Statistical Methods for Machine Learning
Assignment 2: Basic Learning Algorithms

Philip Pickering
pgpick@gmx.at

Marco Eilers
eilers.marco@googlemail.com

Thomas Bracht Laumann Jespersen
ntl316@alumni.ku.dk

1 Neural Networks

1.1 Neural Network implementation

Implement a multi-layer neural network with linear output neuron and a single hidden layer with non-linear neuron. All neurons should have bias (offset) parameters.

To find the derivative of the activation function:

$$\sigma(u) = \frac{|u|}{1+|u|}$$

we apply the quotient rule for differantiation:

$$\frac{d}{du}\frac{f(x)}{g(x)} = \frac{g(x)f'(x) - g'(x)f(x)}{[g(x)]^2}$$

where, in our case $f(u) = |u|$ and $g(u) = 1 + |u|$.

$$\frac{d}{du}\left(\frac{|u|}{1+|u|}\right) = \frac{(1+|u|)\cdot 1 - (0+1)|u|}{(1+|u|)^2} \tag{1}$$

$$= \frac{1+|u|-|u|}{(1+|u|)^2} \tag{2}$$

$$= \frac{1}{(1+|u|)^2} \tag{3}$$

Implement backpropagation to compute gradient of error with respect to the network parameters.

1.2 Neural Network training

Training of our neural network model starts from the file `nntrain.m`. This file takes care of training the model and plotting the results.

The MATLAB function `multiBatchTrain` performs a given number of batch runs, meaning for each run we do a batch training round of forward and backward propagations, accumulate the partial derivatives and do the weight modification. In the training we do 500 iterations of 50 runs for a total of 25,000 iterations. Fig. 1 plots $\frac{\sin x}{x}$ on the range $[-10, 10]$ along with the predictions of our trained NN model for different learning rates and the training data.

Fig. 1(a)–(b) show the neural networks of two hidden neurons. In fig. 1(a) the result when shortcuts are omitted and fig. 1(b) the same model trained using shortcuts.

In fig. 1(c)–(b) is shown neural networks of 20 hidden neurons, where fig. 1(c) is without shortcuts and fig. 1(d) is with.

We found it instructive to include the models trained with and without shortcuts, because in both cases the model using shortcuts performs better on the same learning time independently of the learning rate.

It is also clear that the highest learning rate possible (which didn't produce numerical overflows), gave better solutions on the same learning time.

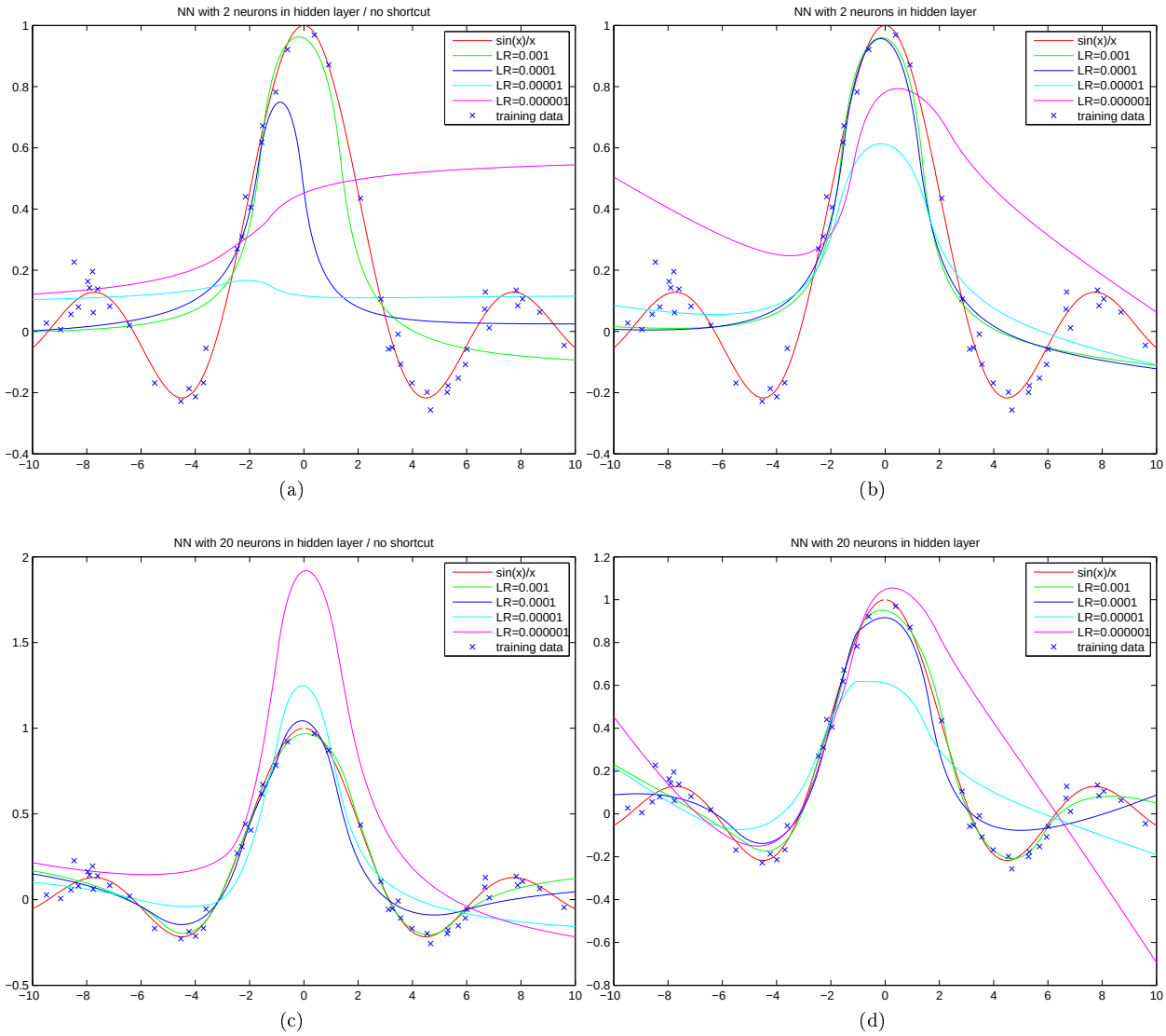

Figure 1: Neural networks for 2 and 20 hidden neurons.

Fig. 2 shows the error trajectories for our different neural networks. A learning epoch on the x-axis corresponds to 50 iterations of batch training, meaning when $x = 100$, a total of 5000 iterations have been performed.

Interestingly, when shortcuts are omitted, the error rate decreases much faster than is the case when shortcuts are in use. One argument for why this is, could be that not using shortcuts leads to a simpler model that is faster to train (but obviously less flexible as demonstrated by fig. 1).

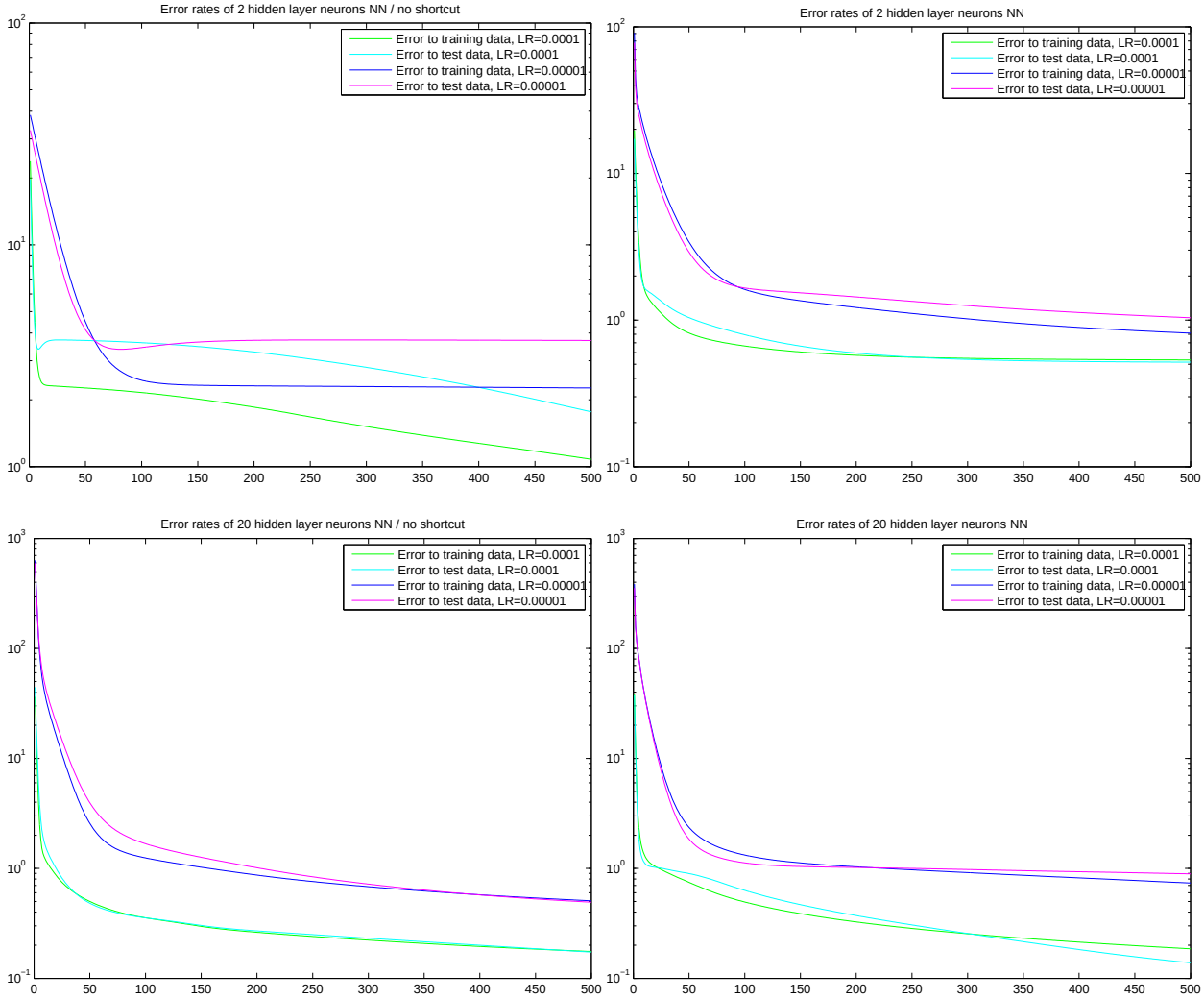

Figure 2: Error trajectories for the different NN models

2 Support Vector Machines

For this part of the assignment we chose to use the LIBSVM library.

2.1 Model Selection

The LIBSVM manual recommends to try an exponentially growing sequence of numbers for γ. Since the default value LIBSVM uses is $1/n$ and our n equals (multiples of) 100, 0.001 should likely be included in this sequence. We therefore did the grid search using the following values of $\gamma : \{0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$.

LIBSVM has built-in functionality to perform n-fold cross validation given a command line option. To perform model selection we iterate through all combinations of C and γ and call a function called `crossval`, which invokes LIBSVM to perform a 5-fold cross validation on the current values of C and γ. When performing n-fold cross validation, LIBSVM returns the accuracy, which we use to keep track of the configuration that gives the highest accuracy.

Applied to the testdata, this gives the results in tables 1 to 3. The resulting optimal parameter

configurations are shown in table 4.

γ \ C	0.1	1	10	100	1000	10000
0.0001	55%	55%	55%	55%	55%	55%
0.001	55%	55%	55%	55%	55%	65%
0.01	55%	55%	56%	57%	65%	98%
0.1	60%	60%	59%	63%	98%	95%
1	57%	58%	59%	92%	97%	94%
10	51%	54%	67%	88%	92%	91%
100	52%	47%	76%	76%	77%	77%

Table 1: Table of all results for model selection using grid-search for `knollC-train100`.

γ \ C	0.1	1	10	100	1000	10000
0.0001	49.5%	49.5%	49.5%	49.5%	50.5%	53%
0.001	49.5%	49.5%	49.5%	59.5%	52.5%	78.5%
0.01	50%	50%	51%	51%	75.5%	97.5%
0.1	48%	49.5%	49%	76%	97.5%	96%
1	50%	51%	59%	96%	97%	96.5%
10	54%	52%	86.5%	96%	93%	94.5%
100	57%	62%	90.5%	91.5%	91%	91%

Table 2: Table of all results for model selection using grid-search for `knollC-train200`.

γ \ C	0.1	1	10	100	1000	10000
0.0001	56%	56%	56%	56%	56.25%	60%
0.001	55.75%	55.75%	55.75%	56%	56.5%	96.75%
0.01	53.25%	53.25%	53.5%	57%	96.5%	97.5%
0.1	52.75%	52%	50.5%	95.75%	97.5%	97.5%
1	54.25%	55%	81.75%	97.5%	97%	96.75%
10	55.25%	61%	94%	96.5%	96%	95.75%
100	59.5%	81.5%	92%	93%	92.25%	90.5%

Table 3: Table of all results for model selection using grid-search for `knollC-train400`.

	C	γ	Acc.
`knollC-train100`	1000	0.1	98%
`knollC-train200`	1000	0.1	97.5%
`knollC-train400`	100	1	97.5%

Table 4: Table of results for model selection using grid-search showing the optimal values for C and γ.

If we train the SVM on all training data sets and run the resulting models on all training sets and the test set, we get result in table 5. We can clearly see that all models behave about equally well on the test data and the other training data sets as on their own particular training set, so there seems to be no overfitting the problem. With more than 96% accuracy for all combinations of model and data, the quality

of the prediction is generally very high, however the results of models with larger training data sets tend to slightly surpass those for lower values on n.

Model \ Data	knollC-train100	knollC-train200	knollC-train400	knollC-test
knollC-train100	98%	97.5%	96.75%	97%
knollC-train200	98%	98%	97.5%	98%
knollC-train400	99%	98.5%	97.25%	98%

Table 5: Percentage of correct predictions when applying the model from each training data set to the training and test data.

2.2 Inspecting the kernel expansion

2.2.1 Visualization

Fig. 3 shows the plot of the knollC-train200 data set, in which the support vectors are circled. The free support vectors are circled in black, and bounded are circled in green. There are 87 bounded support vectors, and just six free ones for a total of 93 support vectors.

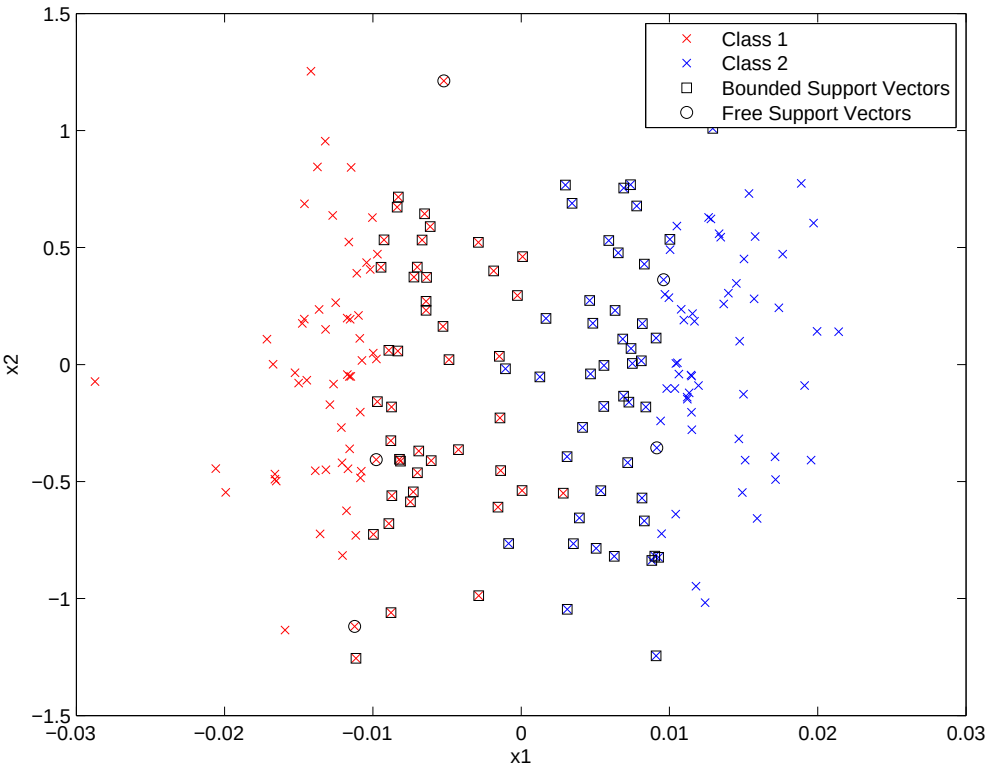

Figure 3: knollC-train200 data set with free support vectors (circles) and bounded support vectors (squares).

2.2.2 Effect of the regularization parameter

The file `regularization.m` performs the outlined procedure, by first training the SVM model using the values for C and γ found during model selection. Then it trains to other models, one in which C is multiplied by a hundred and one in which we divide C by 100.

The most notable change is in the number of support vectors. There's a total of 93 support vectors for the "original" value of C—87 of which are bounded. When C is a hundred times larger, the number of support vectors drop to just 19, all of which are free. Conversely, when dividing C by a hundred we get an increase in the number of support vectors to 199, but again all of them are free.

2.2.3 Scaling behaviour

See table 6 for the number of bounded and free support vectors for all training data sets. The number of support vectors grows (nearly) linearly with the number of data patterns in the training sets, as is to be expected. Since a higher number of support vectors will increase the computational effort for classifying new data patterns, this development needs to be controlled; otherwise, it may lead to a near-quadratic rise in complexity if training and test data grow alike.

In general, a higher number of samples of a distribution with a non-zero Bayes risk means that more samples will be misclassified. Using a Gaussian kernel and therefore an infinite-dimensional feature space, this could of course be prevented, but that would most likely mean that we overfit the model to our training data and is therefore not desirable. To avoid this, i.e. to decrease the penalty for misclassifications in our model, this should mean that a lower value should be chosen for C. This is confirmed nicely by the fact that during our tests, $C = 100$ actually gave the best results for $n = 400$, whereas for lower values of n, $C = 1000$ performed better.

	free	bounded
`knollC-train100`	5	60
`knollC-train200`	6	87
`knollC-train400`	12	153

Table 6: Table of bounded and free support vectors for the three data sets.