

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE

MASTER OF ARTIFICIAL INTELLIGENCE

|  |
| --- |
|  |
|  |
| Artificial neural networksPaper Florentijn Degroote  R0575914  [florentijn.degroote@student.kuleuven.be](mailto:florentijn.degroote@student.kuleuven.be)  2018-2019 |



# Assignment 1

## Perceptron

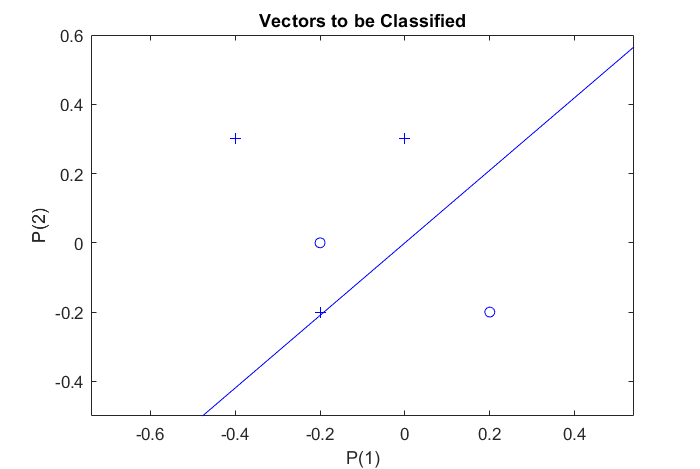
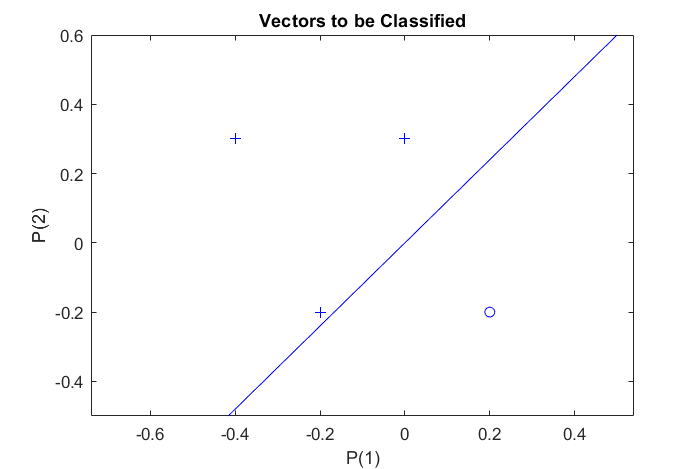


Figure 2

Figure 1



A perceptron is able to classify linearly separable data, as shown in figure 1. Its limits are reached when trying to classify the points as shown in figure 2. The most basic case where the perceptron fails as a perfect classifier is in the XOR problem ([Minsky, 2017](#Minsky2017)).

## Backpropagation in feedforward multi-layer networks

#### Function approximation

In order to approximate the function in the range of , 189 equally spaced points of distance 0.05 were generated. 75 percent of these points were randomly chosen to be part of the training set, thus assigning the remaining 47 points to the test set. For the first bag of tests, no noise was added. The data is fed 50 times (50 epochs) to a feedforward two-layer network, varying the number of neurons (1,5,20,50, and 200) and training algorithms (gradient descent (“traingd”), gradient descent with adaptive learning rate (‘traingda”), Fletcher-Reeves conjugate gradient descent (“traincgf”), Polak-Ribiere conjugate gradient algorithm (“traincgp”), BFGS quasi Newton algorithm (“trainbfg”), Levenberg-Marquardt algorithm (“trainlm”) and Bayesian regularization backpropagation (“trainbr”)) and using the default values for the other parameters. The results for computation time for the training phase and RMSE (root mean squared error) of both the training and test set were measured[[1]](#footnote-1). The results can be found in figures 3,4, and 5, respectively.

Figure

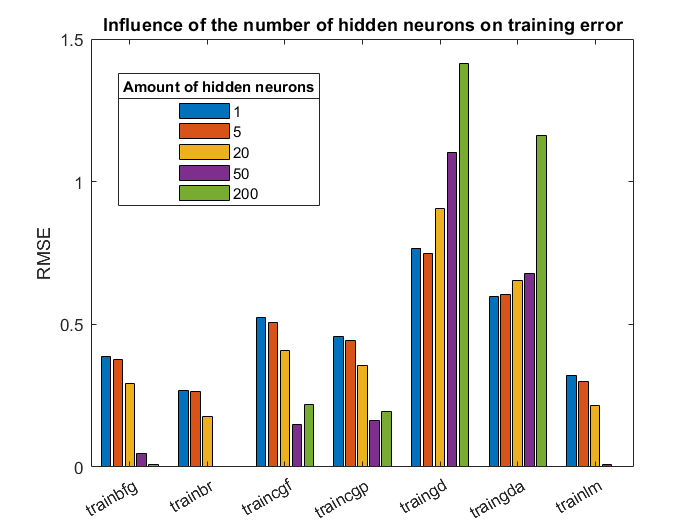
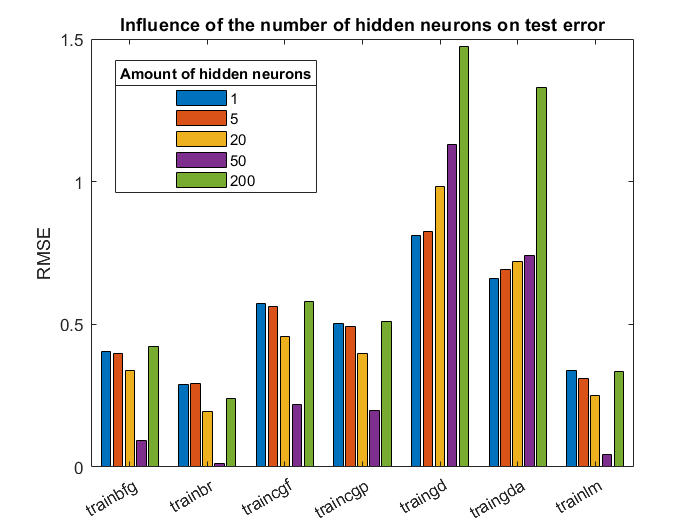


Figure 4

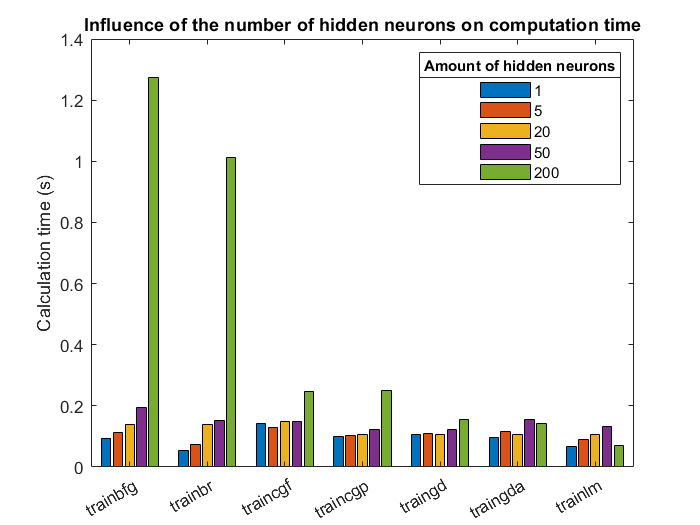


Figure 3

Because a bigger amount of hidden neurons involves more computations, there is a general rising trend in computation time for bigger networks. Computation times for 200 hidden neurons are remarkably big for trainbfg and trainbr in comparison to the others, including traingd. For trainbr this can be explained by the expensive posterior calculations. The decrease in computation time for trainlm is due to only completing 3 epochs before the algorithm stops automatically. This happens because the measured performance is falling below the default threshold. In figure 4 it is shown that for most algorithms, the training set’s RMSE is decreasing for an increasing amount of hidden neurons. Here, the added extra neurons introduce more flexibility and allow the model to fit the training data better. Though, this is not the case in the two gradient descent algorithms. The extra neurons introduce an extra complexity that traingd and traingda can’t cope with. This is a form of overparametrization. Remark that we only have 142 elements in the training set, which is less than the amount of hidden neurons in the most complex case! It is also noted that the same is happening when looking at the graph for the test error. In this graph, the RMSE per input is higher than it is for the training set, overall. This is logical, because the neural net is trained on the training data, not on the test data. The test data is “new” for the neural network. Notice that trainbr, trainbfg and trainlm have quite nice results for the combination of training set and test set for 50 hidden neurons. On the other hand, all except trainlm of these settings do impose a bigger computation time compared to the alternative learning algorithms. For 200 hidden neurons, the differences in RMSE are big, indicating overfitting/overparametrization. In the case of trainbr, the inherent regularizing characteristic limits the amount of overfitting (values of the weights are kept low).

The second bag of experiments were similar to the ones before. The difference here is that there is a gaussian noise added to every target with a mean of 0 and standard deviation of 0,5. The results can be found in figure 6,7, and 8. Again, overfitting is easily detectable for 200 neurons. As expected, the results of computation times and RMSE per input in the training set are also similar to the previous results, although the RMSE of the training set is slightly higher, because the added noise is not modelled perfectly (and shouldn’t be, otherwise the model is overfitted). The RMSE per input in the test set is higher than before, however. Especially for the aforementioned cases of 50 neurons because the noisy data introduces differences in what the model predicts and what the target is.

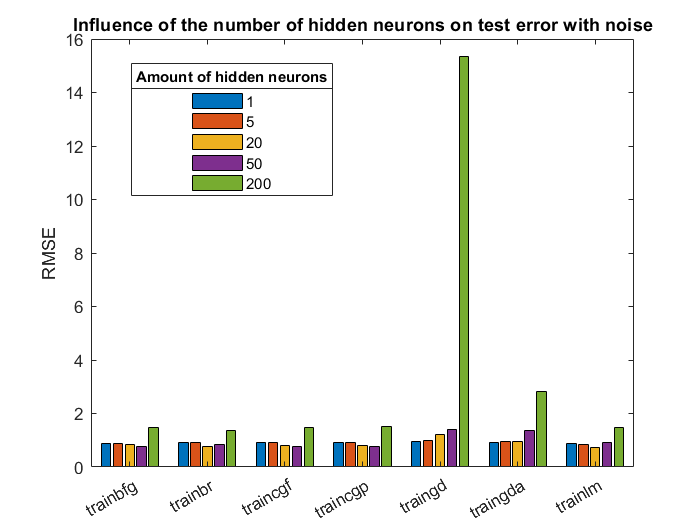


Figure 8

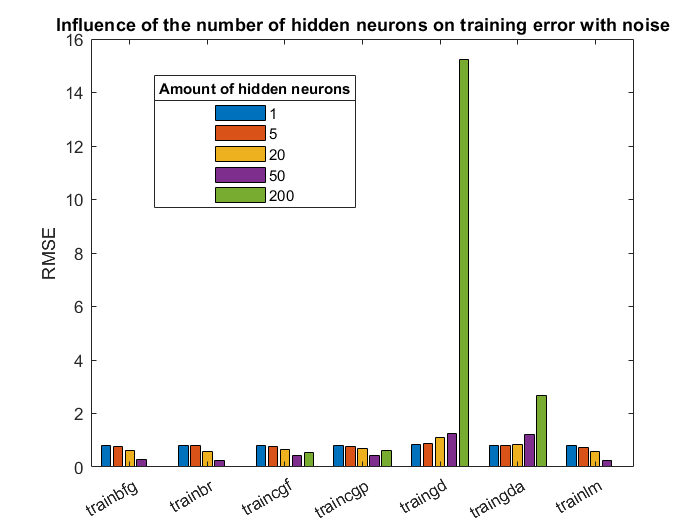


Figure 7

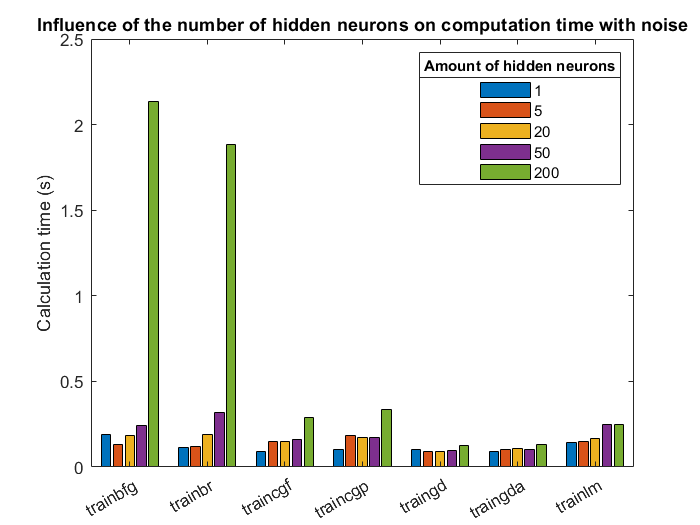


Figure 6

#### Personal regression

Based on my student number, is equal to . Three independent samples were generated (using random generators constructed in bulk), containing 1000 samples each. The first sample will be used as training set and is thus used to train the neural nets with. The next 1000 samples total the validation set, used to tune the architectural/hyperparameters of the model and see how well the model is trained. The last 1000 samples are used as test set. The test set is only used as a last test, to check whether overfitting (to what extent the model was trained to fit the validation set only) occurred. The results of the test set should not be used to alter parameters upon.

As before, during the experiment phase, we will use 50 epochs. Because trainlm, trainbr, and tranbfg gave the best results in the previous tests, because the setting is quite similar (though training set is larger) and because the goal here is to minimize errors, we will check the results of these two. The results in figure 9, 10, and 11 were obtained for a varying amount of neurons and using tansig as activation function in the one and only hidden layer.

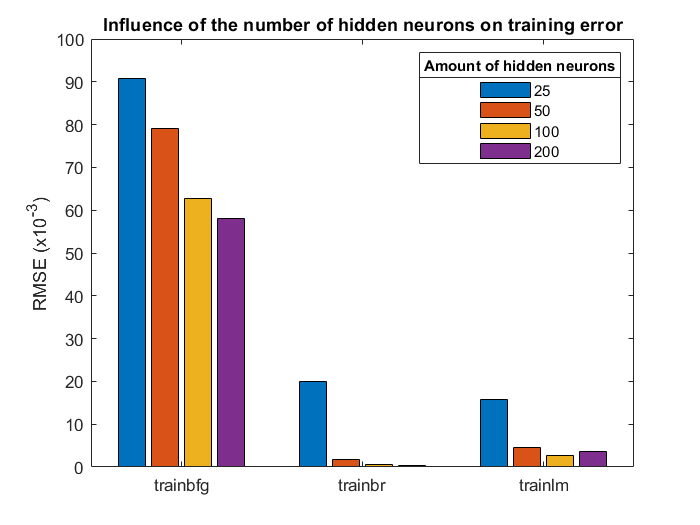
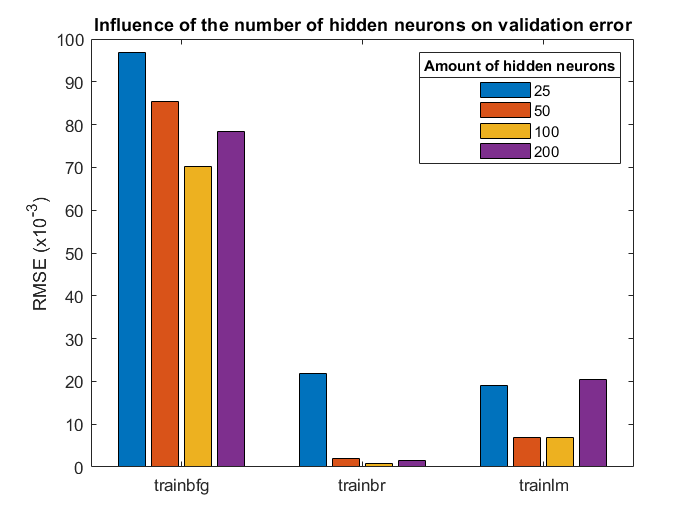


Figure 10



Figure

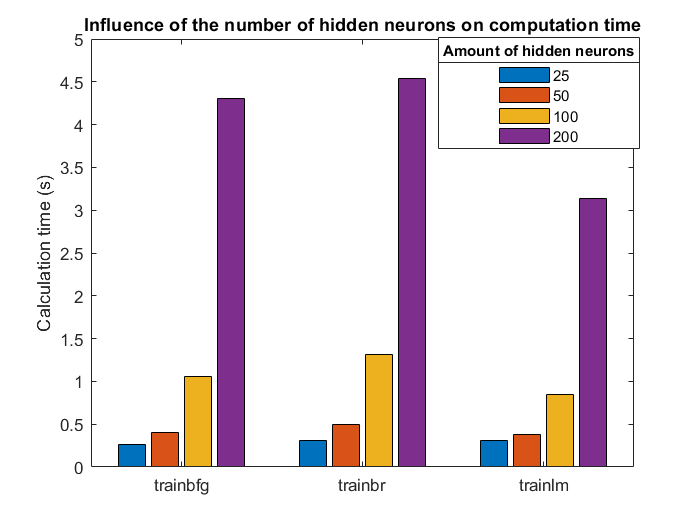


Figure 9

Because the results of Bayesian regularization seem to outperform the Levenberg-Marquardt algorithm, we will continue with this learning algorithm (note that the Levenberg-Marquardt is used in Bayesian regularization to update the weights and bias). Here, 100 hidden neurons offer a good trade-off between computation time and RMSE’s.

In figures 12 and 13, results are shown for 2 and 3 hidden layers of the net, respectively. These shallow networks are computationally expensive and more complex. The examined combinations of number of hidden layers and hidden neurons are not outperforming the one-hidden-layer network, so we will stick with the latter.

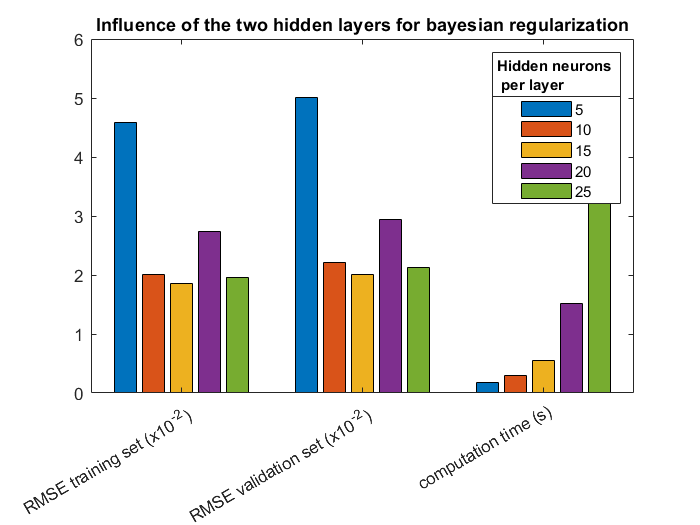


Figure 12

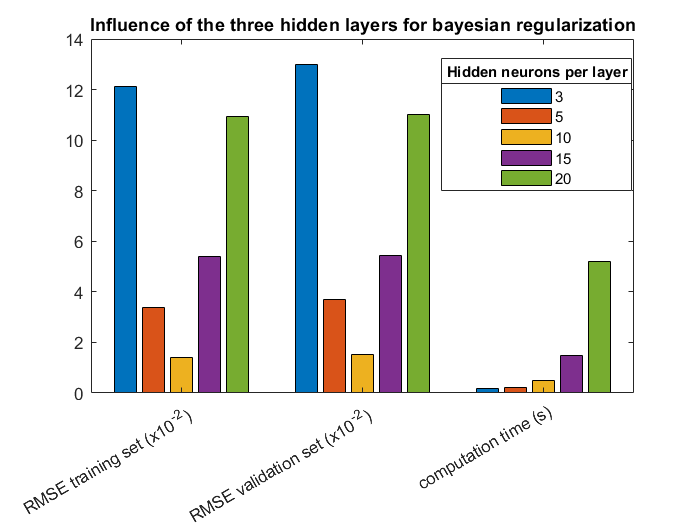


Figure 13

Now that the number of hidden layers and neurons are determined, one can wonder whether the default (tansig : tangent hyperbolic) is really the best performing activation function in this case. On figure 14, the results are shown for four different activation functions. Here, the results of 10 different runs (rng(1) up until rng(10)) were averaged. The tansig seems to be superior to others.

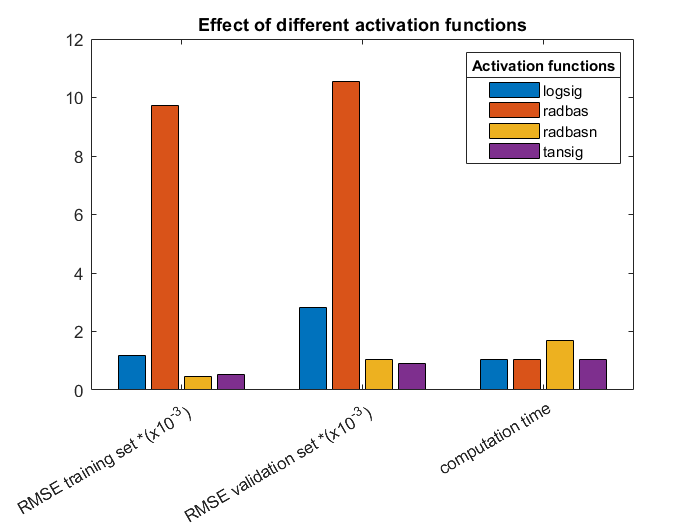
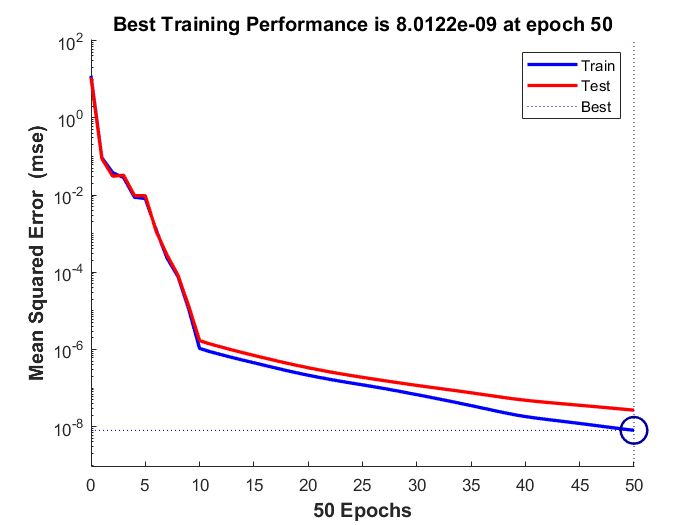


Figure 14

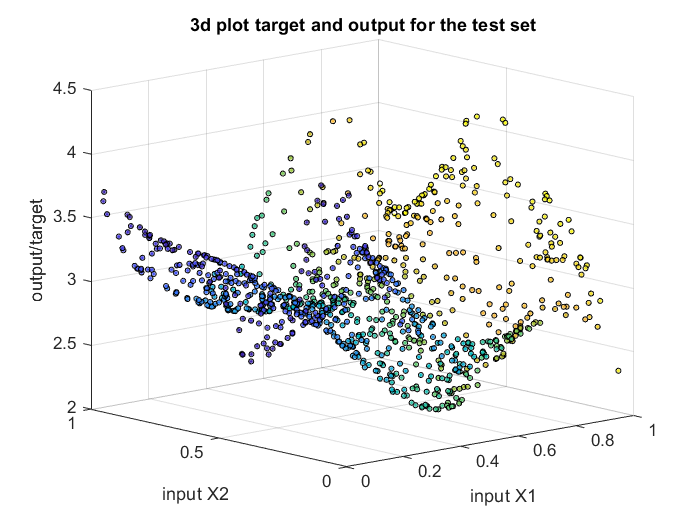
The resulting network is thus composed from a feedforward neural net with one hidden layer containing 100 hidden neurons. The activation function used in this hidden layer is the tangent hyperbolic. For the output layer, the standard hard-limit transfer function is used. Let’s now carry out a performance assessment on the test set. Figure 15 shows a 3D scatterplot of the targets of the training set. Figure 16 shows the targets of the test set (dots changing colours in the X1 direction) and the output of the network on the test inputs (black circles). We can conclude that the test set resembles the same distribution as the training set and that the net is capable of mimicking this distribution just fine. Table 1 shows the squared mean error values of training, validation and test set. These are all extremely low. The difference between test and training set is quite big, in proportion. On figure 17 we can indeed see that there is a visible difference between the two. More epochs for this network could lead to overfitting. On the regression plots generated by the nntraintool, Matlab gives the R squared a rounded value of 1, indicating a near perfect fit.



Figure



Figure



Figure

Of course, this method of setting up hyperparameters in order to find a good/optimal neural net is not guaranteed to generate the best possible neural network. The focus could lie elsewhere; when for example time was the focal point, a network with more hidden neurons would be chosen instead. Also, the order in which the hyperparameters are determined can be altered in order to find another (and maybe better) neural network. Hyperparameters can be tuned repetitively, or more values of hyperparameters could be taken into account in the tests. For illustrative purposes, a test was ran using 950 epochs for both 50 and 100 hidden neurons. We can clearly see that overfitting is

|  |  |
| --- | --- |
| Data set | SME value |
| Training set |  |
| Validation set |  |
| Test set |  |

Table

# Bibliography

Minsky, Marvin, and Seymour A. Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

1. By means of reproducibility, the same random generators are made use of, unless stated otherwise.

   rng(1); % random generator for initializing nets

   s = RandStream('mt19937ar','Seed',1); % random single generator

   [s1,s2,s3] = RandStream.create('mrg32k3a','NumStreams',3); % random generators in bulk

   Also, for every the results of an experiment, the average of 10 rounds’ results is taken [↑](#footnote-ref-1)