Your report was a bit unclear in many sections concerning how the network
was fine-tuned and what exact results you had for every parameter that you finetuned.
Further, points were deducted because you were missing plots/tables to show comparisons.

CO395 - CBC Group 61
Neural Networks Coursework

Belen Barbed, Devin Nanayakkara, Piotr Pomienski, Benjamin Withers

March 8, 2018

Abstract

This report covers our implementation of a neural network trained to classify facial images into one of 7 emotions. The hyperparameter optimisation process is detailed step by step and the performance results of the final network is presented. Finally, the questions given in the coursework specification are answered.

1 Introduction

Previously we have used Decision Trees to classify facial expressions into one of 6 basic emotions: anger, disgust, fear, happiness, sadness and surprise. However, the input data given to us was in the form of Action Units (AU), essentially combinations of different active muscles in a face that result in facial expressions that we as humans can perceive as emotions. Whilst trivial for humans, identifying emotions is still quite difficult for computers to do correctly. We found that Decision Trees were good at classifying emotions from arrays of AUs, however this would not be the case in a practical example, such as providing an image of a person's face as input. For this, a more advanced machine learning technique is required, namely 'Neural Networks'.

For this exercise, we were provided 30x30 pixel greyscale images of people's faces, along with the target for that image, which was one of the 6 basic emotions: anger, disgust, fear, happiness, sadness and surprise, as well as a 7th, neutral. We were provided a toolkit for initialising, training and testing a neural network, as well as steps on how to optimise the hyperparameters (which we will simply refer to as parameters for the rest of this report) in order to achieve the highest classification performance when the network was run on the 'unseen' test set given.

2 Parameter Optimisation

The optimisation process given to us consisted of the 8 steps covered below. We initially believed that each step would seek to improve the neural network's classification performance, however we quickly discovered that by changing or enabling some parameters, earlier changes had to be neglected. A good example was enabling dropout, which we explain in Section 2.4.

2.1 Initial Parameters and Network Architecture

The initial network, before any tweaking, had the following properties:

- Layers: 3 hidden layers, with 500, 500 and 1000 neurons respectively; 7 neurons output layer; Initialised using He method
- Activation functions: Rectified Linear Unit (ReLu) for hidden layers, softmax for output
- Learning rate: Initial LR $lr_0 = 0.001$, decaying from epoch threshold $T = 10$ as per the following:

$$lr_t = \frac{lr_0 * T}{max(t, T)}$$

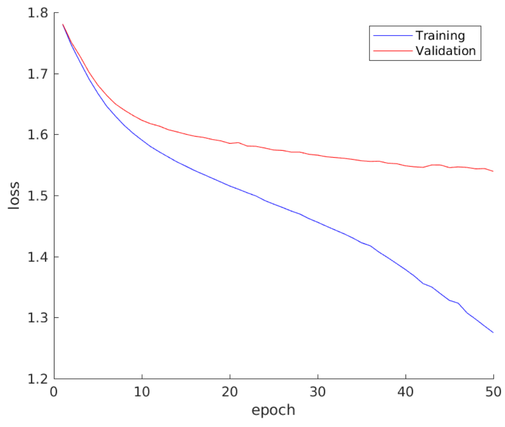
- Momentum: 0.5 initially, linearly increasing to 0.9 between epoch 10 and 40

- Early stopping: if classification error on the validation set rises 5 consecutive times then stop training, otherwise train for 50 epochs
- Regularisation/Dropout: disabled
- Duration: 50 epochs

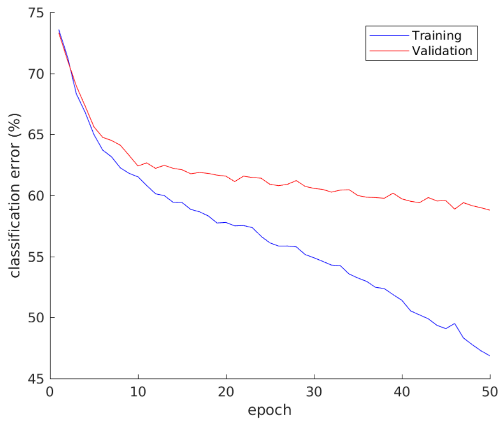
We went with 3 hidden layers of 500, 500 and 1000 neurons initially, as this is what the default configuration given in the toolkit example code was. We left this topology as it was, as it gave us decent performance. Most functions can be approximated using 1 hidden layer, however with a complex classifier like in our case, more layers made sense and the number of neurons looked like a good starting point for later testing. Note that the input layer had 900 neurons, as each row in our dataset had 900 columns, representing the 30x30 pixels per image.

Again, we used the example configuration for the activations functions per layer, three ReLu functions and a softmax output function, for initial testing. We attempted using sigmoid functions, but this gave less desirable results. We discovered later that some combinations of functions resulted in decent performance, which we shall elaborate on later. Note that since we are using continuous data inputs, the input layer uses a linear activation function.

This network scored a classification rate of 42% on both the test set and the validation set, with a training set score of 53%.



(a) Loss



(b) Classification error

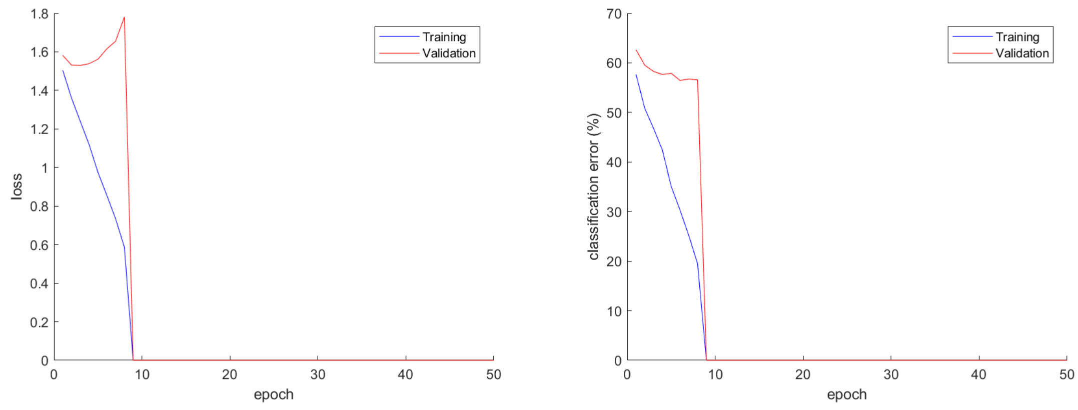
Figure 1: Performance of the network trained using initial parameters

2.2 Initial Learning Rate

Finding a reasonable initial learning rate was the first step in optimisation of the network. This parameter is responsible for the behaviour of the network in the first few epochs, and should be high enough to enable both the loss and classification error to fall rapidly in the first few cycles. The default rate proved to generate a very shallow loss curve, making the learning process to take very long before achieving reasonable results. Therefore, we decided that the rate should be increased, to make the network converge faster.

Extremely high rates, for example 0.100, were giving a small improvement in classification rates compared to the baseline, but after a few epochs they caused the loss to raise sharply, quickly triggering the early stopping criterion. Our early stopping proved to be quite aggressive, as it is relatively easy to have 5 subsequent increases in the error rate. It often stopped training even before 10 epochs, as shown in 2a.

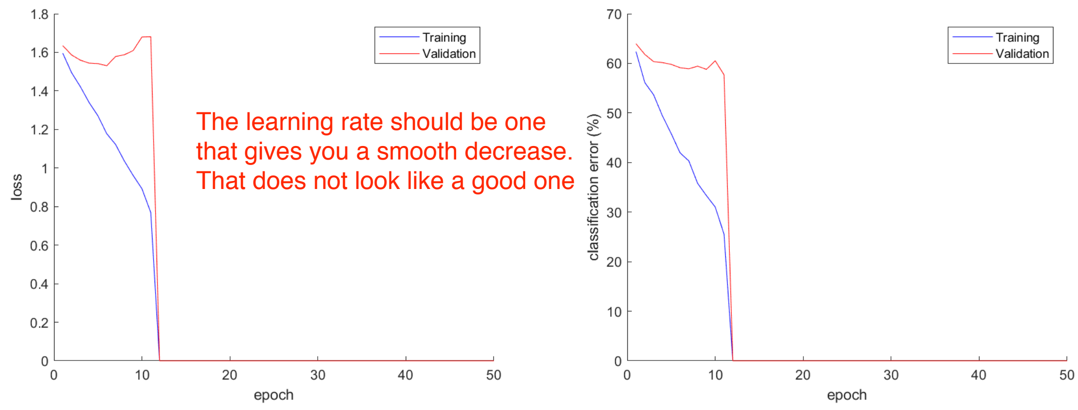
Unclear: What architecture did you use to fine-tune LR, and was regularisation enabled or not? how many values did you use to fine-tune it?
Also missing: it is not clear how you choose the best learning rate? (it should be the one selected that results in an immediate smooth decrease of the training loss



(a) Loss (b) Classification error

Figure 2: Performance of a very high LR network

However, this would potentially be rectified by tweaking the update schedule later on. Therefore, mostly the behaviour and performance in the beginning of the learning process was considered when picking the final value. 0.0125 was accepted as a good compromise between having a good loss and CR right at the beginning while still being able to scale the learning rate later on. This value also resulted in a slight increase in classification rates across all 3 sets.



The learning rate should be one that gives you a smooth decrease. That does not look like a good one

(a) Loss (b) Classification error

Figure 3: Performance of a 0.0125 LR network

2.3 Learning Rate Update Schedule

We considered 3 ways of decaying the learning rate throughout the process:

$$lr_t = \frac{lr_0 * T}{max(t, T)} \tag{1}$$

$$lr_t = lr_{t-1} * scalingFactor \tag{2}$$

$$lr_t = \frac{lr_0}{1 + \frac{t}{T}} \tag{3}$$

The first method was the one used initially. It made no sense to move the threshold further up - the loss curve was rising before epoch 10, so the decay had to start earlier. However, while moving it back improved the shape of the loss curve, the classification rate was worse. Similarly, the best performance when using the third method was worse than the baseline figure, so this method was not used.

Using the second method also resulted in a more controlled loss curve, and though there were no sizeable improvements in terms of classification error, there was no decrease either. Therefore, this method was used from that point on. The optimal value for the scaling parameter was between 0.85 and 0.9, due to the small differences between runs it was impossible to determine a single best value.

Table 1 shows the performance of the network on the validation set using all methods and various parameters. LRscaling for method 2 was set to 0.85 as previously mentioned, and in the case of method 3, the threshold was set to 2 instead of 1, as the equation resolves to NaN if 1 is used.

Method	Thr = 1	Thr = 5	Thr = 10
1	41.21	40.02	43.11
2	41.60	43.16	42.21
3	40.76	42.35	40.35

Unclear: is this the CR on the validation set?

Table 1: Classification rate (%) using different LR decay methods

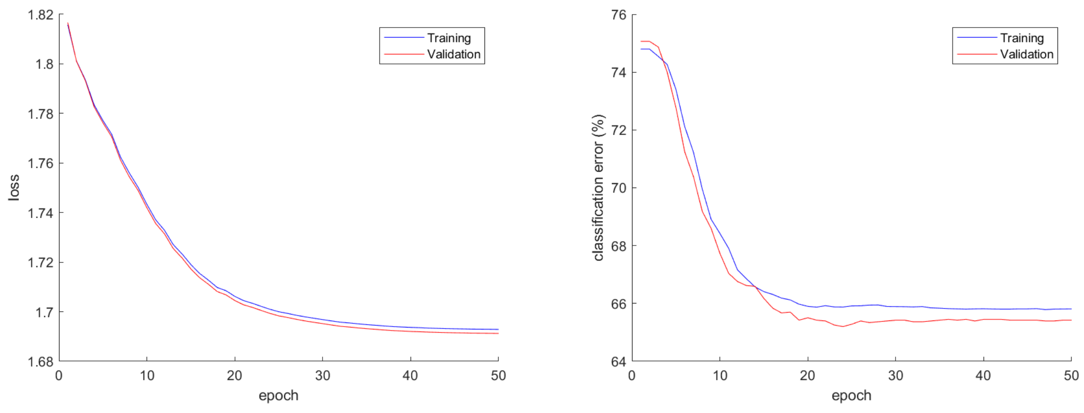
Our neural network's classification rate was not reacting much to these changes, which clearly suggested that there are other parameters that need changing first, before this can be fully optimised.

2.4 Dropout

After tweaking learning rates, we observed very little change, because there was still the major problem of overfitting. This occurs when the learning process adapts to the specifics of the training set too well, losing the capability of generalising on other sets. When this happens, the classification rate continues to fall on the training set, but not on the verification and test sets. Bernoulli dropout aims to prevent overfitting by randomly removing neurons from the network, with a fixed probability - 0.5 for hidden neurons, 0.8 for inputs. Enabling dropout while keeping other network

Are these the default values, or how were the probabilities chosen?

parameters as above gave very poor results. Even though overfitting was less of a problem, the classification rate itself dropped significantly compared to disabling dropout(4b), and the network was taking much too long to learn. Additionally, the classification error graph (4a) is noticeably less smooth especially in the later epochs. This is due to the random nature of dropout, which can cause a temporary increase in loss and/or classification error. Early stopping had to be disabled, as it often triggered because of these spikes.



(a) Loss (b) Classification error

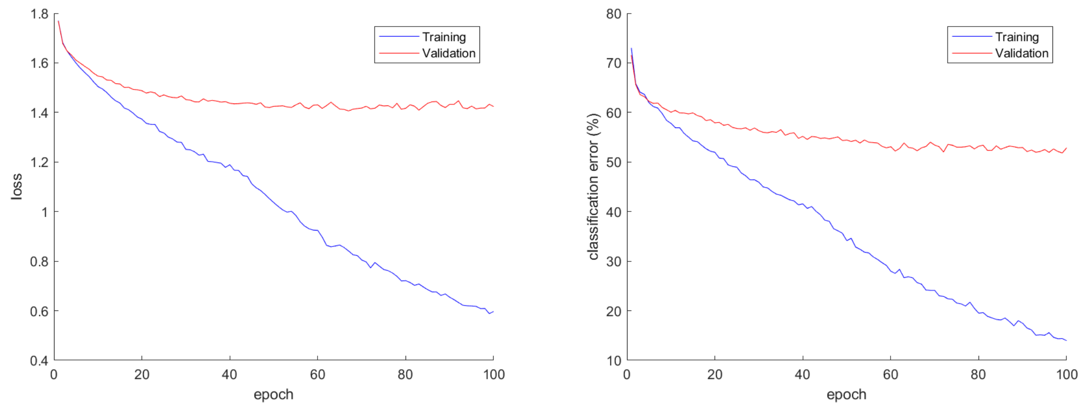
Figure 4: Network from step 3 with Bernoulli dropout enabled

However, it was mentioned in the original research paper [1] that when using dropout, the network needs to be adjusted to achieve good performance. In particular, the following were recommended:

- A higher learning rate (10-100x of the original) with less scaling
- More epochs
- Larger momentum

- Enabling max-norm clamping

Raising the learning rate quite drastically to 0.05, with a much reduced scaling factor of 0.995 gave better results than anything else tried previously. Seeing that after 50 epochs the classification rate was still falling, the number of epochs was also increased to 100, to give really promising results on the validation set.



(a) Loss (b) Classification error

Figure 5: Tuned network with Bernoulli dropout enabled

Running this network on our test set gave a classification rate of 49.1% . This is almost 7% higher than the previously found "best" configuration without dropout.

Max-norm is supposed to enable even higher learning rates, as it clamps the maximum weight of synapses incoming into the neuron. We experimented a bit with this option using the recommended typical values of 3-4, though in the end we found that it made very little difference to the performance of the net. Similarly, neither increasing nor disabling the momentum scaling did not allow us to break the 50% barrier for the classification rate.

The main downside to this method is that the result networks can vary in performance between training runs. This is due to the randomness introduced by dropout, and from what we have seen, the margin is in the region of 2%. However, even the worst case scenario figure of 47% CR is still better than the best non-dropout network configuration we were able to find.

2.5 Regularisation

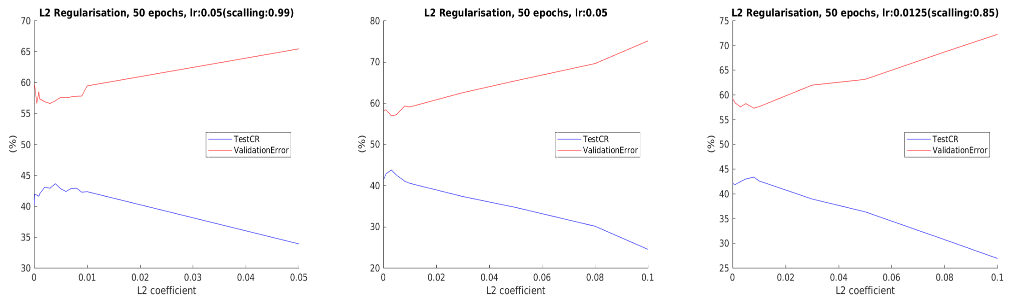
In Section 2.4 we used the dropout technique to observe the performance of our neural network's architecture which resulted in 49.1% classification rate. In this section we examine the performance of our network using the L2 and L1 regularisation methods and compare with the results from dropout. The network parameter settings were the parameters set before dropout and the parameters used for dropout. Larger versions of graphs in Figures 6 and 7 are available in the Appendix.

Why the same parameters used for dropout if you do not use dropout?

2.5.1 L2

L2 regularisation is an attempt to minimise the original error function (E_0) using the following equation:

$$E = E_0 + 0.5 * \lambda_2 \sum_{allWeights} w^2 \quad (4)$$



(a) Setting 1 (b) Setting 2 (c) Setting 3

Figure 6: Performance of Validation Error and CR on Test Set for Different L2 Coefficients

From which range was the optimal L2 found?

Setting	Network Parameters	Optimal L2 Coefficient(λ_2)	CR (%) on Validation	CR (%) on Test Set
1	InitialLR: 0.05, LRscaling: 0.99, Epochs: 50	0.004	43.25	43.81
2	InitialLR: 0.05, LRscheduling: 1, Epochs: 50	0.003	43.90	44.01
3	InitialLR: 0.0125, LRscaling: 0.85, Epochs: 50	0.008	43.12	43.35

Table 2: L2 Regularisation Performance

By observing Figure 6, we can deduce that there exists an optimal value for the λ_2 coefficient. This reflects a property of uniqueness in the solution of a L2 regularisation. In this case, L2 coefficients above 0.05 performed poorly compared to the performance around 0.005. The performance also starts to decrease for coefficients below 0.001.

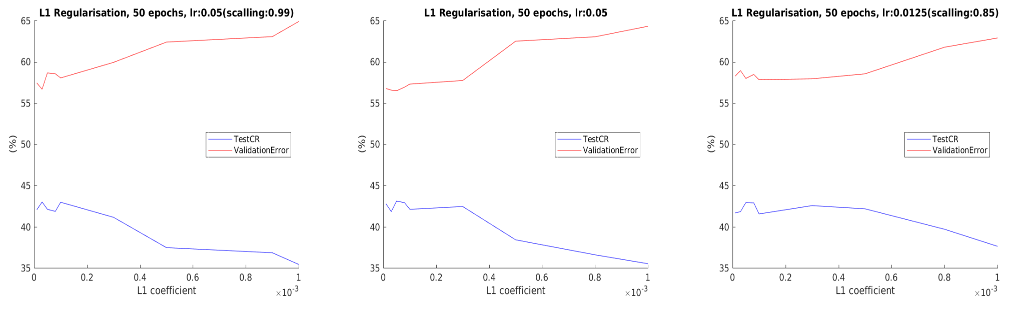
The parameter optimised was the L2 regularisation coefficient (λ_2) defined as the parameter *weightConstraints.weightPenaltyL2*. Setting 3 was used to compare the performance against the network that does not use any regularisation. The classification rate of the validation set (43.12%) performed better by just over 1%. Hence, no significant change was present to be seen. The Setting 1 and 2 were used to compare the performance of L2 regularisation against network with dropout method. The network with L2 regularisation had a classification rate of about 44% on validation and test sets and about 65% on the training set. However the dropout method (49%) outperforms the L2 regularisation method significantly.

It is really difficult to understand, how you actually fine-tuned L2. I cannot seem to recognise any systematic way that you guys did.

2.5.2 L1

L1 regularisation is an attempt to minimise the original error function (E_0) using the following equation:

$$E = E_0 + \lambda_1 \sum_{allWeights} |w| \tag{5}$$



(a) Setting 1 (b) Setting 2 (c) Setting 3

Figure 7: Performance of Validation Error and CR on Test Set for Different L1 Coefficients

Setting	Network Parameters	Optimal L1 Coefficient(λ_1)	CR (%) on Validation	CR (%) on Test Set
1	InitialLR: 0.05, LRscaling: 0.99, Epochs: 50	0.00003	43.33	43.00
2	InitialLR: 0.05, LRscheduling: 1, Epochs: 50	0.00005	43.52	43.10
3	InitialLR: 0.0125, LRscaling: 0.85, Epochs: 50	0.00005	42.02	42.91

Table 3: L1 Regularisation Performance

Unlike Figure 6 in L2 regularisation, Figure 7 does not reflect the existence of a unique solution (similar performance for different λ_1 coefficients). L1 coefficients above 0.008 and below 0.00001 results in poor performance where the validation error is over 70%.

Here the L1 regularisation coefficient (λ_1) defined by *weightConstraints.weightPenaltyL1* parameter was optimised. The Setting 1 was used to perform comparison against the network without regularisation. Similar to L2 regularisation, this did not have a significant performance over the network without regularisation (only 42.91% , less than 1% improvement). The Setting 1 and 2, which were used to perform comparisons with the network with dropout, did not improve any more than the L2 regularisation. The L1 regularisation method gave a classification rate of about 43.5% on validation and 43% on the test set compared to the 49% test score when using dropout.

When the performance of L2 and L1 regularisation methods are compared, it can be said that L2 performs better than L1. Consider the following equations, where the Equations 4 and 5 are differentiated by the weight (w).

$$\text{L2}: \frac{\partial E}{\partial w} = \frac{\partial E_0}{\partial w} + \lambda_2 w \rightarrow \Delta w = -\eta \frac{\partial E_0}{\partial w} - \eta \lambda_2 w \tag{6}$$

$$\text{L1}: \frac{\partial E}{\partial w} = \frac{\partial E_0}{\partial w} + \lambda_1 sign(w) \rightarrow \Delta w = -\eta \frac{\partial E_0}{\partial w} - \eta \lambda_1 sign(w) \tag{7}$$

From Equation 7 it is clear that in L1 regularisation, the weights shrink by a constant amount, thus introducing sparsity to the network. This also introduces a certain degree of feature selection due to some weights being zero or very small. However, L1 regularisation performs better only in sparse feature spaces. If the weights are large, L1 regularisation will not be the best. In comparison, from Equation 6, L2 regularisation shrinks weights by an amount proportional to the weight (w). For a general situation, L2 will perform better than L1 regularisation which is reflected on our results.

Regularisation is mainly used to avoid overfitting. In general, the regularisation term in Equation 4 and 5 prevent the weight coefficients from being absolutely perfect such that the model causes overfitting. Overfitting happens when the model learns the background noise which can lead to false prediction. Regularisation makes the model less complex which avoids the noise in data to a certain extent [2].

2.6 Network Topology

Once the previous parameters were optimised, we went on to alter our network topology to further adapt our network to the problem at hand. In order to do this, two major parameters were changed: the number of layers and the number of hidden neurons per layer.

2.6.1 Number of Layers

The general consensus regarding neural networks is that most problems (save a few complex ones) can be approximated using only one hidden layer between the input and output layers. More complex issues use a second hidden layer, but adding more and more reduces the network's capability to generalise outside the training set. The more unnecessary nodes there are, the more it becomes a memory rather than an equation solver [3]. To avoid such overfitting, keeping the number of layers to a minimum is essential. Even though facial recognition and analysis can be regarded as a complex problem, the best results were obtained using a small number of layers.

Linear problems can be solved without any hidden layers, so when using this approach we expected poor performance. As shown below, the average CR obtained on the test set with this network was approximately 36%. What was deduced from this result is that linear approximation is not enough.

When the first hidden layer was added to the network, performance increased massively, and is extremely close to our optimal solution. Having only one hidden layer guaranteed a CR on the unseen test set of 47%-48%. This performance is a mere 1% off our best network configuration, supporting the claims made in the literature of being able to approximate most problems with only one layer between input and output.

With the objective to get an even more accurate approximation in the classifier, deeper networks were designed and tested. The performance of these Deep Neural Nets peaked at three hidden layers, which was surprising because there is evidence that there is no benefit of employing more than two hidden layers in a network [4]. Despite this, our two layer NN was outperformed by the three-layered version, but beyond that point we experienced diminishing returns when further adding hidden layers to our current topology.

No. of Hidden Layers	Average CR on Test Set
0	36.31%
1	46.89%
2	47.23%
3	49.06%
4	43.91%

Table 4: Relation between the number of Hidden layers in the NN and its performance

2.6.2 Number of Hidden Neurons per Layer

Once it had been found that we obtained best results from having three hidden layers, we sought to find the optimal number of hidden neurons per layer. A good rule of thumb to begin with is to have an amount of hidden neurons per layer between the number of inputs (900) and outputs (7), while also maintaining the total amount of hidden nodes smaller than twice the size of the input layer [4]. We used these heuristics to start tweaking this parameter.

It was found that the network behaved better when the hidden layers decreased in size along the network, which is not unusual in image recognition problems that involve finding features and abstracting them - in our case features like raised eyebrows, open mouths, etc [5].

Once we roughly determined the range within which we obtained the best results (500-750 hidden neurons per layer), finding the exact amount proved difficult, as the stochastic process of dropout meant the same parameters would not result in identical network topology of the final network. It is for this reason that the configuration of [750 750 500] was chosen, and with it we achieved maximum CR on the test set.

Missing: Plot which shows the validation performance as function of the number of hidden neurons

2.7 Activation Functions

Which is?

By this point, our configuration gave very good results, so we tried only changing the hidden layer activation functions. However, after a couple of runs, we discovered that there was almost no change in performance when using ReLu x3 or leakyReLu x3 or even ReLu ReLu sigm. However sigm x3 gave terrible results, roughly in the 35% range compared the 48% range we were getting for the other combinations.

In addition, we attempted a 1 hidden layer configuration with different activation functions, yet this also led to similar results. ReLu and leakyReLu gave similar results in the 47-48% range, sigm resulted in sub-par performance, again around 35%. This led us to believe, at least in these cases, that the activation functions used have very little effect compared to changing the network topology or enabling dropout for example. The only thing to consider is ensuring that the activation functions used are suitable enough, e.g. using ReLu or its derivative, leakyReLu, and not using sigm for all layers.

What are the results using the best activation function?

2.8 Final Parameters and Network Architecture

The most optimal set of parameters was found during our dropout experiments. As we conducted quite an extensive search at that point, there was nothing else left to tweak. Our final network was trained using these values:

- Layers: 3 hidden layers, with 750, 750 and 5 neurons respectively; 7 neurons output layer; Initialized using LeCun method
- Activation functions: Rectified Linear Unit (ReLu) for hidden layers, softmax for output
- Learning rate: Initial LR $lr_0 = 0.05$, decaying from epoch threshold $T = 1$ as per the following:
$$lr_t = lr_{t-1} * scalingFactor$$
- Momentum: 0.5 initially, immediately increasing to 0.9 at epoch 2
- Early stopping: disabled
- Regularisation/Dropout: Bernoulli dropout with p=0.8 for inputs and p=0.5 for hidden layers, L1/L2 disabled
- Duration: 125 epochs

Missing:
* Figures showing the training/validation loss and training/validation classification error for the optimal network and parameters
* Comparison of the training/validation loss curves.
* Comparison of training/validation classification error curves.

3 Results

This section contains the results from testing the performance of our network on the test set. The network was trained with the optimal set of parameters. The 7 classes included the 6 basic emotions and neutral.

3.1 Confusion Matrix

		Predicted Class						
		1	2	3	4	5	6	7
Actual Class	1	150	1	69	91	69	10	101
	2	10	22	2	6	5	1	9
	3	60	1	185	62	89	28	103
	4	50	0	39	601	73	19	97
	5	48	3	72	93	211	9	158
	6	20	0	58	31	26	231	50
	7	44	0	47	108	73	9	345

Table 5: Confusion Matrix

3.2 Evaluation

	Class							
	1	2	3	4	5	6	7	Average
Recall Rate	30.55	40.00	35.04	68.37	35.52	55.53	55.11	45.73
Precision Rate	39.27	81.48	39.19	60.58	38.64	75.24	39.98	53.48
F1 Measure	34.36	53.66	37.00	64.24	37.02	63.90	46.34	48.07
CR per Class*	84.04	98.94	82.45	81.36	79.99	92.73	77.74	85.32
Overall CR**	48.62%							

Table 6: Evaluation Results (%)

*The CR per class is measured by dividing the sum of true positives and true negatives by the total number of examples.
**The Overall CR (for the whole model) is measured by dividing the trace of the confusion matrix by the total number of examples. This is the classification rate provided in the code.

From Table 6, we can observe that the emotion classes 2 and 6 show great performance. Emotion class 2 has a low recall rate but significantly high precision rate, hence this is a perfect example for an instance where a lot of positive examples are missed but examples which are predicted as positive are very likely to be positive. Emotion classes 1,3 and 5 have a low recall rate and a low precision rate which implies that a lot of positive examples are missed and the examples predicted as positive are more likely to be incorrect.

4 Questions

4.1 Question 1: Decision Trees vs Neural Networks

We cannot claim that a single algorithm is better than the other in general because the performance of a model is dependent on the data. The best performing model is the one which most accurately fits the given data. If it is online learning, this requires continuous updates in the model because data arrives continuously. In this instance, neural networks can efficiently update the model by just changing the weights. Whereas, for decision trees, you maybe required to reconstruct the trees. In contrary, decision trees may work better than neural networks in offline learning.

Neural networks can model more arbitrary functions which results in good accuracy, but it is very much prone to overfitting. Decision trees however are much simpler, easy to interpret and can easily avoid overfitting by pruning the trees.

4.2 Question 2: Adding New Emotions

For decision trees classifiers, if new emotions are added to the existing dataset, all the trees must be recreated. Since there are new emotions, the addition of new data may change the information gain of each attribute for the existing emotion classes. Hence the current structure of trees will no longer be valid. Therefore the decision trees algorithm must be run again which will create the new set of trees for each emotion class. Addition of new emotions has no change in the code of the algorithm.

What about just training a new tree?

If new emotions are added to the existing dataset of the neural networks classifier, the network must be readjusted for the additional outputs. This means that the current weights do not hold and hence must be changed. Thus all hyperparameters will need to be adjusted to satisfy the new set of emotions (outputs). Addition of new emotions will result in the change of the whole neural network architecture.

What does that mean exactly?

References

[1] "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", N. Srivastava, 2014. [Online]. Available: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf. [Accessed: 08-Mar-2018].

[2] "What is Regularization in Machine Learning? – codeburst", codeburst, 2018. [Online]. Available: https://codeburst.io/what-is-regularization-in-machine-learning-aed5a1c36590. [Accessed: 07- Mar-2018].

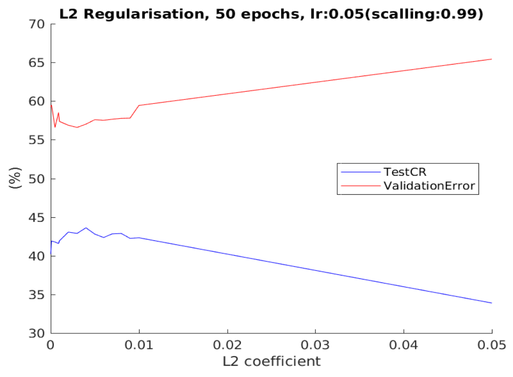
[3] "How to decide the number of hidden layers and nodes in a hidden layer?", ResearchGate, 2018. [Online]. Available: https://www.researchgate.net/post/How_to_decide_the_number_of_hidden_layers_and_nodes_in_a_hidden_layer. [Accessed: 07- Mar- 2018].

[4] "Introduction to Neural Networks with Java, Second Edition", Heaton, 2008.

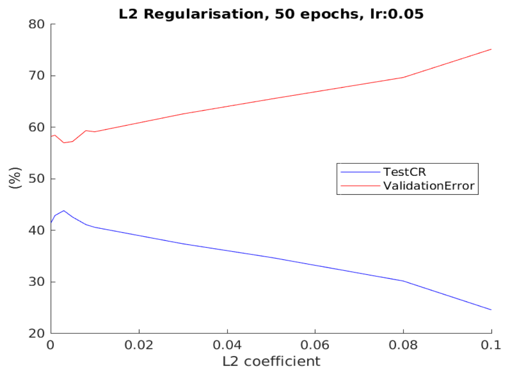
[5] "Neural Networks and Deep Training, Chapter 5: Why are deep neural networks hard to train?", Michael Nielsen, 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap5.html. [Accessed: 07- Mar- 2018].

5 Appendix

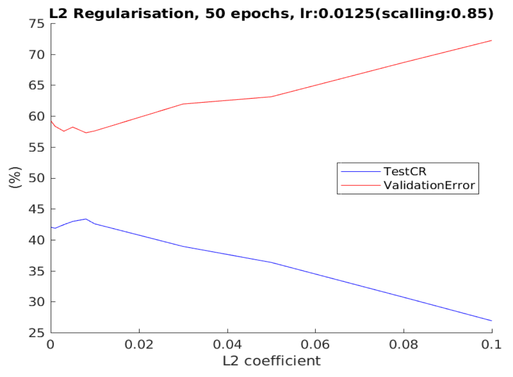
5.1 L2 Regularisation Graphs



(a) Setting 1



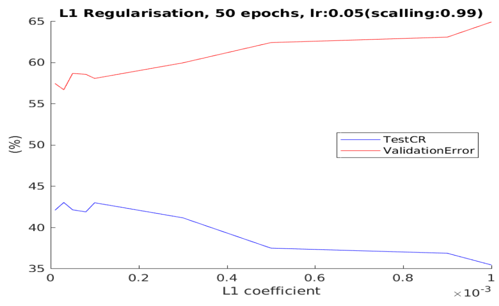
(b) Setting 2



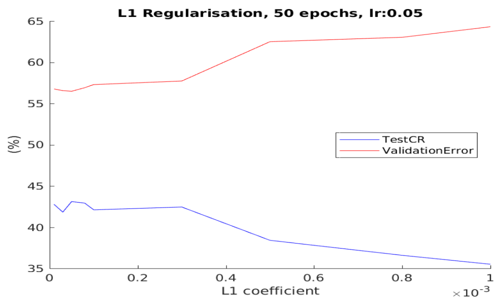
(c) Setting 3

Figure 8: Performance of Validation Error and CR on Test Set for Different L2 Coefficients

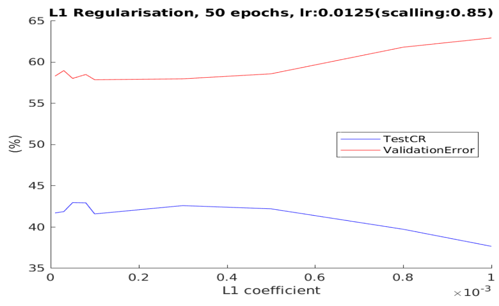
5.2 L1 Regularisation Graphs



(a) Setting 1



(b) Setting 2



(c) Setting 3

Figure 9: Performance of Validation Error and CR on Test Set for Different L1 Coefficients