Report on Assignment 2 - Neural Networks

Group 23: David Toth, Vinea Au Yeung, Tina Liu

1 Introduction

Artificial neural networks (ANNs) are another way to generalize the target function given its inputs and outputs. One neuron is a linear combination of the attributes and the weights transformed by an activation (or transfer) function. Choosing a non-linear activation function in combination with multiple neurons and layers increases the expressivity sufficient enough to express any nonlinear continuous function. In this sense, ANNs have the potential of being more expressive than linear regressions. On the other hand, with the power of expressivity comes the risk of overfitting. Thus our task is to find the ANN topology and parameters powerful enough to express a target function with a reasonably small error, but still simple and general enough to classify correctly the instances from the real data that could not be used for the training.
In our experiments we considered the two types of networks: one with six output neurons corresponding to each class (network A) and a set of six binary classification networks (network B). We have included our Network A (one network with 6 outputs) which is trained on our entire data set using the optimal parameters found. This network can be found in network_trained_on_all_data.mat.

2 Implementation Details

2.1 Network optimization

We have written a script *find_best_ann.m* which takes a list of ANNs generated by *gen_anns.m*. ANNs are generated with different parameters - a training algorithm, an activation function, a number of layers, a number of neurons in a hidden layer. Then ANNs are trained on a training set (67% of the dataset). The mean squared error (mse) of an ANN is calculated on a validation set (33% of the dataset). We consider an ANN with the lowest mse to be the one with the optimal parameters - a discussion follows later. Some networks were found manually in the same way with *simple_test_ann.m* which returns mse for a manually specified network.

2.2 Statistics and Cross-validation

The *get_stats.m* function performs cross-validation by taking as inputs all the examples, the corresponding targets and the network with the optimized parameters, the validation is performed by calling *cross_validate.m* 10 times, each time using a different tenth of the data for testing and the other ninth for training and validation, according to the fold number that is passed in. Each time, cross_validate.m returns a structure containing the predictions, confusion matrix, error rate, recall rates, precision rates, and F1 measures. We use a 1x10 cell array to store these data for each of the 10 classifiers in a cell element. Final statistics are obtained as an arithmetic mean over the individual statistical measures of the 10 classifiers.

2.3 Classification

For a six-output network with six output neurons, the neuron with the highest output value indicates the emotion label under which an example should be classified. For the set of six single-output networks, each network output produces a value representing the strength of the association of an emotion with an example. We combine the results into a single classifier by choosing the label that corresponds to the network with the highest output (a strongest association) for that example.

3 Design Overview

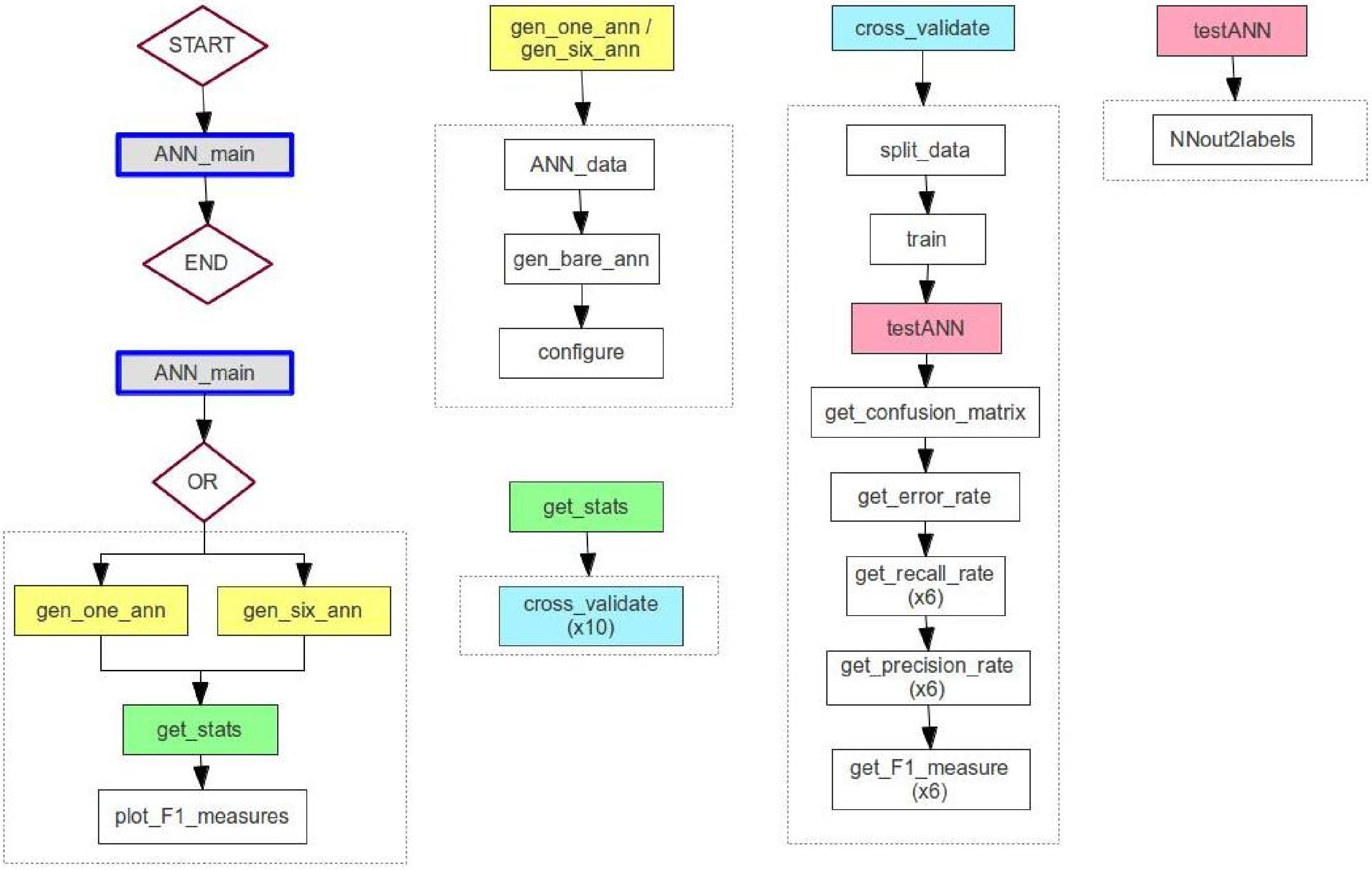


Figure 1: Flow chart showing the sequence of invoked functions

3.1 Program Flow

The above flowchart (Figure 1) describes the stages of our running program. The sequences of functions enclosed in the dotted lines allow us to represent our function hierarchy. For instance, ANN_main.m calls either gen_one_ann.m or gen_six_ann.m, followed by get_stats.m and plot_F1_measures.m. Then gen_one_ann.m (highlighted in yellow), in turn, calls ANNdata.m, gen_bare_ann.m and configure.m. We have optimised the topology and parameters of the network in a separate, independent program which is run using the function find_best_ann.m (see Figure 2 below).

3.2 Details on Main Functions

Our main function is ANN_main.m which takes as inputs: the matrix of examples and the target vector. It generates either a six-output neural network or 6 single-output neural networks. These networks are configured but not trained. We find their optimal topology and parameters independently using our find_best_ann.m function. The gen_bare_ann.m function is used to generate a basic network with certain parameters, without configuring or training the data, therefore once we have obtained the optimal parameters, we set these manually in gen_bare_ann.m for each of the two types of network.

The cross-validate.m function is invoked 10 times by get_stats.m. Its purpose is to carry out one fold of the cross validation so we obtain a training set and validation set (67% and 33% correspondingly from 9/10th of the original data set) and a test set (remaining 1/10th of the original data set). Once it has trained the networks on the allocated training data using the training function specified in gen_bare_ann.m, testANN.m tests the performance of the networks on the test data. The testANN.m function is also responsible for combining the 6 binary networks into one classifier. Based on the results of this test, it computes the following statistics for each fold: a confusion matrix, the error rate, and for each class the precision and recall rates, and the F1 measures. Once this process is completed for all 10 folds, we obtain the average of these results in get_stats.m. For both types of network, ANN_main.m also plots the average F1 measure over all 6 folds against each fold of the cross validation in a bar chart.

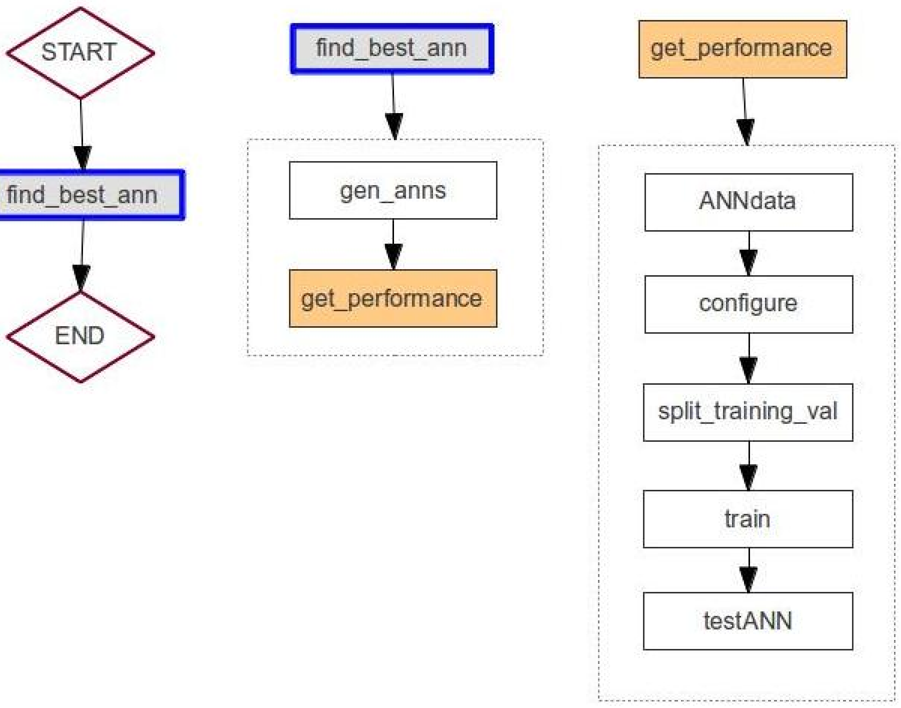


Figure 2: Flow chart for parameter optimisation process

Figure 2 shows the process of the independent program that we implemented to find the optimal parameters and topology of the networks. The find_best_ann.m function firstly invokes gen_anns.m, which generates a cell array of networks, each one with varying possible parameters and topologies. The get_performance.m function splits the entire data set into training set (67%) and validation set (33%) before training each network on the training set and retrieving the performance measure for each network. Since the optimal parameters are different for the six-output network and the 6 single-output networks, we obtain them separately and set them manually in gen_bare_ann.m.

4 Optimization of ANNs

4.1 Choice of Parameters

Statistical analysis of all networks with their possible parameters is computationally intractable as having more than 100 binary choices introduces more than 2^100 possible parameter configurations. We have eliminated these combinations by some theoretical choices explained in an answer to a question 1. Further, we admit that networks with more efficient parameters may exist. However, finding the most optimized one was not our goal.
Currently, we ensure that the data splitting into a training and a validation set is carried out in the same way for every network with different parameters - by default network.divideFcn = ‘dividerand’, we changed it to network.divideFcn = ‘divideint’. But we leave the initial weights to be initialized randomly as MATLAB default.
Nevertheless, we do not think that either way is better in our process of optimization. The theoretical reason is that choosing a particular way of splitting or weight initialization may be biased towards certain types of networks - our decision of the weights, training data and validation data determines performance measures of the networks that we are going to measure.
The negative effects of randomization could be mitigated by cross-validation as it performs tests on 10 folds and then takes a mean of multiple (10) values resulting in a lower standard deviation[1]. However this approach would require around 10 times more computational time for the statistical analysis.

4.2 Optimization of a six output ANN (Network A)

We have tested 16 ANNs with different sets of parameters. Our results in Figure 3 show that a choice of a training algorithm can influence the results significantly: mse of networks with trainlm are in a range [0.5, 1.0] whereas networks with traingd have their mse in a range [1.0, 2.0]. Other observation includes that networks with fewer neurons had lower mse as displayed in a Figure 4. The lowest mse was recorded for a network number 3. Thus our network with optimized parameters has a trainlm training algorithm, a purelin activation function, 2 layers, 10 neurons per layer.

ANN No.	Training Algorithm	Activation Function	Layers	Neurons per Layer	Mse
1	trainlm	purelin	1	10	0.06604
2	trainlm	purelin	1	40	0.060283
3	**trainlm**	**purelin**	**2**	**10**	**0.055851**

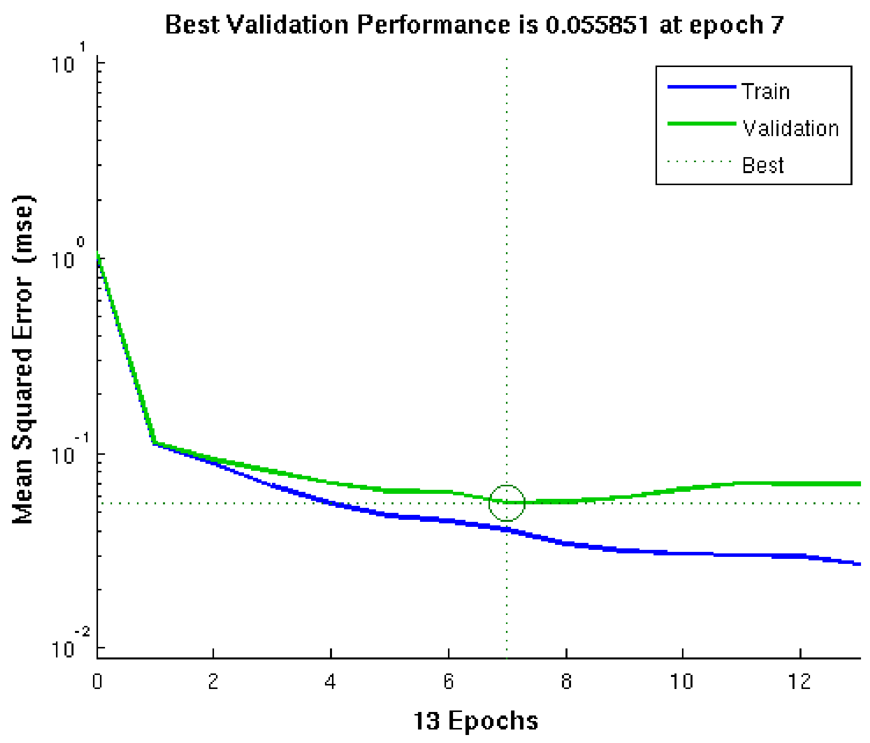
[1] It is a known fact from statistics that a sum of random variables is distributed according to the distribution with a lower standard deviation than standard deviations of original distributions.

4	trainlm	purelin	2	40	0.074605
5	trainlm	tansig	1	10	0.065389
6	trainlm	tansig	1	40	0.071556
7	trainlm	tansig	2	10	0.058631
8	trainlm	tansig	2	40	0.095502
9	traingd	purelin	1	10	0.13204
10	traingd	purelin	1	40	0.10858
11	traingd	purelin	2	10	0.18346
12	traingd	purelin	2	40	0.38984
13	traingd	tansig	1	10	0.15043
14	traingd	tansig	1	40	0.19257
15	traingd	tansig	2	10	0.1418
16	traingd	tansig	2	40	0.15047

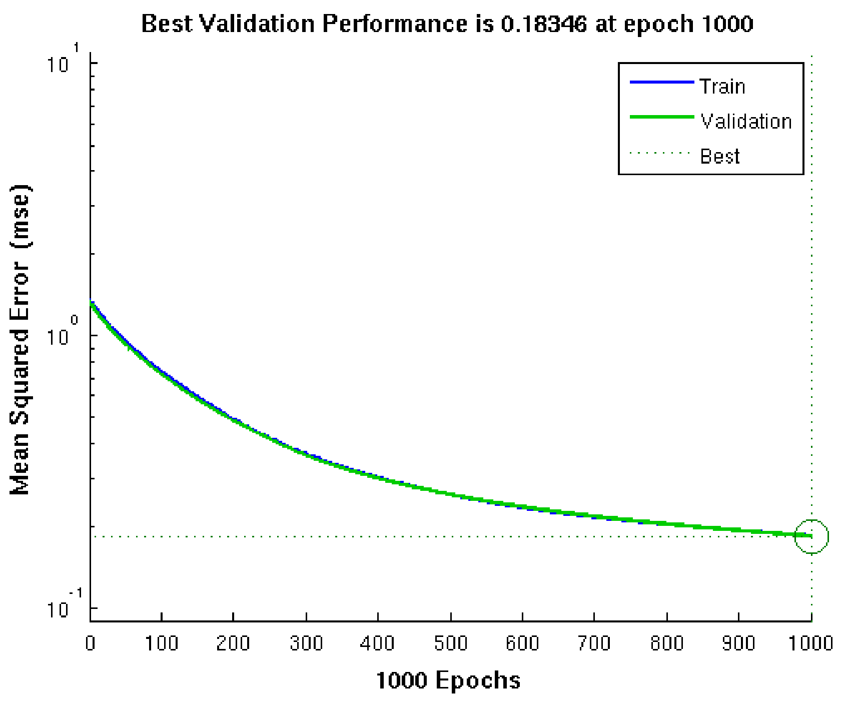
Fig. 3

Mean Mse of ANNs	**1 Layer**	**2 Layers**	**Mean**
10 neurons	0.1035	0.1099	0.1067
40 neurons	0.1082	0.1776	0.1429
Mean	0.1059	0.1438	

Fig. 4



ANN 3 performed the best reaching its lowest mse after the 7th iteration of training.



An example ANN 11 with a gradient descent training algorithm. These types of networks performed worse and required a longer training.

4.3 Optimisation of Six Single-Output ANN (Network B)

ANN No.	Training Algorithm	Activation Function	Layers	Neurons Layer	Mse per
1	trainlm	purelin	1	10	0.0630
2	trainlm	purelin	1	20	0.0619
3	trainlm	purelin	2	5	0.0660
4	**trainlm**	**purelin**	**2**	**10**	**0.0566**
5	trainlm	tansig	1	10	0.0649
6	trainlm	tansig	1	20	0.0632
7	trainlm	tansig	2	5	0.0656
8	trainlm	tansig	2	10	0.0631
9	traingd	purelin	1	10	0.1419
10	traingd	purelin	1	20	0.1512
11	traingd	purelin	2	5	0.1737
12	traingd	purelin	2	10	0.1878
13	traingd	tansig	1	10	0.1528
14	traingd	tansig	1	20	0.1626
15	traingd	tansig	2	5	0.1397
16	traingd	tansig	2	10	0.1434

Fig. 5

Parameters of a network B were optimized in the same way as parameters of a network A. The network with of type B with the lowest mse at 0.0566 uses a trainlm training algorithm, a purelin activation function and 2 layers, each with 10 neurons.
According to the experiment both types of the networks have the same optimal parameters while a network of type A is performing slightly better with a mse at 0.055851.

5 Cross-Validation Results

5.1 Classification Rates

During our network optimization stage we have chosen the following networks with their optimal parameters: a network A: trainlm, purelin, 2 layers and 10 neurons in each layer; the optimized network B had the coincidentally the identical parameters. We perform cross validation on the two types of networks with these optimised parameters. From the results in Fig 6, our six-output network on clean data has a higher classification rate than our six single-output networks on clean data, confirming a correlation of a mse to a classification rate. On noisy data, the average classification rate is also higher for our six-output network.

Network Type	Data Type	Average Classification Rate
Six-Output	Clean	80.09
Six-Output	Noisy	70.43
Six Single-Output	Clean	71.01
Six Single-Output	Noisy	64.15

Fig 6. shows the average classification rates for our two types of network when carrying out cross-validation on both clean and noisy data.

There are two primary measures to determine the performance of our classifier: recall rate and precision rate. The recall rate indicates the accuracy of our classifier, i.e. how often our predicted class matches the target value, while the precision rate indicates the precision of the classifier, i.e. how many elements of a predicted class truly belong to that class.

5.2 Evaluation on the Six-Output Network

5.2.1 Analysis on Clean Data

The figures on the next page(Fig 7, 8, 9) are the average cross-validation results produced using the provided clean data. There are a range of recall rates and precision rates over the 6 emotions. The rates are generally high, especially for emotions 4 and 6, which are above 90% (for the recall rates) and high 80’s% (for the precision rates). In both cases, emotion 5 exhibits a lower rate. The average F1 measures calculated over all 10 folds for each class also reflect these results, in particular, that better performance is shown when classifying emotions 4 and 6, while emotion 5 is classified less accurately and precisely.

When we look at the confusion matrices produced during each fold of the cross validation, 5 out of the 10 folds show that performance in classifying emotion 4 is the best and the other 5 folds show emotion 6 as the most accurately classified emotion. Also, 6 out of the 10 folds show emotion 5 as the poorest in terms of classification accuracy and precision, whereas the other 4 folds indicate the lowest rates for either emotion 1 or emotion 3. Therefore, the consistency amongst all 10 folds is slightly varied, but overall they agree that classification of emotions 4 and 6 are the most accurate and precise.

In Figures 7 and 10 we represent the entries of the confusion matrix as percentages of the total actual positives for each class. Hence, the values along the highlighted diagonal are the recall rates. In Figures 8 and 11, the values along the diagonal represent the precision rates.

		Predicted Emotions						
		1	2	3	4	5	6	Total
Actual Emotions	1	75.0	5.3	3.0	2.3	10.6	3.8	100.0
	2	6.6	75.8	3.0	3.5	9.1	2.0	100.0
	3	5.9	3.4	71.4	4.2	4.2	10.9	100.0
	4	0.0	3.7	0.0	94.0	0.9	1.4	100.0
	5	12.9	15.9	6.1	5.3	58.3	1.5	100.0
	6	0.0	1.9	4.3	0.5	1.4	91.8	100.0

Fig 7. Six-Output Confusion Matrix on Clean Data (as percentages of the true positives for each class)

		Predicted Emotions					
		1	2	3	4	5	6
Actual Emotions	1	72.8	3.6	3.6	1.3	11.8	2.3
	2	9.6	77.3	5.4	3.1	15.1	1.8
	3	5.1	2.1	75.9	2.2	4.2	6.0
	4	0.0	4.1	0.0	89.8	1.7	1.4
	5	12.5	10.8	7.1	3.1	64.7	0.9
	6	0.0	2.1	8.0	0.4	2.5	87.6
	Total	100.0	100.0	100.0	100.0	100.0	100.0

Fig 8. Six-Output Network Confusion Matrix on Clean Data (as percentages of the predicted positives for each class)

	Emotions					
	1	2	3	4	5	6
Average Recall Rate	75.0	75.8	71.4	94.0	58.3	91.8
Average Precision Rate	72.8	77.3	75.9	89.8	64.7	87.6
Average F1 Measure	73.9	76.5	73.6	91.9	61.3	89.7

Fig 9. Six-Output Network statistics on Clean Data

5.2.2 Analysis on Noisy Data

The above 3 figures are the average cross-validation results produced on the noisy data. Here, there is a much wider range of recall rates and precision rates over the 6 emotions. We can see that the rates are particularly low for emotion 1. The low recall rate means that there is a high false negative rate, while the low precision rate implies that the number of false positives is high. Overall, our network performs worst when trying to classify into emotion 1. The next worst performance is in classifying emotion 5. However, our network classifies reasonably accurately and precisely for emotions 2, 4 and 6.

The confusion matrices produced from each fold of the cross validation all individually show a low recall and precision rate for emotions 1 and 5. In addition, the rates are consistently high for emotions 2, 4 and 6. Folds 3, 4, 6 and 8, however, produce confusion matrices in which the rates for emotion 3 are lower than the rates for emotions 2, 4 and 6. In fold 8, the emotion 3 rates are extremely low. This would explain why the average results above show that the classification of emotion 3 performs slightly worse than emotions 2, 4 and 6.

		Predicted Emotions						
		1	2	3	4	5	6	Total
Actual Emotions	1	8.0	21.6	27.3	12.5	23.9	6.8	100.0
	2	1.6	82.9	7.5	3.7	2.7	1.6	100.0
	3	1.6	7.5	72.2	7.5	3.7	7.5	100.0
	4	0.0	6.2	5.7	82.3	1.9	3.8	100.0
	5	3.6	10.0	12.7	6.4	53.6	13.6	100.0
	6	0.0	3.2	5.9	5.9	4.5	80.5	100.0

Fig 10. Six-Output Network Confusion Matrix on Noisy Data (as percentages of the true positives for each class)

		Predicted Emotions					
		1	2	3	4	5	6
Actual Emotions	1	41.2	8.7	11.3	4.9	19.8	2.7
	2	17.6	70.8	6.6	3.1	4.7	1.3
	3	17.6	6.4	63.7	6.3	6.6	6.3
	4	0.0	5.9	5.7	76.8	3.8	3.6
	5	23.5	5.0	6.6	3.1	55.7	6.7
	6	0.0	3.2	6.1	5.8	9.4	79.4
	Total	100.0	100.0	100.0	100.0	100.0	100.0

Fig 11. Six-Output Network Confusion Matrix on Noisy Data (as percentages of the predicted positives for each class)

	Emotions					
	1	2	3	4	5	6
Average Recall Rate	8.0	82.9	72.2	82.3	53.6	80.5
Average Precision Rate	41.2	70.8	63.7	76.8	55.7	79.4
Average F1 Measure	13.4	76.4	67.7	79.5	54.6	79.9

Fig 12. Six-Output Network statistics on Noisy Data

5.3 Evaluation on the 6 Single-Output Networks

5.3.1 Analysis on Clean Data

On the clean data, our classifier generally has different performances on different classes, as we can see that the F1 measures across all the 6 classes(See Fig 15) have values that are extremely varied, ranging from around 50% to around 90%. This might indicate that our parameters perform much better for certain classes, such as emotion 4 where the precision rate is high, and that it performs worse on some others, such as emotions 1 and 5 (See Fig 13).

We noticed that across all confusion matrices from cross validation, emotion 1 and 5 have very low recall rate and precision rate in each of the fold while other emotions are generally similar, which shows that the folds are very consistent to each other and hence we have a representative average confusion matrix.

We also noticed that some examples are wrongly classified as certain emotions at a high rate, for example a person with emotion 1 is classified as having emotion 2 approximately every fourth time (25%), and these might be the cases where the network output for emotion 2 was higher than emotion 1 so that it was chosen according to our outputs-combination algorithm (the highest output is always chosen). This would be reflected in the low recall rates (See Fig 14) as the emotion has not been successfully recalled due to the fact that our network wrongly recognised another emotion as present.

		Predicted Emotions						Total
		1	2	3	4	5	6	
Actual Emotions	1	53.0	25.0	4.5	2.3	9.8	5.3	100.0
	2	9.6	72.2	3.0	3.0	7.6	4.5	100.0
	3	5.9	4.2	68.9	2.5	2.5	16.0	100.0
	4	0.5	3.7	1.9	90.3	1.4	2.3	100.0
	5	18.2	21.2	6.8	3.8	42.4	7.6	100.0
	6	2.4	3.9	8.2	1.9	2.9	80.7	100.0

Fig 13. 6 Single-output Networks Confusion Matrix on Clean Data (as percentages of the true positives for each class)

		Predicted Emotions					
		1	2	3	4	5	6
Actual Emotions	1	55.6	14.7	4.8	1.4	13.5	3.2
	2	15.1	63.6	4.8	2.8	15.6	4.1
	3	5.6	2.2	66.1	1.4	3.1	8.8
	4	0.8	3.6	3.2	90.3	3.1	2.3
	5	19.0	12.4	7.3	2.3	58.3	4.6
	6	4.0	3.6	13.7	1.9	6.3	77.0
Total		100.0	100.0	100.0	100.0	100.0	100.0

Fig 14. 6 Single-output Networks Confusion Matrix on Clean Data (as percentages of the predicted positives for each class)

	Emotions					
	1	2	3	4	5	6
Average Recall Rate	53.0	72.2	68.8	90.3	42.4	80.7
Average Precision Rate	55.6	63.6	66.1	90.3	58.3	77.0
Average F1 Measure	54.3	67.6	67.4	90.3	49.1	78.8

Fig 15. 6 Single-output networks statistics on Clean Data

5.3.2 Analysis on Noisy Data

On the noisy data, our classifier generally didn't perform as great as on the clean data, as reflected in the lower F1 measures on all classes (See Fig 18) than those on clean data. Particularly our classifier has a very bad precision rate of 10.2% (See Fig 16) on emotion 1, which is much lower than on the clean data. We notice from all the 10 confusion matrices generated from cross validation, emotion 1 is rarely recognised (in some folds the emotion has never even been recognised. Since we trained the network until the best epochs (on minimum mean squared error) reached, the network is unlikely to be extremely overfitted. It is therefore likely that the phenomenon is caused by the extreme difference between the clean data and noisy data on that particular emotion.

		Predicted Emotions						Total
		1	2	3	4	5	6	
Actual Emotions	1	10.2	22.7	22.7	12.5	20.5	11.4	100.0
	2	0.5	77.5	10.2	4.8	2.7	4.3	100.0
	3	1.1	10.2	57.2	12.3	4.3	15.0	100.0
	4	0.0	4.8	7.2	76.6	2.9	8.6	100.0
	5	4.5	14.5	16.4	8.2	35.5	20.9	100.0
	6	0.9	3.6	5.5	4.5	2.7	82.7	100.0

Fig 16. 6 Single-output Networks Confusion Matrix on Noisy Data (as percentages of the true positives for each class)

		Predicted Emotions					
		1	2	3	4	5	6
Actual Emotions	1	47.4	9.2	10.5	5.0	22.0	3.7
	2	5.3	66.5	9.9	4.1	6.1	3.0
	3	10.5	8.7	56.0	10.4	9.8	10.4
	4	0.0	4.6	7.9	72.1	7.3	6.7
	5	26.3	7.3	9.4	4.1	47.6	8.6
	6	10.5	3.7	6.3	4.5	7.3	67.7
Total		100.0	100.0	100.0	100.0	100.0	100.0

Fig 17. 6 Single-output Networks Confusion Matrix on Noisy Data (as percentages of the predicted positives for each class)

	Emotions					
	1	2	3	4	5	6
Average Recall Rate	10.2	77.5	57.2	76.6	35.5	82.7
Average Precision Rate	47.4	66.5	56.0	72.1	47.6	67.7
Average F1 Measure	16.8	71.6	56.6	74.3	40.7	74.5

Fig 18. 6 Single-output networks statistics on Noisy Data

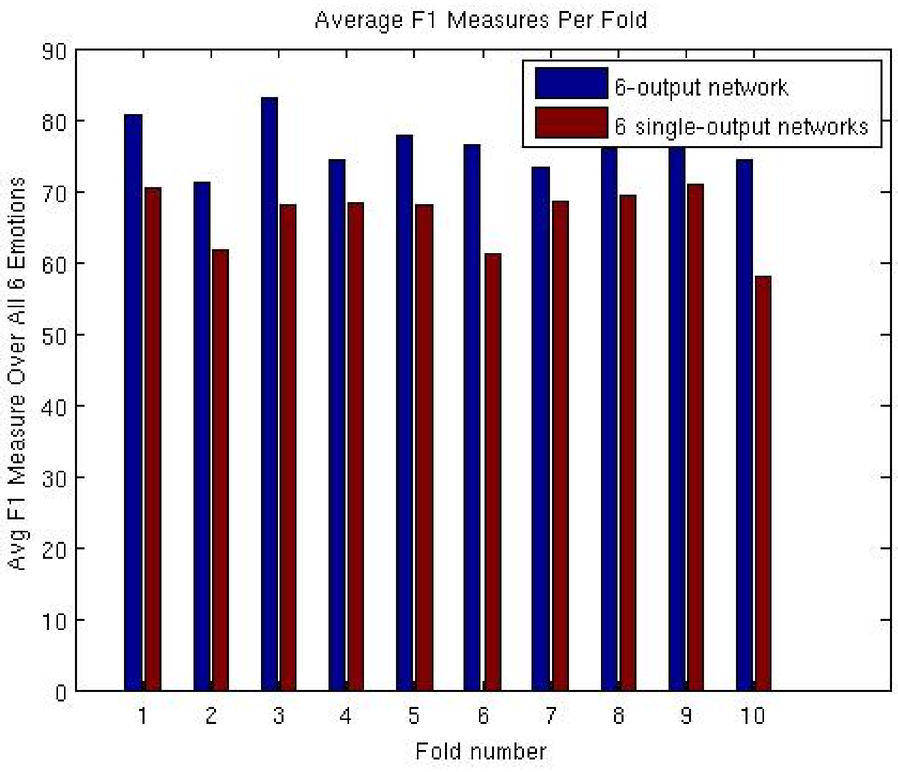


Fig 19. Bar chart of average F1 measures per fold over the 6 classes calculated on clean data

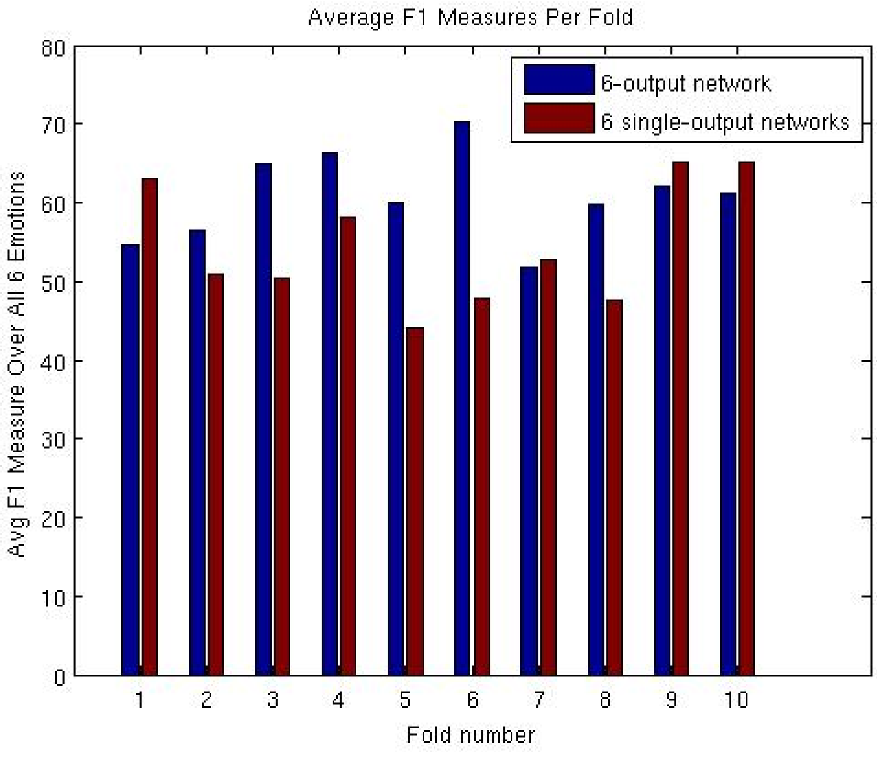


Fig 20. Bar chart of average F1 measures per fold over the 6 classes calculated on noisy data

6 Questions

1. Discuss how you obtained the optimal topology and optimal values of network parameters. Describe the performance measure you used (and explain why you preferred it over other measures) and the different topologies / parameters you experimented with.

For information on how the optimal topology and optimal values of network parameters were obtained, we have discussed in **section 4:** Optimization of ANNs
The set of parameters of an ANN on which we performed the optimization are an activation function, a training function, a number of layers, and a number of neurons in one layer.

Performance measure

We have chosen the mean squared error (MSE) as a measure of our performance. Other candidates included a classification rate, a precision rate, a recall rate, a F-alpha measure, an area under the ROC curve. Since it was not our aim to increase a precision or recall rate we did not consider them together with corresponding f-alpha measure and an area under the ROC curve. Taking a classification rate as a performance measure would not capture the idea that the network although classifying falsely might be close to classifying an example correctly and vice versa. Particularly, a value of 0.5 on an output neuron in a binary network corresponds when rounded to a class 1, however a slight adjustment of the weights might change the output to 0.49 resulting in a different classification. A family of square error functions like MSE or SSE captures the proximity to the correct classification and that was our main reason for choosing one of them. There is no difference in their power of expressivity as MSE is just a mean of squared errors and SSE is a sum of them.

ANN Topology

The **number of hidden layers** corresponds to having nonlinear compositions of linear combinations of the input vector and the neuron weights. Additional layers may provide an overfitted approximation to the target function. However, if the problem domain cannot be expressed with a simple general hypothesis producible by one layer with a simple activation function, a risk of overfitting is outweighed by an advantage of having possibly a more exact approximation of a target function. Given that decision trees with their modest expressive power (as compared to ANNs) performed reasonably well on our problem (classification rate around 69.5%), we have opted for ANN topologies with only one or two hidden layers. The **number of neurons per hidden layer** provides a higher granularity to the approximation of the function, but does not have an effect on overfitting. Hence we could prefer many neurons to fewer and we would do no worse. However, multiple neurons mean more computation and slower learning. Thus we want to choose always as few neurons as possible but still with a reasonable small classification error rate.

Activation Function

An activation function of one neuron is a building block for approximating the target function. Non-linear functions with non-constant derivatives have a higher expressive power than linear functions since they are better tailored to fit the training set. However, this would mean a higher possibility of overfitting.
Additionally, different types of machine learning problems are representable by different statistical distributions. The knowledge of a particular distribution might enlighten the process of decision of an

activation function. Not being familiar with our problem domain, we have chosen the following general MATLAB built-in functions as our candidate activation functions: *tansig* (symmetric sigmoid transfer function) - from a family of highly expressive functions, and *purelin* (linear transfer function) - a function with less expressive power.
We found that *purelin* produces a better performance than *tansig* for our classification, and this may be due to the fact that we have 2 hidden layers. With *tansig*, we apply it again to each hidden layer, therefore resulting in a very complex function to separate the classes during classification, whereas the *purelin* function always produces a linear hyperplane. The complex function produced by *tansig* would imply that the network has been trained to overfit the training data and would not be a good classifier to unseen data.

Training Function

A training function and its associated training algorithm both specify how the weights of the neurons are updated during the learning process. We have opted for a classical backpropagation algorithm with gradient descent (*traingd*) and for a backpropagation with Levenberg-Marquardt algorithm (*trainlm)*. The *trainlm* function interpolates between Gauss-Newton algorithm and gradient descent method. *Trainlm* is the most efficient known training algorithm for many machine learning problems. Both of the algorithms guarantee a convergence to a local minimum, with the later one being potentially faster. We have chosen a default **learning rate** supplied with MATLAB built-in training algorithms. Increasing the learning rate might increase the speed of learning or might cause the divergence away from the local minimum. Our performance measures showed that the network was learning - a training error was decreasing with the increased number of training samples provided. We considered additional configurable parameters to be of minor importance as their default configuration was already reasonably good.

2. Explain what strategy you employed to ensure good generalisation ability of the networks and overcome the problem of overfitting.

As mentioned earlier, during the topology design stage we kept in mind a problem of overfitting. We considered only ANNs with a few hidden layers to restrict the expressivity of an approximated target function and thus enforcing only a simple (more probable) hypothesis to be learnt. Another step was to include a linear activation function as one of the candidates for an optimal network parameter. Other parameters do not impose as strong risk of overfitting. We performed also cross-validation tests to find out how our chosen model of the network is likely to perform against the real data.
Additionally, we let MATLAB stop training our chosen network when we have reached the optimal number of epochs, which is determined by the minimum mean squared error against the number of epochs we have carried out for our validation set. This is because while the error on training set gets lower with more epochs, the error on the validation set stops decreasing at some point, meaning if we keep training the network after that point, it will just become more overfitting and lead to bad performance on unseen data.

3. In Part VIII you used the optimal parameters that you found in part VI to train your networks. However, there is a problem with this approach, the data you used for validation at some point will be used for testing in cross-validation. Can you explain why this a problem? Ideally how should you optimise the parameters in cross-validation?

The problem with optimising our parameters before running cross-validation is that our parameters are not necessarily the optimal ones for the specific networks generated in each fold. Also, the validation data used for optimisation of parameters may overlap with the testing data during cross-validation. If part of this validation data is reused for testing in cross-validation, then the test results will be biased on these parameters and the performance displayed will not be reliable. This means that the performance would appear much better than the network's true performance if it was given a completely new and unseen test set. Ideally, we should never test the network with previously seen data. A way of doing this would be to instead of cross-validating a network with optimal parameters that were determined beforehand, we find our optimal parameters during each cross-validation fold before testing. For a fixed set of potential networks with varying parameters, they will be trained and validated using the proportions shown in Figure 21. When the network with the optimal parameters is found, it is tested on the remaining 1/10.

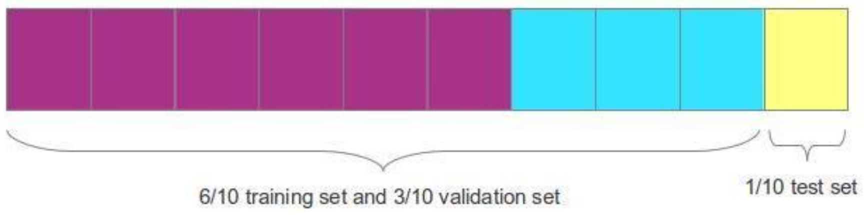


Fig. 21: Ideal splitting of data for parameter optimisation and cross validation

4. Is there any difference in the classification performance of the two different classification approaches. Discuss the advantages / disadvantages of using 6 single-output NNs vs. 1 six-output NNs.

According to our results, Network B (the 6 single-output networks) performed worse than Network A (the six-output network). This can be seen from our average classification rates for both the clean and noisy data (Fig 6). Network A produced a classification rate of about 10% higher than Network B. The bar charts show that the average F1 measures are generally significantly higher for Network A than for Network B (Figures 19 and 20).
A disadvantage in using 6 single-output networks could be that, even though we make sure that each of the 6 networks is trained on the exactly the same data, the initial weights of each network is set randomly. Hence, the initial weights will be different for each of the 6 networks so the training process will progress slightly differently. On the other hand, the six-output network is initialised with just one set of random weights, therefore the overall classifier is trained consistently. This could be a reason why Network A performs better when classifying the test data, compared to Network B.

7 Summary

We have considered a six output ANN and six binary classification ANNs with different parameters. We chose our optimal networks to be the ones with the least mse in a simple performance validation test. The two types of optimized networks had the same set of parameters and in our cross-validation tests, the single 6-output network performed better, with a classification rate of 80.09%.