TIPR SECOND ASSIGNMENT
TOPIC: MLNN

Submitted by Akshaya A Mukundan
M.Tech DESE, 2018-20
SR No.15521

Part One:
The task here is simple yet subtle. You need to build a Optical Numeral Recognition (ONR) system using neural network. You have to design a Multi- Layer Neural Network (MLNN) from scratch - everything starting from initialization, forward propagation and backward propagation(commonly coined as back-prop) - everything by yourself. NO LIBRARY IS ALLOWED.

Step-1 (5 Points) : Initialize parameters.
Step-2 (15 Points) : Implement forward propagation.
Step-3 (15 Points) : Code out Backward propagation.
Step-4 (5 Points) : Design the cost function.

You need to write four different function for the above four steps and then connect accordingly. Write your code in such a way that the architecture of neural network can be plugged in from outside. Your code should be able to take the number of layers, activation function and number of neurons in each
layer as external input (format mentioned later). Remember that the external layer
of your neural-net should be a softmax layer. You should use Acccuracy and F1-score as evaluation metrics for you model.

<u>Result:</u>
Implemented ML-NN from scratch.

<u>Step 1:</u>
A layer class was defined and each layer required in the neural network was created as an object falling in this class with its attributes as number of neurons in that layer(mentioned as numneuron in code), vj (W.T * X + b), activation function(mentioned as layeractivfunc), layer output (mentioned as layerout, which is activation function(vj)), local gradient (mentioned as localgrad).

All the layers needed in a neural network of certain configuration is combined in the class named as mynn, which has the attributes num of layers in the neural network (input layer + hidden layers + output layer), number of neurons in each layer and the learning rate used for the neural network. the learning rate can also be used separately for each layers also.

Initialization done for various parameters such as learning rate (mentioned as eta in the code), weights connecting various layers(mentioned as weighttrans, it's the transpose of the weight matrix) and bias inputs (mentioned as bias) to the various layers.
To select the best learning rate, eta was varied and it was found to give good results for eta =0.1 , 0.08, 0.01, etc. Most of the tasks are performed with eta = 0.1.
For most of the tasks, initially bias is set as a ndarray with all elements as 0.25. It was chosen after varying bias values in the range of -2 to + 2; 0.25 was found to give the best result. Although these bias gets updated in subsequent epochs, the value of bias chosen gave accuracy with less number of epochs.
For most of the tasks, weights are generated randomly from a normal distribution of standard deviation 1/sqrt(next_layer_number of _neurons).
Reference: Neural network, A comprehensive foundation, Simon Haykin

<u>Step 2:</u>
Forward propagation When an input is provided above functionality was peformed to find vj.
vj = Weightinbetweenlayers.T * prev_layer_out +bias_layerinbetweenlayers
Then, activation function was performed on each vj obtained in each layer depending on the configuration of activation function provided to give the next_layer_output.
next_layer_output = activation_function(vj_obtained_for_that_layer).
Here, layer_0 is taken as the input value.
The activation function of hidden layers are configurable, and for the input layer an activation function called buffer is defined which passes the value as such. Output layer activation function used is softmax for all the tasks.

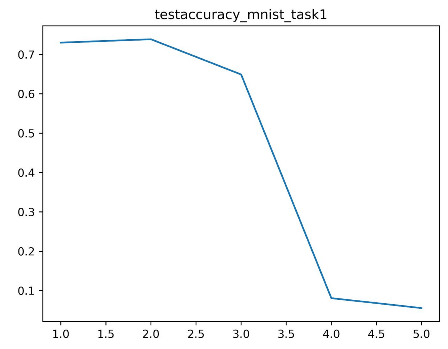
Activation functions used are sigmoid, tanh, ReLu, swish and softmax. These are described in method under class mynn as activfunc in the code. their respective derivatives are defined as a method named activfuncderiv.

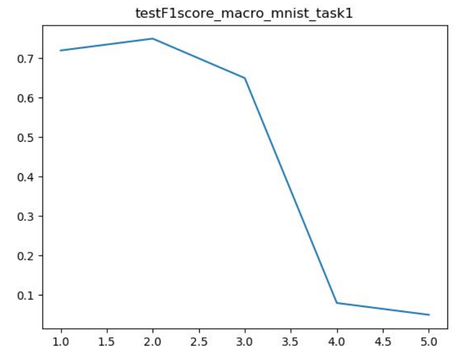
<u>Step 3:</u>
Back propagation After the layer out of output layer is obtained, the error in the estimated value of output layer is to be found. mean squared error is used to find the cost.
Cost_MSE =
 0.5 * sum_over_j((desired_response_of_jth_neuron - obtained_response_of_jth_neuron)2)
delta weights and delta biases where found corresponding to each weight and bias in the neural network using local gradients.
Finally, weights and biases were updated.

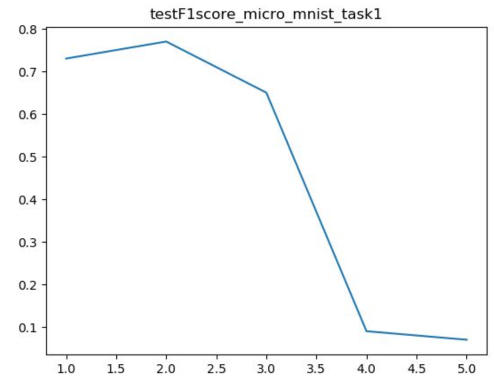
<u>Step 4:</u>
Cost function was found using mean square error.
Cost_MSE =
 0.5 * sum_over_j((desired_response_of_jth_neuron - obtained_response_of_jth_neuron)2)

4 different functions were defined for above 4 steps inside the class mynn, so that everytime a neural network is created all these 4 methods in th nn can be accessed with ease.
Architecture of the neural network including number of hidden layers, number of neurons in each layer and activation function for each layer is provided as input functions from keyboard.
External layer of the neural network uses softmax layer.

Task - I (2.5 Points)
Experiment with different number of layers of your neural network and give a plot of accuracy vs. layer count, macro-F1 vs. layer count and micro-F1 vs. layer count.
Plots obtained are attached below:



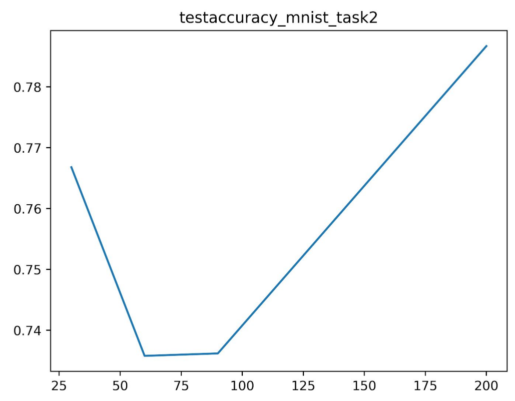


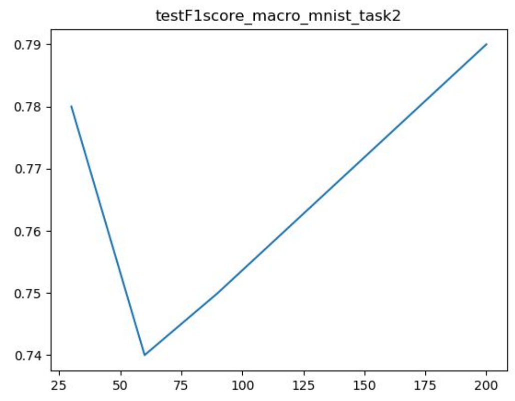


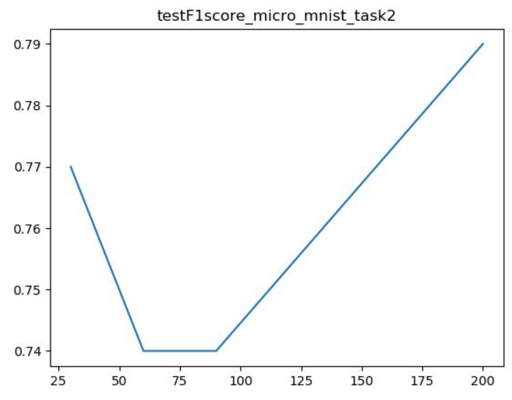
Layer range is varied from 1 to 5. Number of neurons in each layer is 30. Activation function used is sigmoid. Number of neurons and activation function used are same so that they do not interfere the performance variation due to the variation provided in number of layers
Accuracy is found to be highest for number of layers = 2, it provides equally high performance for layer number = 1.

Task - II (2.5 Points)

Try out various configurations of number of neurons in each layer and find the best one. Provide a suitable plot of the same.



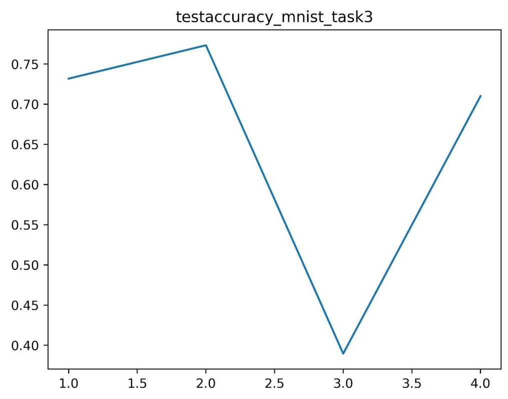


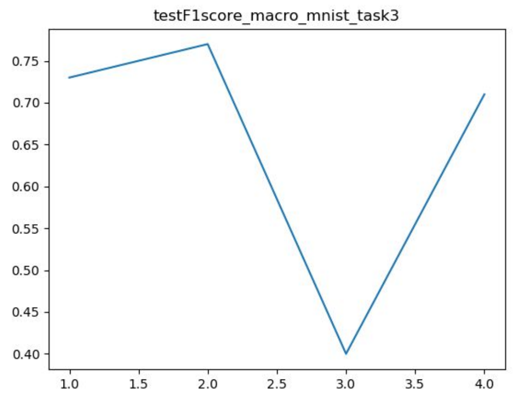


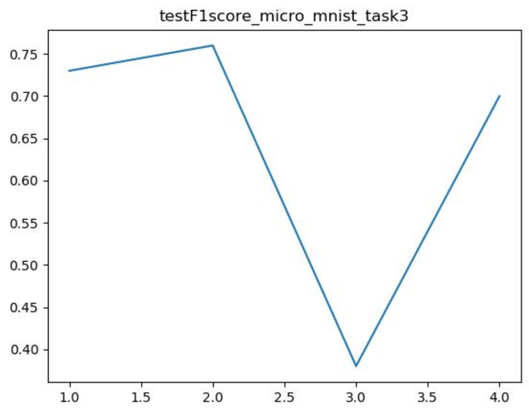
Layer number is chosen as 1 as it provided high performance in task 1. All activation function are sigmoid. MNIST dataset is found to provide high accuracy for all range of neurons chosen. All in range of 70%

Task - III (2.5 Points)

Fix the number of layers and neurons in each layer and feed the hidden layer with the following activation functions - **sigmoid(1), tanh(2), relu(3) and swish(4)**. Evaluate the performance of your model. Do it for different configurations of your neural net. Provide a suitable plot of the results obtained.





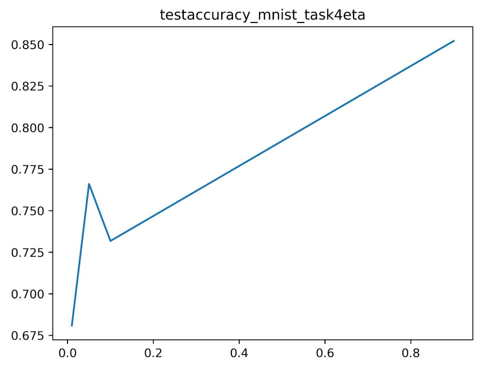


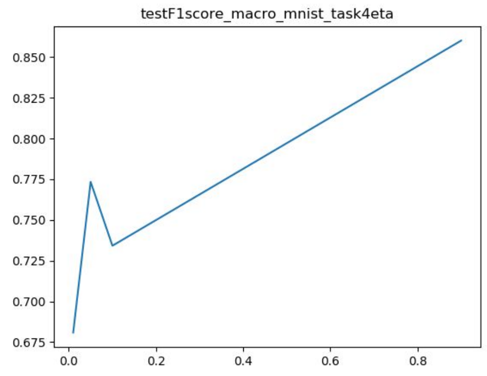
High performance is found for sigmoid, tanh and swish. Although relu is found to show much low performance.

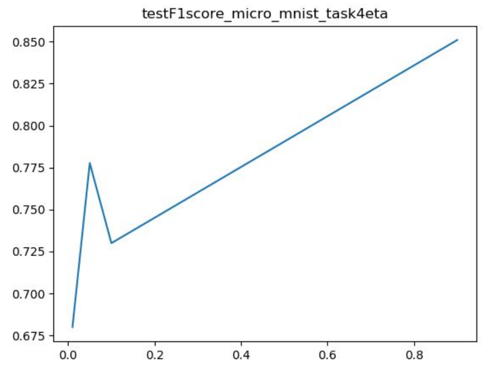
Task - IV (2.5 Points)

Try out various initialization techniques and observe the results.

Variation of eta:

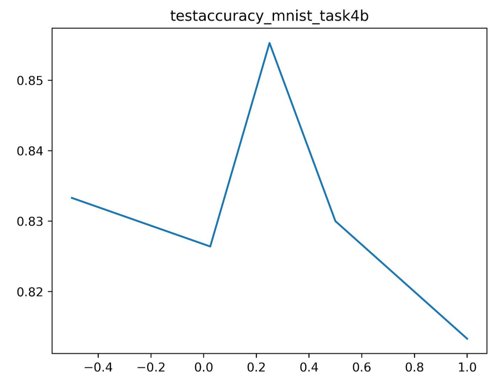


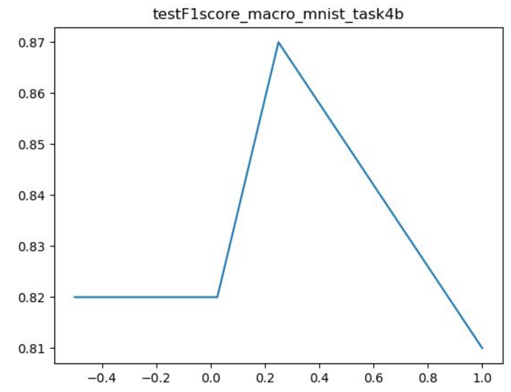


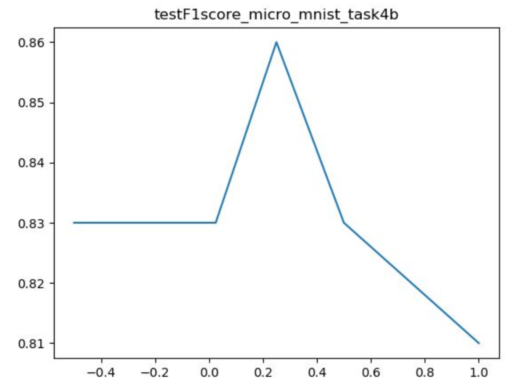


Performance in terms of accuracy and F1 Score is found to be high for a learning rate of 0.9; For other learning rates the accuracy and F1 Score are almost of comparable value with that of the performance with 0.9. This is because, learning rate is just a parameter that scales the value with which weights and biases have to be updated. Unless it makes the local gradient to fall in a undesired local minima, it would provide almost same accuracy.

Variation of bias:







Initial value of bias gets updated in epochs to reach to a final decision boundary that separates the objects of different classes. Hence, bias variation may not provide much variation in accuracy. Although, the MLP using BPA is online mode, hence if there are multiple decision boundaries, then the accuracy and the number of epochs needed may vary. This is shown in the above plots. Performances are almost in the range of 80% for all values of biases tried.(b = -0.5 to +1)

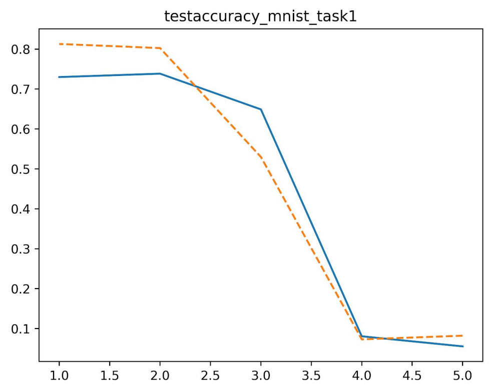
Task - V (10 Points)

Check with keras library and its MLP implementation and compare the results.

Task 1 comparison: (layer count)

-- line : MLP implementation using keras library

blue line: MLP algorithm implemented

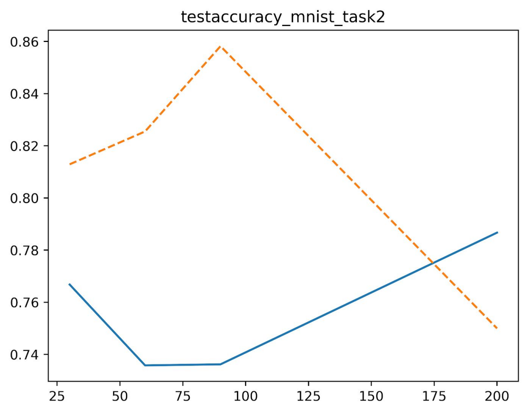


Inference: Performance of both are comparable with addition of layers.

Task 2 comparison: (neuron count)

-- line : MLP implementation using keras library

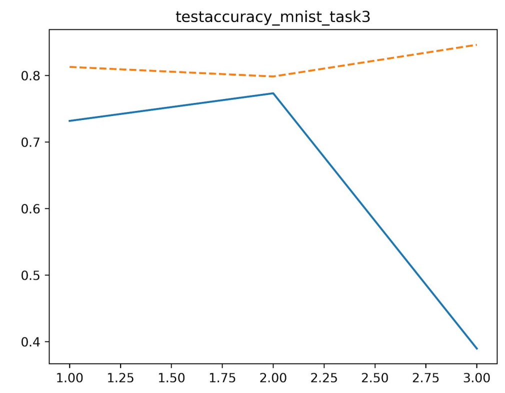
blue line: MLP algorithm implemented



Inference: 1→[30];2→[60];3→[90];4→[200]

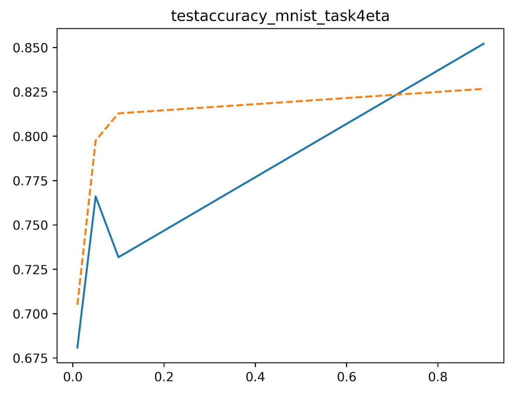
Performance of keras library is found to be better for different sets of neurons chosen

Task 3 comparison: (activation function)
-- line : MLP implementation using keras library
blue line: MLP algorithm implemented



Inference: 1→ sigmoid; 2→ tanh; 3→ relu
There was no inbuilt function for swish in keras library, hence it’s not used for comparison. Performance of the implemented is comparable for sigmoid and tanh, but is poor for relu as compared to the one implemented using keras library.

Task 4 comparison: (parameter variation)
-- line : MLP implementation using keras library
blue line: MLP algorithm implemented

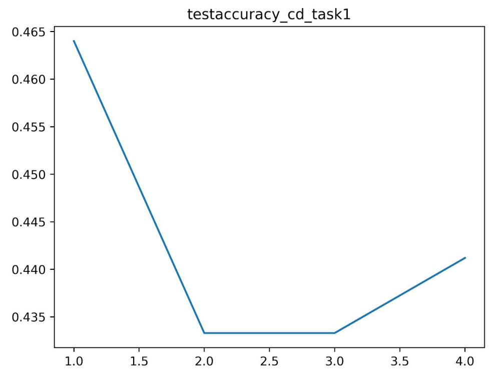


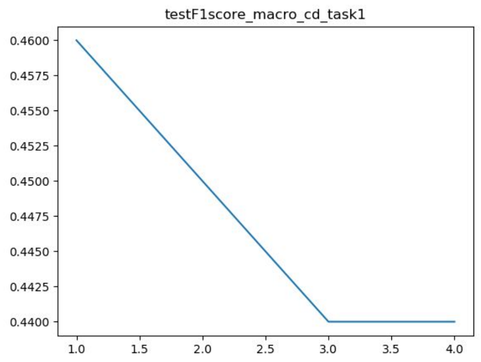
Inference: As the value of learning rate parameter is varied, the performance is almost comparable except in the range from 0.2 to 0.5

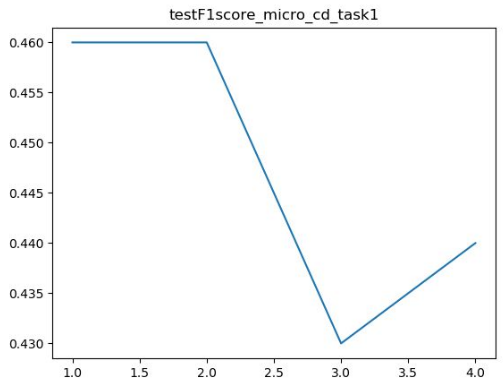
Part - Two: Cat-dog dataset

Task - I (2.5 Points)

Experiment with different number of layers of your neural network and give a plot of accuracy vs. layer count, macro-F1 vs. layer count and micro-F1 vs. layer count.



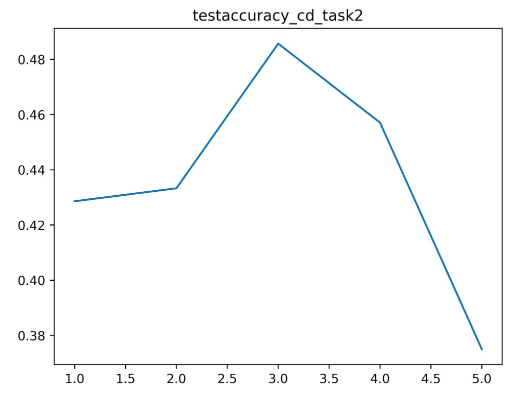


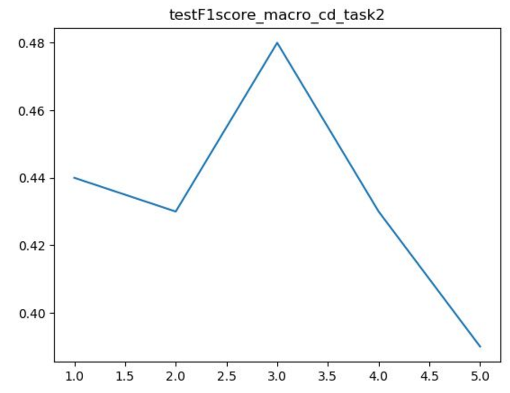


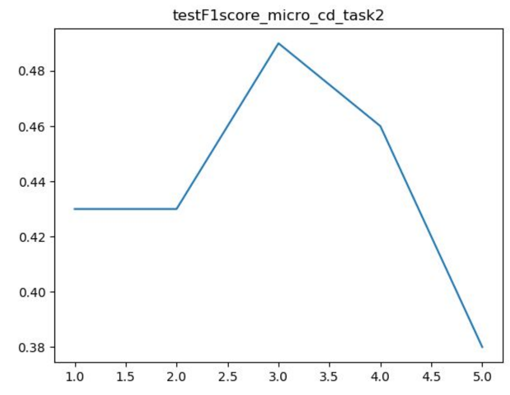
Keeping the number of neurons fixed for each layer(=30) and same activation function(sigmoid is used) various plots for layers ranging from 1 to 5 is plotted. Cat-dog dataset is found to provide higher accuracy for single layer of hidden layer. Hence, for subsequent tasks single hidden layer is chosen

Task - II (2.5 Points)

Try out various configurations of number of neurons in each layer and find the best one. Provide a suitable plot of the same.







Layer number is fixed as 2.

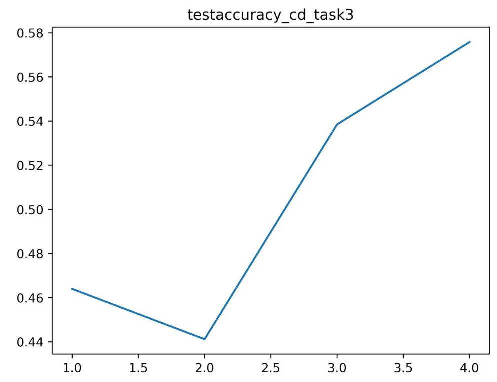
1→[20 10]; 2→[30 30]; 3→[60 40]; 4→[90 30]; 5→[200 50]

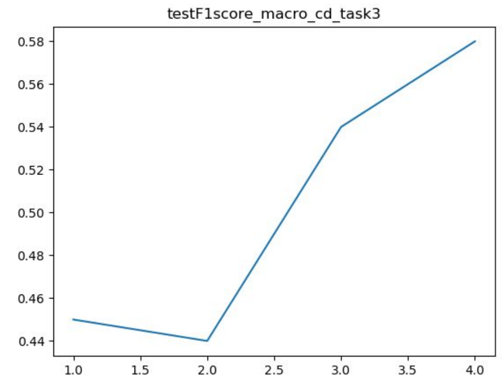
Configuration of [60 40] is found to give the best results.

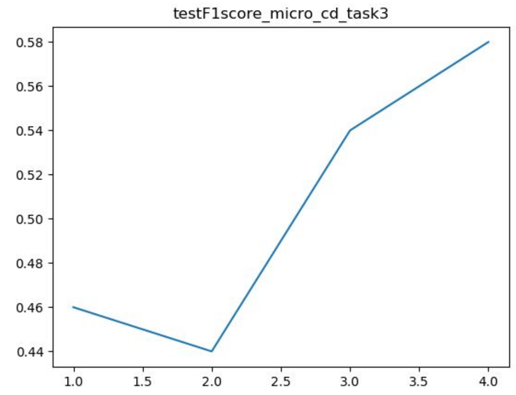
According to bounds of approximation, the number of neurons can’t be too large neither too small. The above plot shows the same.

Task - III (2.5 Points)

Fix the number of layers and neurons in each layer and feed the hidden layer with the following activation functions - **sigmoid(1), tanh(2), relu(3) and swish(4)**. Evaluate the performance of your model. Do it for different configurations of your neural net. Provide a suitable plot of the results obtained.





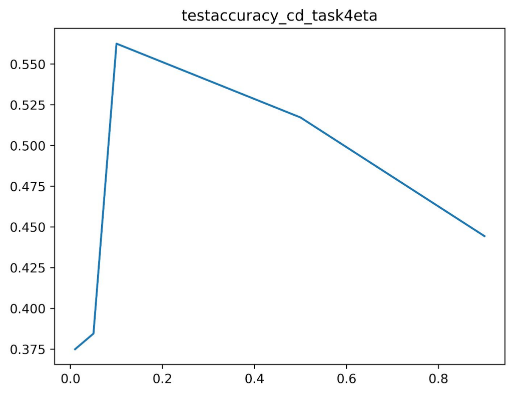


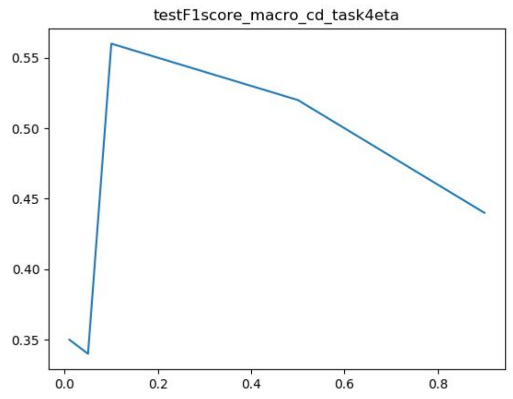
Accuracy, F1 Score is found to be high for swish activation when compared to others, although all of them are in almost the same range.

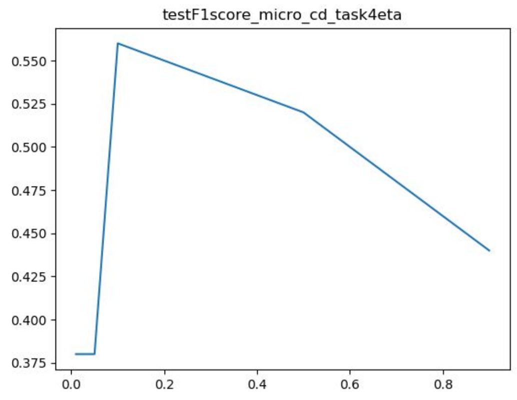
Task - IV (2.5 Points)

Try out various initialization techniques and observe the results.

Variation of eta:

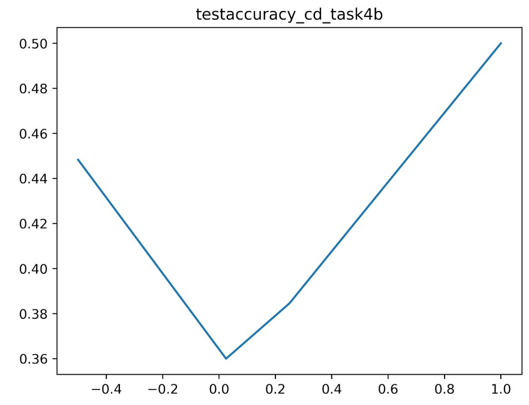


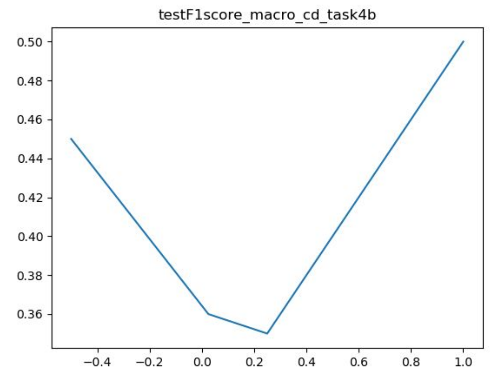


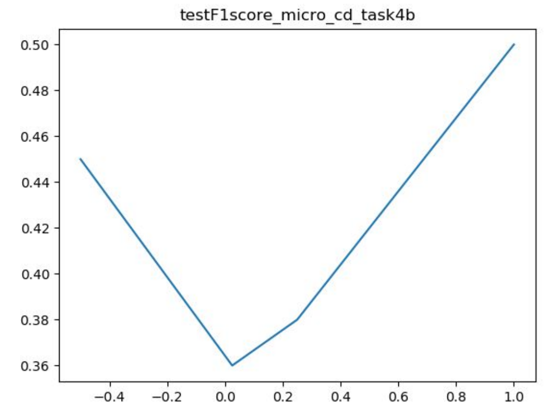


Accuracy and F1 Score gives good results with a learning rate of 0.1, in other ranges the decrease in accuracy might be due to settling into an undesired local minima in gradient descent.

Variation of bias:



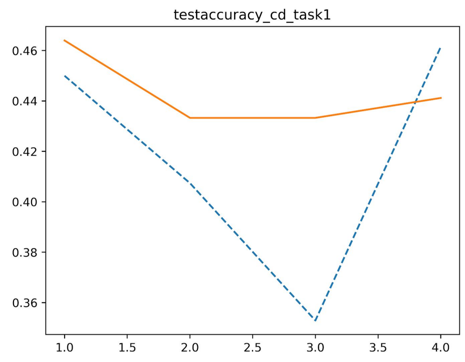




Accuracy, F1 Score is found to increase as the magnitude of the bias increases. As the number of epochs are increased the accuracy would be almost in the same range.

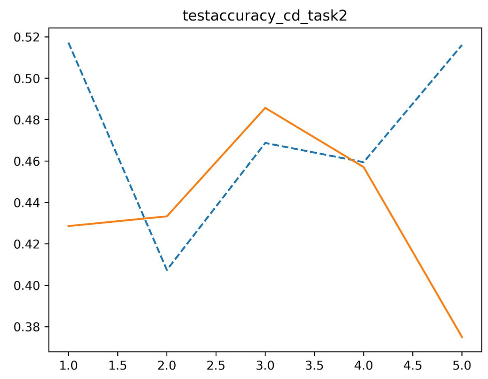
Task - V (10 Points)
Check with keras library and its MLP implementation and compare the results.

Task 1 comparison: (layer count)
-- line : MLP implementation using keras library
orange line: MLP algorithm implemented



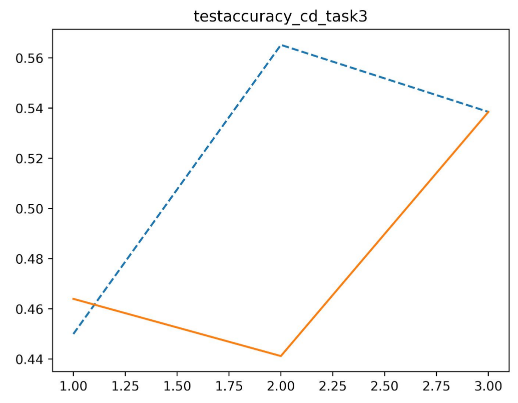
Inference: Accuracy is found to be comparable except when the number of layer is 3.

Task 2 comparison: (number of neurons)
-- line : MLP implementation using keras library
orange line: MLP algorithm implemented



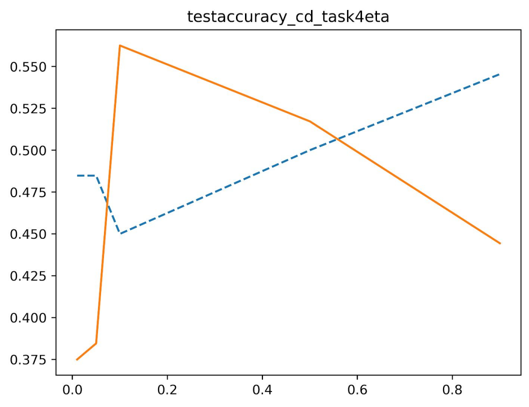
Inference:
1→[20 10]; 2→[30 30];3→[60 40];4→[90 30];5→[200 50]
Accuracy is found to be comparable in both implementation except when number of neurons is increased upto [200 50]

Task 3 comparison: (activation function)
-- line : MLP implementation using keras library
orange line: MLP algorithm implemented



Inference:
1→ sigmoid; 2→ tanh; 3→ relu
Implemented code shows poor performance for tanh. For other 2 activation function performance is comparable. Swish implementation is not available in keras, hence not included.

Task 4 comparison:
-- line : MLP implementation using keras library
orange line: MLP algorithm implemented



Inference:
Variation of learning rate is found to provide different performance in the 2 algorithms implemented.

Part -Three:
For all the three datasets provided in the first assignment, check if the MLNN is performing better than what you obtained using Bayes classifier or NN-classifier.

Dolphins:

	Accuracy	F1 Score macro	F1 Score micro
MLNN	0.952	0.7119	0.9268
NN	0.9286	0.88	0.93
NB	0.91	0.85	0.92

Pubmed:

	Accuracy	F1 Score macro	F1 Score micro
MLNN	0.217	0.117	0.213
NN	0.38	0.33	0.37
NB	0.395	0.37	0.395

Twitter:

	Accuracy	F1 Score macro	F1 Score micro
MLNN	0.5504	0.4025	0.7439
NN	0.509	0.44	0.51
NB	0.2562	0.14	0.26

Inference:
Dolphins dataset- MLNN is found to have a higher accuracy over kNN and Naive Bayes algorithm, although all the 3 have high accuracy.
Pubmed dataset- MLNN is found to have much lower accuracy over kNN and Naive Bayes which provide adjustable level of performance.
Twitter dataset- MLNN is found to provide much higher accuracy compared to kNN and Naive Bayes algorithm