Name - Abhisek Mohanty
Roll - 18EC3AI20

MIES Coding Assignment README file

Details about the folder contents

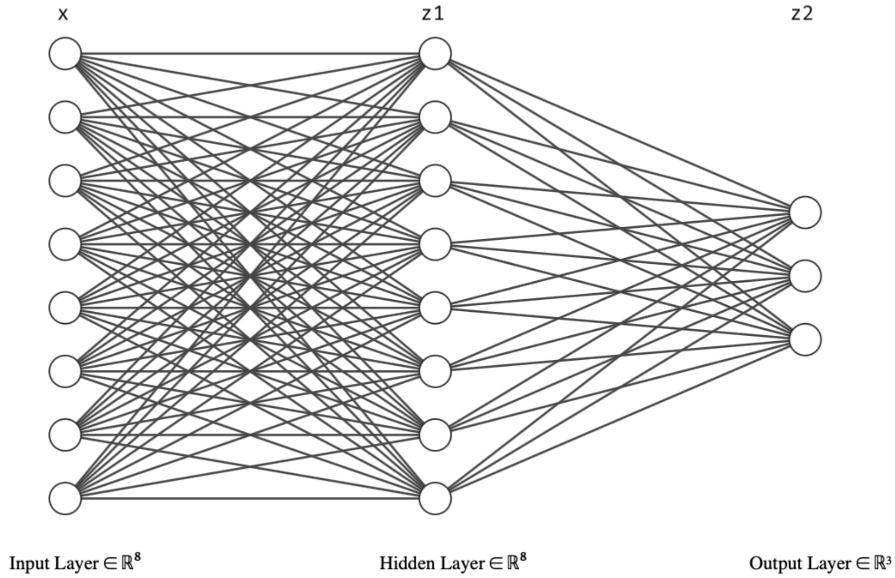
```
.
├── MIES_18EC3AI20_Assignment_3.ipynb
├── Network.png
├── Neural\ network\ computer_Assignment.docx
├── Snails.csv
├── __pycache__
│   ├── create_dataset.cpython-38.pyc
│   ├── nn.cpython-38.pyc
│   └── plots.cpython-38.pyc
├── calc.png
├── create_dataset.py
├── main.py
├── nn.py
└── plots.py
```

1. ***main.py*** - The main python file to execute
2. ***nn.py*** - The file contains the class for Neural network
3. ***plots.py*** - This contains the functions to do the scatter plots of the dataset
4. ***create_dataset.py*** - This contains the function to split the dataset into train and test
5. ***MIES_18EC3AI20_Assignment_3.ipynb*** - This is python notebook with all outputs
6. ***Snails.csv*** - The dataset

Steps to run

1. Open the IPython notebook to get the results.
2. Using terminal -> $ **python main.py**

Neural network architecture and the governing equations



Forward pass

$a_1 = W_1 x + b_1$

$z_1 = \sigma(a_1)$

$a_2 = W_2 z_1 + b_2$

$z_2 = \sigma(a_2)$

$Thus\ the\ predicted\ output\ is\ = z_2$

Backpropagation

$Loss(J) = \frac{1}{2}(z_2 - y)^2; where\ y\ is\ the\ actual\ label$

$$\frac{\partial J}{\partial z_2} = z_2 - y$$

$$\frac{\partial J}{\partial a_2} = \frac{\partial J}{\partial z_2} \times \frac{\partial z_2}{\partial a_2} = (z_2 - y) \times \sigma(a_2) \times (1 - \sigma(a2))$$

$$\frac{\partial J}{\partial W_2} = \frac{\partial J}{\partial a_2} \times \frac{\partial a_2}{\partial W_2} = \frac{\partial J}{\partial a_2} \times z_1$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial a_2} \times \frac{\partial a_2}{\partial b_2} = \frac{\partial J}{\partial a_2} \times 1$$

$$\frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial a_2} \times W_2$$

$$\frac{\partial J}{\partial a_1} = \frac{\partial J}{\partial z_1} \times \frac{\partial z_1}{\partial a_1} = \frac{\partial J}{\partial z_1} \times \sigma(a1) \times (1 - \sigma(a1))$$

$$\frac{\partial J}{\partial W_1} = \frac{\partial J}{\partial a_1} \times \frac{\partial a_1}{\partial W_1} = \frac{\partial J}{\partial a_1} \times x$$

$$\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial a_1} \times \frac{\partial a_1}{\partial b_1} = \frac{\partial J}{\partial a_1} \times 1$$

Details about the code

1. **nn.py -> class Dense_Layer()**
   - **__init__**: This function takes the inputs and initialises the weights and biases, weights_layer1 and bias_layer1 are the weights and biases for hidden layer and are of dimension 8x8 and 1x8 respectively and weights_layer2 and bias_layer2 are the weights and biases for final/output layer and are of dimension 8 x 3 and 1 x 3 respectively.
   - **sigmoid(self,input)** : return the sigmoid value of input
   - **sigmoid_derivative(self,input)** : returns the derivative of the sigmoid for the input.
   - **forward(self,input)** : Calculates the outputs for each layer, output_layer1 and activated_output_layer1 are the output and activated sigmoid output of hidden layer(dimension:Batch_sizex8). output_layer2 and activated_output_layer2 are the output and activated sigmoid output of the output layer(dimension:Batch_sizex3).
   - **calc_loss(self,predicted_label,actual_label)** : computes the MSE loss after feed forward pass and returns the normalized loss
   - **backward(self,input,y_pred,y_actual)** : This calculates the partial derivatives of the Loss (J) with respect to activated_output_layer2, output_layer2, weights_layer2 and bias_layer2 for the output layer and activated_output_layer1, output_layer1, weights_layer1 and bias_layer1 for the hidden layer
   - **update_parameters(self,dW1,db1,dW2,db2)** : It updates the weights and the biases of both the layers using the partial derivatives that we have passed
   - **train(self,input,y_actual)** : This trains the model by first calling self.forward, the self.calc_loss to find the loss, the self.backward to find the derivatives of Loss(J) wrt model parameters and self.update_parameters to backpropagate and update the parameters.
   - **predict(self,input)** : Predicts the output label for an input
   - **calc_accuracy(self,input,y_actual)** : It calculates the accuracy by first passing the batch through self.forward and then predicting the labels.

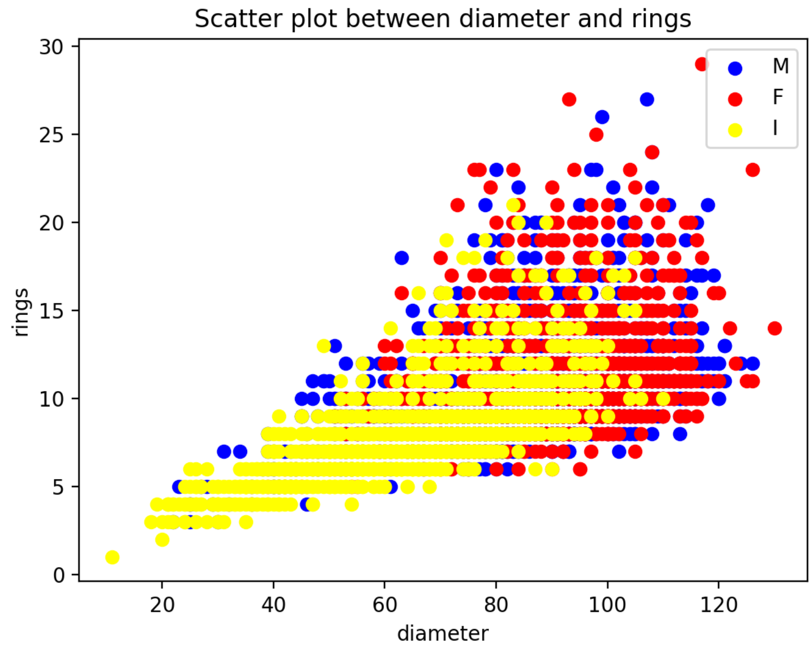
2. **create_dataset.py**
   - **split_dataset(df,ratio)** : It takes the dataframe df and the ratio with which we want to divide the training data. It splits the dataset into train and test, and also stores the labels in Y_train and Y_test. Finally returns ***train,Y_train,test,Y_test***
3. **plots.py**
   - **scatter_plot(df,gender_label,column1,column2)** : It takes the dataframe df as input and also the gender_label which is df['sex'] column. Then it also takes two features in column1 and column2 respectively. Finally plots the scatter plot for the two features
   - **scatter_plot_normalized (df,gender_label,column1,column2)** : It takes the dataframe df as input and also the gender_label which is df['sex'] column.It then normalizes the dataframe df after converting it to a numpy array. Then it also takes two features in column1 and column2 respectively. Finally plots the scatter plot for the two features
4. **main.py**

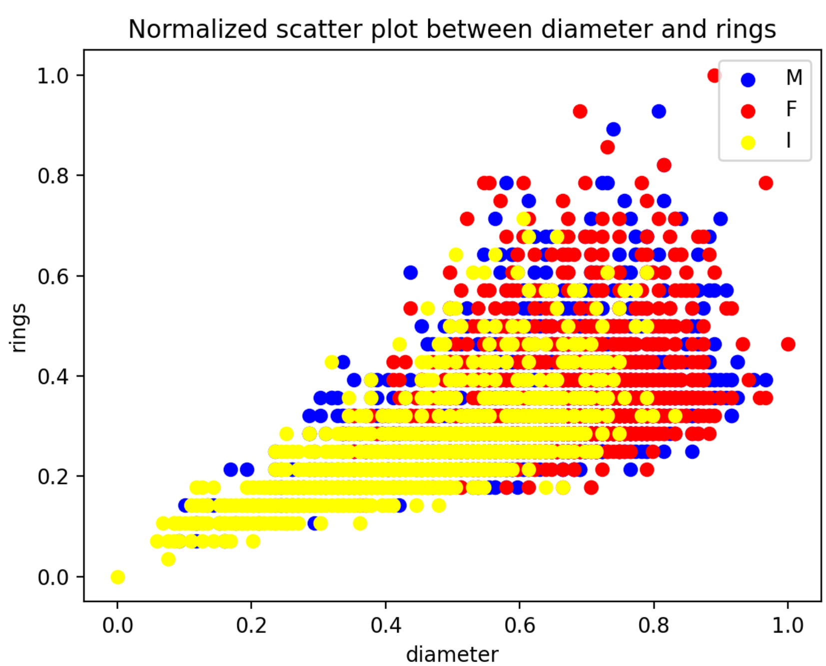
   In the main function we are doing the following things

   1. Reading the dataset and storing it in a variable df
   2. Plotting the scatter plots wrt any two features as per choice
   3. We define our model in the variable 'neural_net'
   4. Divide the dataset into train and test with labels Y_train and Y-test respectively
   5. Train the model for 500 epochs, and after every epoch we calculate and store the loss, training accuracy and the testing accuracy respectively
   6. Then we print the final training and test accuracies
   7. Then we plot the model statistics and analyse the plots

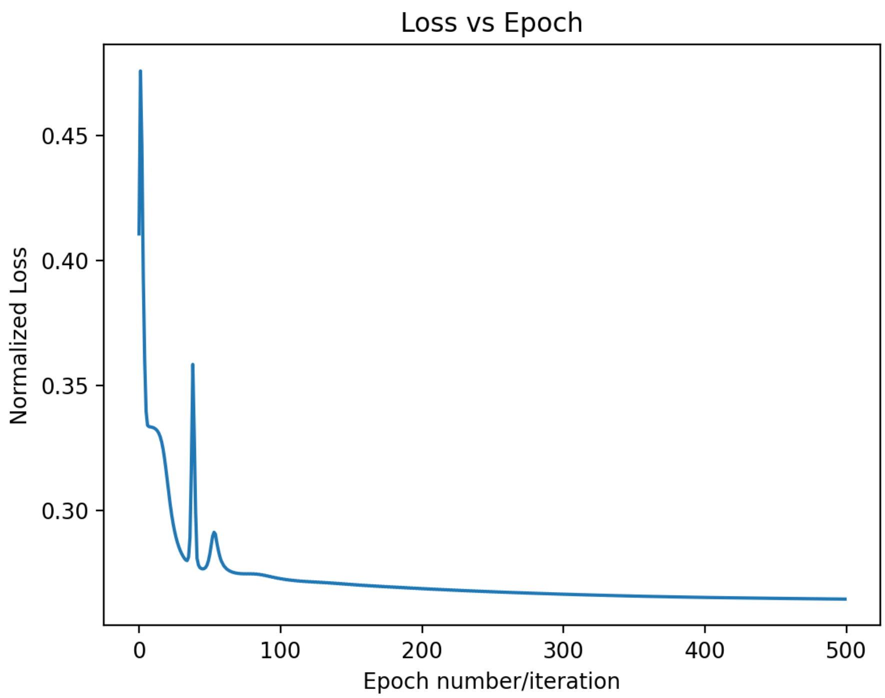
Scatter plots

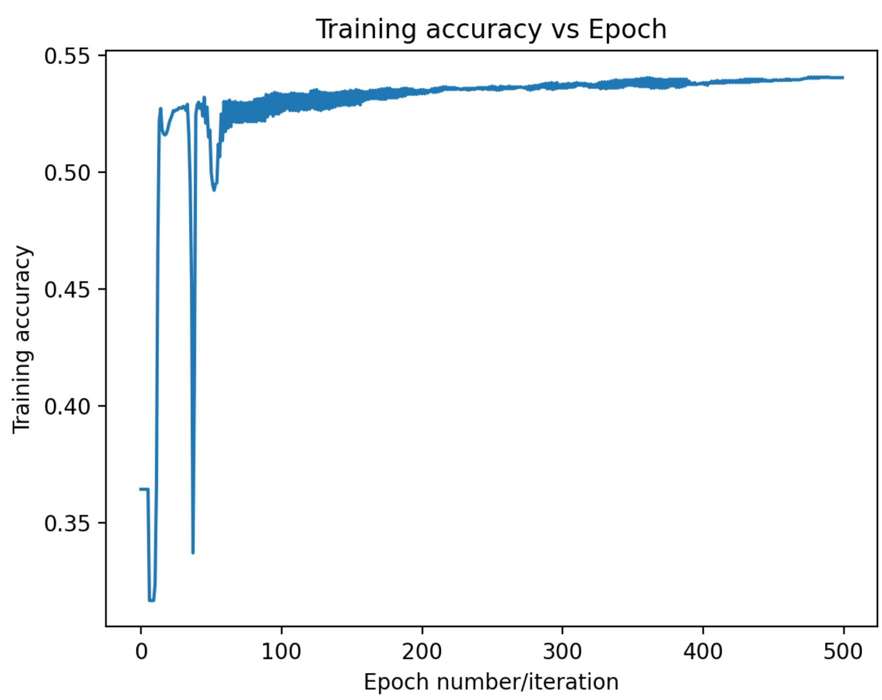
Scatter plot for features ‘diameter’ and ‘rings’ for both normalized and non-normalized dataset

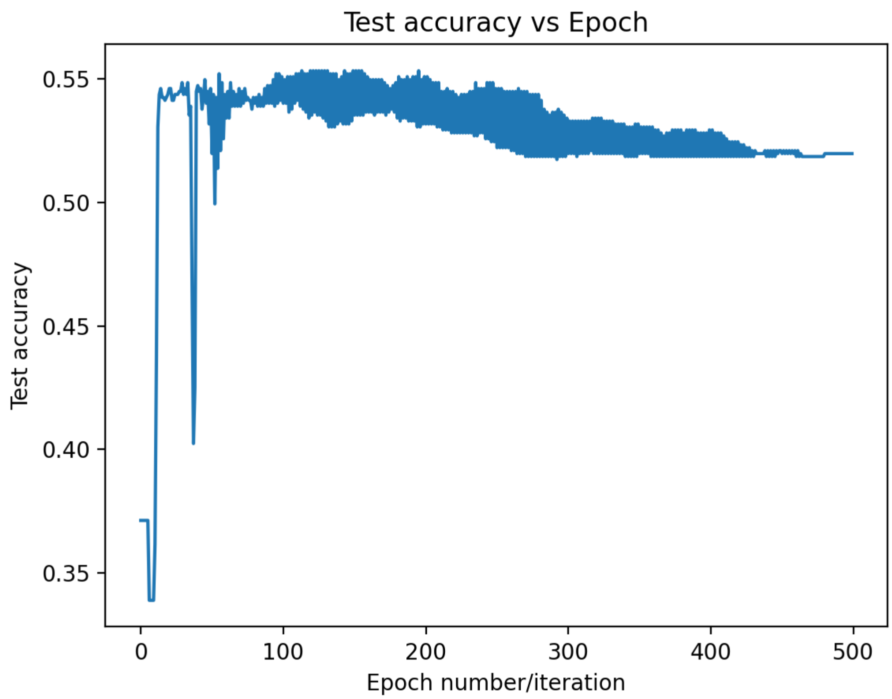




Model statistics and results








```
➜  MIES 3 git:(main) ✗ python main.py
Training accuracy =  0.540394973070017 9
Test accuracy =  0.5197604790419161
```