CW1: Backpropagation and SoftMax

06-32212 Neural Computation (Extended)

Group - 25

2218037 - VENKATA LAHARI BALANTRAPU

As per the instructions in the task 6, various experiments conducted on the different set of hyper-parameters such as activation function, training time, learning rate, network topology, mini-batch size and their influence on the classification accuracy of the model will be discussed in this report.

Data Normalization:

The image data set has varying values of different RGB ranges, as different scales of the data might slower the convergence, we shall first normalise the data. This model seemed to give better results after normalising the data. This has been achieved by dividing the training input values by 255.

Activation Function:

The first hyper-parameter that I chose to work with is the activation function, The activation function plays a major role in determining the output and the accuracy of the learning model as, it performs non-linear transformation on the input making it capable of learning complex tasks.

There are three activation functions that were used on this model, they are:

1. Sigmoid
2. Tanh or Hyperbolic
3. ReLu (Rectified Linear Unit)

Sigmoid Function: $f(x) = 1/(1+\exp(-x))$

$$f'(x) = f(x) * (1-f(x))$$

the values of this function range in between 0 and 1, the output of the sigmoid is not zero centred , when I was working with the sigmoid there is no learning when the value of x is relatively small or big the slope is zero which resulted in just a few output classes missing and a bad accuracy with greater loss. The sigmoid function works perfectly fine with binary classification but not an optimal choice for data set with multiple classes.

Tanh or Hyperbolic: $f(x) = \tanh(x) = \exp(x) - \exp(-x) / \exp(x) + \exp(-x)$

$$f'(x) = 1-f(x)^2$$

tanh function is quite similar to the sigmoid but in this function the values are mapped between -1 and 1, the output is zero centred as the values go -1< output < 1 which makes the learning for the next layer easily. The only drawback of tanh is it suffers from the vanishing gradient problem. While working with the tanh it was observed that the activation function works well with data set as it allows the negative values too unlike the sigmoid and ReLu.

ReLu: $f(x) = \max(0, x)$

the values range from 0 to infinite, as the function gives an output x is the value is positive and 0 if the value is negative. ReLu is computationally faster and easier because of its less complicated mathematics. ReLu rectifies the vanishing gradient problem unlike sigmoid and tanh. The only drawback of ReLu is that it could result in dead neurons to overcome this a simple modification called Leaky ReLu has been introduced. Also, the problem that has been observed while working with ReLu was it had the similar kind of behaviour on the data set as it deactivates neurons that has negative values resulting in no learning for certain classes. ReLu is one of the most widely used activation function in the deep learning and neural networks models.

Applying all the three activation functions on the model, it was observed that the tanh to be giving quite the better performance for the data set alongside the other hyper-parameters. Therefore, the tanh activation function has been used in this model.

Network Topology: The number of hidden layers and the neurons per each layer do contribute a lot to the classification accuracy. The deeper the layers the more complex information they extract from the data. For this we shall start adding one layer after the other and check if there are any drastic changes in the accuracy by keeping all the other hyper-parameters constant. By default, we have 3 hidden layers in the network and 20 neurons in each layer. After conducting several combinations of these layers and neurons, there wasn't much difference in the accuracy due to which, adding one hidden layer i.e., 4 hidden layers with 20 neurons each deemed to be an optimal network for the model.

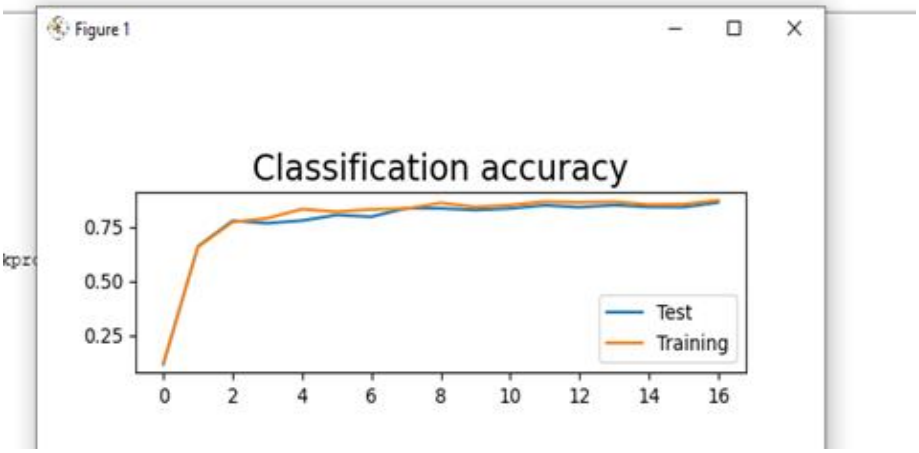


Figure 1: 4 hidden layers and 20 neurons per layer with default hyper-parameters.

Learning Rate and Batch Size: Generally, the higher the batch size the larger learning rate it requires. From the observations, when the learning rate is too low the progress was slow whereas if the learning rate is set too high the loss function showed an undesirable behaviour. Also, the higher values of mini batch size required a lot of memory space. Therefore, finding the perfect balance of these two would improve the performance of the system drastically and was the most important task.

The model was tested against pairs of learning rate and batch sizes and the accuracy of such pairs have been noted to find the optimal values. Where, the learning rate lies in the range [0.01,0.16] increasing the value times 2 and the batch sizes in the range [1,60] increasing the value by multiplying

with 2 at every step and looping through the learning rates for every batch size we study the accuracy to find the best combination of the learning rate and batch size.

After analysing the accuracy and the performance of the data under these conditions a batch size of 36 with a learning rate of 0.16 is used in the final model.

Training Time: For this hyper-parameter the first set of experiments that were done was to take a large value of epochs and perform early stopping. Early stopping is to end the training based on certain metrics such as when the data starts over-fitting, or when the loss value increases else when the accuracy decreases. Although, this seemed to be a good solution practically it seemed a bit complex, as the loss values kept fluctuating as shown in the Figure 2. Therefore, to decide only based on that metric would not be ideal also if we were to consider both loss test accuracy as well as the training accuracy, there will always be a compromise.

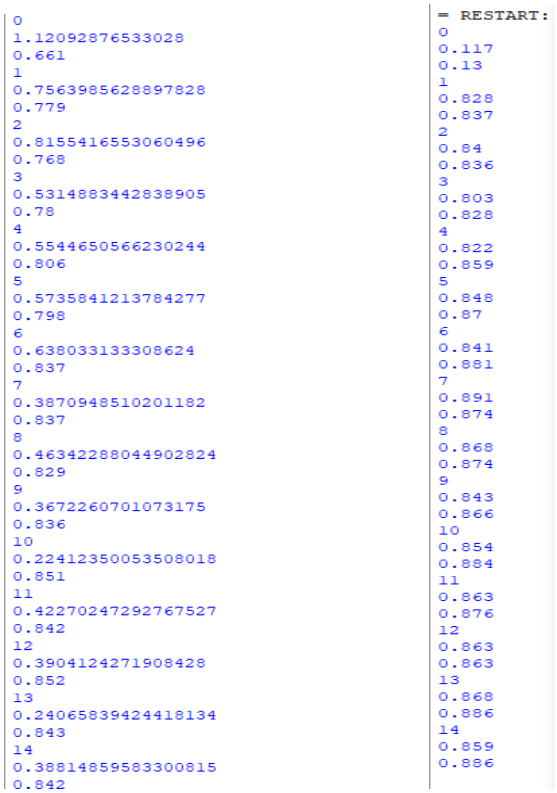


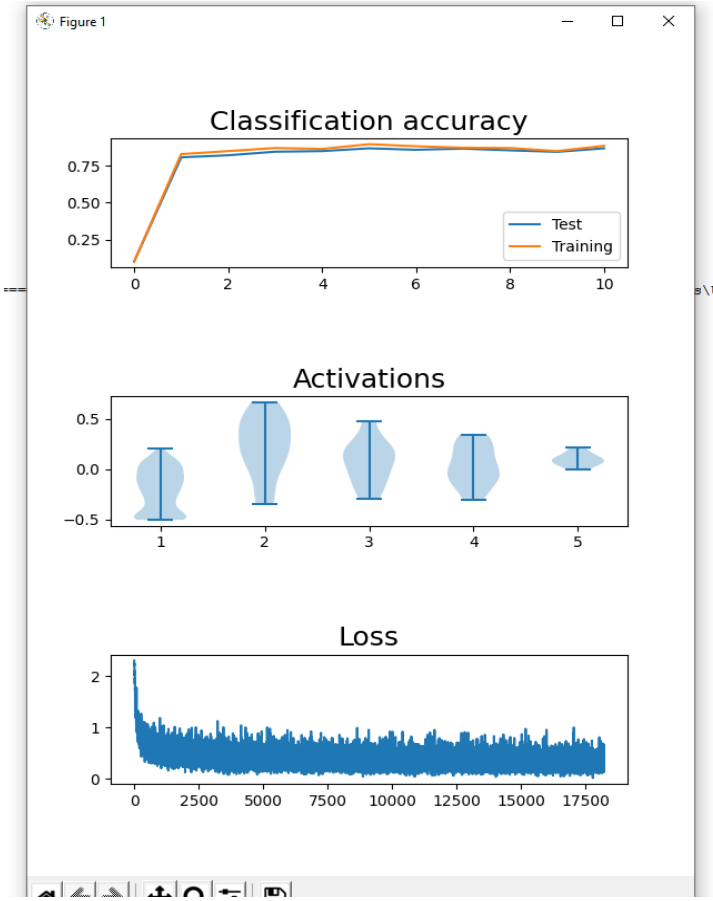
Figure 2: Loss, test and training accuracy fluctuations.

The way that this problem was approached was to manually use the epochs values and check the behaviour of the model for the obtained learning rate and the batch size. At this learning rate the model generally started overfitting after a certain value of epoch usually between 10-12 epochs after reaching the maximum accuracy of 0.87 as shown in the figures 3. Hence, we just stop the training after the accuracy reach a value of 0.87 or train for 30 epochs until we get the maximum accuracy. This approach avoids the overfitting problem and reduces the computational time. The average accuracy of the final model with the given hyper-parameters was about 0.858.

```
2
0.84
0.835
3
0.846
0.878
4
0.866
0.864
5
0.848
0.875
6
0.857
0.873
7
0.854
0.878
8
0.866
0.889
9
0.845
0.887
10
0.853
0.869
11
0.87
0.9
12
0.843
0.885
13
0.859
0.857
14
0.837
0.904
>>>
```

```
0.83
0.849
3
0.86
0.879
4
0.855
0.842
5
0.854
0.856
6
0.852
0.862
7
0.854
0.868
8
0.868
0.891
9
0.843
0.891
10
0.844
0.879
11
0.87
0.848
12
0.889
0.899
13
0.838
0.884
14
0.84
0.893
```

Figure 3: Observations of test and training accuracy.




```
==========
0
0.101
0.102
1
0.808
0.83
2
0.822
0.85
3
0.846
0.871
4
0.85
0.865
5
0.869
0.897
6
0.859
0.883
7
0.867
0.873
8
0.855
0.871
9
0.845
0.85
10
0.869
0.886
11
0.876
>>>
```

Figure 4: The classification accuracy of the final model