INDEX

Acknowledgment

I have completed my assignment by learning from the following sources

1 Given Support Materials

a) https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc
b) https://builtin.com/data-science/gradient-descent
c) https://medium.com/@devalshah1619/activation-functions-in-neural-networks-58115cda9c96
d) https://en.wikipedia.org/wiki/Artificial_neural_network
e) http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/
f) https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

2) https://www.freecodecamp.org/news/how-to-build-your-first-neural-network-to-predict-house-prices-with-keras-f8db83049159/
3) https://machinelearningmedium.com/2017/10/03/neural-networks-cost-function-and-back-propagation/
4) https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8
5) https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications
6) https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/
7) https://www.geeksforgeeks.org/difference-between-batch-gradient-descent-and-stochastic-gradient-descent/

8) https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4
9) http://neuralnetworksanddeeplearning.com/chap3.html
10) https://www.pluralsight.com/guides/machine-learning-neural-networks-scikit-learn
11) https://towardsdatascience.com/build-neural-network-from-scratch-part-2-673ec7cdd89f
12) https://warwick.ac.uk/fac/sci/mathsys/news/readinggroups/machinelearningrg/presentation.pdf
13) https://github.com/ebi84/MLP
14) http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/
15) https://github.com/manoharmukku/multilayer-perceptron-in-c
16) Other Books and study materials
17) My previous submitted assignments

NOTE –

I have not copied anything from these sites, such as programs,

Only have taken the idea of the general syntax and working of the programme

I have taken some code same as my previous assignments.

Pre-Requisite

How To Make Executable and Run Programme

1) Unzip File at location x
2) Open Terminal and change directory to that location x
3) Now enter command make.
   This will create Executables
4) Now type
   export LD_LIBRARY_PATH=${pwd} //give the path of the current working directory.
5) Then type executable name with the arguments
6) For the plot of cross-entropy, type make plot1
7) For the plot of MSE, type make plot2
8) For different propagation comparisons, type make plot3
9) Type make run for classification data set with tanh
10) Type make run1 for regression data set with tanh
11) Type make run2 for vanishing gradient with logistic
12) Type make run3 for solving vanishing gradient through relu
13) Type make run8 for seeing the accuracy of the normalized data set
14) Type make run9 for seeing the accuracy of the standardized data set
15) Type make run4 for classification in python
16) Type make run5 for Regression in python
17) Type make run6 for own data set
18) Type make run7 for own dataset in python

Structure Of Programme

Program is made up of directory structure which includes

1) src – Source Folder, It contains all necessary .c files to make object files
   a) main.c files – contains extracting of data and passing arguments to the neural library for classification data set
   b) main2.c - contains extracting of data and passing arguments to the neural library for own data set
   c) main3.c - contains extracting of data and passing arguments to the neural library for regression data set
   d) main4.c – contains gradient vanishing problem
   e) main5.c – contains classification for normalized data, can run relu on layer 4, show how to solve vanishing problem.
   f) Main6.c – contains classification for normalized data
   g) Main7.c – contains the classification for standardized data
   h) Mainclass.py- contains python code for classification
   i) Mainown.py- contains python code for own data set
   j) Mainregress.py- contains python code for Regression
2) obj – contains object file created from above code
3) inc – contains header file useful at the time of linking
4) lib – contains a neural library
5) data – contains running data from which the plot can be made
6) venv – contains virtual environment file for python

MLP Neural Network

As, Our brain helps to identify and solve the problem with the help of several neurons, which multiply previous neuron results with some weight and add some bias to it and add the result to its learning part.

Similarly, neural networks can be made in the computer, which can be composed of several layers, each layer containing several neurons with some weight and bias and function associated with it. When a set of input arrives at that neuron, their respective weight is applied, and bias is added to the final result, followed by a function. Then the result is passed to another node.

Formula can be summarized as –

$$a(l,i) = f\left[b(l,i) + \sum_{j=0}^{n(l-1)} W(l,i,j)*aj)\right]$$

$$z(l,i) = \left[b(l,i) + \sum_{j=0}^{n(l-1)} W(l,i,j)*aj)\right]$$

Different activation functions can be maintained for hidden and output layers. This formula is called forward propagation and helps in predicting the result. We can calculate loss or error by using cross-entropy or MSE.

Now if error is weight and bias has to be adjusted and it is done using back propagation algo.

The basic back propagation algo is

$$\delta(l) = (W(l)*\delta(l+1))*f'(z(l))$$

$$\nabla W(l) = \delta(l+1)(a(l))$$

$$\nabla W(l) = \delta(l+1)(a(l))$$

$$\Delta W(l) = \Delta W(l) + \nabla W(l)$$

$$W(l) = W(l) - \propto \left[\left(\frac{1}{m} * \Delta W(l)\right) + \gamma W(l)\right]$$

Types of Back Propagation

Basic – In this, for all data set for one iteration, the $\nabla W(l)$ is accumulated, then at the end of the iteration, it is updated.

Stochastic – In this, for each data set, the $\nabla W(l)$ is updated.

Batch – In this, for specific batch data set in 1 iteration, the $\nabla W(l)$ is accumulated, then at the end of an iteration or Batch, it is updated.

Types of Activation Function

RELU – In this for all result which is positive are kept it is and rest negative result turned to 0

Tanh – In this result is converted in range -1 to 1.

Logistic – In this result is converted in the range of 0 to 1, used in case of predicting probabilities

None/Identity – In this result is kept as it is

Softmax – In this, all result is seen, and their probabilities are returned by seeing their logistic(exponential) type of result.

Data Set /Library Specification

Data Set

1) Classification Data set – Data.csv is a classification data set that contains 32 columns, 1st column is for identity, 2nd column is output with B as of class 1 (Value 0) and M as of class 0 (Value 1). Rest 30 Columns are input sets. It predicts cancer.

2) Own Classification Data set – Data2.csv is a classification data set which contains four columns, 1st column is for identity, 2nd column is output with 0 as of class 1 (Value 0), and one as of class 0 (Value 1). Rest 2 Columns are input sets. It forms a square of size 4*4.divided into 4 parts of 2*2. With rightmost part lower part as value 0 and upper part as value 1 and leftmost part lower part as value 1 and upper part as value 0

3) Regression Data set – Data3.txt is a classification data set that contains 14 columns. The last column predicts house value. Rest Columns are input sets.

4) Normalized Data set – Data4.csv contains normalized data set of data of classification.

5) Standardized Data set – Data5.csv contains standardized data set of data of classification.

Library Specification

My code is a sort of mix of soft and hardcoded, just like python neural network specification, which needs data to be filled manually and only pass the parameter to the library as asked in the assignment. When to change data set, it is to be changed by going in main, by

changing its column, total size, training size, input nodes, output nodes, and name and path of File.

When changing attributes of the neural network, it can be done in 3 ways

1st way – Passing only layers, out function, hidden function, cost function, backpropagation.

2nd way - Passing only layers, out function, hidden function, cost function, backpropagation, learning rate, iteration, regularization, followed by Batch if needed

3rd way - Passing only layers, out function, hidden function, cost function, backpropagation, learning rate, iteration, regularization, followed by Batch (necessary), followed by no of neurons in the hidden layer (supported up to 3 layers)

Note, the layer should not be one, and no of neurons in the hidden layer should not be 1. Also, the layers include the output layer also.

In main, if any modification is needed, we can uncomment some of the line parameters and change the parameters.

Note in classification by default output function is 5, the hidden function is tanh, backpropagation is 1, iteration is 200, the learning rate is 0.001, regularization is 0.0001, layers=7 with all 30 neurons, default batch size is 10

Note the type structure numbers

Basic=1 , Stochastic=2, Batch =3

Sigmoid=1, Tanh=2, Relu=3, None=4, Softmax=5

Cross-Entropy=1 , MSE=2

Gradient Vanishing Problem

This problem generally occurs when the effect of weights update of higher layers is not able to update weights of lower layer correctly due to delta tending to 0. Several reasons like the high no of layers and use of limited range activation function for the hidden layer are responsible for this. Several methods like changing activation, which have more range like relu, has been proposed

This problem leads to neural networks not learning anything and mostly giving biased results. This can be clearly seen by low changing or constant output loss function

An example of gradient vanishing can be seen by running make run2 in our library – it is for seven layers with activation function as logistic, the learning rate is 0.001

Gradient Vanishing can be seen most of the time in 3rd layer for logistics activation function also

Here 1st column, as stated above, is serial number, 2nd and 3rd column is for class probability, and 4th column is predicted output

We can clearly see in the picture that probabilities stay the same, and the predicted value is biased to 0 for all answers.

Although accuracy is 77% but, it is by chance as the test contains most of the values having 0.

This problem can be solved by using tanh to some extent and by relu if data is normalized, as given below.

It can also be solved by batch normalization if logistics is to be used.

Main5.c has normalized data set so that somewhat relu can run on it, thus giving a solution to solve this.

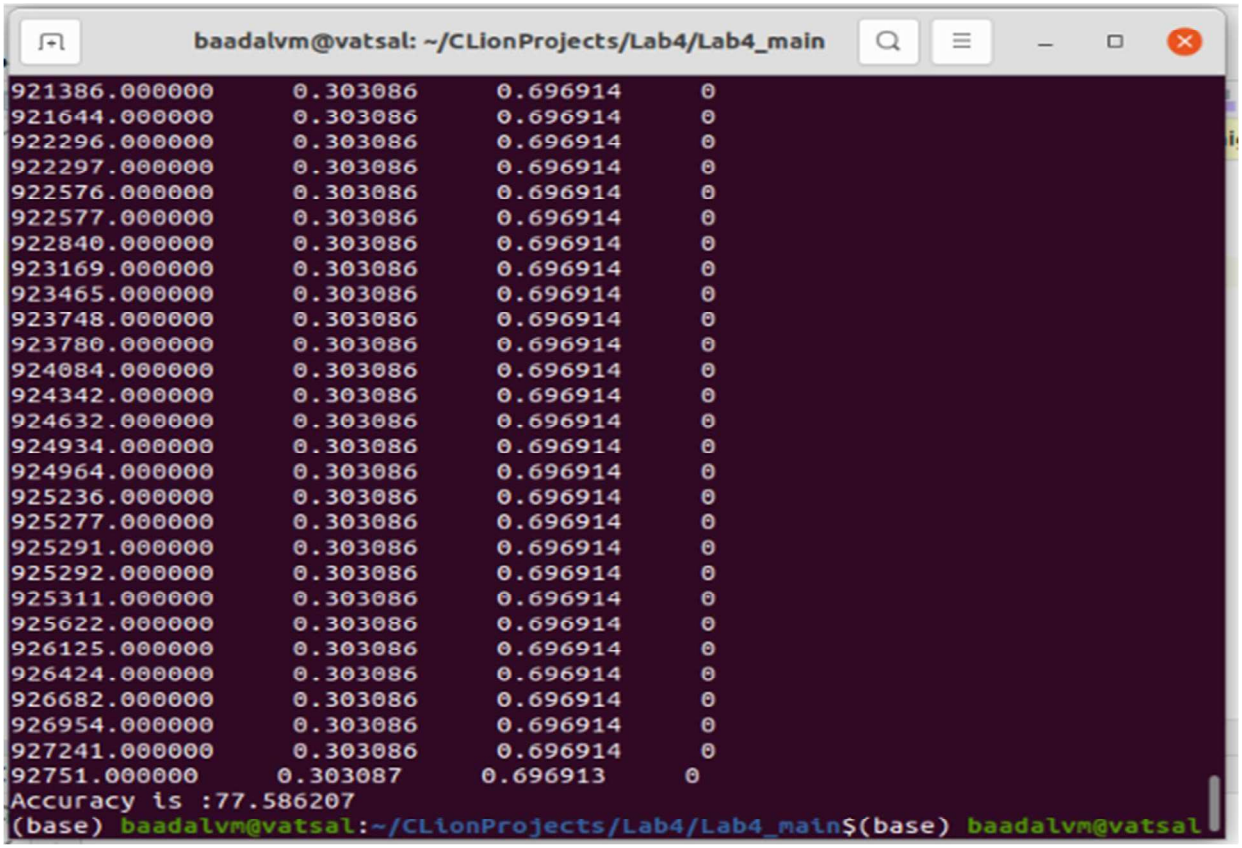

Fig1. Gradient Vanishing Problem using logistic

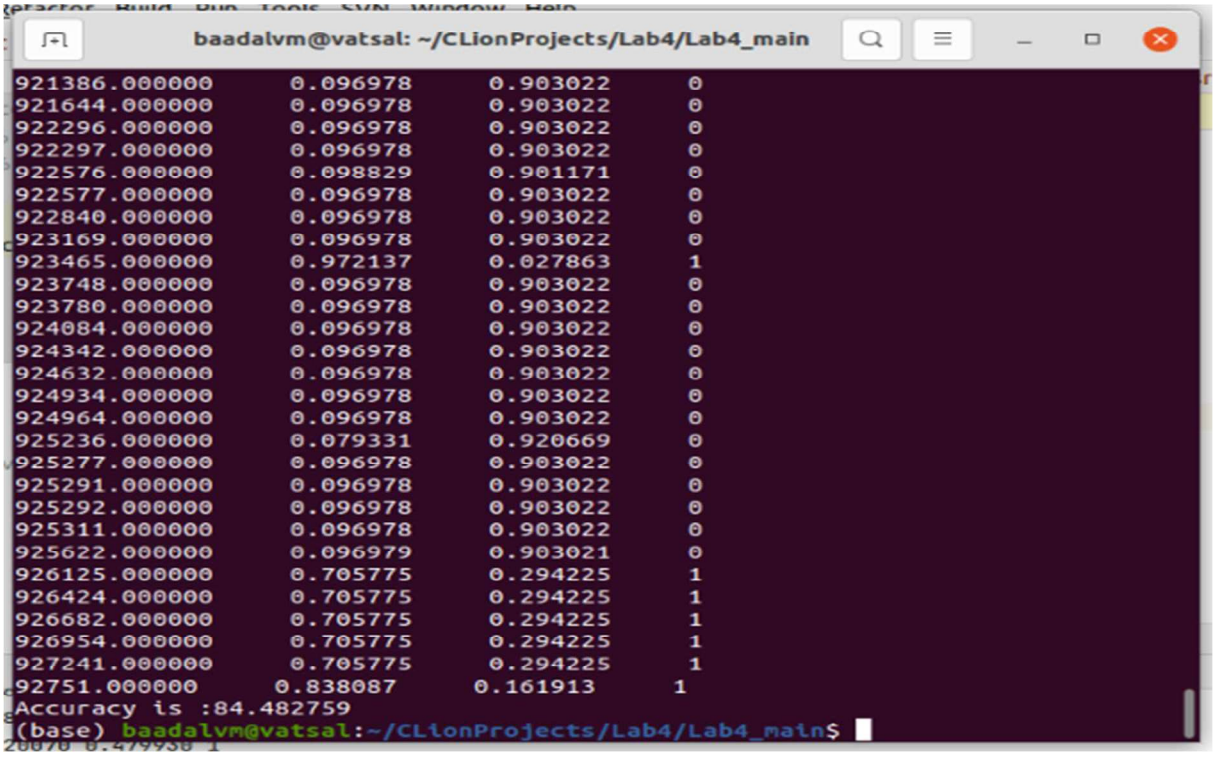

Fig2. Gradient Vanishing Problem (solved) using Tanh

Result

1) I have made an MLP Neural Network Library that performs all significant functions like any other MLP Library
2) The library works similar to as sklearn library in neural network
3) I have implemented it with random weight and bias initialization with srand. So each time we run, we may get a different answer.
4) Gradient vanishing Problem can be seen in high no of layers
5) Relu may not work correctly because of value overflow, so a possible way is to limit the range of relu by dividing its answer by some fixed quantity. This changes its derivative part also.
6) Suppose we divide relu +ve part with x; then its derivatives will also be divided by x because the derivative of f(y) = y/x is 1/x
7) Softmax is used as an output activation function in my classifier
8) The output function can be changed also
9) The identity function is used as output activation function in Regression
10) MSE is checked by the regression data model, and Cross entropy is checked by the Classification model
11) I have implemented three types of gradient backpropagation methods – basic, stochastic, and Batch. Best performance is seen in basic
12) Batch also gives optimal performance if the Batch is chosen wisely.
13) As mentioned above, I have implemented two ways of error checking (loss function) -MSE and cross-entropy
14) Logistic activation function is good as output activation function, but it sometimes results in gradient vanishing for less no of layers also.
15) Tanh is found to be best in all cases.
16) Performance on many runs of python scikit learn, and my c neural network is found to be the same for any data set.
17) For small input and less complex data set, the accuracy level reaches 100%.

18) Nan occurs in relu because of the number exceeding the range of double.

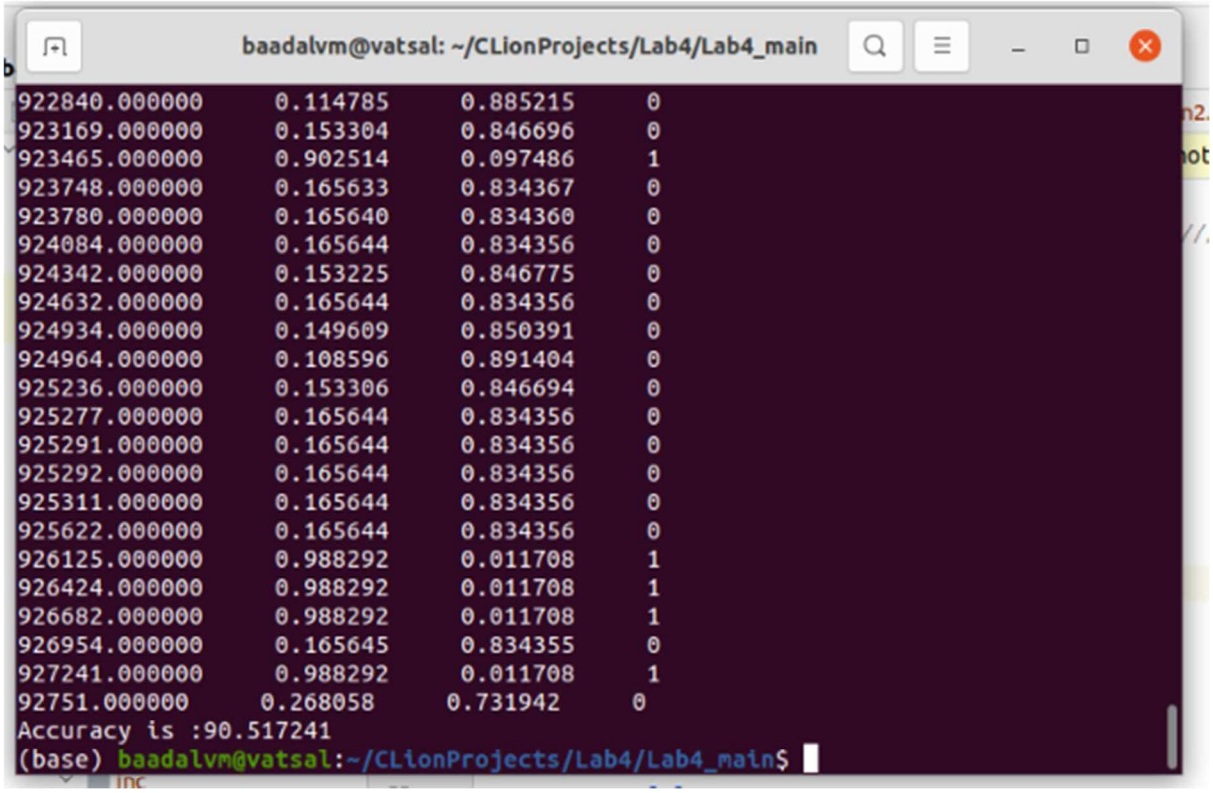

Fig 3 . Tanh Implementation for Classification

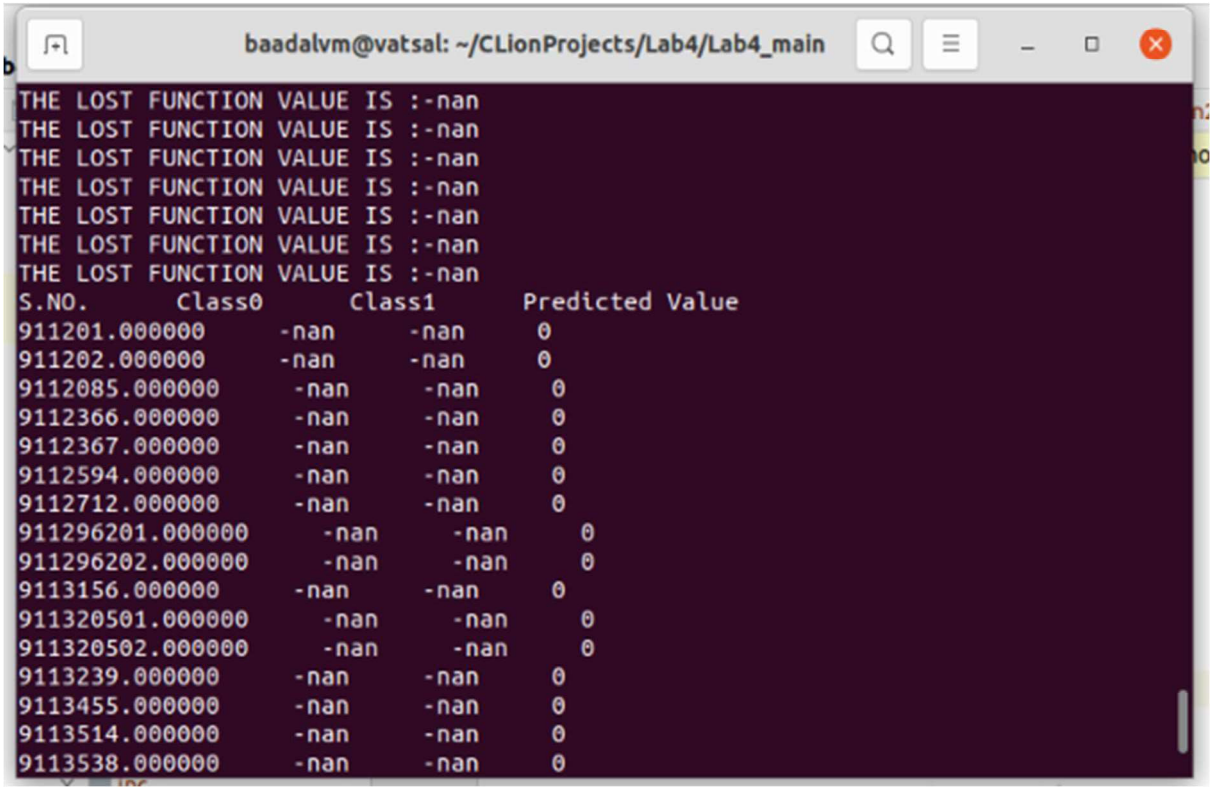

Fig 4 . Relu Implementation for Classification

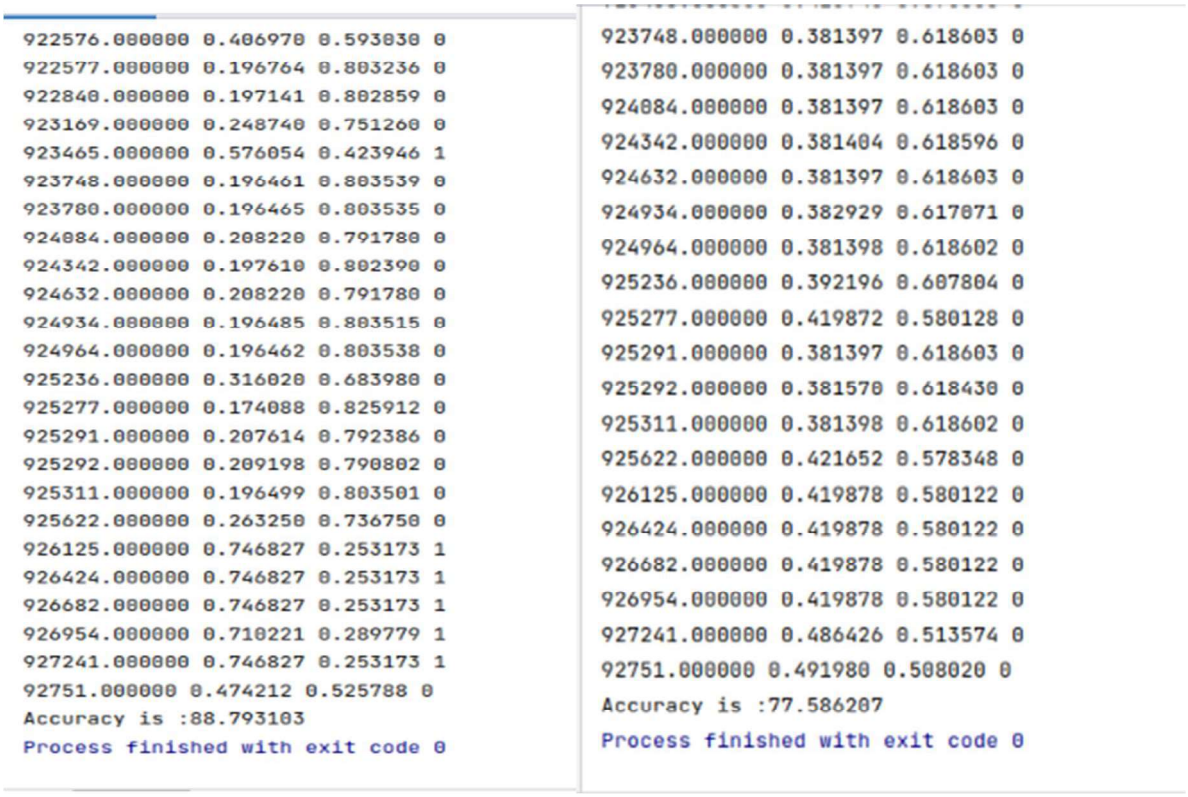

Fig 5 . Sigmoid Implementation for Classification with 100 neurons and 30 neurons in 3 layers each

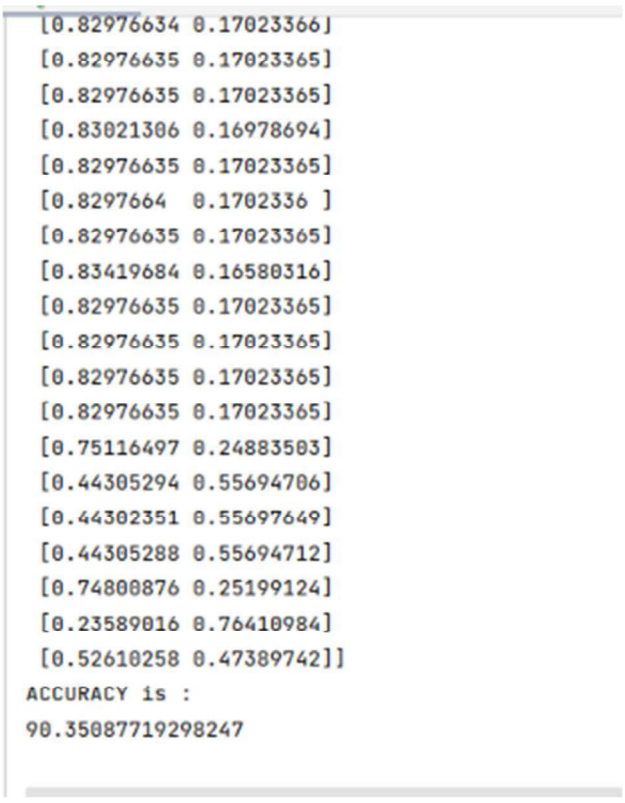

Fig 6. Tanh Implementation for Classification in python

```
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]
 [0.60010075 0.39989925]]
ACCURACY is :
77.19298245614034
```

Fig 7 . Relu Implementation for Classification in python

```
[0.61642546 0.38357454]
[0.61642362 0.38357638]
[0.61642044 0.38357956]
[0.61648116 0.38351884]
[0.61642044 0.38357956]
[0.61644351 0.38355649]
[0.61643645 0.38356355]
[0.61651463 0.38348537]
[0.61642036 0.38357964]
[0.61642223 0.38357777]
[0.61642035 0.38357965]
[0.61642639 0.38357361]
[0.61637593 0.38362407]
[0.61570189 0.38429811]
[0.61570187 0.38429813]
[0.61570188 0.38429812]
[0.61581765 0.38418235]
[0.61570186 0.38429814]
[0.6165427  0.3834573 ]]
ACCURACY is :
77.19298245614034
```


```
[0.61642546 0.38357454]
[0.61642362 0.38357638]
[0.61642044 0.38357956]
[0.61648116 0.38351884]
[0.61642044 0.38357956]
[0.61644351 0.38355649]
[0.61643645 0.38356355]
[0.61651463 0.38348537]
[0.61642036 0.38357964]
[0.61642223 0.38357777]
[0.61642035 0.38357965]
[0.61642639 0.38357361]
[0.61637593 0.38362407]
[0.61570189 0.38429811]
[0.61570187 0.38429813]
[0.61570188 0.38429812]
[0.61581765 0.38418235]
[0.61570186 0.38429814]
[0.6165427  0.3834573 ]]
ACCURACY is :
77.19298245614034
```


Fig 8. Sigmoid Implementation for Classification with 100 neurons and 30 neurons in 3 layers in python

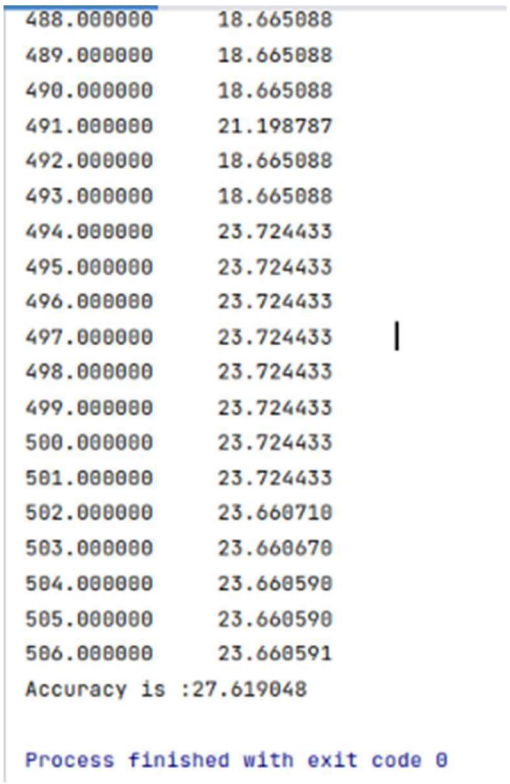

Fig 9 . Tanh Implementation for Regression

```
Predicted:
[27.73873308 27.73873309 27.73873308 27.73873308 27.73873308 27.73873301
 20.40407355 27.73873308 27.73873312 27.73873314 27.73873308 27.73873308
 20.40407355 27.73873308 20.40407355 20.40407355 20.40407355 27.73873309
 27.73873308 27.73873309 27.73873308 27.73873308 27.73873308 27.73873312
 27.73875595 27.7387331  27.73873308 11.5265993  27.73873311 11.52796634
 27.73873308 27.73873308 20.40407355 27.7387331  27.73873308 20.40407355
 27.73873308 27.73873308 27.73873308 11.52661532 20.40407355 27.73873308
 20.40407355 27.73873312 20.40407355 27.73873308 27.73873308 27.73873308
 27.73873308 27.73873308 20.40407355 15.33287136 27.73873314 27.73873308
 27.73873308 27.73873309 27.73873308 27.73873308 27.73873308 27.7387331
 27.73873308 27.73873308 27.73873308 27.73873308 27.73873308 20.40407355
 27.73873296 20.40407355 27.73873309 27.73873308 27.73873308 20.40407447
 27.73873314 15.33353656 27.73873308 27.73873311 27.73873311 27.73851309
 27.73873308 27.73873313 27.73873309 27.73873308 11.5265994  11.52660356
 27.73873308 27.73873308 27.7387331  11.52658766 27.73873308 20.40407355
 27.73873312 20.40407355 27.73873308 11.52659949 20.40407355 20.40407355
 27.73873308 27.73873308 27.73873308 27.73873309 27.73873308]
ACCURACY is :
16.860860535475474
```

Fig 10. Tanh Implementation for Regression in python

Conclusion

1. Gradient Vanishing Problem can be seen clearly with seven layers
2. With 3 Layer Gradient vanishing Problem occurs sometimes depending on random initialization with logistics
3. Relu most of the time give NAN because of range overflow
4. The best-hidden layer function is Tanh
5. Increasing no of neurons to 100 gives the best answer but increases time drastically
6. The optimal no of neurons is found out to be 30, which takes less time and also performs best
7. The best no of hidden layers is found to be 3 with tanh
8. Other activations need 2 or 1 layer for their consistent result
9. Modified relu is giving the same result as logistics.
10. For the classification data set, accuracy ranges from 70 to 95% for tanh with a 0.001 learning rate with three layers of 100 neurons each.
11. Accuracy with sigmoid ranges from 20% to 75% for three layers
12. Tanh also gives the same accuracy
13. Batch gives the best result, but batch size finding is difficult optimum half of training size, gives accuracy 90%.
14. Stochastic gives result in less iteration, but the result cannot be of high accuracy sometimes, but most of the time it ranges to 75%, can be biased sometimes. It can contain noise.
15. Basic gives the highest accuracy ranging to 90%
16. Basic gives high accuracy but again needs time
17. Normalized data set gives an accuracy of 98% on avg.
18. For classification, I have used Cross entropy
19. For MSE in classification, the accuracy severely decreases to 60% for tanh
20. For regression data set again, the best model is found to be tanh

21. Accuracy range from 8% to 40% for tanh with three layers, 100 neurons each.
22. Accuracy dips to 0% to 20% for other activation function
23. Regression gives an accuracy of 70% with a +-2 error rate and a max 40% with a +-1 error rate
24. For other relatively less complex data set accuracy to turn out to be ranging from 87% to 100%
25. Increasing no of layers to four increases accuracy for tanh
26. Increasing no of layer till two increase accuracy for logistic and relu
27. Increasing no of neurons beyond 150 has negative effect only
28. More no of neurons, more time will be needed for calculation
29. Python calculation time is relatively short as compared to ours
30. Scikit also calculate relu for some cases (not all) before crossing maximum bound
31. For tanh and logistics, our and python library performance is similar, considering avg performance for large run
32. In one run, only any of the python and my implementation can perform better
33. Loss function value for MSE and cross-entropy for different functions in python and my implementation is given below
34. For normalized data set, both python and c implementation works till layer 4 for relu
35. Normalized and normal data set has the almost same accuracy
36. Standardized data set have somewhat less performance than normalized
37. I have not implemented any normalization or standardization function. Instead, I have used weka for this
38. For Regression, relu is not working for later 4 in python
39. For classification, relu is working for layer 4 in python
40. Most of the time gradient vanishing problem is seen in layer 4 for logistic

41. For normalized data set, this can be seen clearly that running relu for layer 4 removes the gradient vanishing problem
42. Loss function graphs are given below
43. Note in python plot straight line after some point is due to no significant change in the loss. Thus copying the last value for all iterations
44. Relu for c is modified in both loss graphs because the original relu is giving nan due to the large number
45. Relu modified is when derivative and original value is divided by 100000
46. Python gives nan for relu of MSE. Thus no plot for it

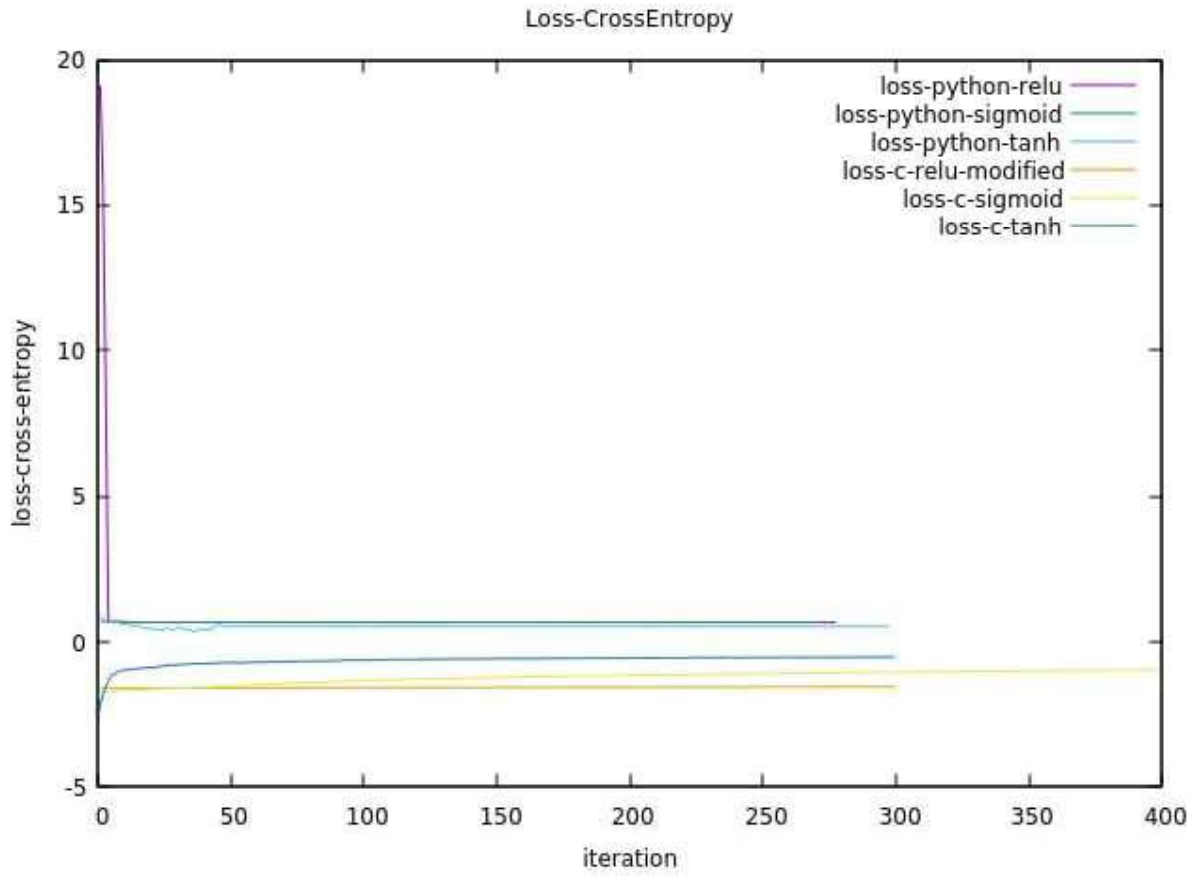

Fig 11. Loss for Cross-Entropy

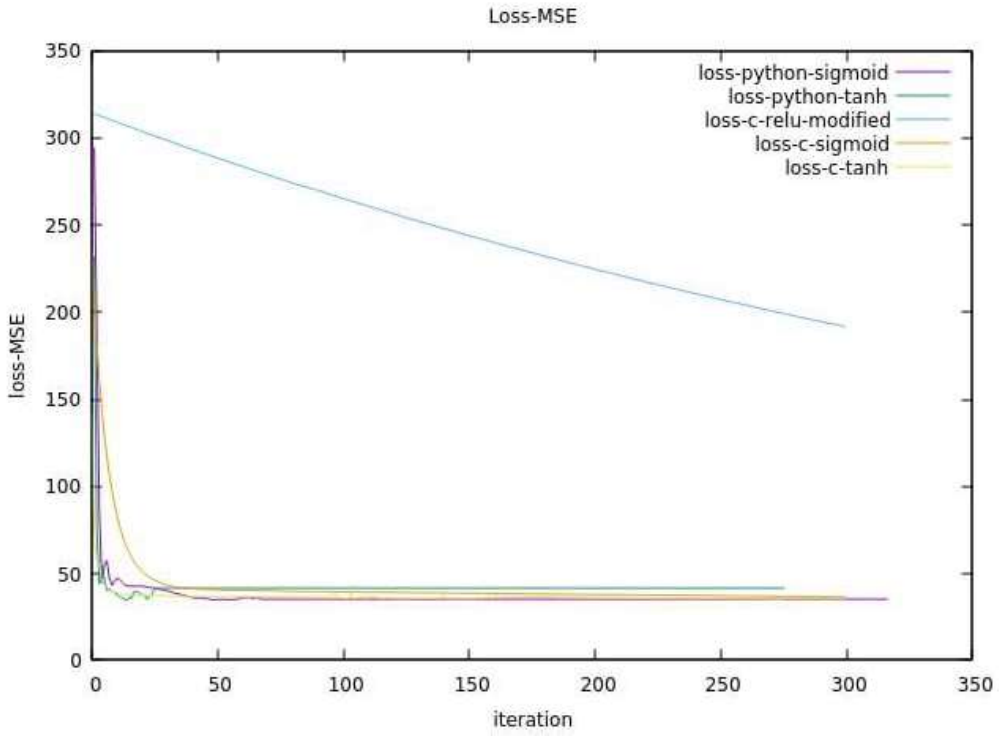

Fig 12. Loss for MSE

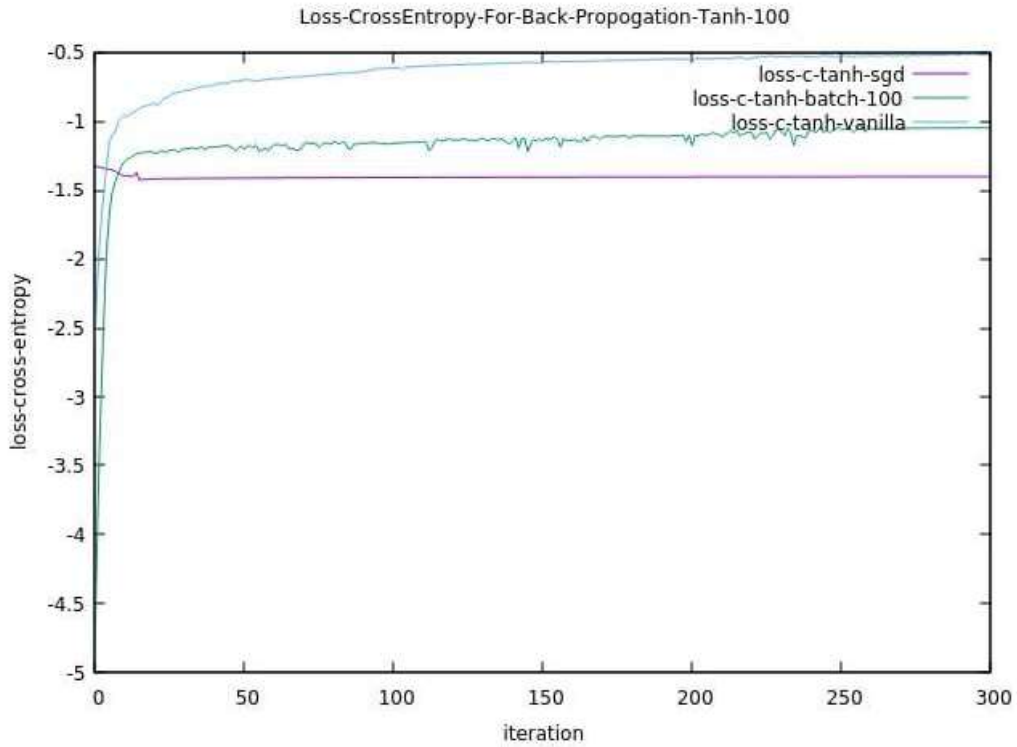

Fig 13. Loss for different backpropagation with batch size 100 and 100 neurons in 3 hidden layers with tanh

Future Scope

1) More combination of the hidden layer can be tried with different neurons and different functions
2) The optimum batch size can be improved
3) Better ways to remove noise can be applied
4) Preprocessing can also be applied
5) Reports can be matched with high accuracy tools like weka
6) Relu like function can be modified, and new methods can also be applied