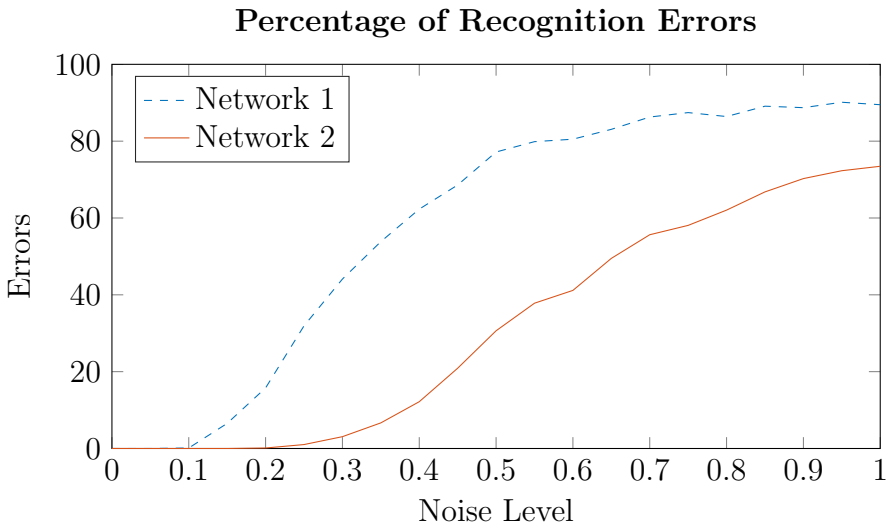


Figure 2.10: Network performance of networks trained in the presence and absence of noise.

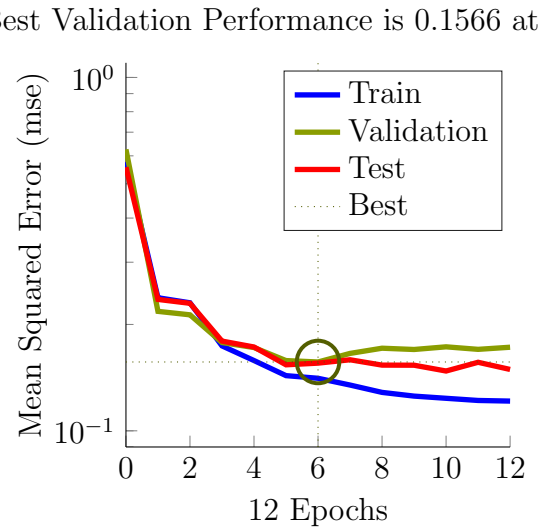


Figure 2.11: MLP performance during the Training process on the training, validation and test sets.

2.3 Prima Indians Diabetes

The diabetes data set contains 8 data points per person. Using these values, which could be for example blood pressure, hight, weight, etc. Using this data set and a vector which classifies the individuals in the data set into a healthy and unhealthy group, a neural network will be trained to identify individuals with diabetes. A total of 768 humans is included in the data of which 268 have diabetes (34.1 %). Matlab's `patternnet` networks will be used to solve this problem. These nets are specialized for applications in pattern recognition and can be trained to classify inputs according to target classes. Before the network is trained the target vector is remapped to contain a zeros for healthy individuals and a 1 for persons with diabetes. Again 70% of the data are used for learning process, wile 30% will be split evenly among training and testing data. The evolution of the mean square error during the learning phase is shown in figure 2.11. The training process is stopped, when the classification error on the validation data is observed to rise. A detailed picture of the classification results of the network is shown in the confusion matrix in figure 2.12. On surprisingly the performance is best on the training data. On the validation and test data the correct classification various roughly between 77 and 74%. A slightly better result has been achieved using lest squares support vector machines in the last session.

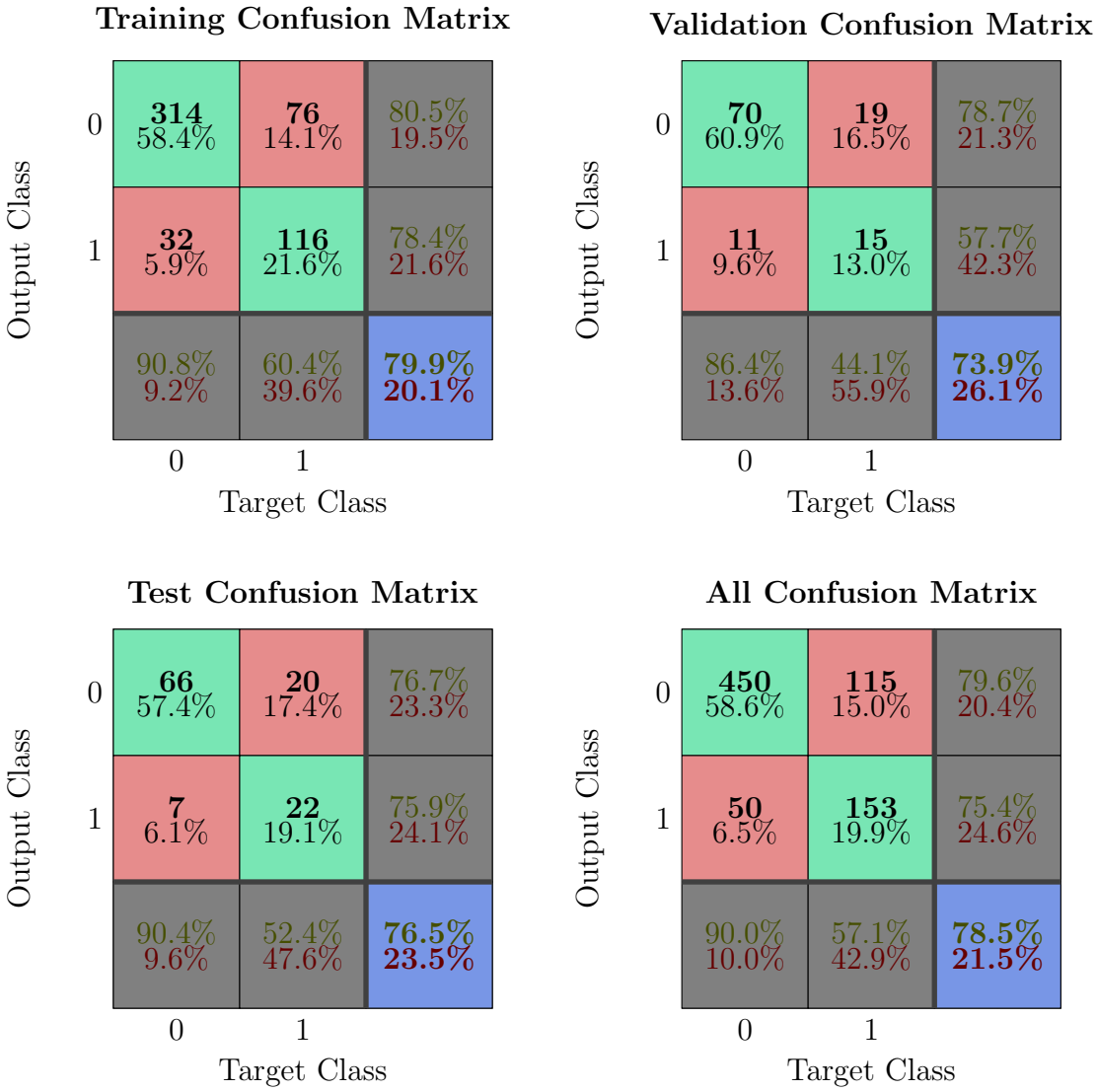


Figure 2.12: Classification results on the training, validation and test set as well as combined.

Session 3

Dimension reduction and input selection

3.1 Dimension reduction by principal component analysis

In this section blood sample data will be preprocessed using principal component analysis. The 21x264 input matrix contains twenty-one spectral measurements of 264 blood samples. A network will be trained to identify the levels of three types of cholesterol LDL, VLDL, HDL from the sampled data. The target data is stored in a 3x264 matrix. The dimension reduction process starts by computing the covariance matrix of the given data. A whitened version of the problem which is easier to handle is found trough an eigenvalue decomposition. In order to reduce the dimension only eigenvalues and corresponding eigenvectors larger then a certain threshold value (i.e. 0.001) are selected. The new problem is then given by

$$\mathbf{z} = \mathbf{U}_{trunc}^{T}\mathbf{x}. \tag{3.1}$$

where $\mathbf{U}_{trunc}^{T}$ denotes the eigenvector matrix with vectors corresponding to eigenvalues above the cutoff value. Figure 3.1 shows a plot of the squared and normalized diagonal of the singular matrix and the treshold which separates used and neglected values from each other. For numerical reasons the singular value decomposition of the raw data is preferred over the eigendecomposition of the covariance matrix. Due to the choice of the cutoff value in this case the dimension is reduced from 24 to 4. In order to assess the effect of dimensionality reduction networks will be trained on the normalized data and on inputs that have been reduced as well as normalized. In any case the 264 available data points are evenly split into training and validation/testing data-points. Using the data points $1, 3, 5 \ldots 263$ as training data, the values at $2, 6, 10 \ldots 262$ as test and finally the points at $4, 8, 12 \ldots 264$ as validation data. Thus 50% of the data is used for training purposes, while 25% each is used for validation and testing. Using the data distribution outlined above a network as shown in figure 3.2 on the left with 5 hidden neurons has been trained. With the same amount of hidden neurons the process has been repeated for the data of reduced dimension. In order to avoid over-fitting early stopping is used. An exemplary training process using Levenberg-Marquardt is shown in figure 3.3 on the right. In order to avoid over fitting the training process is stopped, when a mean square error minimum on the validation data set is reached. The test data set serves as a backup

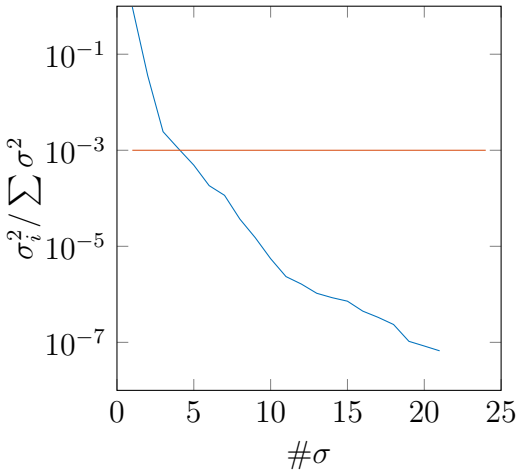


Figure 3.1: A plot of the condition used in the dimension reduction process. Eigenvalues and corresponding eigenvectors of the covariance matrix smaller then a given threshold are cut.

Figure 3.2: Architecture of the cholesterol prediction network trained on the normalized data (left) and layout of the one trained on the reduced and normalized data(right).

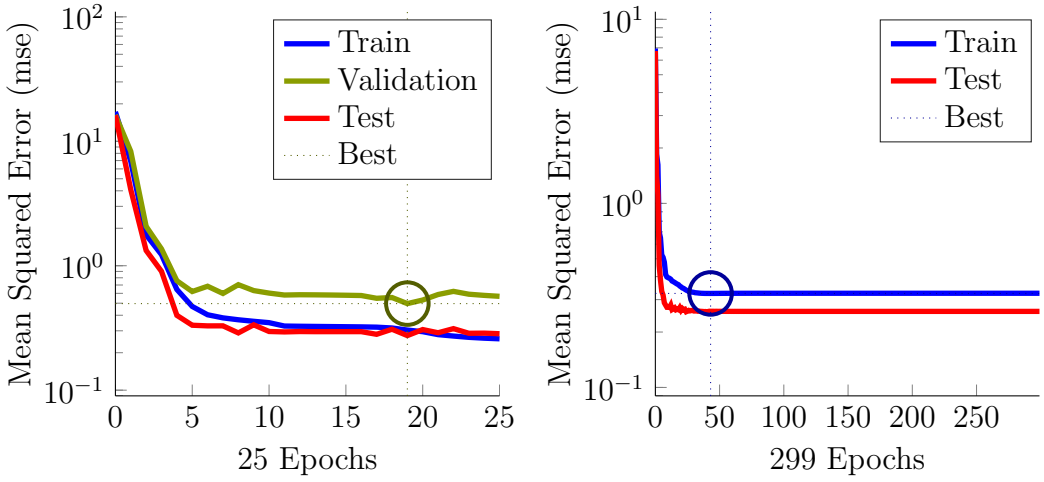


Figure 3.3: Performance plots of training results using Levenberg-Marquardt and Bayesian regularization backpropagation training. On the left the best validation performance is 0.49627 at epoch 19 on the right it is 0.32417 at epoch 43.

	training mse	validation mse	training time [s]
LM normalized	0.3175	0.2852	0.0952
BR normalized	0.3098	0.2573	1.5644
LM normalized, reduced	0.3667	0.3326	0.0548
BR normalized, reduced	0.3462	0.2734	0.2906

Table 3.1: Each row shows the average performance as well as the mean training time of 100 networks. The networks have been trained using Bayesian regularization and Levenberg-Marquardt optimization.

to guard against poor division of training and validation data. If the mean square error minimum is attained on the training data set on a significantly different location the data should be split in a another manner. Table 3.1 reveals that dimension reduction reduces training time at the cost of higher mean square errors, in both training and validation data. Bayesian regularization works by minimizing the cost function

$$\min_{\omega} M(w) = \beta E_D(w) + \alpha E_W(w) \tag{3.2}$$

where E_D is the squared error and E_W the square of the network weigts. α and β are the objective function parameters, where α regulates the importance of reducing the error and β the penalty on the network weights. Keeping the weights small has a smoothing or regularization effect on the network output. Minimizing the objective function given above is equivalent to maximizing the posterior probability $P(\mathbf{w}|D, \alpha, \beta, M)$[1], for a given model M or in this case a network. Another look at table 3.1 shows that Bayesian regularization produces better results then pure Levenberg-Marquardt backpropagation training. This is due to the fact that Bayesian regularization, incorporates Occams razor. When comparing the probability that the observed data has been produced by a given model[2], or equivalently when the value of α is increased. Another issue with pure Levenberg-Marquardt training is the choise of a suitable regularization parameter, which is not necessary when a Bayesian approach is taken. However the pure LM training is significantly faster then Bayesian regularization. Regularization can speed up the training process significantly, in particular in combination with Bayesian regularization where the training process can be completed about five times faster on average (in this example). Bayesian regularization (`trainbr`) should be the algorithm of choice due to the better prediction performance in this case.

3.2 Input selection and automatic relevance detection

3.2.1 `demev`

In the second experiment the file `demev1.m` is run. Here noise distributed according to $\mathcal{N}(0.25, 0.02^2)$ serves as the input. The function is sampled along the input and an

[1]GAUSS-NEWTON APPROXIMATION TO BAYESIAN LEARNING, F. Dan Foresee* and Martin T. Hagan, 1997 IEEE

[2]Cursustekst page 102

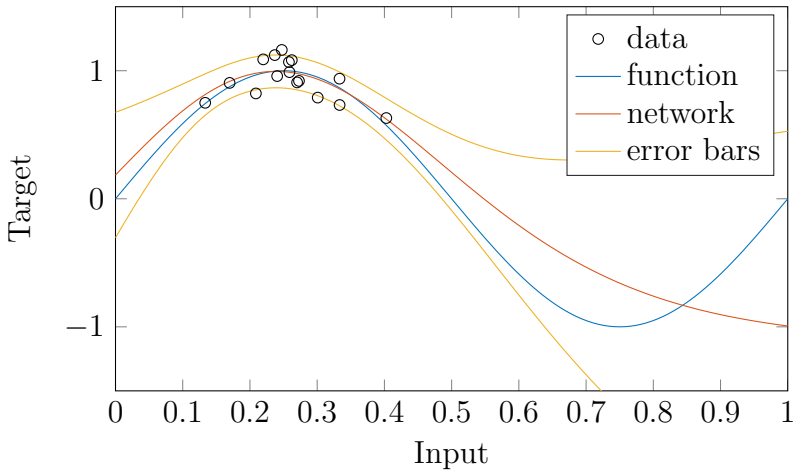


Figure 3.4: `demev1` output.

additional disturbance distributed according to $\mathcal{N}(0, 0.1^2)$ is added. Next a network with two layers and three hidden perceptrons is trained three times. After each iteration, when the training process has reached its minimum, the cost function parameters are re-estimated using a Gaussian approximation[3]

$$a^{new} = \frac{\gamma}{2E_w} \tag{3.3}$$

$$\beta^{new} = \frac{N - \gamma}{2E_d}. \tag{3.4}$$

Where γ is a relative measure of how many of the model parameters are used to reduce the error function and N gives the total number of parameters in the network. This training estimation combination is probably also used in the `trainbr` function. Figure 3.4 shows the result. The estimation works better where numerous data points are available. Error bars are created by along one standart deviation from the mean which corresponds to the network output.

3.2.2 `demard`

In a first experiment an automatic relevance detection demonstration file from the `netlab`[4] toolbox called `demard.m` is executed. In the file a three different input random sets are created. The first one is drawn from the uniform distribution scaled into $[0, 1]$. The second set is a noisy version of the first. Gaussian noise with the distribution $\mathcal{N}(0, 0.02^2)$ is added. The last set are values from the normal distribution $\mathcal{N}(0.5, 0.2^2)$ unrelated to the other two sets. Therefore x_1 is very relevant for estimating the target value, x_2 is somewhat important, yet x_3 may be considered almost irrelevant. Next the targets are found from sampling the function $\sin(2\pi x)$ at the points specified in the first input set x_1. In order to classify the relevance of the three input sets described above an multilayer perceptron is trained and retrained repeatedly, each input-set is now assigned its own hyperparameter α_i. From these the relative importance of each input can be evaluated.

[3] Netlab, Algorithms for pattern recognotion, Ian Nabney page 346

[4] `http://www.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/`

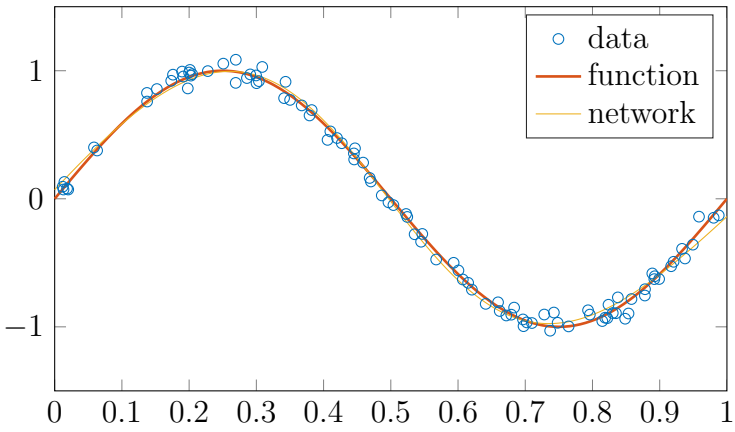


Figure 3.5: `demard` output.

In the case treated in the `demar.m` demo file the parameter values are:

$$\alpha_1 = 0.17555$$
$$\alpha_2 = 21.07742$$
$$\alpha_3 = 153182.97363$$

Which shows that during the optimization process the weight of the regularization term increases as the input data becomes less relevant. An observation that is confirmed by the two layered network weights

$$\mathbf{w} = \begin{pmatrix} -3.18149 & 1.10024 \\ -0.24288 & 0.05027 \\ 0.00119 & -0.00048 \end{pmatrix}$$

where the rows correspond to the different inputs. One observes that the greatest importance is given to values coming from the first set of input values, some to the second and always none to the third. Which aligns with the conclusions drawn from the hyperparemeter values. Thus the network correctly identified the first set as the most important. Figure 3.5 shows the noise samples from the most important input set x_1 as well as the sampled function and its network approximation.

3.2.3 Ionospere data Set

In this subsection classification of the ionosphere data set will be undertaken. The UCI machine learning repository[5] describes the data set as :
"This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. See the paper for more details. The targets were free electrons in the ionosphere. Good radar returns are those showing evidence of some type of structure in the ionosphere. Bad returns are those that do not."
The data set contains 351 data points and 33 inputs. In the following experiments will be concerned with determining the most important inputs. Figure 3.6 shows the procedure

[5] `https://archive.ics.uci.edu/ml/datasets/Ionosphere`

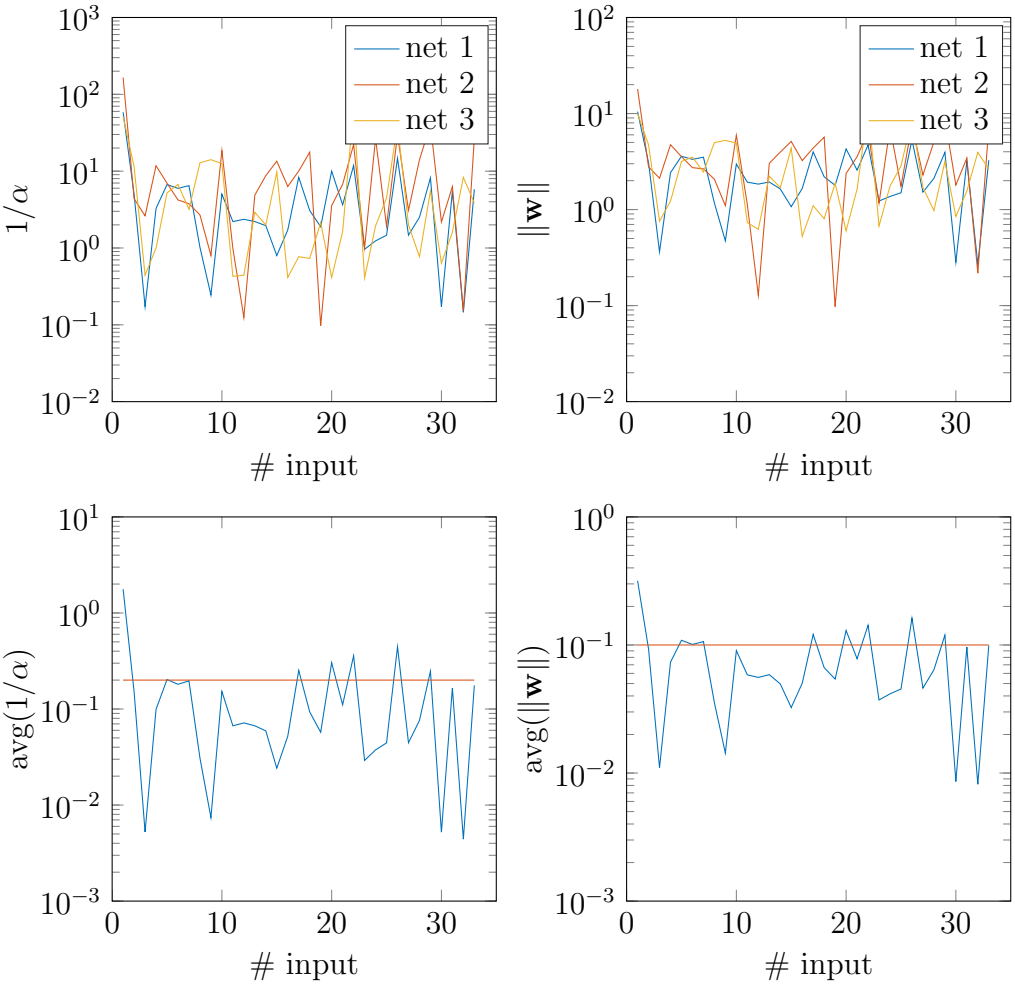


Figure 3.6: Hyperparameter alpha and input layer weights. The top row the results for three different networks trained are depicted. In the bottom row their average values and exemplary cuttoff values are shown.

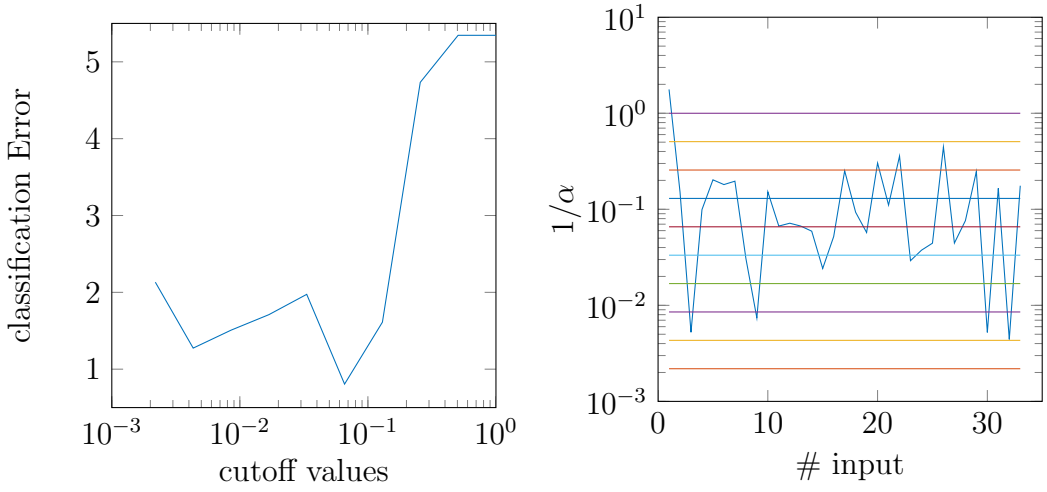


Figure 3.7: Performance of a retrained committee of three networks when various different cutoff values are used.

outlined in the previous section applied to the ionosphere classification problem. Three networks with the same structure as in the previous experiment have been trained. In order to select the most important inputs the inverse of the hyperparameter α and the input weights are considered. The figure 3.6 shows that the obtained values resemble each other. Ten times the input average weight is roughly the values of the inverse parameter $1/\alpha$. In order to systematically asses the effect of neglecting inputs, the network committee will be retrained while increasing the cutoff value for input neglection after every training iteration. The cutoff values used are shown in figure 3.7 on the right. The inputs that remain for every given cutoff value are tabulated in table 3.2. The classification error in relation to the cutoff value is shown in figure 3.7 on the left. In order to asses the classification error the comittee output has been run trough a `sign` function, mapping positive values to 1 and negative ones to -1. Here on can observe that it is possible to remove inputs until the cutoff value $10^{-1.822}$, where 20 inputs remain, whitout significantly impeading the comittee performance. When even more inputs are removed the performance decreases significantly.

-2.6600	-2.3644	-2.0689	-1.7733	-1.4778	-1.1822	-0.8867	-0.5911	-0.2956	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2			
3	3								
4	4	4	4	4	4				
5	5	5	5	5	5	5			
6	6	6	6	6	6	6			
7	7	7	7	7	7	7			
8	8	8	8						
9	9								
10	10	10	10	10	10	10			
11	11	11	11	11	11				
12	12	12	12	12	12				
13	13	13	13	13	13				
14	14	14	14	14					
15	15	15	15						
16	16	16	16	16					
17	17	17	17	17	17	17			
18	18	18	18	18	18				
19	19	19	19	19					
20	20	20	20	20	20	20	20		
21	21	21	21	21	21				
22	22	22	22	22	22	22	22		
23	23	23	23						
24	24	24	24	24					
25	25	25	25	25					
26	26	26	26	26	26	26	26		
27	27	27	27	27					
28	28	28	28	28	28				
29	29	29	29	29	29	29			
30	30								
31	31	31	31	31	31	31			
32	32								
33	33	33	33	33	33	33			

Table 3.2: Logarithms of the cutoff values as shown in figure 3.7 and corresponding inputs fed into the network.

Session 4

Density estimation and self organizing maps

4.1 Density Modeling and Clustering

4.1.1 The expectation maximization algorithm

The first experiment considered comes from the `demgmm1.m` file which is part of the Netlab toolbox. In the experiment 40 data points in a 2-dimensional space are considered. These points have been drawn from a mixture of 2 Gaussian distributions, with means (0.3,0.3) and (0.7,0.7) respectively the common variance is 0.01. The challenge now is that the two initial distributions are considered unknown. In such a case the distributions will be recovered as precisely as possible using the expectation-maximization algorithm. Initially a mixture model containing M components is defined,

$$p(x) = \sum_{j=1}^{M} P(j)p(\mathbf{x}|j). \tag{4.1}$$

In general the mixing coefficients are constrained $\sum_{j=1}^{M} P(j) = 1$ similarly for the component density functions $\int p(\mathbf{x}|j)d\mathbf{x} = 1$ is used. The constraints guarantee that the model represents a density function. When the algorithm starts no assignment of data points to component distributions is available. It will have to be estimated iteratively. The steps this algorithm follows are shown in figure 4.1. In a first step a new Gaussian mixture model is created with their means initially assumed at (0.2,0.8) and (0.8,0.2) and a variance of 0.01 for both distributions, which turns out to be quite a poor initial guess. The distributions are plotted as a circle of one standard deviation around their mean. The following step is an expectation (E) step. During this step the points are assigned to the distributions, which are part of the mixture model based on each distributions probability to generate the data point under consideration. In figure 4.1 this step is shown by coloring the data points according to the distribution, which is most likely to have generated it. Red and blue are used for the two gaussians considered. Points which are equally likely to belong to each of the available distributions are colored with a shade of purple mixing both colors. Each expectation step is followed by a maximization (M)-step. During this step the mean and variance of the distributions under consideration are re-estimated using

$P(j)$	μ_x	μ_y	σ	$P(j)$	μ_x	μ_y	σ
0.2943	1.9941	3.4742	0.0415	0.3000	2.0000	3.5000	0.0400
0.2022	0.1395	1.9759	1.0499	0.5000	0	0	0.2500
0.5036	0.0306	0.0026	0.2451	0.2000	0	2.0000	1.0000

Table 4.1: EM approximation results for the spherical example(left) and the parameters of the three distributions the data was sampled from(right).

the point assignments from the previous expectation step. In figure 4.1 convergence of in total 9 EM cycles can be observed. However only the iterations `1,2, 7` and `10` are shown.

4.1.2 Gaussian mixture models with spherical,diagonal and full covariance

In a similar experiment now three distributions are to be estimated. The EM-algorithm converges nicely in ten iterations as shown in figure 4.2 and table 4.1. It estimates means, variances and priors nicely. Further examples explore the EM-Algorithms ability to deal with data covariance matrices of various forms. A spherical covariance matrix for example means, that the diagonal entries are the same which will always lead to spherical shaped plot ring of one standard deviation around the mean. If a diagonal covariance matrix is chosen the entries on the diagonal do not have to be the same. This ellipsoid standard deviation plots become possible. Last a full covariance matrix additionally allows it to rotate the ellipsoids around the mean. In all examples the EM-Algorithm is able to find good approximations for mean, covariance and priors.

4.2 Self-organizing maps (SOM)

Self organizing maps are a visualization tool for high dimensional data.[1] To obtain a self organizing map a distribution of high dimension is mapped to a grid, which is often two by two. Self organizing maps compress information, while at the same time preserving the most important topological and metric relationships. SOMs often consist of two dimensional grids of nodes. The nodes are organized in such a way, that similar models are close to each other.

The model vectors are updated according to the process below,

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{c(x),i}(\mathbf{x(t)} - \mathbf{m_i}(\mathbf{t})). \tag{4.2}$$

To find $h_{c(x),i}$ the winning model has to be used which may be found from

$$\forall i \|\mathbf{x}(t) - \mathbf{m}_c(t)\| \leq \|\mathbf{x}(t) - \mathbf{m}_i(t)\| \tag{4.3}$$

which is saying that one is primarily interested in using the one model, that is closest to $\mathbf{x}(t)$. Therefore it is also called the winning model. If multiple winners occur, which is

[1]This paragraph is based on: The self-organizing map Teuvo Kohonen, Helsinki University of Technology, Neural Networks Research Centre, P.O. Box 2200, FIN-02015 HUT, Neurocomputing 21 (1998) 16

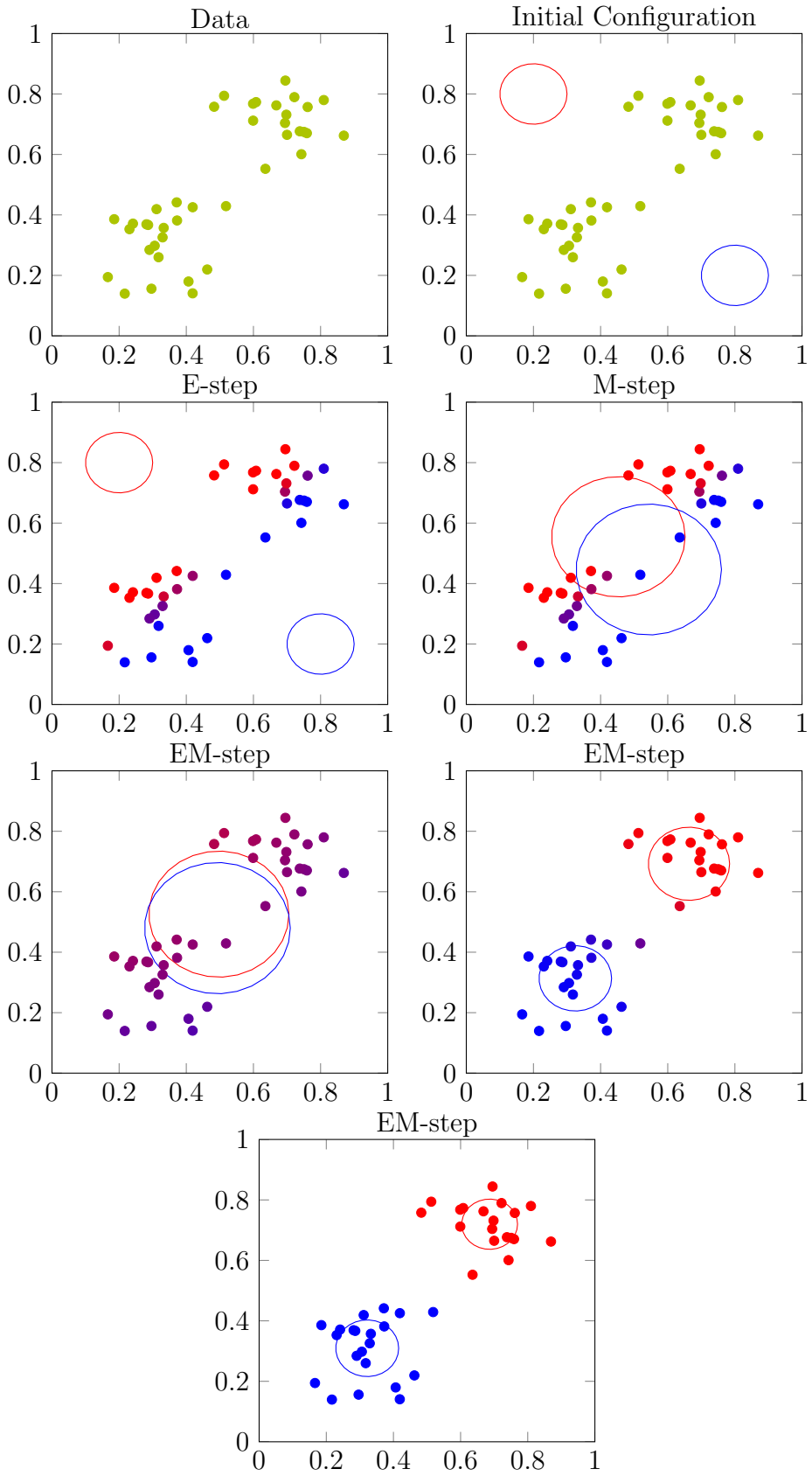


Figure 4.1: Maximum likelihood fitting of a mixture of Gaussians to a data set, also called EM (expectation-maximization).

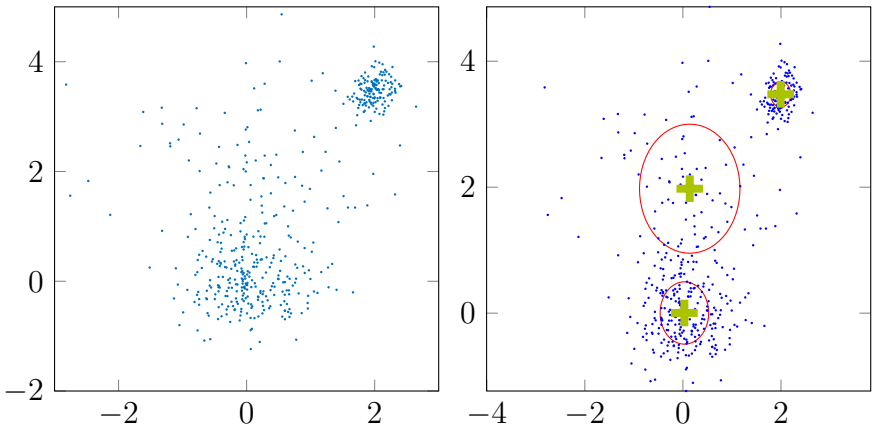


Figure 4.2: Spherical Gaussian mixed model example.

possible for discrete-valued variables the winner should be selected at random. A simple definition for the neighborhood function is

$$h_{h_c(x),i} = \alpha(t) \text{ if } \|\mathbf{r}_i - \mathbf{r}_c\| < c \tag{4.4}$$

$$h_{h_c(x),i} = 0 \text{ else.} \tag{4.5}$$

Which means the we the neighbourhood is a set of points within a radius c. In order to set up a batch map procedure the concept of the Voroni set is needed. It is the set of those samples $\mathbf{x}(t)$ that lie closest to the model $\mathbf{m}_i$. N_i is the set of nodes within a certain radius around node i in the array. The union of the Voroni sets V_i of to the nodes in N_i is called U_i. Ultimately it containts the data values $\mathbf{x}_i$ associated to the winning model and those models close to it. The model can then be defined as an average

$$\mathbf{m}_i = \frac{\sum_{\mathbf{x}(t)\in U_i} \mathbf{x}(t)}{n(U_i)} \tag{4.6}$$

where $n(U_i)$ denotes the number of elements in U_i. The training procedure can then be described by [2]

1. Initialize the models $\mathbf{m}_i$. Random vectors will work, but the process is faster if values from the data eigenspace are chosen.
2. Go trough the data points $\mathbf{x}(\mathbf{t})$, and list each one under its closest model $\mathbf{m}_i$.
3. Use $\mathbf{m}_i = \frac{\sum_{\mathbf{x}(t)\in U_i} \mathbf{x}(t)}{n(U_i)}$ to update the models.
4. Loop from step two until a steady state is reached.

The result of the organization process can be seen in figure 4.3. The net clearly moves into the data set it is supposed to represent. It initially resides within the plane $x \in [-3, 3], y \in [-3, 3]$, throughout the training process it nicely expands to cover the whole domain of the mexican hat function which resides in $x \in [-5, 5], y \in [-5, 5]$. Additionaly the map is able to nicely emulate the behavior of the hat in the z domain. After training the quantization error has decreased to `q = 0.4` from `q0 = 0.88`.

[2] based on: The self-organizing map Teuvo Kohonen, Helsinki University of Technology, Neural Networks Research Centre, P.O. Box 2200, FIN-02015 HUT, Neurocomputing 21 (1998) 4

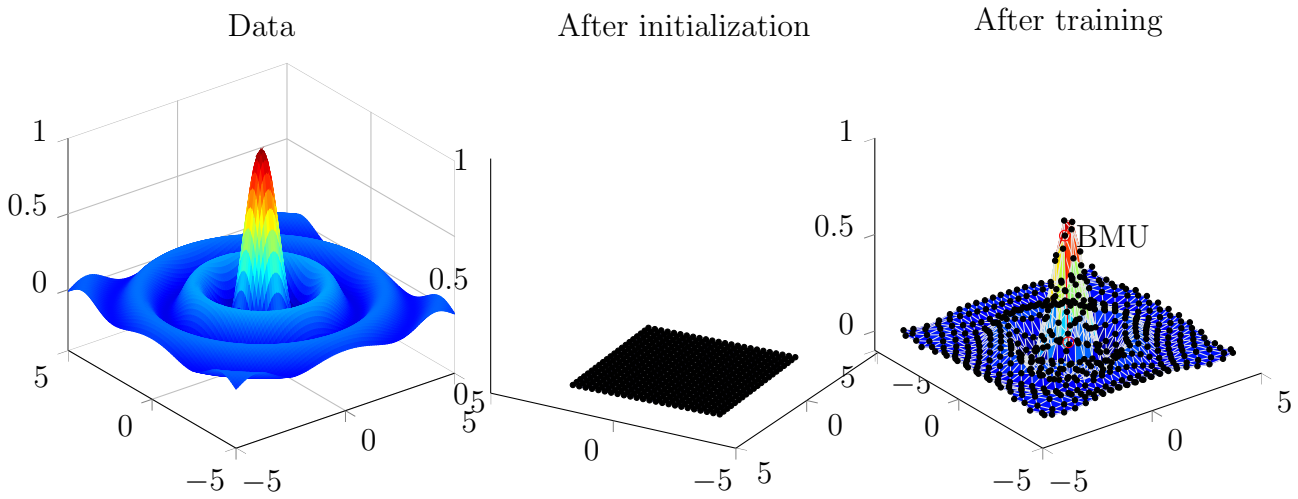


Figure 4.3: An example of the SOM organization process. Based on `som_demo1.m` from the som toolbox.

4.2.1 The Iris dataset

Yet a more important property of self organizing maps is their ability to visualize high dimensional data. Using the UCI iris data set this will be illustrated in the upcoming experiment .[3] The iris data set consists of 50 samples from three different kinds of Iris flowers. This the set contains 150 samples in total. For each sample the width and hight of the sepal and petal leaves have been measured. The data set labes all the measurment according to the name of each species, "Setosa", "Versicolor" and "Virginica". Figure 4.4 shows that self organizing maps are able to distinguish the different kinds of plants from another fairly well. Without knowing which plant would belong the which data point, using the map it would have been at least possible to separate the "Setosa" plants from the others. Using a supervised training process the results are better. In figure 4.5 the classification map is shown. The color bar shows the error on the different nodes. A comparison to figure 4.4 shows that the nodes where the plant data is not mixed are classified correctly.

4.2.2 The burpa dataset

The bupa dataset named after the donating company BUPA Medical Research Ltd., contains blood and alcohol consumption data from 345 men.[4] For every set of data points it is known whether a liver condition is present or not. 1 indicates a healthy liver, 2 points to the contrary. Figure 4.6 contains denormalized data plots for the U-matrix on the very left and component planes for each measured value. It is important to note that the U-matrix has more hexagons then the component planes, because distances are shown in relation the neighboring cells. The component plane plots use lesser hexagons as they only show absolute values. The component planes reveal that the two blood values `sgpt` and `sgot` are highly correlated, both are somewhat correlated to the `gammagt` blood value. The U-matrix can be divided in two regions a large plain at the top and a somewhat more

[3] `https://archive.ics.uci.edu/ml/datasets/Iris`
[4] `http://archive.ics.uci.edu/ml/datasets/Liver+Disorders`

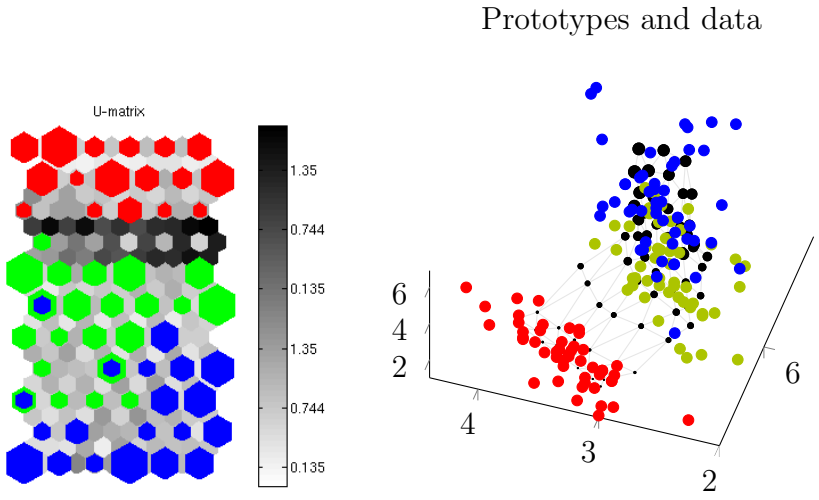


Figure 4.4: An example of the SOM analysis of the UCI setosa dataset. On the left the two dimensional visualization of the self organizing map. The colors indicate classification of the different plant types red for "Setosa", green for "Versicolor" and blue for "Virginica". On the right the same map is shown inside the input space.

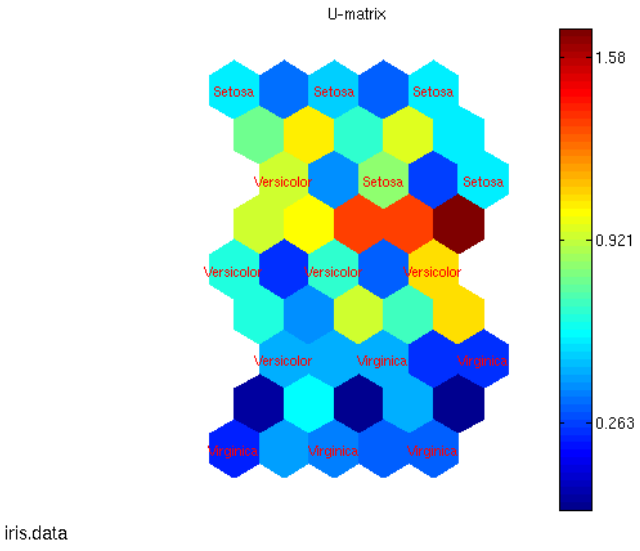


Figure 4.5: Iris classification results using supervised training.

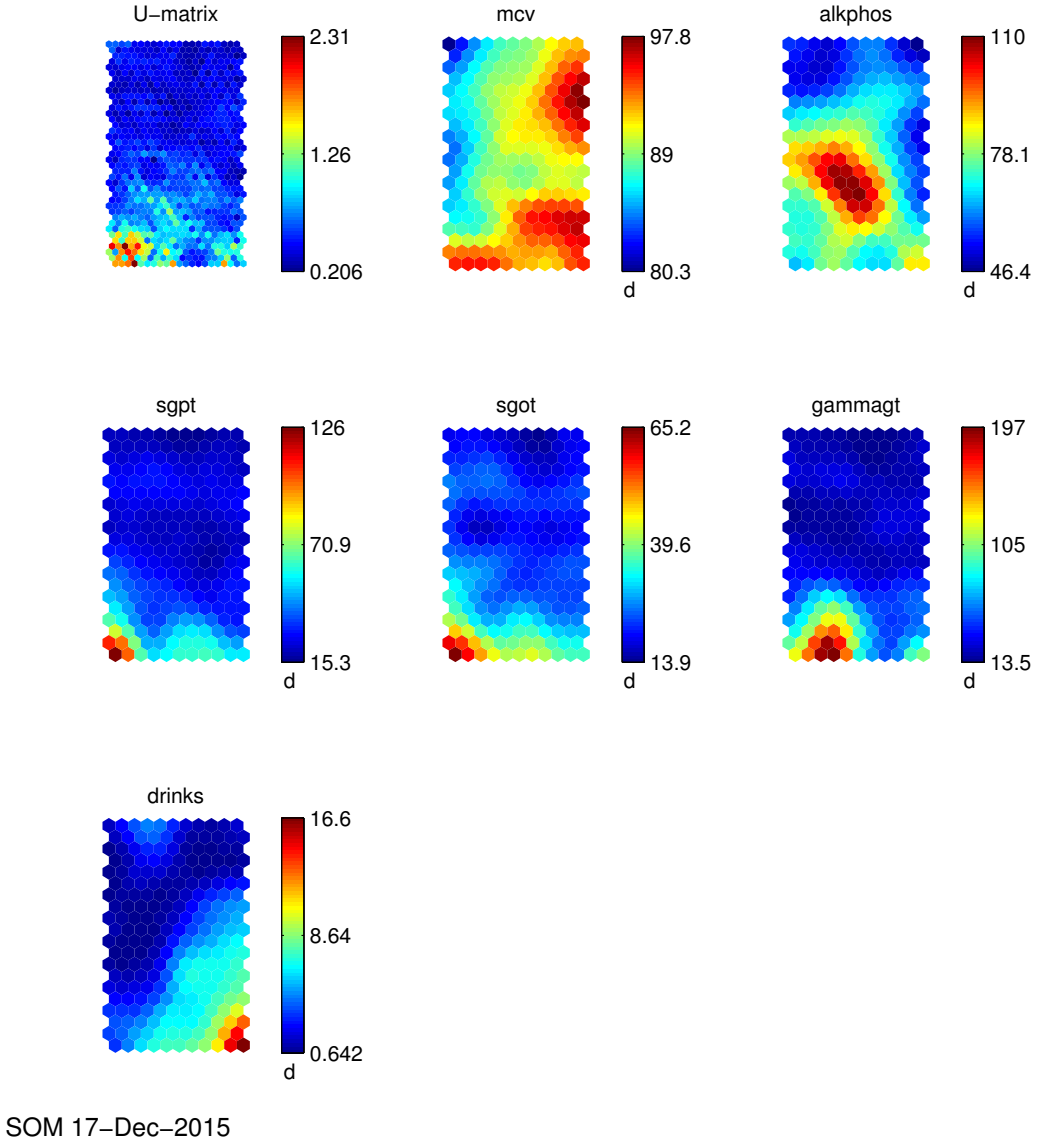


Figure 4.6: Self organizing map plots for the bupa dataset.

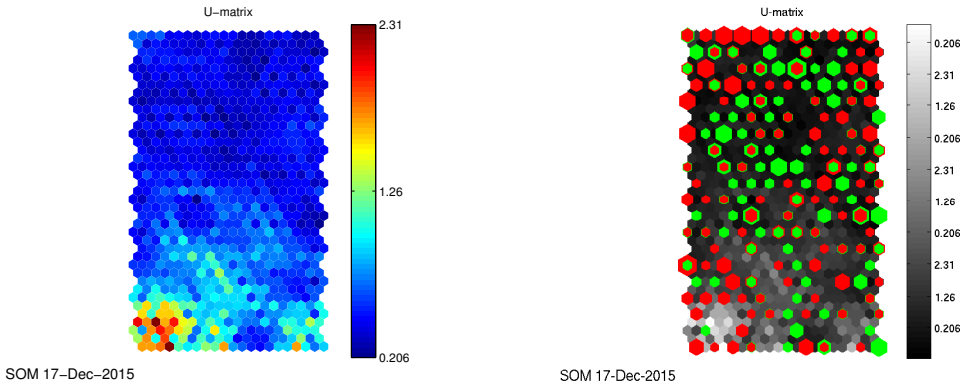


Figure 4.7: U-matrix without and with data labels. On the left green labels indicate data from healthy and red from unhealthy patients.

mountainous region at the bottom. Elevated `sgpt`, `sgot` and drinking are characteristics of the bottom cluster. `mcw` and `alkphos` don't seem too related to the culstering. Adding the classifications on top of the U-matrix gives additional information. Figure 4.7 shows the original matrix and a copy with added labels side by side. Contrary to what one would expect the unhealthy livers are not located in the bottom region, but spread out over the whole plane. Figure 4.8, explores the notion of distance further. Similarity coloring is used to make sure similar nodes are given a analogous color. The bottom left region of the U-matrix (shown in yellow) seems to be particularly distant from the others. Which is not surprising as the elevated values for `sgpt` and `sgot` are found nowhere else. It remains to investigate the effects of various map sizes on the accuracy. Here data representation accuracy and topology quality have to be considered. Figure 4.9 shows surface plots for measures of both, for maps sizes ranging from two to thirty. It follows that the larger the map the better the quality will be. However training time has not been taken into account, which would lead into the other direction. As computational effort increases quadratically with the number of neurons chosen. In general it is desirable that the width and the hight of the map at least roughly correspond to the number of significant principal components of the input data set.[5] For the bupa data set the singular values are `1.0e03 * (2.3853, 0.9876, 0.3185, 0.2636, 0.1128, 0.0572)`, which means at least five of not six significant directions. And indeed, the representation accuracy falls under one for more then six rows and columns. Another way to chose the neurons is to set their number to $b = 5\sqrt{N}$.[6] In this case this would mean 93 neurons for $N = 345$. Which is in this case a little stricter then taking at least as many rows and columns as principle components.

[5] `http://www.scholarpedia.org/article/Kohonen_network`
[6] Suykens, Cursustekst Data Mining and Neural netowrks, page 116

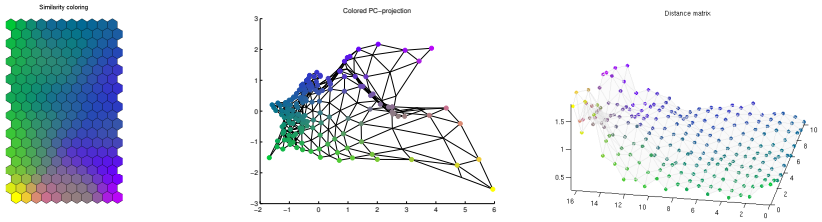


Figure 4.8: Similarity coding colored principal component projection and distance surface plot with colored nodes. All nodes are colored using similarity coloring.

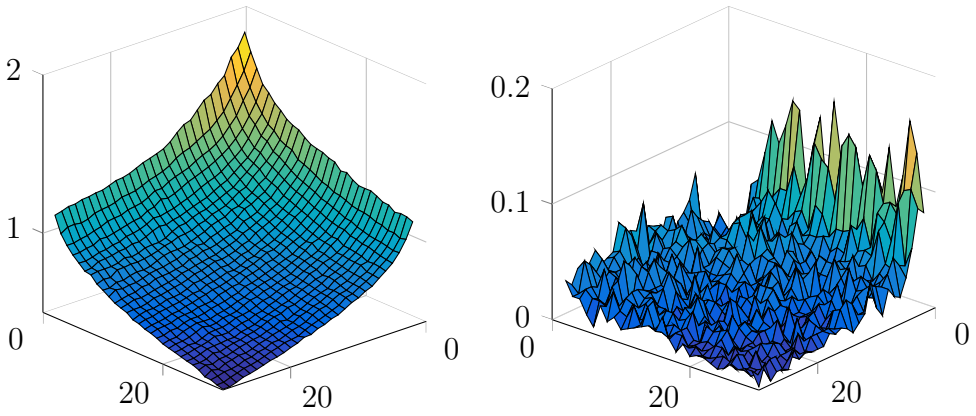


Figure 4.9: Date representation accuracy and data set topology representation accuracy.

Session 5

Support vector machines

5.1 Vapnik Support vector machines

Neural networks have various nice properties for example their universal approximation ability. Unfortunately drawbacks also exist. Most prominently the existence of many local minimal solutions. Despite the fact that these are often good approximations, these minima sometimes make it hard to reproduce results if a previous solution cannot be found again. Thus it could be beneficial to look into other algorithms. In this last assignment classical Vapnik support vector machines will be considered. At the hart of vapnik's theory is the optimal hyperplane algorithm.[1] In the linear the hyperplane condition is given as

$$y_k[\mathbf{w}^T\mathbf{x}_k + b] \geq 1, \quad k = 1, \ldots, N \tag{5.1}$$

The next step is to formulate the optimization problem

$$\min_{w,b} \frac{1}{2}\mathbf{w}^T\mathbf{w} \text{ such that } y_k[\mathbf{w}^T\mathbf{x}_k + b] \geq 1. \tag{5.2}$$

Formulating the Lagrange dual, and taking the gradient of $\mathcal{L}(\mathbf{w}, b; \alpha)$ with respect to $(\mathbf{w}, b)$ leads to a problem in α. The lagrange multipliers are called α, but in this context they will be called support vectors of the solution. From an optimization point of view, the support vectors are Lagrange multipliers with active set indices. A problem reformulation as

$$y_k[\mathbf{w}^T\phi(\mathbf{x}_k) + b] \geq 1 \tag{5.3}$$

allows for different kernel options. The classifier is given by

$$y(\mathbf{x}) = \text{sign}(\sum_{k=1}^{N} \alpha_k y_k K(\mathbf{x}, \mathbf{x}_k) + b). \tag{5.4}$$

The following three kernels will be considered here[2]

$$K(\mathbf{x}, \mathbf{x}_k) = \mathbf{x}_k^T\mathbf{x} \qquad \text{(linear SVM)}, \tag{5.5}$$

$$K(\mathbf{x}, \mathbf{x}_k) = (\mathbf{x}_k^T\mathbf{x} + \eta)^d \qquad \text{(polynomial SVM)}, \tag{5.6}$$

$$K(\mathbf{x}, \mathbf{x}_k) = \exp(-\|\mathbf{x} - \mathbf{x}_k\|_2^2/\sigma^2) \qquad \text{(RBF Kernel)}. \tag{5.7}$$

[1] Suykens, Data Mining and Neural Networks, Cursustekst page 134.
[2] Suykens, Data Mining and Neural Networks, Cursustekst page 140.

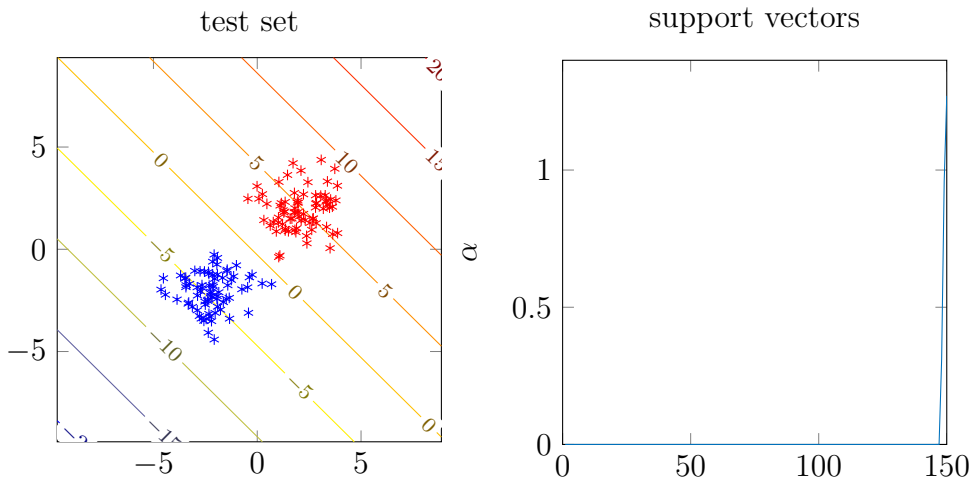


Figure 5.1: Linear Vapnik-SVM classification of separable data (left) and support vectors (right).

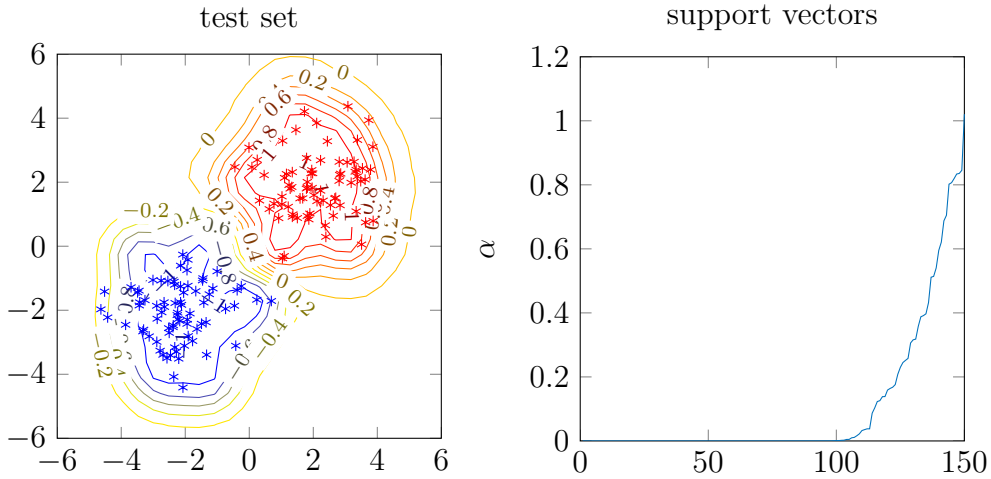


Figure 5.2: Gaussian Vapnik-SVM classification of separable data and (left) support vectors (right).

If the problem is not separable, it means that classification cannot be done without error. In the underlying optimization problem slack variables have to be included in the formulation. Figure 5.1 shows classification results using a linear SVM on a separable data set. It consists of two normally distributed data sets with distributions $\mathcal{N}((2,2)^T, \mathbf{I})$ and $\mathcal{N}((-2,-2)^T, \mathbf{I})$. In total 150 values are sampled from both distributions. Sampling was done twice first for the training set and once more for the test set. The linear SVM classifies the two sets correctly. In this case without miss-classification of a single data point. Two achieve this results the machine uses three support vectors α. Using a more complex radial basis function (RBF) kernel might seem promising due the its similarity to the gaussian distribution, where the values have been sampled from. However as this problem is separable the linear classifier was already sufficient. In this case the more complex RBF kernel is overmodelling the problem, which can be seen in by the fact, that is utilizes almost 100 support vectors. Nonetheless it is able to correctly classify all points. The harder problem is the case where not all points can be classified without error. A slight modification of the distributions to $\mathcal{N}((0.5,0.5)^T, \mathbf{I})$ and $\mathcal{N}((-0.5,-0.5)^T, \mathbf{I})$ produces

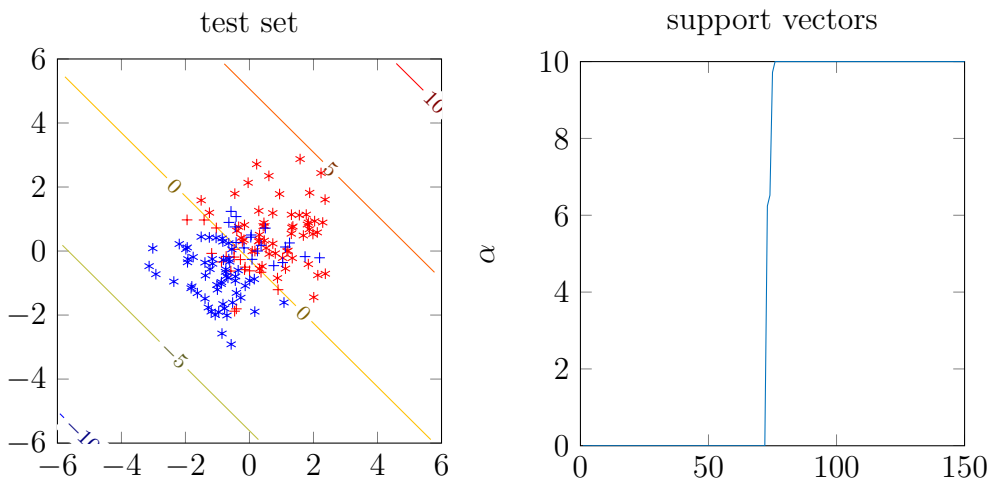


Figure 5.3: Linear Vapnik-SVM classification of indivisible data (left) and support vectors (right).

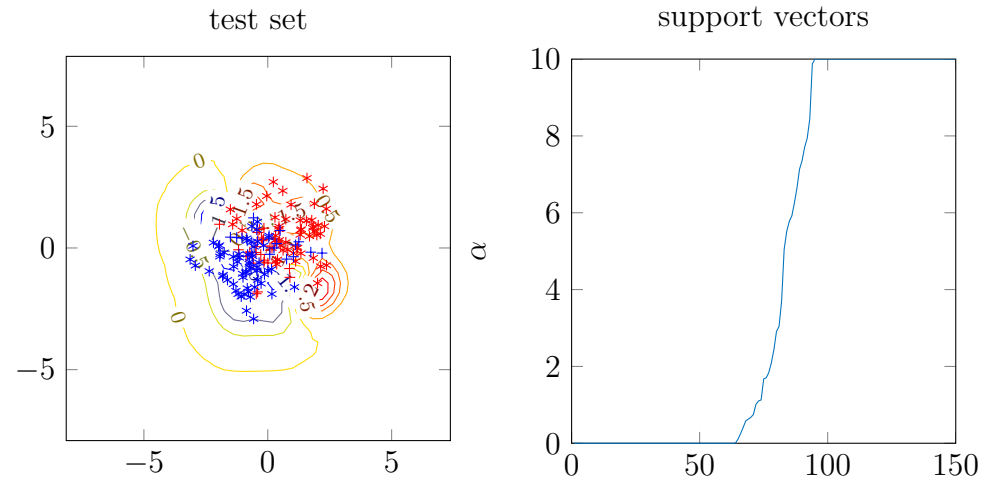


Figure 5.4: RBF Vapnik-SVM classification of indivisible (left) data and support vectors (right).

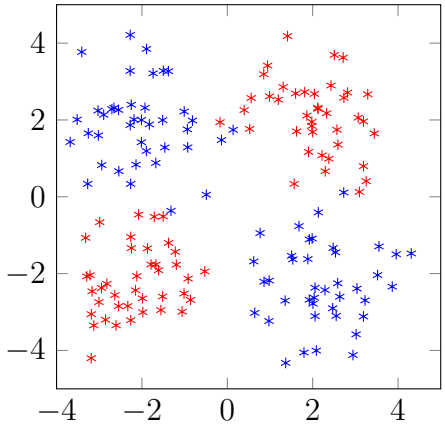


Figure 5.5: An even harder problem sampled from four normal distributions.

such a scenario. Figure 5.3 shows the performance of a linear SVM faced with such an inseparable problem. On the training set it misclassified 35 points or 23.3% of the data. On the test set again 35 points are misclassified. In this experiment using a polynomial kernel did not improve matters. 40 points or 26.7% where incorrectly assigned during training and 37 points or 24.7% on the test data. The gaussian kernel led to improvements during training where only 17.3% of the data was incorrectly assigned to the wrong group however this effect did not reoccur during testing, where 35 data points or 23.3% where wrong.

A more complicated problem can be created by sampling four normals with distributions $\mathcal{N}((2,2)^T, \mathbf{I})$, $\mathcal{N}((-2,-2)^T, \mathbf{I})$, $\mathcal{N}((2,-2)^T, \mathbf{I})$ and $\mathcal{N}((-2,2)^T, \mathbf{I})$. The first two are placed in a first group and the data from the last two normals is assigned to a second. This way the data shown in figure 5.5 has been obtained. Using 160 data points and sampling again two times for different training and test sets, three Vapnik-SVMs have been trained for classification. If a linear Kernel is used the result shown in figure 5.6 is obtained. Results are catastrophic 68.8% of the data is incorrectly classified during training and 65.0% during testing. The Linear kernel is too simple for the complexity of the problem. The high number of support vectors confirms this conclusion. A plot of the results is given in figure 5.6. A more complex polynomial kernel delivers significantly better results. 2.5% of the available data points are incorrectly classified during training and 4.4% during testing. Figure 5.7 shows the results which are satisfactory overall. Finally figure 5.8 depicts the results obtained when a RBF-Kernel is used. In this case all training data points are classified correctly, but during testing the missclasification rises to 8.1%. Which indicates an over-fitting problem.

So far the box constraint c, which sets the upper limit for the magnitude of any α has been kept constant at 10. c determines the penalty of miss-classification during the training process. Reducing c allows the optimization algorithm to allow more missclasification in order to increase the margin between the involved classes. Thus allowing for greater generalization. And indeed if c is decreased to $c = 1$, the training misclassification rises to 1.9% while test misclassification falls to 5.0%. Related plots are shown in figure 5.9. Please note an interesting detail. The miss-classified points (indicated by +) at the outer edges have disappeared. Miss-classifications now only occur in the center where the distributions overlap more frequently. In the polynomial case nothing changes if c is set to one. But here no over-fitting problem existed in the first place.

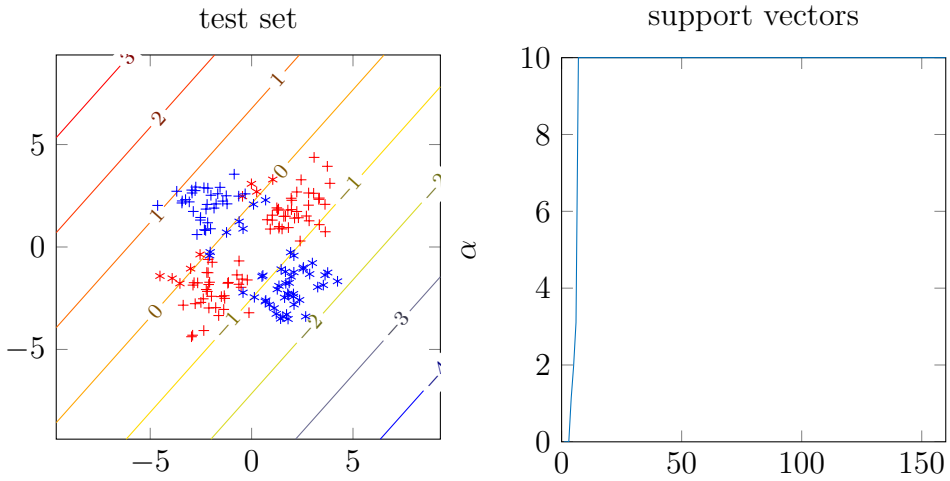


Figure 5.6: Linear Vapnik-SVM classification of the four distributions problem .

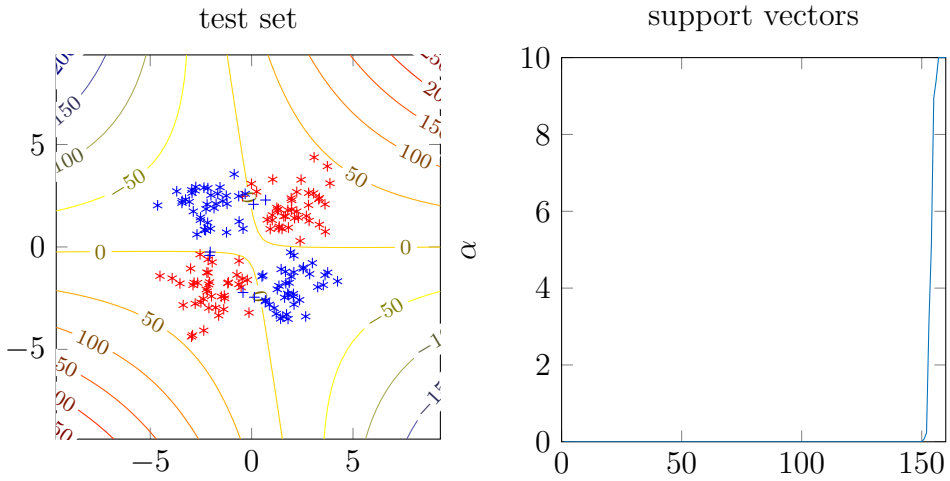


Figure 5.7: Polynomial Vapnik-SVM classification of the four distributions problem.

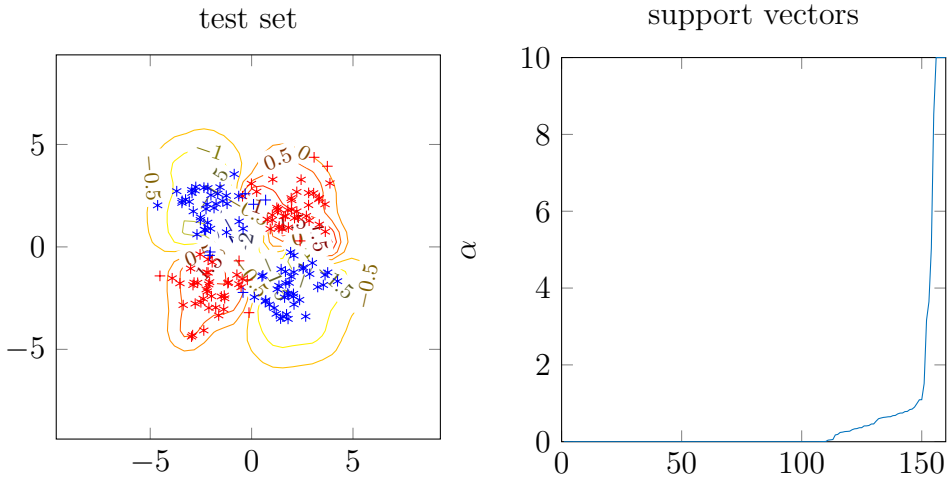


Figure 5.8: RBF Vapnik-SVM classification of the four distributions problem.

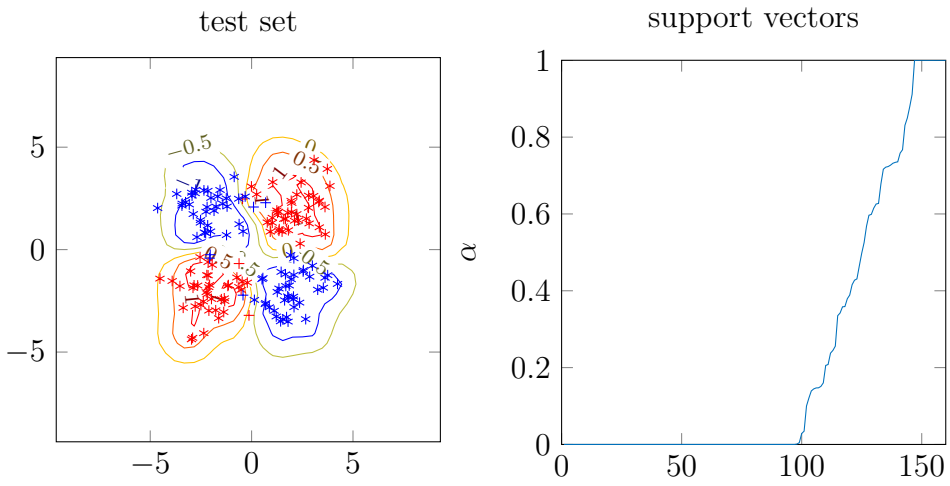


Figure 5.9: RBF-Vapnik SVM classifiaction of the 4 distributions problem with $c = 1$.

5.2 Least-squares support vector machines

SVMs can also be formulated in a least squares sense. This extension leads to a linear system without loosing the advantages of the standard SVM formulation.[3] Additionally equality constraints are used instead of inequalities. As a consequence a linear system yields the solution instead of a quadratic program. Another nice feature is that the support vectors are proportional to the errors in the least squares case. [4]

5.2.1 LS-SVM - Diabetes classification

Once more the Pima diabetes classification data set is considered.[5] Least squares support vector machines with a RBF-Kernel will be utilized. Two thirds of the total 768 available data sets are used for classification. The last third is set aside for verification. To start the training process two parameters σ^2 and γ have to be chosen. When γ is set to a low value minimizing of the complexity of the model is emphasized. Choosing a high value for γ emphasizes good fitting of the machine to the training data points. Increasing the RBF kernel bandwidth given by σ^2 has a smoothing effect on the model. In order to choose these parameters correctly the least squares support vector machine classification performance on the validation data is shown in figure 5.10. The optimal value at $\gamma = 30$ and $\sigma^2 = 13$ is marked with `x`.

5.2.2 LS-SVM - Santa Fee prediction

Finally the Santa Fe Laser data prediction problem will be solved using ls-svms. Before training the data $\mathbf{x}$ have been mapped into the interval $[-1, 1]$. Additionally the data has been placed into windows using the ls-svm toolbox function `windowize`.[6] Trough tuning of the two parameters $\gamma = 30$ and $\sigma = 10$ have been determined as good ls-svm

[3] Sykens, Data Mining and Neural Networks, Cursustext, page 146

[4] J.A.K. SUYKENS and J. VANDEWALLE, Least Squares Support Vector Machine Classifiers, 1999 Kluwer Academic Publishers, pages 294 and 299

[5] `https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`

[6] `http://www.esat.kuleuven.be/sista/lssvmlab/`

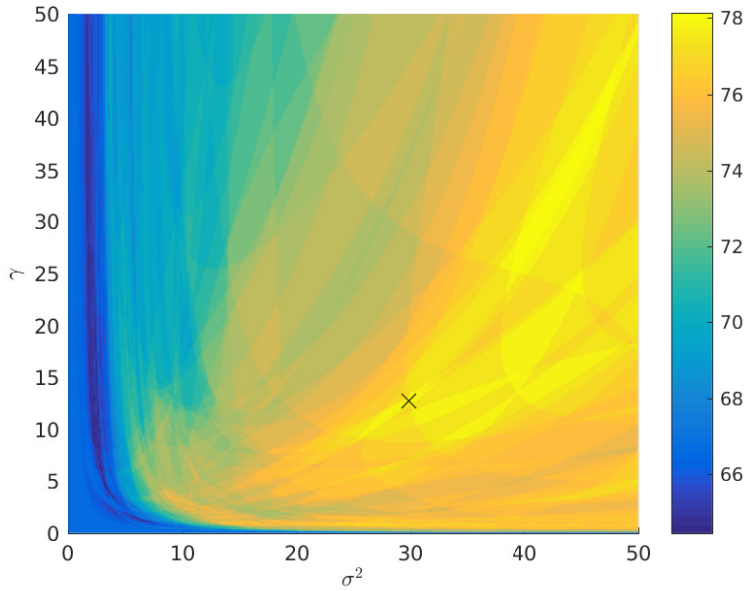


Figure 5.10: Correct diabetes classifincation in percent for various sigma and gamma values ranging from one to fifty.

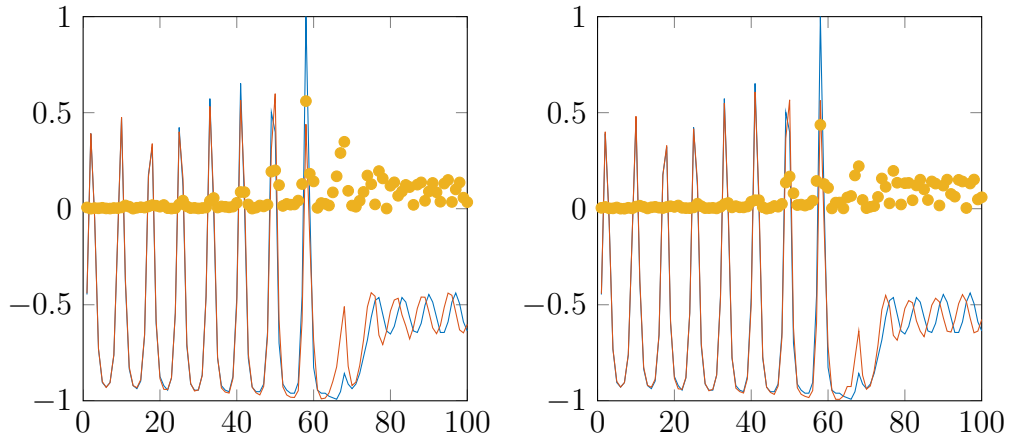


Figure 5.11: Santa Fe data prediction using $\gamma = 10$ and $\sigma = 10$(left) and $\gamma = 30$ and $\sigma = 10$(right).

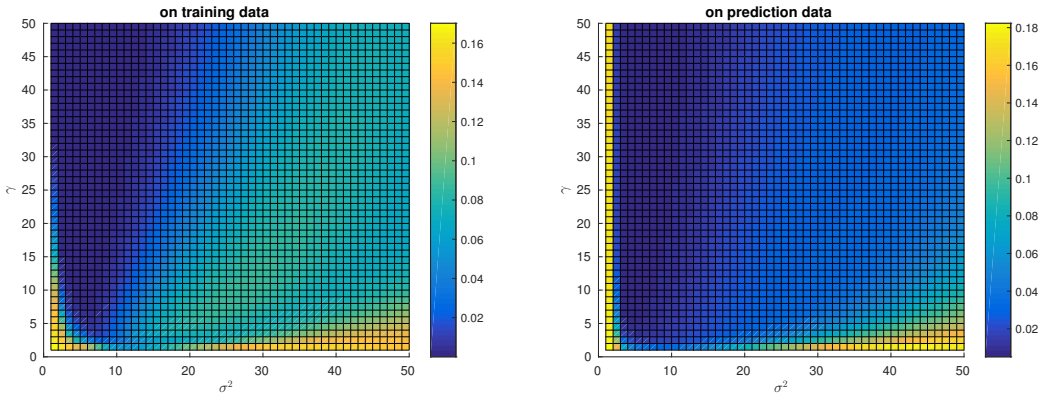


Figure 5.12: Least square estimation error on training and prediction data for various ls-svm parameters.

parameters. Figure 5.11 shows the svm prediction performance which rivals that of the neural committee network used earlier, but it came cheaper in terms of computational cost. Looking at figure 5.12 the parameter choice $\gamma = 30$ and $\sigma = 10$ can be confirmed to be a decent choice. During training only the left plot is available. Knowing that placing too little emphasis on reducing the model order one will not choose a very high γ and a small kernel density σ^2 due to the overfitting problem. The yellow bar on the left in the plot on the prediction data mean squared error confirms over-fitting for the top right corner.