CS 6375

ASSIGNMENT _______2_______

Names of students in your group:

Ching-Yi Chang - CXC190002

Xiaokai Rong - XXR230000

Number of free late days used: ________0________

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.

UCI ML Repository: https://archive.ics.uci.edu/dataset/9/auto+mpg Auto MPG dataset

In this part, you will code a neural network (NN) having **at least one hidden layers, besides the input and output layers.** You are required to pre-process the data and then run the processed data through your neural net. Below are the requirements and suggested steps of the program

- The programming language for this assignment will be Python 3.x
- **You cannot use any libraries for neural net creation.** You are free to use any other libraries for data loading, pre-processing, splitting, model evaluation, plotting, etc.
- Your code should be in the form of a Python class with methods like pre-process, train, test within the class. I leave the other details up to you.
- As the first step, pre-process and clean your dataset. There should be a method that does this.
- Split the pre-processed dataset into training and testing parts. You are free to choose any reasonable value for the train/test ratio, but be sure to mention it in the README file.
- Code a neural net having **at least one hidden layer**. You are free to select **the number of neurons in each layer**. Each neuron in the hidden and output layers should have a bias connection.
- You are required to add an optimizer on top of the basic backpropagation algorithm. This could be the one you selected in the previous assigment or a new one. Some good resources for gradient descent optimizers are:

https://arxiv.org/pdf/1609.04747.pdf

https://ruder.io/optimizing-gradient-descent/

https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9

- You are **required to code three different activation functions**:
    1. Sigmoid
    2. Tanh
    3. ReLu

The earlier part of this assignment may prove useful for this stage. The activation function should be a parameter in your code.

- Code a method for creating a neural net model from the training part of the dataset. Report the training accuracy.
- Apply the trained model on the test part of the dataset. Report the test accuracy.
- You have to tune model parameters like learning rate, activation functions, etc. Report your results in a tabular format, with a column indicating the parameters used, a column for training accuracy, and one for test accuracy.

Dataset

You can use any one dataset from the UCI ML repository, or any other standard repository such as Kaggle.

What to submit

You need to submit the following for the programming part:

- Link to the dataset used. Please do not include the data as part of you submission.
- Your source code and a README file indicating how to run your code. Do not hardcode any paths to your local computer. It is fine to code any public paths, such as AWS S3.
- Output for your dataset summarized in a tabular format for different combination of parameters
- A brief report summarizing your results. For example, which activation function performed the best and why do you think so.

Any assumptions that you made. Report:

Use Pyton 3.8

Step 1: Load the dataset

url = 'https://raw.githubusercontent.com/SparrowChang/CS6375_assignment1/main/auto%2Bmpg/auto-mpg.data'

Read the CSV file from the URL into a DataFrame, in the original dataset, total with 8 features ('cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model year', 'origin', 'car name'), 1 target('mpg’). MPG means miles per gallon, the number of miles your vehicle can travel down the streets of Deltona on one gallon of gas.

Step 2: Pre-processing

a. Convert categorical variables to numerical variables

b. Drop the columns 'car name' that are not relevant for the regression analysis

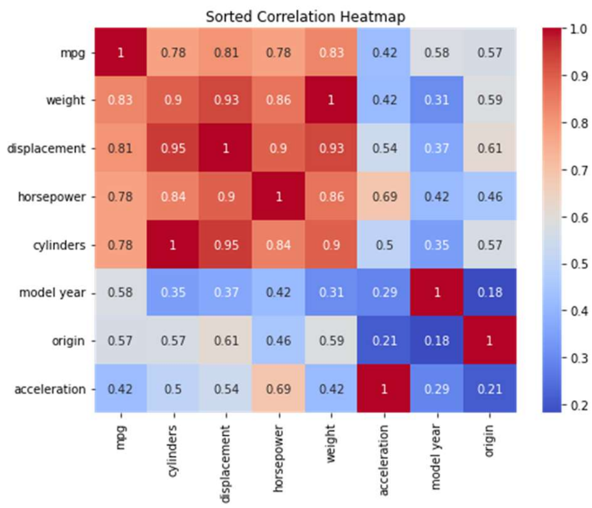
c. Create a StandardScaler object and normalized the dataframe

d. Remove null or NA values, dropna()

e. Remove redundant rows

f. Calculate the correlation matrix. Sort absolate correlation with 'mpg'. We would know ‘weight’, ‘displacement’, ‘horsepower’, ‘cylinders’ are high correlation with target.

Finally, the dataset is with 7 features ('cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model year', 'origin') and 392 instances.



Step 3: Split the dataset into training and test sets

We split the training and test sets to 80/20 (test_size = 0.2, random_state =42)

Step 4: To define Activation Functions

a. sigmoid(x): 1 / (1 + exp(-x)) and its derivative: sigmoid(x) * (1 - sigmoid(x))

b. tanh(x): tanh(x) and its derivative: 1 - tanh(x)**2

c. relu(x): maximum(0, x) and its derivative: 1 (x>0)

Step 5: To define Neural Network Class

```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, activation):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.activation = activation

        # Initialize weights and biases
        self.W1 = np.random.randn(self.input_size, self.hidden_size)
        self.b1 = np.zeros((1, self.hidden_size))
        self.W2 = np.random.randn(self.hidden_size, self.output_size)
        self.b2 = np.zeros((1, self.output_size))

    def forward(self, X):
        # Input to hidden layer
        self.z1 = np.dot(X, self.W1) + self.b1

        # Apply activation function to the hidden layer
        if self.activation == 'sigmoid':
            self.a1 = sigmoid(self.z1)
        elif self.activation == 'tanh':
            self.a1 = tanh(self.z1)
        elif self.activation == 'relu':
            self.a1 = relu(self.z1)

        # Hidden to output layer
        self.z2 = np.dot(self.a1, self.W2) + self.b2
        self.output = self.z2  # Since it's a regression problem, no activation on output

        return self.output
```

```
    def backward(self, X, y, output):
        # Compute the loss and the derivative of the loss with respect to output
        loss = np.mean((output - y)**2)
        d_loss_output = 2 * (output - y) / X.shape[0]

        # Backpropagate the error
        if self.activation == 'sigmoid':
            d_output_z2 = sigmoid_derivative(output)
        elif self.activation == 'tanh':
            d_output_z2 = tanh_derivative(output)
        elif self.activation == 'relu':
            d_output_z2 = relu_derivative(output)

        d_loss_z2 = d_loss_output * d_output_z2
        d_loss_a1 = np.dot(d_loss_z2, self.W2.T)
        d_loss_z1 = d_loss_a1 * sigmoid_derivative(self.a1)  # Use sigmoid derivative for sigmoid activ
```

```python
        # Compute the gradients
        d_loss_W2 = np.dot(self.a1.T, d_loss_z2)
        d_loss_b2 = np.sum(d_loss_z2, axis=0, keepdims=True)
        d_loss_W1 = np.dot(X.T, d_loss_z1)
        d_loss_b1 = np.sum(d_loss_z1, axis=0, keepdims=True)

        return loss, d_loss_W1, d_loss_b1, d_loss_W2, d_loss_b2

    def update_weights(self, d_loss_W1, d_loss_b1, d_loss_W2, d_loss_b2, learning_rate):
        # Update weights and biases using gradient descent
        self.W1 -= learning_rate * d_loss_W1
        self.b1 -= learning_rate * d_loss_b1
        self.W2 -= learning_rate * d_loss_W2
        self.b2 -= learning_rate * d_loss_b2
```

```python
    def train(self, X, y, learning_rate, num_epochs, tolerance):
        for epoch in range(num_epochs):
            # Forward pass
            output = self.forward(X)

            # Backward pass
            loss, d_loss_W1, d_loss_b1, d_loss_W2, d_loss_b2 = self.backward(X, y, output)

            # Update weights and biases
            self.update_weights(d_loss_W1, d_loss_b1, d_loss_W2, d_loss_b2, learning_rate)

            if (epoch + 1) % 100 == 0:
                print(f'Epoch {epoch+1}/{num_epochs}, Loss: {loss:.2f}')

            # check if the loss is below the tolerance level
            if loss < tolerance:
                print('Traning converaged!')
                break

        print('Training completed!')

    def predict(self, X):
        output = self.forward(X)
        return output
```

Step 6: To set up Neural Network model

We create 1 hidden layer (64 hidden units) and 1 output layer (1 output units). In each layer, with its own weights and bias. 3 activations (sigmoid, tanh, relu)

For hyperparameters:

epochs: 100, 500, 1000.

learning rate: 0.001, 0.01, 0.1, 0.5

```python
iteration_range = [100, 500, 1000]
learning_rate_range = [0.001, 0.01, 0.1, 0.5]
activ = ['sigmoid', 'tanh', 'relu']
```

Step 7: train and evaluate test result

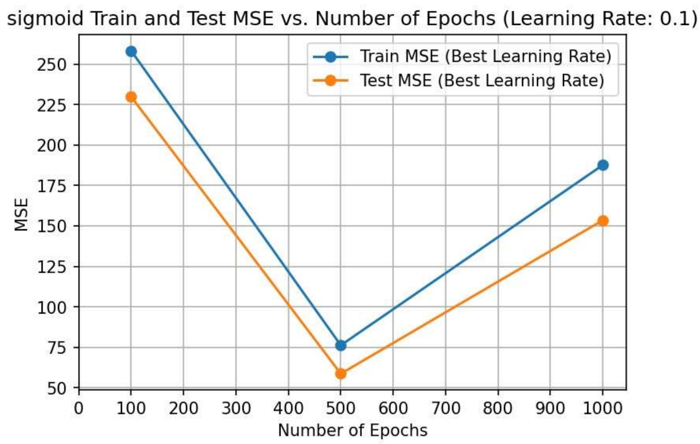
```python
model = NeuralNetwork(input_size, hidden_size, output_size, activation=activ[1])

# Train the model
model.train(X_train, y_train, learning_rate, num_epochs, tolerance)
```

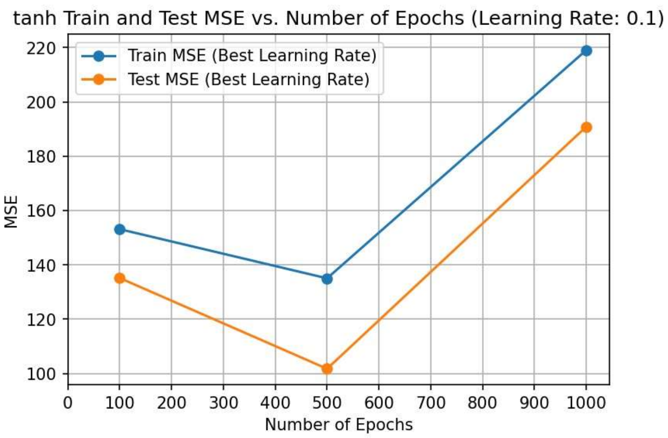
```python
# Evaluate the model
train_predictions = model.predict(X_train).flatten()
train_loss = np.mean((train_predictions - y_train.flatten())**2)
train_accuracy = np.mean(np.abs(train_predictions - y_train.flatten()))

test_predictions = model.predict(X_test).flatten()
test_loss = np.mean((test_predictions - y_test.flatten())**2)
test_accuracy = np.mean(np.abs(test_predictions - y_test.flatten()))
```

```
sigmoid Learning Rate: 0.001, Num Epochs: 100, Train Loss: 433.08, Train Accuracy: 19.11, Test Loss: 413.88, Test Accuracy: 18.88
----------------
sigmoid Learning Rate: 0.001, Num Epochs: 500, Train Loss: 347.36, Train Accuracy: 16.72, Test Loss: 299.86, Test Accuracy: 15.70
----------------
sigmoid Learning Rate: 0.001, Num Epochs: 1000, Train Loss: 254.82, Train Accuracy: 14.28, Test Loss: 231.58, Test Accuracy: 13.85
----------------
sigmoid Learning Rate: 0.01, Num Epochs: 100, Train Loss: 325.55, Train Accuracy: 16.45, Test Loss: 291.38, Test Accuracy: 15.67
----------------
sigmoid Learning Rate: 0.01, Num Epochs: 500, Train Loss: 269.90, Train Accuracy: 14.73, Test Loss: 227.94, Test Accuracy: 13.64
----------------
sigmoid Learning Rate: 0.01, Num Epochs: 1000, Train Loss: 242.60, Train Accuracy: 13.48, Test Loss: 223.47, Test Accuracy: 13.20
----------------
sigmoid Learning Rate: 0.1, Num Epochs: 100, Train Loss: 258.28, Train Accuracy: 14.21, Test Loss: 229.88, Test Accuracy: 13.60
----------------
sigmoid Learning Rate: 0.1, Num Epochs: 500, Train Loss: 76.20, Train Accuracy: 6.32, Test Loss: 58.62, Test Accuracy: 5.55
----------------
sigmoid Learning Rate: 0.1, Num Epochs: 1000, Train Loss: 187.66, Train Accuracy: 11.22, Test Loss: 153.52, Test Accuracy: 10.12
----------------
sigmoid Learning Rate: 0.5, Num Epochs: 100, Train Loss: 2327.21, Train Accuracy: 47.28, Test Loss: 2336.21, Test Accuracy: 47.56
----------------
sigmoid Learning Rate: 0.5, Num Epochs: 500, Train Loss: 478.53, Train Accuracy: 17.80, Test Loss: 452.05, Test Accuracy: 16.56
----------------
sigmoid Learning Rate: 0.5, Num Epochs: 1000, Train Loss: 206.68, Train Accuracy: 12.98, Test Loss: 226.90, Test Accuracy: 13.96
----------------
Best Mean squared error (MSE):58.62
Best parameters:{'learning_rate': 0.1, 'num_epochs': 500}
--------------------------------------------------------------------------------
```

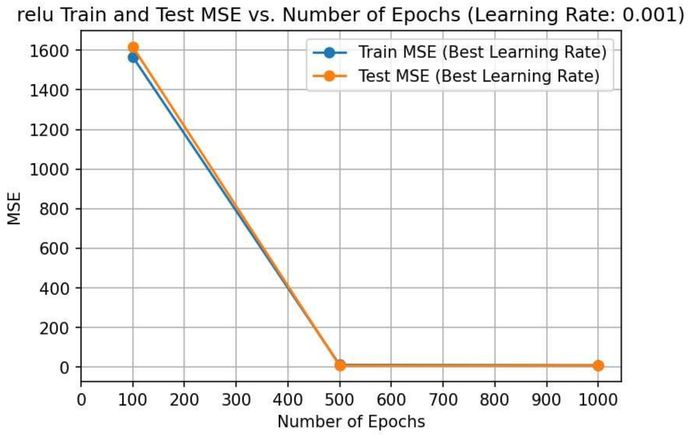


```
tanh Learning Rate: 0.001, Num Epochs: 100, Train Loss: 231.46, Train Accuracy: 12.65, Test Loss: 209.87, Test Accuracy: 12.30
-----------------
tanh Learning Rate: 0.001, Num Epochs: 500, Train Loss: 287.76, Train Accuracy: 15.06, Test Loss: 260.48, Test Accuracy: 14.69
-----------------
tanh Learning Rate: 0.001, Num Epochs: 1000, Train Loss: 269.16, Train Accuracy: 14.09, Test Loss: 243.44, Test Accuracy: 13.61
-----------------
tanh Learning Rate: 0.01, Num Epochs: 100, Train Loss: 253.81, Train Accuracy: 13.12, Test Loss: 215.33, Test Accuracy: 12.39
-----------------
tanh Learning Rate: 0.01, Num Epochs: 500, Train Loss: 289.12, Train Accuracy: 13.80, Test Loss: 257.49, Test Accuracy: 13.25
-----------------
tanh Learning Rate: 0.01, Num Epochs: 1000, Train Loss: 266.76, Train Accuracy: 12.69, Test Loss: 214.48, Test Accuracy: 11.67
-----------------
tanh Learning Rate: 0.1, Num Epochs: 100, Train Loss: 153.14, Train Accuracy: 9.77, Test Loss: 135.16, Test Accuracy: 9.70
-----------------
tanh Learning Rate: 0.1, Num Epochs: 500, Train Loss: 135.05, Train Accuracy: 9.00, Test Loss: 101.75, Test Accuracy: 7.83
-----------------
tanh Learning Rate: 0.1, Num Epochs: 1000, Train Loss: 219.07, Train Accuracy: 11.91, Test Loss: 190.89, Test Accuracy: 11.52
-----------------
tanh Learning Rate: 0.5, Num Epochs: 100, Train Loss: 872.79, Train Accuracy: 27.97, Test Loss: 869.75, Test Accuracy: 27.63
-----------------
tanh Learning Rate: 0.5, Num Epochs: 500, Train Loss: 754.34, Train Accuracy: 26.19, Test Loss: 782.79, Test Accuracy: 26.32
-----------------
tanh Learning Rate: 0.5, Num Epochs: 1000, Train Loss: 286.96, Train Accuracy: 14.88, Test Loss: 258.81, Test Accuracy: 14.09
-----------------
Best Mean squared error (MSE):101.75
Best parameters:{'learning_rate': 0.1, 'num_epochs': 500}
-------------------------------------------------------------------------------------
```




```
relu Learning Rate: 0.001, Num Epochs: 100, Train Loss: 1566.87, Train Accuracy: 34.75, Test Loss: 1618.12, Test Accuracy: 35.82
-----------------
relu Learning Rate: 0.001, Num Epochs: 500, Train Loss: 11.60, Train Accuracy: 2.53, Test Loss: 7.62, Test Accuracy: 2.09
-----------------
relu Learning Rate: 0.001, Num Epochs: 1000, Train Loss: 9.60, Train Accuracy: 2.35, Test Loss: 8.87, Test Accuracy: 2.32
-----------------
relu Learning Rate: 0.01, Num Epochs: 100, Train Loss: 10.77, Train Accuracy: 2.41, Test Loss: 10.99, Test Accuracy: 2.32
-----------------
relu Learning Rate: 0.01, Num Epochs: 500, Train Loss: 28.62, Train Accuracy: 4.58, Test Loss: 30.10, Test Accuracy: 4.71
-----------------
relu Learning Rate: 0.01, Num Epochs: 1000, Train Loss: 20.25, Train Accuracy: 3.55, Test Loss: 23.41, Test Accuracy: 3.83
-----------------
relu Learning Rate: 0.1, Num Epochs: 100, Train Loss: 92955287.86, Train Accuracy: 8018.12, Test Loss: 96327574.86, Test Accuracy: 8404.64
-----------------
relu Learning Rate: 0.1, Num Epochs: 500, Train Loss: 2294.41, Train Accuracy: 46.92, Test Loss: 2144.32, Test Accuracy: 45.35
-----------------
relu Learning Rate: 0.1, Num Epochs: 1000, Train Loss: 2823269717.64, Train Accuracy: 30560.29, Test Loss: 3348023156.30, Test Accuracy: 31785.55
-----------------
relu Learning Rate: 0.5, Num Epochs: 100, Train Loss: 366332254753.51, Train Accuracy: 388048.93, Test Loss: 322842765523.19, Test Accuracy: 392196.52
-----------------
relu Learning Rate: 0.5, Num Epochs: 500, Train Loss: 5781450939.20, Train Accuracy: 59280.89, Test Loss: 5450243051.49, Test Accuracy: 59599.63
-----------------
relu Learning Rate: 0.5, Num Epochs: 1000, Train Loss: 67321445363041.20, Train Accuracy: 5931569.51, Test Loss: 71302520400159.77, Test Accuracy: 6156249.34
-----------------
Best Mean squared error (MSE):7.62
Best parameters:{'learning_rate': 0.001, 'num_epochs': 500}
-------------------------------------------------------------------------------------

Execution time: 24.97 seconds
Current date and time: 2023-07-03 17:07:40.041273
```



Train/ Test MSE of MPG (miles per gallon) comparison											
Activations	sigmoid			tanh			relu			LR by gradient descent	LR by linear regression library 10.71
(R) epochs (D) learning rate	100	500	1000	100	500	1000	100	500	1000	1000	
0.001	433.08 413.88	347.36 299.86	254.82 231.58	231.46 209.87	287.76 260.48	269.16 243.44	1566.87 1618.12	11.60 7.62	9.60 8.87		
0.01	325.55 291.38	269.90 227.94	242.60 223.47	253.81 215.33	289.12 257.49	266.76 214.48	10.77 10.99	28.62 30.10	20.25 23.41		
0.1	258.28 229.88	76.20 58.62	187.66 153.52	153.14 135.16	135.05 101.75	219.07 190.89	$>10^7$ $>10^7$	2294.41 2144.32	$>10^7$ $>10^7$	10.73	
0.5	2327.21 2336.21	478.53 452.05	206.68 226.90	872.79 869.75	754.34 782.79	286.96 258.81	$>10^7$ $>10^7$	$>10^7$ $>10^7$	$>10^7$ $>10^7$		

We aim to compare the results with those obtained in Assignment 1. For the prediction of MPG (miles per gallon), the test mean squared error (MSE) achieved using the linear regression library is 10.71. When applying the gradient descent algorithm with 1000 epochs and a learning rate of 0.1, the test MSE is slightly higher at 10.73.

Surprisingly, the ReLU activation function yields a test MSE of 7.62, which not only matches our Assignment 1 result but is even lower. This suggests that the ReLU activation function is effective for this particular task.

However, the test MSE results for MPG using the Sigmoid and Tanh activation functions are unstable. The inconsistency in the test results suggests that these activation functions may not be as suitable for this specific prediction task.

By comparing and analyzing these outcomes, we can gain valuable insights into the performance of different methods and activation functions for the MPG prediction problem.

We have identified a few potential reasons for the observed behavior:

a. The dataset used for training is relatively small, containing only 392 instances. Limited data can sometimes lead to suboptimal model performance due to insufficient patterns for the algorithm to learn from.

b. The learning rate used for both the Sigmoid and ReLU activation functions is set to 0.1. This relatively high learning rate may cause the optimization process to converge to local minima instead of the global minimum, impacting the model's overall performance.

c. It is possible that some features in the dataset may not be highly informative or relevant for the target variable. Utilizing domain knowledge and performing feature selection techniques can potentially improve the model's performance by focusing on the most important features.

d. The algorithm implemented from scratch is a simple model. Alternatively, incorporating a penalty term, such as regularization techniques like L1 or L2 regularization, could potentially help in reducing the mean squared error (MSE) further and improving the model's generalization capability.

Consider exploring these suggestions to address the observed issues and enhance the performance of the model.