Neural Networks

Implementation Details

Our implementation works as follows: the function createNN takes the data to train on, the training parameters, and a flag for whether we want 1 or 6 trees outputted, and builds a network on that data using the Matlab ANN functions (feedforwardnet,configure,train). We used the 'useParallel' and 'useGPU' flags to speed up the training process.

bestMatch in testANN normalises the results found by running a simulation on the 6-output network, and combined the outputs of the six 1-output networks: for each of the 1004 examples, it takes the highest value in the 6 emotions and set it to 1, and all the others are set to 0. It then produced the predicted labels y by passing through NNout2labels.

In crossValidateANN, fold generated the same folds as in the decision tree exercise. We then performed parameter optimisation using paramOpt, which iterated through a set of parameters as below for the first fold training and validation data. Then those optimal parameters are used to generate the ANNs for the rest of the folds, and returns the classification rates for those folds in an array.

Average classification rate of 20 iterations of cross validation

	Clean	Noisy
One neural network	81.9%	70.3%
Six neural networks	83.2%	72.1%

Noisy One		***Predicted Class***					
		Anger	Disgust	Fear	Happiness	Sadness	Surprise
Actual Class	Anger	12	14	19	8	16	4
	Disgust	5	124	11	5	6	1
	Fear	3	11	105	11	7	16
	Happiness	2	7	9	135	4	6
	Sadness	9	9	13	6	45	10
	Surprise	2	3	9	6	6	143

Noisy Six		***Predicted Class***					
		Anger	Disgust	Fear	Happiness	Sadness	Surprise
Actual Class	Anger	14	12	21	9	15	4
	Disgust	4	125	11	4	4	1
	Fear	5	10	105	11	6	15
	Happiness	3	6	8	137	4	5
	Sadness	9	9	13	5	48	9
	Surprise	2	2	10	6	6	144

Clean One		***Predicted Class***					
		Anger	Disgust	Fear	Happiness	Sadness	Surprise
Actual Class	Anger	77	10	5	2	12	2
	Disgust	8	126	2	5	14	2
	Fear	4	3	78	2	3	12
	Happiness	1	5	1	163	1	2
	Sadness	11	15	4	4	62	4
	Surprise	0	2	10	2	4	148

Clean Six		***Predicted Class***					
		Anger	Disgust	Fear	Happiness	Sadness	Surprise
Actual Class	Anger	77	9	5	3	11	2
	Disgust	8	130	2	5	12	1
	Fear	3	2	81	2	2	11
	Happiness	0	4	1	166	1	1
	Sadness	9	15	3	4	66	4
	Surprise	0	2	6	3	3	152

Average Precision		*Class*					
		Anger	Disgust	Fear	Happiness	Sadness	Surprise
Neural Network	Clean One	76.2%	78.3%	78.0%	91.6%	64.6%	87.1%
	Clean Six	79.4%	80.2%	82.7%	90.7%	69.5%	88.9%
	Noisy One	36.4%	73.8%	63.3%	78.9%	53.6%	79.4%
	Noisy Six	37.8%	76.2%	62.5%	79.7%	57.8%	80.9%

Average Recall		*Class*					
		Anger	Disgust	Fear	Happiness	Sadness	Surprise
Neural Network	Clean One	71.3%	80.3%	76.5%	94.2%	62.0%	89.2%
	Clean Six	72.0%	82.3%	80.2%	96.0%	65.3%	91.6%
	Noisy One	16.4%	81.6%	68.6%	82.8%	48.9%	84.6%
	Noisy Six	18.7%	83.9%	69.1%	84.0%	51.6%	84.7%

F1-Measure		*Class*					
		Anger	Disgust	Fear	Happiness	Sadness	Surprise
Neural Network	Clean One	73.7%	79.2%	77.2%	92.9%	63.3%	88.1%
	Clean Six	75.5%	81.3%	81.4%	93.3%	67.3%	90.2%
	Noisy One	22.6%	77.5%	65.8%	80.8%	51.1%	81.9%
	Noisy Six	25.0%	79.9%	65.6%	81.8%	54.5%	82.8%

From this, we can see some differences in terms of accuracy in different emotions. Disgust, happiness and surprise are quite accurate (> 75% precision, recall and F1), with happiness and surprise being above 90% on the clean data, while anger is not accurate at all on the noisy data (< 25% accuracy on both the clean and the noisy data). This could be because anger does not have selective enough properties, therefore it can easily be mismatched for another emotion.

The figures also show that building six neural networks gives slightly better results (~2% better for each emotion) with a peak for happiness with 93.3% accuracy for the F1-measure.

Optimisation

The parameters we optimised were:

- topologies = {40,20,10,[40,20],[30,10]};
- Training functions:
    - Levenberg-Marquardt backpropagation
    - Resilient backpropagation
    - Gradient descent with momentum backpropagation
- Lean Rates = {0.5,0.6,0.7,0.8,0.9,1}

- momentums = {0.5,0.6,0.7,0.8,10.9}
- activation functions: Tan-sigmoid and Log-sigmoid
- epochs = [10,100,1000];

We performed an exhaustive search on the possible combinations to ensure we found the optimum values. These values were then saved for later use in our validation (as the search took multiple hours to run). The optimum values we found were:

One Neural Network:
Topology: [40,20]
Training fn: trainrp
Learn Rate: 0.8
Momentum: 0.6
Activation fn: Tan-sigmoid
Epochs: 1000

Six Neural Networks:
Topology: [40,20]
Training fn: trainrp
Learn Rate: 0.7
Momentum: 0.7
Activation fn: Tan-sigmoid
Epochs: 1000

Because these values are optimal, we have avoided the problem of overfitting.
Logically, if our parameter values overfitted the data then they would not be optimal as the classification rate would fall.

Comparing classification approaches

Using 6 one-output neural networks resulted in a slightly higher classification rate. However, there is a roughly six-fold time penalty as each network has to be trained and tested separately. However, this problem could be mitigated on a multi-core system as the networks can be run in parallel. Although, as Matlab has built-in concurrency support, this would not provide an advantage for our implementation as the single six-output network can be run concurrently too.

Ideal parameter optimisation

Imagine we run 10 fold cross validation on our neural network for a given set of parameters, and the classification rate for each fold is given below:

60%	70%	75%	69%	72%	64%	63%	58%	69%	66%

This set of parameters would be judged solely on the performance of its initial fold, which has a classification rate of 60%, compared to an average that is obviously higher. We may find that another set of parameters gives the following classification rate

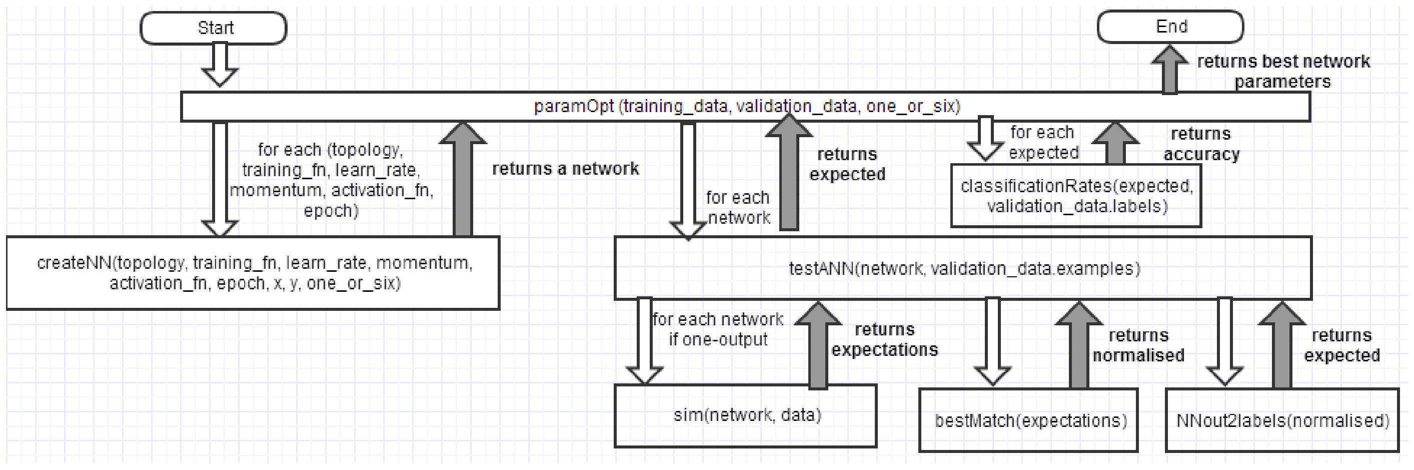
66%	55%	62%	61%	52%	53%	63%	58%	67%	56%

This parameters set would be judged to be better performing simply because the first fold performs better, whereas by inspection it is quite easy to see that our first parameter set performs substantially better.

It may seem as though the examples are quite contrived, but in actuality our data has shown that the variance of the classification rates between the folds can be as high as 15% in absolute terms.

Ideally we would compare every parameter set through complete 10-fold cross validation, as it would give us more reliable data, although in practice, it may not be possible due to time constraints.

Flowchart illustrating how code works


Start
End
returns best network parameters
paramOpt (training_data, validation_data, one_or_six)
for each (topology, training_fn, learn_rate, momentum, activation_fn, epoch)
returns a network
for each network
returns expected
for each expected
returns accuracy
classificationRates(expected, validation_data.labels)
createNN(topology, training_fn, learn_rate, momentum, activation_fn, epoch, x, y, one_or_six)
testANN(network, validation_data.examples)
for each network if one-output
returns expectations
returns normalised
returns expected
sim(network, data)
bestMatch(expectations)
NNout2labels(normalised)