Assignment 3: Artificial Neural Network

Implementation Details

Cross Validation

Cross Validation was implemented through the use of the NFoldCrossValidation functions which were closely modelled on the one used for Assignment 2. The exact same partitioning technique was used and the only differences in implementation were:

- calling ANNData to convert x and y to the correct format for creating an ANN
- use of createNetwork functions instead of decisionTreeLearning for training
- calling testANN instead of testTrees to compute the predicted classifications for the examples

Classification

Classification was performed using the testANN function which runs examples through the given network, takes the resulting output vectors and calls NNout2Labels to get a classification from each output vector. For each output vector, NNout2Labels returns the index of the first output element with maximal value; this index represents the assigned classification.

Training Functions

Each training function has its own implementation of the createNetwork, nFoldCrossValidation and optParams functions. This is because createNetwork takes different parameters for different training functions and nFoldCrossValidation makes calls to createNetwork. optParams also requires training function specific implementations because of the differing parameters to be optimised. For more information on the process of optimising parameters see the Optimisation of Topology and Parameters section.

Optimisation of Topology and Parameters

We choose to optimise the topology of the network, the parameters of the individual training functions and the ratio of training to validation data by having multiple nested loops within reasonable ranges. We computed the F1 values for each network created using 10 fold cross validation and compared it to the current best. At first we thought it best to take the network with the smallest value for the gradient, however we very soon noticed that this would completely disregard training data and thus be a very poor performance measure. Having noticed this we decided the most useful statistic would either be the F1 measure because it takes into account the recall and the precision rate.

Each member of the team was responsible for optimising all parameters including the topology of the network for one of the 4 training types: Gradient Descent with Backpropagation, Gradient Descent with adaptive learning rate backpropagation, Gradient descent with momentum backpropagation and Resilient backpropagation.

To optimise the topology of the network we selected the following ranges for the topology related parameters:

- Number of hidden neurons per layer: 40 – 85

  Having 45 inputs we thought that having less hidden neurons per layer may result in inaccuracy. Having a hidden layer size of more than double the number of inputs could lead to overfitting so we chose an upper limit of 85.

- Number of hidden layers: 1 - 3
  We wanted to avoid overfitting and considering any function can be approximated to arbitrary accuracy with 2 layers we figured the optimal must be close to this value.

We also set the range for possible ratios of training to validation data to be from 0.1 to 0.2. We initially optimised this parameter in isolation using the Gradient Descent with Backpropagation training method. We then took what appeared to be the best range and used that for optimisation in conjunction with the other parameters.

For the training functions which use a learning rate we set the range for this parameter to be 0.04 - 0.08. This range was chosen based on research into what MATLAB generally uses as a value. In general, having a learning rate greater than 0.08 would cause the weights to be too volatile and thus make the network unstable. Having a learning rate of less than 0.04 would cause the network to learn too slowly.

Gradient Descent with Backpropagation

We found the following optimal parameters:

No. Hidden Neurons per Layer	No. Hidden Layers	Training to Validation Ratio	Learning Rate
85	3	0.14	0.08

When using these we achieved an optimal F1 of 78.3 on the clean data set, and an F1 of 59.2 on the noisy data set.

Gradient Descent with Adaptive Learning Rate Backpropagation

We chose to use the following ranges for optimising the parameters:

- Learning rate increase: 1.0 – 1.1

  MATLAB suggests an lr_inc of 1.04, the notes suggest the range 1.05 - 1.1

- Learning rate decrease: 0.65 – 0.75

  MATLAB suggests an lr_dec of 0.7, the notes suggest the range 0.5 – 0.7

The purpose of these parameters is to make the learning rate responsive to the complexity of the local error surface. If lr_inc were too large or lr_dec too small, the learning rate would change too much and make the network unstable. If lr_inc were too small or lr_dec too large, there would be little effect and the result would be similar to normal Gradient Descent with Backpropagation.

We found the following optimal parameters:

No. Hidden Neurons per Layer	No. Hidden Layers	Training to Validation Ratio	Learning Rate	Learning Rate Increase	Learning Rate Decrease
70	2	0.1	0.08	1.0	0.75

Using these we achieved an optimal F1 of 76.9 on the clean data set and 58.1 on the noisy data set.

Gradient descent with Momentum Backpropagation

We chose to use the following ranges for optimising the parameters:

- Momentum constant: 0.85 – 0.95

  The momentum constant determines how much the weight change depends on the previous weight change. If it is too small the network is more susceptible to getting stuck in a local minimum whereas if it is too large the weight change will ignore the local gradient. This particular range was selected based on the general rule of thumb presented in the notes.

We found the following optimal parameters:

No. Hidden Neurons per Layer	No. Hidden Layers	Training to Validation Ratio	Learning Rate	Momentum Constant
40	1	0.2	0.04	0.85

Using these we achieved an optimal F1 of 59.9 on the clean data set and 38.5 on the noisy data set.

Resilient Backpropagation

We chose to use the following ranges for optimising the parameters:

- Delta used for incrementing: 1.05-1.4

  We chose these bounds by relying on the notes, which said that an optimal solution is usually found around 1.2.

- Delta used for decrementing: 0.4-0.7

  Again, we chose these bounds by relying on the notes, according to which an optimal solution can be found around 0.5.

The purpose of the two coefficients is to control the rate at which the weights are changing by taking into account the sign of the partial derivative. If the derivative maintains its sign during two consecutive turns, then the convergence is accelerated by multiplying with the Delta for incrementing; otherwise, it is slowed down, by multiplying with the Delta for decrementing. the fact that the Delta used for incrementing is close to 1, while the Delta used for Decrementing is far from 1 indicates that some examples were difficult to distinguish and needed careful approximation(obtained by quickly decelerating and slowly accelerating).

We found the following optimal parameters:

No. Hidden Neurons per Layer	No. Hidden Layers	Training to Validation Ratio	Delta for Incrementing	Delta for Decrementing
70	3	0.2	1.1	0.5

Using these we achieved an optimal F1 of 82.0 on the clean data set and F1 64.2 on the noisy data set.

Selection and Optimisation of The Network (Part VII)

The result of creating a network is non-deterministic given the same parameters, meaning that our F1 measures found in the Optimising Parameters stage may not be reliable. However, to gain more reliable results before selection would require repetitions of the experiment to calculate an average. Therefore, in the interest of time we have selected our optimal function and parameters based on the first results returned by our optParams functions.

We created the network by using the Resilient Backpropagation training method and the optimal parameters found for this method because they yielded the best F1 measures for both clean and noisy data.

Evaluation

The following results are from performing 10 fold cross validation on the full clean data set and the full noisy data set respectively. We used the optimal training function trainrp and its optimal parameters as found in the Optimisation of Topology and Parameters section.

Clean Data

	1: Anger	2:Disgust	3: Fear	4: Happiness	5: Sadness	6: Surprise
1: Anger	96	12	4	2	17	2
2: Disgust	13	156	2	6	18	4
3: Fear	5	4	90	1	5	15
4: Happiness	1	8	2	202	1	4
5: Sadness	10	17	4	3	93	6
6: Surprise	0	4	10	5	4	187

	1	**2**	**3**	**4**	**5**	**6**	**Average**
Recall Rate	72.2	78.4	75	96.7	69.9	89.0	79.5
Precision Rate	76.8	77.6	80.4	92.2	67.4	85.8	80.0
F1 measure	74.4	78.0	77.6	92.4	68.8	87.4	79.7
Classification Rate	0.935	0.913	0.949	0.967	0.916	0.947	0.938

Noisy Data

	1: Anger	2:Disgust	3: Fear	4: Happiness	5: Sadness	6: Surprise
1: Anger	15	18	21	13	19	3
2: Disgust	6	143	21	9	8	2
3: Fear	10	19	112	14	12	22
4: Happiness	12	15	23	152	6	4
5: Sadness	15	12	17	6	49	11
6: Surprise	11	26	20	8	10	146

	1	**2**	**3**	**4**	**5**	**6**	**Average**
Recall Rate	16.9	75.7	59.3	71.7	44.5	66.1	55.7
Precision Rate	21.7	61.4	52.3	75.2	47.1	77.7	55.9
F1 measure	19.0	67.8	55.6	73.4	45.8	71.4	55.5
Classification Rate	0.873	0.865	0.823	0.891	0.885	0.884	0.870

Analysis of Cross Validation Experiments

10 Fold cross validation showed that the best recognised emotions were Happiness (F1 of 92.4 for clean and 73.4 for noisy data) and Surprise (F1 of 89.0 for clean and 71.4 for noisy data). The emotions recognised with the least accuracy were Anger (F1 of 74.4 for clean and 19.0 for noisy data) and Sadness (F1 of 68.8 for clean and 45.8 for noisy data).

The average values for recall, precision and f1 are all very close together (varying by <1%) on both datasets. Average F1 measures for both clean and noisy data sets were considerably lower than the average classification rates.

Average classification was very high at 0.938 for clean data and 0.870 for noisy data. This could indicate that the classification rates were misleadingly high at times and not a good indicator for the performance of the network, particularly for the class Anger for which there was a large number of false positives and false negatives in the noisy data.

Across the classes, the classification rate varied little in clean and noisy data whereas the F1 measures were very different, particularly in the noisy data set where they ranged from 19.0 to 73.4.

Surprise and Fear were sometimes confused with one another; in the noisy data set 20 instances of Fear were classified as Surprise and 22 instances of Surprise were classified as Fear. Anger, Disgust and Sadness were also sometimes confused. These confusions can also be seen to a lesser extent in the clean data.

Performance on Clean and Noisy Datasets

When comparing the clean and noisy datasets we can see roughly what was expected; most of the classes dropped by the same proportion in all their measures, with only 1 notable exception. All measures for anger but the classification rate fell by a very large amount (Recall by 55.3%, Precision by 55.1% and F1 by 55.4%). This indicates that there was a serious problem with our network learning from the various examples on anger provided in the noisy dataset, possibly as a result of having some parameters set to a suboptimal value.

One other point of interest is that the recall rate for Disgust fell by barely 3%. This means that it has the highest recall rate for any class on the noisy data, in contrast to being 3rd highest on the clean data. The examples given for disgust in the noisy data set could have much less interference than the emotions with the best performance in the clean data set: happiness and surprise.

Overfitting could be a major cause for the performance drop from clean data to noisy data. In particular, the number of hidden layers used for these 10 fold cross validation experiments was 3 which is relatively large. Having a large number of hidden layers is known to make a network more susceptible to overfitting (see the Prevention of Overfitting section) and this could mean that our network is fitting too closely to noise in the training data thus making it perform worse on the test data as shown in our findings for the noisy data set.

One would expect the performance of the network to drop considerably from clean to noisy data because, during the optimising parameters stage, we calculated the performance of the networks being trained and tested on clean data. Therefore the optimal combination of parameters for the noisy data set could have been very different from those used in the cross validation experiments. For example, if we had repeated the process of parameter optimisation for noisy data, the optimal number of hidden layers is likely to have been smaller than 3 to avoid overfitting.

Prevention of Overfitting

Fewer Hidden Layers

Backpropagation in networks with many hidden layers is more susceptible to overfitting because errors are decreased to the point where the network only performs well on the training data and do not generalise well for unseen test data. Networks with more hidden layers also take longer to train.

For this reason, when trying to optimise the number of hidden layers, we chose a small range (from 1 to at most 5 hidden layers) to prevent overfitting. This proved to be effective, as the optimal number of layers was on all occasions at most 3.

Early Stopping

Overfitting can be detected during backpropagation when the error in the training set is decreasing but the in the validation set it is increasing. If this has been the case after a certain number of iterations, early stopping can be used to halt the backpropagation process and therefore stop the network from overfitting further. It is important to consider how many iterations can pass before early stopping is invoked because error in the validation set can reach local maxima before decreasing again. To prevent overfitting, MATLAB sets the maximum number of iterations to 6.

Regularisation

Another method which could have been used to avoid overfitting would be to decrease the weights by a small weight decay factor during each epoch. Networks with larger weights are prone to overfitting because larger weights are needed to accommodate outliers. Selection of this weight decay factor is important to ensure that the weights remain within a reasonable range.

MATLAB provides some options for regularisation with a regularisation constant in net.performParam which is used by the performance function net.performFcn when checking the network performance.

One Network Approach versus Six Networks Approach

Besides the required theoretical comparison, we also did a practical comparison by constructing both one network and six networks. However, when we constructed six neural networks, we found it difficult to combine the results of the neural networks in the situations where more than two networks reported a positive result on the same example or when none reported a positive result, as we had to determine which network is more accurate or less inaccurate. This comparison was hard to make because the six networks differed only by their weights and it's difficult to compare their 'quality' based on this difference alone. Therefore, the One Networks Approach proved more effective as a whole. However, the Six Networks approach could produce more accurate results for each emotion individually as it approximates a simpler function and therefore it may need a smaller neural network, which in turn will reduce the possibility of overfitting.

A way of decreasing the errors for both the One Network Approach and the Six Network Approach is to create for each example networks using all four network training functions and then to choose the emotion obtained by the majority of training functions. Assuming that the probability of getting an erroneous result by using the network/combination of networks obtained by a training function is p (where 0 <= p <= 1), and that the training functions make independent errors, then the probability of k out of n networks producing an error is:

$$Comb(n, k) * pk * (1 - p) n - k$$

where Comb(n, k) means combination of n taken k. Therefore, the possibility of the majority being wrong is:

$$k > n/2n\ Comb(n, k) * pk * (1 – p) n - k$$

To illustrate this, if there are four training functions, each with an accuracy of 70%, then the possibility of the majority being wrong is 8.37%; if there are four functions, each with accuracy 60%, then the possibility that the majority is wrong will be 17.92%. We can see that in both cases, the error rate has been reduced significantly; however, the more accurate the training functions, the bigger the improvement. While this may not matter very much for the One Network Approach, it is highly relevant for the Six Networks Approach, where each individual network is significantly more accurate than the combination of the networks. Therefore, the optimal procedure for the Six Networks Approach would be to apply the majority decision for each individual emotion, then to combine the results.