Neural Networks for Classification of Facial Expressions

George Wright, Weixiong Tay, Nattapat Juthaprachakul and Yicheng Xia
{gw17, wt814, nj2217, yx5017 }@doc.ic.ac.uk

Group 60
Course: CO395, Imperial College London

29th January, 2019

0 Introduction

This report details the implementation of a feed-forward neural network for classification of facial expressions by emotion written in Python 3. The code was first tested for its ability to classify CIFAR-10 images into 10 classes, at which it achieved approximately 50% accuracy. It was then trained with the FER-2013 images and achieved approximately 25% validation accuracy and a classification rate of approximately 25% during testing. The same data was also used to train a convolutional neural network, which achieved a similar classification rate.

1 Question 1

When training with FER-2013 data, the neural network had an input layer with 2304 (48×48) nodes. The output was a softmax layer, with one node for each of the seven classes. In between was a variable number of hidden layers. The hidden layers were linear layers with ReLU activations.

During the forward pass, the linear layer first applies a linear transformation to the vector of input data X, with a vector of weights W and a scalar bias b.

$$y = WX + b$$

The output of the linear transformation is then passed through the ReLU activation function.

$$ReLU(X) = max(0, X)$$

This causes negative outputs from the linear transformation to be set to zero and positive outputs to be left unaltered. The outputs of the ReLU function are then passed on to the next layer.

Weights and biases are updated based on the results of the backward pass. The linear backward function calculates and returns the rate in the change of the cost with respect to the upstream inputs, weights and biases. These are calculated using the chain rule with the derivatives from the upstream layers.

The ReLU function behaves the same in the backward pass as in the forward pass; it passes each gradient of the error with regard to X to the next layer if it is larger than zero and otherwise passes zero.

2 Question 2

A deep fully-connected network with a large number of parameters will often overfit the data. This is when the model learns to classify only the training set of data but fails to generalize to new data (the test set). We apply dropout by randomly turning off sets of neurons(along with their connections) from the network during training. This helps solve the problem of co-dependency and co-adaptations which arise when some neuron units may help fix mistakes made by other units which makes neurons fail to learn robust features but instead to take random noise into consideration. We used Inverted Dropout since we are able to scale training phase by factor $\frac{1}{q}$; thus,test phase is left unchanged.

$$Train\ phase : O_i = \frac{1}{q} X_i a \sum_{k=1}^{d_i} (w_k x_k + b)$$

$$Test\ phase : O_i = a \sum_{k=1}^{d_i} (w_k x_k + b)$$

q = Bernoulli probability of keeping neurons switched on

Forward Propagation: The probability of keeping a neuron switched on is q.

$$f(h) = D^n \odot a(h)$$

where f(h) is the output activation, D is a vector of Bernoulli variables in n-dimensional space and a(h) is activation function before dropout.

Backward Propagation: There are no learned parameters that need to be adjusted in the dropout layer. We propagate back the derivatives through neurons that are not turned off during forward propagation since turned-off neurons do not change the output; their gradients are zero.

3 Question 3

A Softmax classifier can interpret the scores from the outputs as the unnormalized log probabilities. Exponentiating and division implements the normalization hence the probabilities add up to one, which can represent the possibility of each sample being classified to the corresponding class. Cross-entropy functions are utilized together with softmax as the loss function.

$$Softmax\ Classifier : f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

$$cross - entropy\ loss : L_i = -log(\frac{e^{f_{yi}}}{\sum_j e^{f_j}})$$

When coding the Softmax function, $e^{f_{yi}}$ and sum may be very large due to the exponentials, which may lead to overflow and instability. A constant $logC = -max(e^{f_{yi}})$ can be added to the exponent so that we can shift all the values and make the new highest value zero. This operation will not change the value of result an0 increases numerical stability during computation.

In backward pass, the process to calculate the gradient of Softmax is elegant if cross-entropy is used as loss function. Where p_k is the softmax value of the sample for k class. By abstracting 1 from the possibility of correct class and keeping the rest of the values, the gradient of dscore can be calculated.

4 Question 4

4.1 Overfitting with the CIFAR-10 Data

To conduct the first sanity check, we overfit the CIFAR-10 Data with 50 samples: 49 for training and 1 for validation. The architecture of the network of overfitting was to have 2 hidden layers each of size 40. No dropout or regularisation was implemented. The hyperparameters used were as follows: Learning rate 1e-3,

- **Update Rule**: Stochastic Gradient Descent
- **Learning Rate** : 1e^{-3}
- **Learning Rate Decay**: 0.9
- **Batch size** of 20
- 20 **Epochs**

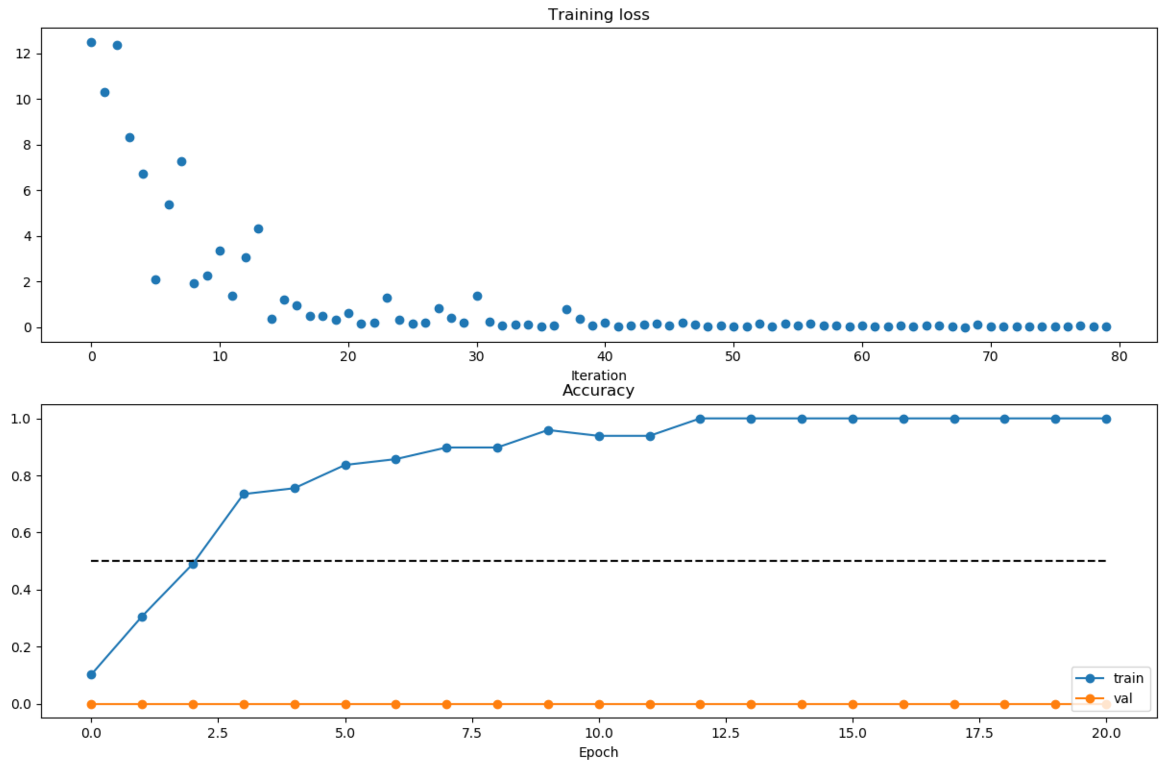

Figure 1: Training loss and accuracy for the overfit sanity check

4.2 Attaining 50% Accuracy on the CIFAR-10 Data

After conducting the overfit test, we then trained a two-layered fully connected network on CIFAR-10. Each layer had 80 neurons and achieved 61% accuracy on the training set and 45% accuracy on the validation set.

The hyperparameters for this particular network were as follows:

- **Update Rule**: Stochastic Gradient Descent
- **Learning Rate** : 1e^{-3}
- **Learning Rate Decay**: 0.9
- **Batch size** of 10
- 30 **Epochs**

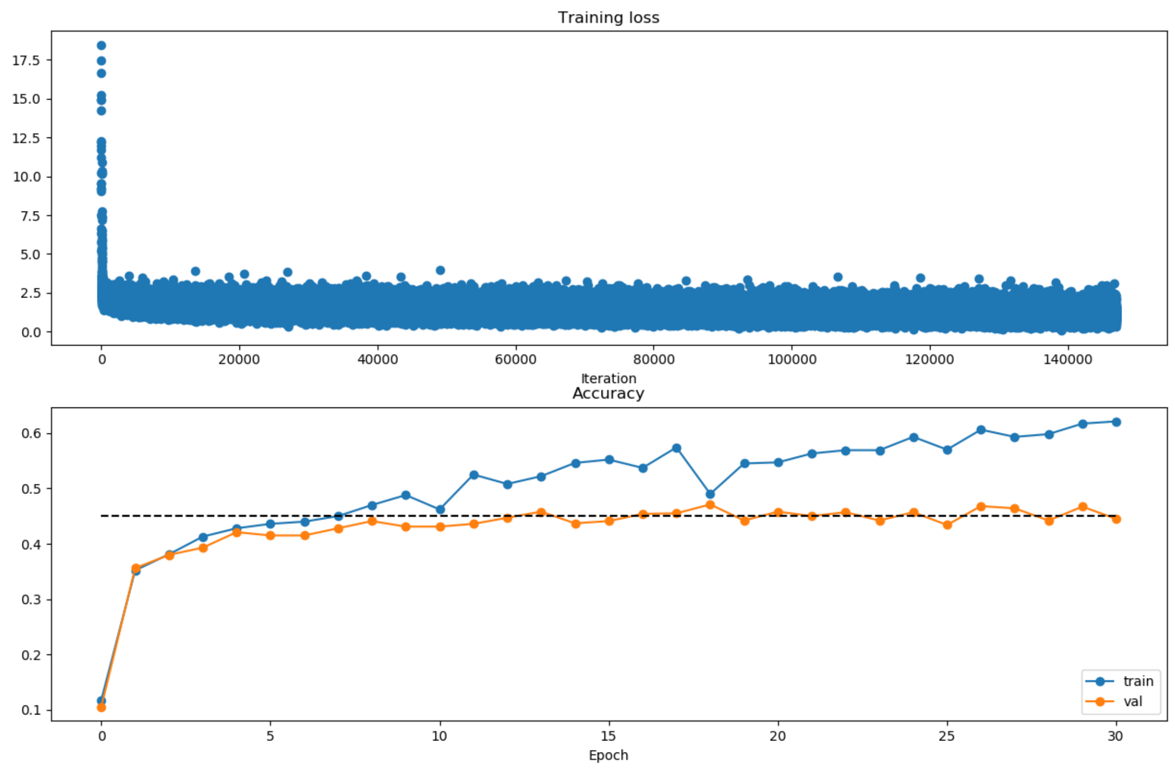

Figure 2: Training and validation accuracy for the CIFAR-10 data

5 Question 5

5.1 Fine-Tuning the Hyper-Parameters

We picked an initial learning rate of 0.01 as this is a common starting point when searching for the optimum.

Several methods were utilized to fine-tune the neural network. The provided solver function can keep track of the validation accuracy after every epoch and save it as a checkpoint. After the training process it will save the set of parameters with the best performance.

We sampled the hyperparameters on log range, since hyperparameters such as learning rate have multiplicative effects on the training process. A fixed interval has no significant effect on accuracy when the value is high.

Furthermore, we start the search from coarse ranges, and narrow the range based on the best results. We trained the neural network for 1 or 2 epochs on coarse ranges to see if the model does not predict the emotion at all, and perform a finer search with 5 to 10 epochs to see if it can be fine-tuned.

Strategies - for hyper-parameter tuning strategy, our group looks into two widely popular strategies: grid search and random search. In grid search, this is the simple form of algorithm for optimizing hyper-parameters by defining sets of parameters and training our data with all possible parameter combinations according to our defined sets. We later choose the best combination based on the cross validation score. On the other hand, for random search, the idea is very similar to grid search but instead of training our data with all possible defined sets of parameters, we train our data with random combination of all possible parameters. We decide to use random search over grid search for the following reasons.

While grid search can give insights of how each parameter affect our model, training with grid search take a lot of resources such as time and computational power as grid search explores a parameter space systematically but it looks only at fixed points. For example, we can train our data with grid search with small sets of parameters very quickly; however, if we need to optimize 6 parameters with 10 possible values, we need to make 1,000,000 valuations. Meanwhile given the same resources available like computational power, random search finds better models by effectively searching a larger combinations of parameters.The reason is that some value of parameter or parameters have less effect on our loss function than others. Random search allows us to skip and explore more values of each parameter given the same amount of trials.

In summary,compared to grid search, random search is more practical than grid search as it can test with a wide range of values and parameters, reaching desired results in less amount of time and less computational budget.

5.2 Performance of the Network with Optimal Parameters

After searching for a configuration that provided the highest validation accuracy, we found the network to perform best with the hyper-parameters displayed in table 1:

Hyper-parameter	Value
Learning rate	0.001
Learning rate decay	0.001
Momentum	0.5
Batch size	10
Number of layers	2
Nodes per layer	1000

Table 1: The optimal hyper-parameters for the neural network

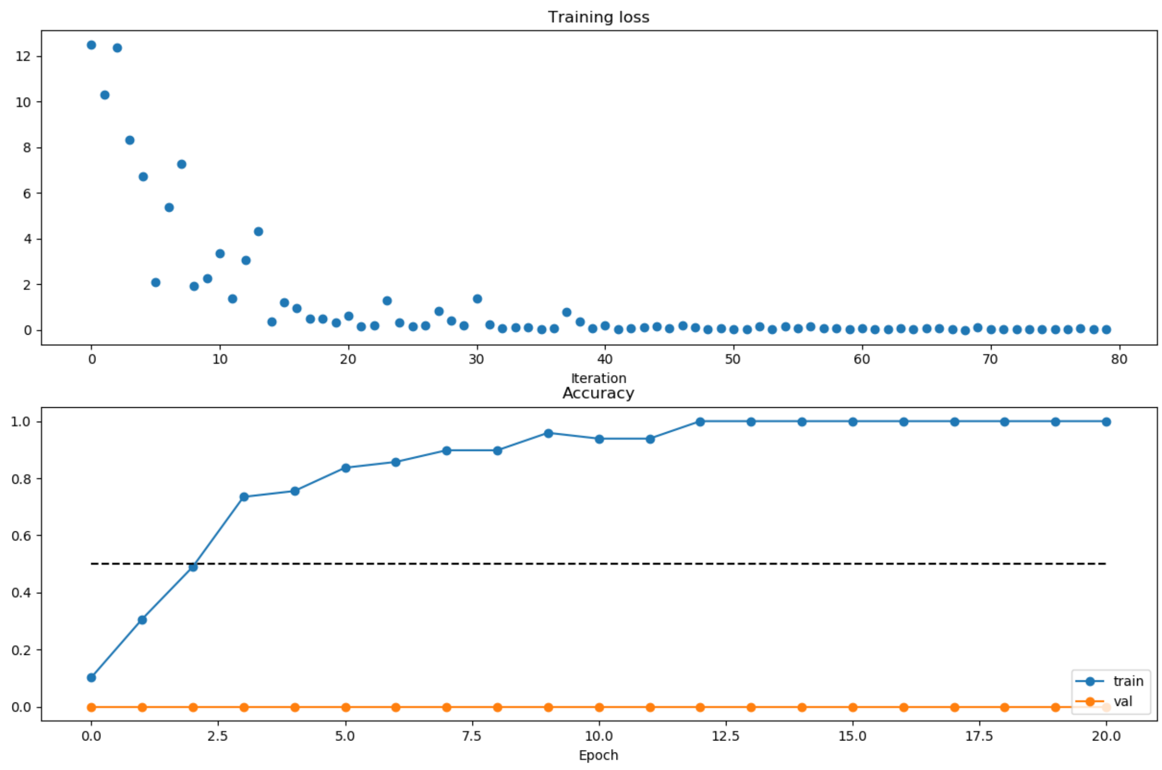

Figure 3: Training loss and accuracy for the overfit sanity check

This configuration was able to achieve both a training and validation accuracy of approximately 25%. The network was then tested with a set of test data from the FER-2013 images. Table 2 shows the confusion matrix constructed according to the neural network's predictions.

		Actual emotion						
		0	1	2	3	4	5	6
Predicted emotion	0	1	0	1	0	1	0	3
	1	0	0	0	0	0	0	0
	2	3	0	1	6	1	1	1
	3	459	56	490	885	648	412	603
	4	1	0	2	1	2	0	0
	5	1	0	2	3	0	1	0
	6	2	0	0	0	0	1	0

Table 2: Confusion matrix for the neural network with optimal hyper-parameters

The neural network achieved a classification rate of 24.8% on the test data. The recall, precision and F1 measure for each class are shown in table 3.

The performance of the network is disappointing as it has clearly learned to classify nearly all instances as belonging to class 3. The network therefore has the highest recall, precision and F1 when classifying class 3. This is in part due to the distribution of the training data which contains many more instances of class 3 than other classes. Since class 3 represents the emotion of happiness, one could conclude that the network has learned a very optimistic world view!

Class Number	Recall	Precision	F1 Measure
0	0.002	0.167	0.004
1	0.000	nan	nan
2	0.002	0.077	0.004
3	0.989	0.249	0.398
4	0.003	0.333	0.006
5	0.002	0.143	0.005
6	0.000	0.000	0.000

Table 3: Recall, precision and F1 for each class

6 Training a Convolutional Neural Network

We trained a convolutional neural network using Keras as an interface to the Tensorflow library. It achieved a classification rate of 24.4% which is slightly lower than the performance of the feed-forward network. The basic architecture was using 5 convolutional layers, 3 average pooling layers and 3 fully connected layers at the end. The overview of the network was as follows:

- **First Layer**: convolutional layer (36 filters of 5x5) and an average pooling layer
- **Second Layer** : 2 convolutional (48 filters of 3x3) layers followed by an average pooling layer
- **Third Layer**: 2 convolutional (48 filters of 3x3) layers followed by an average pooling layer
- **Fully-Connected Layer**: 3 fully-connected layers with dropout layers in between

Zero-padding was also used in between each conv-conv-pooling layer and dropout (0.5) was implemented as a form of regularization in between the fully connected layers. Zero-padding of size 1 was implemented to try and preserve the volume in between the 5 convolutional layers. Dropout was implemented to prevent overfitting. Without the dropout, the network overfits after 20 epochs. Pooling layers were also added to prevent overfitting as the network also overfits without the layers. Average pooling was used instead of max pooling because the network performed better in preventing overfitting with average pooling.

The optimizer used was a stochastic gradient descent with a learning rate of 0.001, decay of 1e^{-5} and momentum of 0.9. Each layer was activated using ReLU. A batch size of 32 was used along with 50 epochs to train the network.

Overall, the network failed to get out of a local minima and mostly predicted the data as a single class.

6.1 Performance of the Network

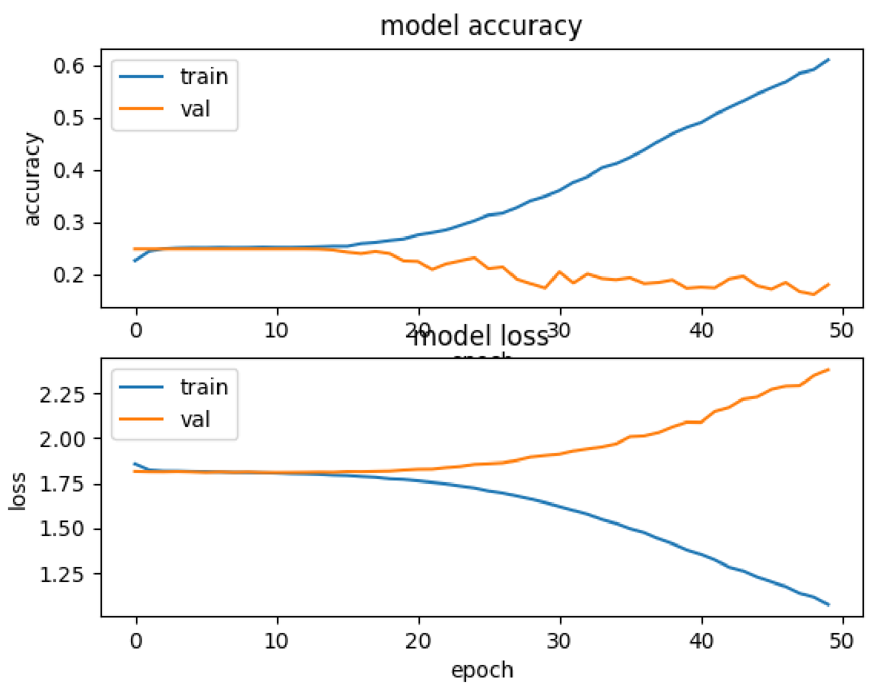

Figure 4: Overfit sanity check

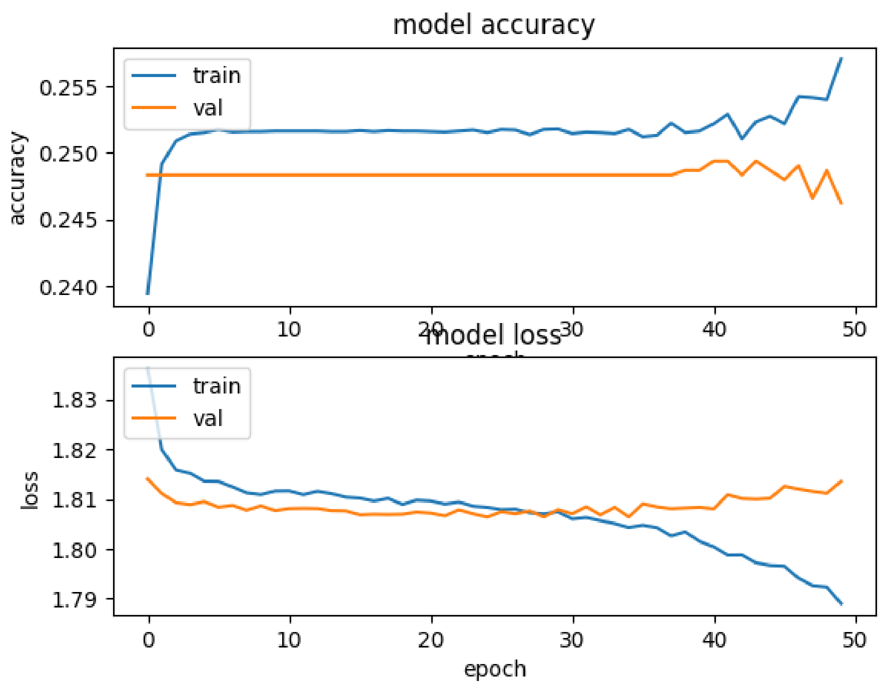

Figure 5: Training and validation loss and accuracy

6.2 Performance of the Network

The recall, precision and F1 measure for each class given by the convolution neural network are shown in table 5.

		Actual emotion						
		0	1	2	3	4	5	6
Predicted emotion	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0
	2	0	0	1	4	1	0	4
	3	493	52	478	849	631	389	588
	4	9	2	3	6	9	5	9
	5	0	0	0	0	0	0	0
	6	19	2	14	36	12	21	17

Table 4: Confusion matrix for the convolutional neural network with optimal hyper-parameters

Class Number	Recall	Precision	F1 Measure
0	0.000	nan	nan
1	0.000	nan	nan
2	0.002	0.100	0.004
3	0.949	0.244	0.388
4	0.014	0.209	0.026
5	0.000	nan	nan
6	0.028	0.140	0.046

Table 5: Recall, precision and F1 for each class

6.3 Comparison of the CNN and Feed-forward Network

The performance of the convolutional neural network was no better than the performance of the two-layer feed forward network. The classification rates and precision, recall and F1 measure for each class on the two networks were similar.

One possible improvement to make the models better is to pre-process the images before training the network such as rotation, shearing or applying filters. This could be used to reduce the noise in the images and allow the networks to train better.

7 Additional Questions

7.1 A1

Both neural network and decision tree are widely used because both models can give us ability to model data based on non-linear relationship between variables. However,choosing suitable models depend not only on accuracy but also on other things like model interpretability. Though in many cases neural network is chosen because of its accuracy, it has several drawbacks. For example, deep neural network suffers a "black box" problem which is when models fail to give explanation how results are obtained or how best parameters are reached while decision tree is more easier to be interpreted and explained.

Another drawback of neural networks is that they require a relatively large amount of time to train. It is therefore not necessarily appropriate to use a neural network if the performance is not much better than that of a decision tree. Because of the time required to train a neural network, neural networks are also less portable. If the computer being used to run the algorithms has poor specifications, for example no GPU, the neural network becomes less tractable.

7.2 A2

When using decision tree learning, if a new emotion is added to the data set, a new decision tree must be constructed which classifies the emotion. The decision trees for the other emotions do not necessarily need be altered as they simply classify each instance as either true (if it belongs to the emotion) or false (if it does not).

It will only be necessary to re-train the decision trees for the other emotions if when adding the new emotion, some of the items in the data set are reclassified as belonging to the new emotion. If the items in the data set from before the new emotion was added remain unchanged, their decision trees need not be re-trained.

When adding a new emotion to the data set for a neural network, a new output node is required for the new classification. This will have to be connected to the nodes in the last hidden layer and the weights can be initialized randomly (or through another method of weight initialization). The weights between the existing nodes should be kept the same as they will already be close to the optimum configuration, assuming the neural network has already been trained to reach a high accuracy.

Backpropagation should then be used to retrain the network. All of the weights in the network will need to be updated as weights between nodes in the hidden layers will need to be altered in order to correctly classify the new emotion. Updates to these weights mean that the weights connecting the output layer will also need to be altered. Retraining should not take as long as the initial bout of training as the previous weights will already have been close to the optimum.

The distribution of the new data set should be of the same type as the distribution of the data set before the new emotion was added.

8 Conclusion

Both the fully-connected network and the convolutional neural network failed to exit the local minima when training the on the FER2013 sets. Possible improvements would be to normalize the dataset before training to have an equal dataset as there was a disproportionate amount of class 3 examples in the data. Another way could be to pre-process the data to reduce noise. The hyperparameters of the 2 networks were also less than optimal and needs to be tuned in order to give a better performance. Some examples would be to use a different optimiser such as Adam or adaptive subgradient methods. The activation function could also be tuned by using a leaky ReLU instead. Finally, a possible issue might also be that the 2 networks were not given enough time to train and thus increasing the epochs to higher numbers i.e. 1000 might be enough to allow the networks to escape from a local minima.

References

[1] Y. Bengio and X. Glorot. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, Proc. of *The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS10).* pages 249256, 2010.

[2] Bengio, Y. (2000). Gradient-Based Optimization of Hyperparameters. *Neural Computation.* , 12(8), pp.1889-1900.

[3] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Z. Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML07).* , pages 473480. ACM, 2007.

[4] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, London, UK, 1995.

[5] CS231n: Convolutional Neural Networks for Visual Recognition, `http://cs231n.stanford.edu/`