CO395 Machine Learning
CBC #2
Artificial Neural Networks

Group 1

Yong Wen Chua, ywc110

Thomas Morrison, tm1810

Marcin Baginski, mgb10

Marcin Kadziela, mk4910

Contents

Implementation Details

Overview

This report describes the work conducted in the second assignment of the Machine Learning course. The main goal of this coursework was to investigate and learn how to use the tools provided by the Neural Network Toolbox. The task that we were given was a typical pattern recognition and classification problem. We were supposed to build one network with multiple outputs and multiple single-output neural networks (one for each class). We started this assignment by optimising parameters for each of these networks. The approach taken was optimising topology together with training function first. In later phases we focused on other parameters (such as minimum gradient or goal). Later on we used the 10-fold cross-validation in order to compare performance of the two network types that we have created.

Description of the MATLAB functions

- `nFoldValidate(examples, classifications, n, networkType)` - performs the n-fold cross-validation. Takes as arguments the examples, classifications, number of folds and the network type (`'single'` or `'multi'`). Based on the network type, it either generates multiple single-output networks or one multi-output network and validates them. It returns a cell array which contains the confusion matrices for each fold

- `generateMultiOutputNetwork(xANN, yANN, layers, neurons, transferFcn, trainingFcn, lr, grad, goal)` - generates a multiple-output network based on the arguments which are passed to the function. These include the number of layers, neurons in a layer, training function, transfer function, learning rate, stopping gradient and the performance foal. It is used within `nFoldValidate()`

- `generateSingleOutputNetworks(xANN, yANN, layers, neurons, transferFcn, trainingFcn, lr, grad, goal)` - generates multiple single-output networks based on the same arguments as the previous function

- `testANN(network, inputs)` - produces a vector of predictions for the inputs using provided neural network (either the multi or single output one)

- `recallPrecisionF1(confusionMatrix)` - calculates the recall, precision and F1 measures after taking the confusion matrix as an argument

- `plotAverageF1Measure(confusionMatrixSingle, confusionMatrixMulti)` takes two arrays of confusion matrices (one for each fold) from the single and multi-output networks and plots the average F1 measure for each fold. Its output is a direct answer to the question from Part VIII

- `confusion_matrix(actual, predicted, possible_outcomes)` - calculates the confusion matrix given a column vector of actual and predicted outcomes

Evaluation

The average F_1 measure per fold for both types of networks is presented below:

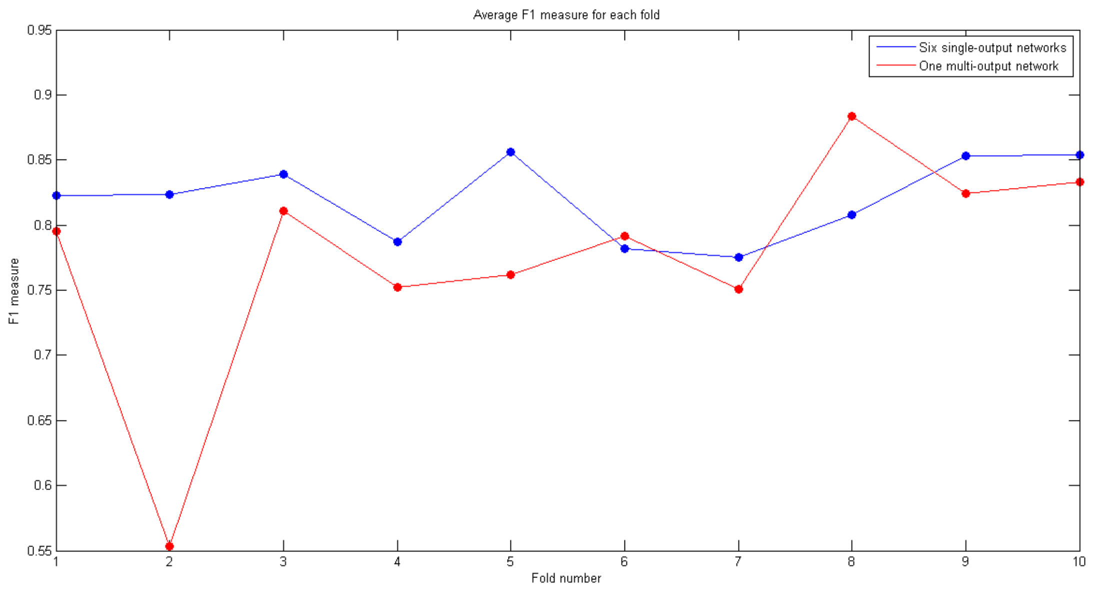

Figure 1: Average F_1 measure per fold for both types of networks

Clean dataset

Single six-output network

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**86**	18	6	6	14	2
	2	9	**167**	6	5	11	0
	3	2	5	**97**	3	5	7
	4	0	6	1	**203**	4	2
	5	7	26	2	3	**92**	2
	6	0	3	12	2	2	**188**

Table 1: Confusion Matrix for single six-output ANN for the *clean* dataset

		Recall	Precision	F_1
Actual class	1	65%	83%	73%
	2	84%	74%	79%
	3	82%	78%	80%
	4	94%	91%	93%
	5	70%	72%	71%
	6	91%	94%	92%

Table 2: Recall, precision and F_1 measure for single six-output ANN for the *clean* dataset

$$C = \frac{833}{1004} = 83.0\%$$

Figure 2: Classification rate for single six-output ANN for the *clean* dataset

Six single-output networks

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**99**	12	7	3	10	1
	2	9	**168**	3	6	11	1
	3	4	2	**98**	2	3	10
	4	2	4	2	**205**	1	2
	5	8	21	7	4	**89**	3
	6	0	3	10	5	2	**187**

Table 3: Confusion Matrix for six single-output ANNs for the *clean* dataset

		Recall	Precision	F_1
Actual class	1	75%	81%	78%
	2	84%	80%	82%
	3	82%	77%	80%
	4	95%	91%	93%
	5	67%	77%	72%
	6	90%	92%	91%

Table 4: Recall, precision and F_1 measure for six single-output ANNs for the *clean* dataset

$$C = \frac{846}{1004} = 84.3\%$$

Figure 3: Classification rate for six single-output ANNs for the *clean* dataset

Noisy dataset

Single six-output network

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**11**	16	22	7	25	7
	2	4	**149**	18	6	6	4
	3	2	15	**132**	10	9	19
	4	3	7	13	**175**	4	7
	5	7	13	14	5	**56**	15
	6	1	4	15	5	9	**186**

Table 5: Confusion Matrix for single six-output ANN for the *noisy* dataset

		Recall	Precision	F_1
Actual class	1	13%	39%	19%
	2	80%	73%	76%
	3	71%	62%	66%
	4	84%	84%	84%
	5	51%	51%	51%
	6	85%	78%	81%

Table 6: Recall, precision and F_1 measure for single six-output ANN for the *noisy* dataset

$$C = \frac{709}{1001} = 70.8\%$$

Figure 4: Classification rate for single six-output ANN for the *noisy* dataset

Six single-output networks

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**15**	11	25	8	23	6
	2	3	**154**	15	6	7	2
	3	1	11	**139**	8	11	17
	4	1	9	11	**176**	6	6
	5	4	8	19	4	**67**	8
	6	1	3	13	4	9	**190**

Table 7: Confusion Matrix for six single-output ANNs for the *noisy* dataset

		Recall	Precision	F_1
Actual class	1	17%	60%	27%
	2	82%	79%	80%
	3	74%	63%	68%
	4	84%	85%	85%
	5	61%	54%	58%
	6	86%	83%	85%

Table 8: Recall, precision and F_1 measure for six single-output ANNs for the *noisy* dataset

$$C = \frac{741}{1001} = 74.0\%$$

Figure 5: Classification rate for six single-output ANNs for the *noisy* dataset

Discussion of results

Questions

Optimal topology

The final, optimal topology which we used for the networks are:

1. Single six-output network:
   - Hidden layers: 1
   - Hidden neurons: 23
   - Transfer function: `'tansig'` (sigmoid)
   - Training function: `'trainscg'`
   - Learning rate: 0.01
   - Minimum gradient: $3 * 10^{-6}$
   - Goal: $2 * 10^{-3}$
2. Six single-output networks:
   - Hidden layers: 1
   - Hidden neurons: 9
   - Transfer function: `'tansig'` (sigmoid)
   - Training function: `'trainscg'`
   - Learning rate: 0.01
   - Minimum gradient: $5 * 10^{-6}$
   - Goal: $3 * 10^{-3}$

We arrived at these values by performing a series of simulations of the performance of networks with different values on the first fold on the *clean* dataset. In the first experiment, we tried to optimise the topology i.e. number of hidden layers, number of hidden neurons in each layer as well as the training function. We did this my repeatedly training the network with different parameters and plotted the results which are presented in **Figures 6, 7, 8, 9**. From the analysis of the results it can be concluded that the networks with just one hidden layer perform slightly better than those with two hidden layers, and so we trained our final networks with a single hidden layer. In case of the single six-output network, the peak is achieved for 32 hidden neurons, however as our final topology we selected 23 hidden neurons, since the classification rate for this topology is insignificantly smaller, yet the number of neurons is reduced by a third. For the six single-output networks, we decided to choose 9 hidden neurons.

In terms of the training function, it looks that `'trainscg'` and `'trainrp'` perform similarly, while `'trainlm'` almost always exhibits the worst performance. Due to the speed of training and the classification rates at our chosen numbers of hidden neurons, we ultimately choose `'trainscg'` as our training function for both types of networks.

We selected these values because we needed eventually to decide on the topology of the network. However, by looking at the plots, it is clear that the number of hidden layers and neurons (at least for the range of values which we investigated) does not cause the performance to vary significantly beyond some minimum number of neurons and before some maximum. As long as the number of hidden neurons is in some reasonable range (above 20 for the single six-output network and between 6 and 30 for six single-output networks), it does not matter themendously what exact value we would choose.

Overfitting

Six-output vs. single-output networks

Ideal parameter optimisation

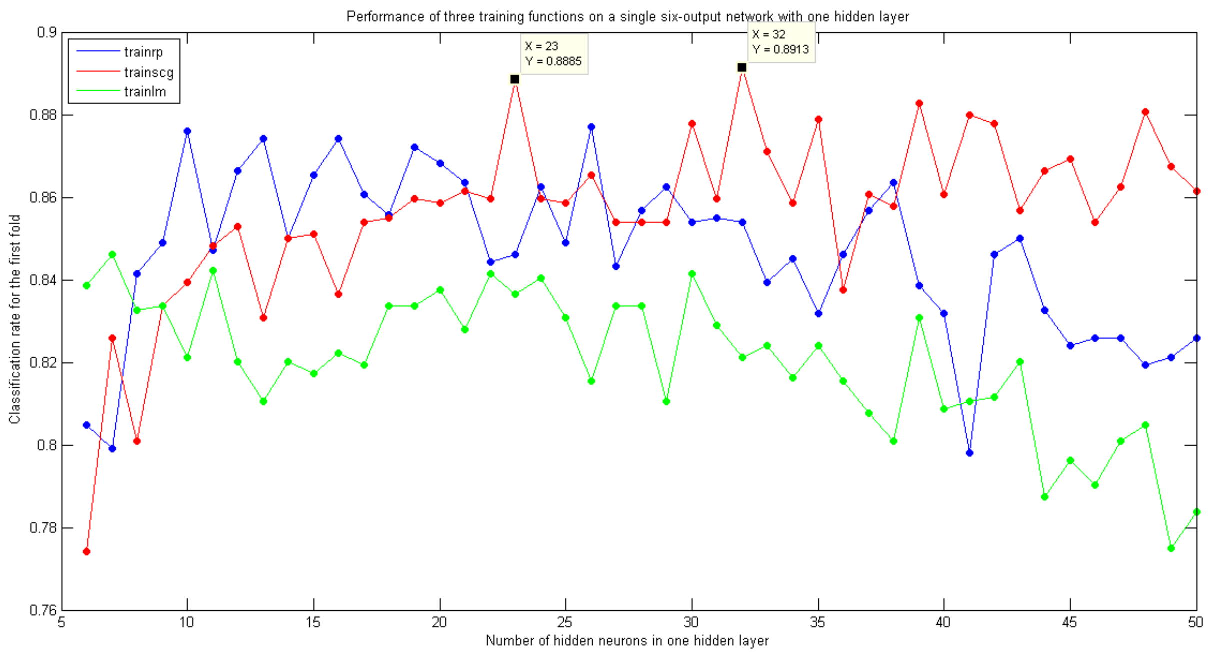

Figure 6: Classification rate for the first fold of the *clean* dataset with respect to the training function and the number of hidden neurons for a single six-output network with one hidden layer

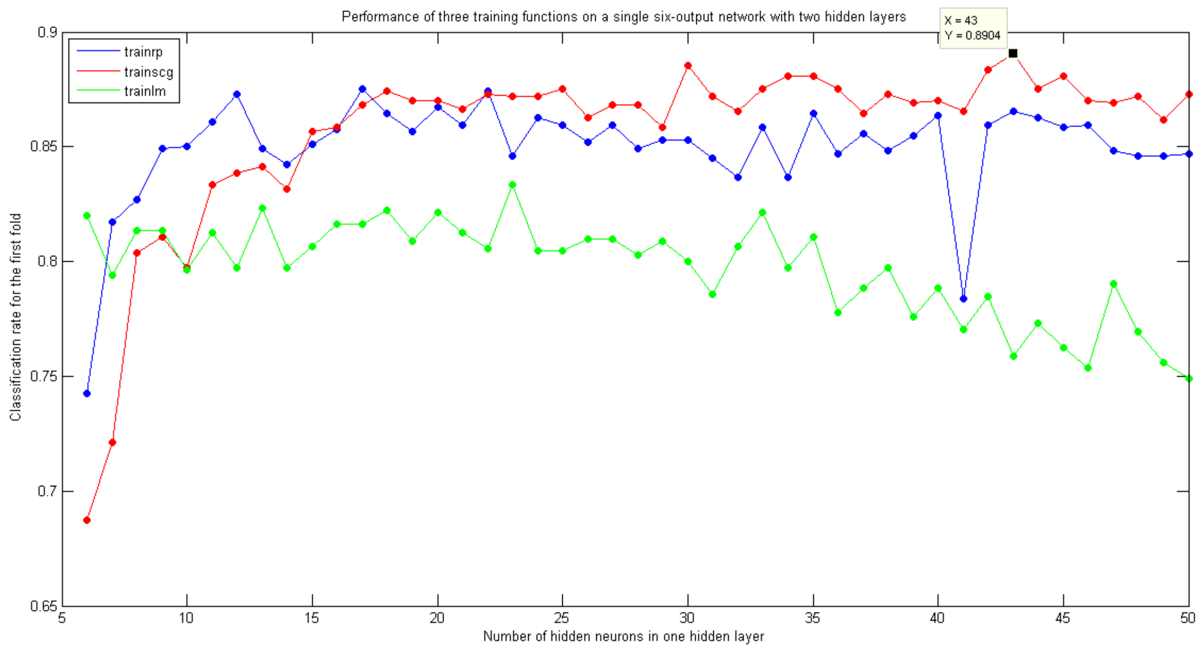

Figure 7: Classification rate for the first fold of the *clean* dataset with respect to the training function and the number of hidden neurons for a single six-output network with two hidden layers

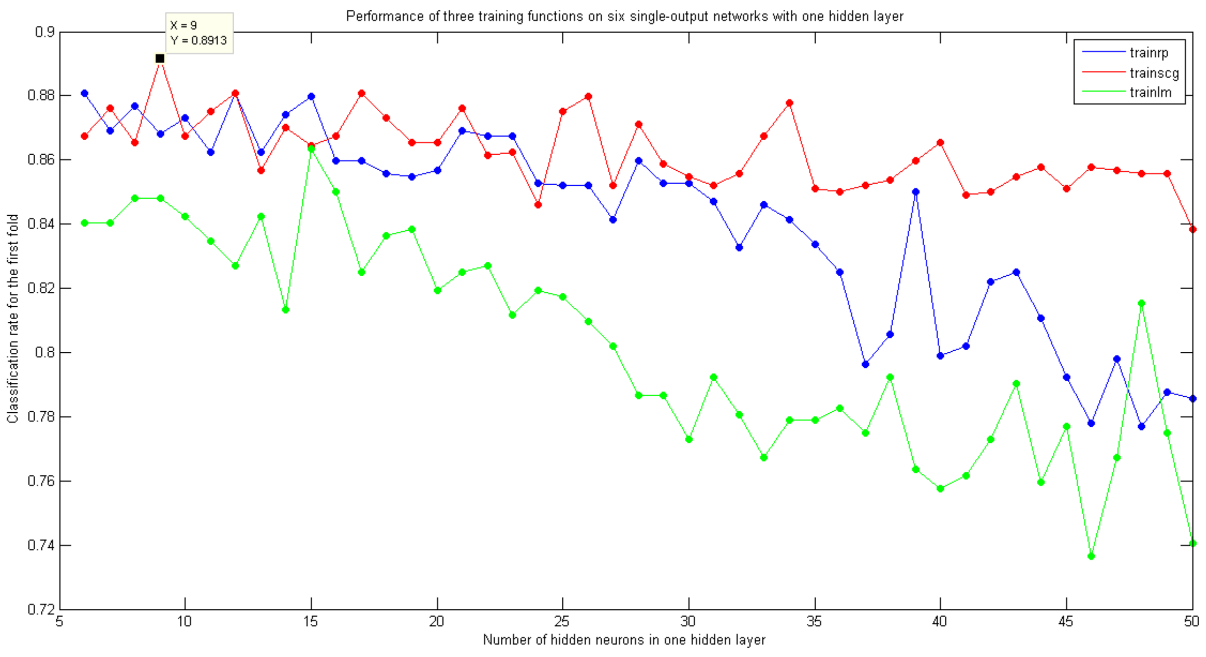

Figure 8: Classification rate for the first fold of the *clean* dataset with respect to the training function and the number of hidden neurons for six single-output networks with one hidden layer

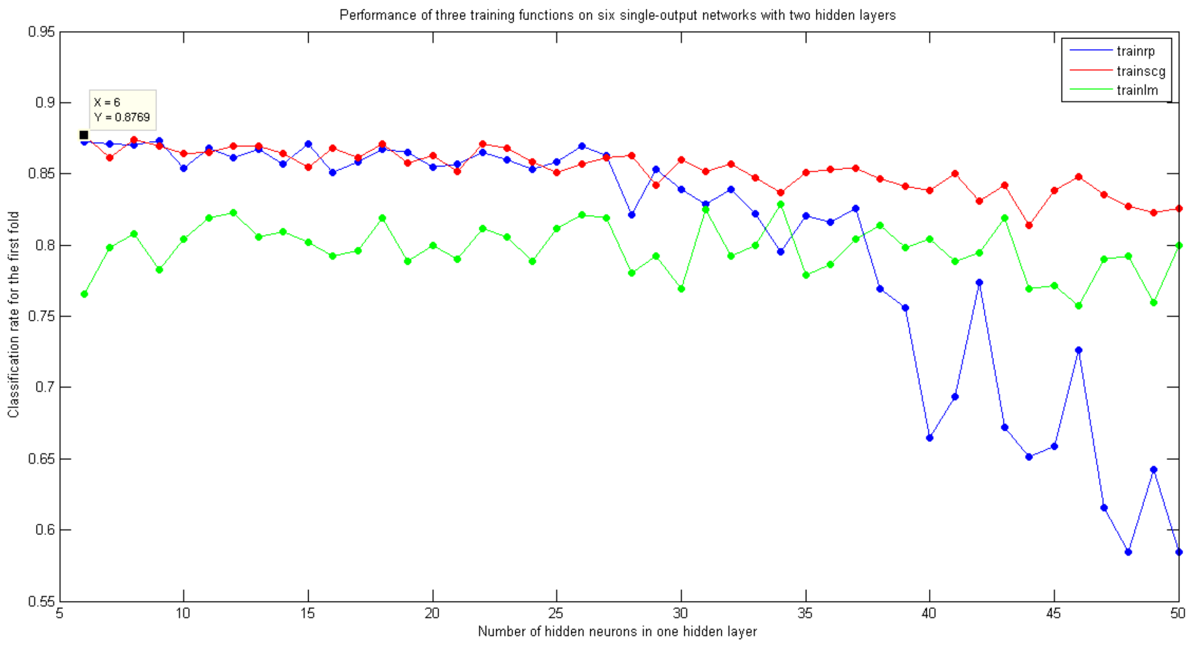

Figure 9: Classification rate for the first fold of the *clean* dataset with respect to the training function and the number of hidden neurons for six single-output networks with two hidden layers

Code Flowchart