Group 10
Sahithi Kotagiri – Se21uari122
Veda Miracle Palaparthy – Se21uari188
Sri Sanjana Ravuri – Se21uari135
Vaishnavi Neela – Se21uari179

Listing out the step-by-step backpropagation algorithm, with the equations and control actions of the program

1. Initialize Network: Begin by setting up the neural network with random weights and biases.

2. Forward Pass:

- Input the training data and propagate it through the network layer by layer.
- For each layer l, calculate the weighted sum of inputs z^l and the activation a^l using the defined equations.

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$$
$$a^{(l)} = \sigma(z^{(l)})$$

- Where:
  - $W^{(l)}$ is the weight matrix for layer l.
  - $a^{(l-1)}$ is the activation of the previous layer.
  - $b^{(l)}$ is the bias vector for layer l.
  - σ is the activation function.

- Utilize an activation function, such as the sigmoid function, to compute the activation.

3. Compute Loss: Calculate the discrepancy between the predicted output and the actual target using a suitable loss function like Mean Squared Error (MSE)

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- n is the number of samples.
- y_i is the actual target.
- $\hat{y}_i$ is the predicted output.

4. Backward Pass:

- Compute the gradients of the loss with respect to the output layer activations and propagate them backward through the layers using the chain rule.
- Update the gradients of the weights and biases by applying the computed gradients and the activation of the previous layer.

$$\frac{\partial L}{\partial a^{(L)}} = \frac{1}{n}(2(\hat{y} - y))$$

- Backpropagate the gradients through the layers using the chain rule:

$$\frac{\partial L}{\partial z^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \cdot \sigma'(z^{(l)})$$
$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial z^{(l)}} \cdot a^{(l-1)}$$
$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial z^{(l)}}$$
$$\frac{\partial L}{\partial a^{(l-1)}} = (W^{(l)})^T \cdot \frac{\partial L}{\partial z^{(l)}}$$

Where:

- σ' is the derivative of the activation function.

5. Update Weights and Biases:

- Use the computed gradients to adjust the weights and biases of the network, employing an optimization algorithm like Gradient Descent with a predetermined learning rate.

$$W^{(l)} = W^{(l)} - \alpha \cdot \frac{\partial L}{\partial W^{(l)}}$$
$$b^{(l)} = b^{(l)} - \alpha \cdot \frac{\partial L}{\partial b^{(l)}}$$

Where:

- α is the learning rate.

6. Repeat:

- Iteratively execute the forward pass, backward pass, and weight updates for a predefined number of epochs or until convergence.

- Monitor the loss function throughout the training process to ensure convergence.

- Fine-tune hyperparameters such as the learning rate and batch size to optimize performance.

- Employ regularization techniques like L2 regularization or dropout to prevent overfitting.

Control Actions:

- Implement the forward pass, backward pass, and weight updates iteratively for a specified number of epochs.

- Manage memory efficiently, especially for large datasets and complex network architectures.

- Monitor and analyse the loss function's behaviour during training to evaluate convergence.

- Experiment with various hyperparameters and regularization techniques to enhance model performance.

- Optimize computational efficiency using techniques like mini-batch gradient descent and momentum.

- Incorporate appropriate error handling and logging mechanisms for debugging and performance evaluation.

TOY PROBLEM

1. Start by importing necessary libraries like NumPy for numerical computations and Matplotlib for plotting.

2. Generating Data: Create synthetic training and validation data by generating sine wave data within a specific range.

3. Normalization: Define a function to normalize the data to a range of -1 to 1, which helps make training more stable.

4. Activation Functions: Define the tanh activation function and its derivative. Tanh is used because it squashes values between -1 and 1, similar to sigmoid but with a range from -1 to 1.

5. Neural Network Parameters: Specify the architecture and hyperparameters of the neural network, including the sizes of input and hidden layers, learning rate, number of epochs, batch size, momentum, and L2 regularization parameter.

6. Weights Initialization: Initialize the weights of the network randomly. Proper initialization is crucial for the network to learn effectively.

7. Momentum Initialization: Initialize momentum terms to keep track of how the weights should change based on previous updates.

8. Training Loop: Loop through the data for a number of epochs. Within each epoch, iterate through the training data in mini-batches. For each mini-batch:

- Perform a forward pass through the network to compute the output.

- Compute the error and update the weights through backpropagation using the gradient descent algorithm with momentum and L2 regularization.

9. Validation Error Calculation: At the end of each epoch, calculate the validation error to monitor how well the model generalizes to unseen data.

10. Training Error Calculation: Also calculate the training error to monitor the model's performance on the training data.

11. Plotting Results: Plot the training data along with the output of the neural network and visualize the training and validation errors over epochs.

CCPP_ANN

1. Import necessary libraries:

- Import required libraries including pandas, numpy, matplotlib, and scikit-learn.

2. Load the dataset:

- Load the dataset from an Excel file using pandas.

3. Handle missing values:

- Remove any rows with missing values in the dataset.

4. Separate features and target variable:

- Separate the features (input variables) and the target variable (output) from the dataset.

5. Split the data:

- Split the dataset into training, validation, and test sets using the `train_test_split` function from scikit-learn.

6. Standardize the data:

- Standardize the features using `StandardScaler` to scale them to have mean 0 and standard deviation 1.

7. Define activation functions and their derivatives:

- Define activation functions such as sigmoid, relu, and tanh, along with their derivatives.

8. Define optimizer functions:

- Define optimizer functions such as stochastic gradient descent (SGD) with momentum and Adam.

9. Define MAPE error calculation:

- Define a function to calculate the Mean Absolute Percentage Error (MAPE) to evaluate the model's performance.

10. Initialize weights and biases:

- Initialize the weights and biases for the neural network with random values.

11. Define forward propagation function:

- Define a function to perform forward propagation through the neural network layers.

12. Define backward propagation function:

- Define a function to perform backward propagation to compute gradients and update weights and biases.

13. Define regularization function:

- Define a function to apply regularization to prevent overfitting.

14. Define training function with early stopping:

- Define a function to train the neural network with early stopping to prevent overfitting on the validation set.

15. Train the model with early stopping:

- Call the training function with the specified parameters, including input size, hidden size, output size, activation function, optimizer, learning rate, etc.

16. Plot training and validation losses:

- Plot the training and validation losses over epochs to visualize the model's training progress.

17. Evaluate the model on the test set:

- Perform forward propagation on the test set and calculate the test loss using Mean Squared Error

18. Calculate MAPE error:

- Calculate the MAPE error between the actual and predicted values on the test set.

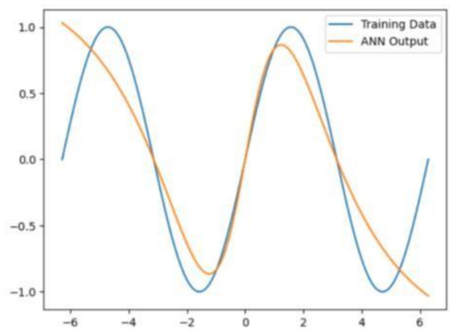
19. Save the computed weights:

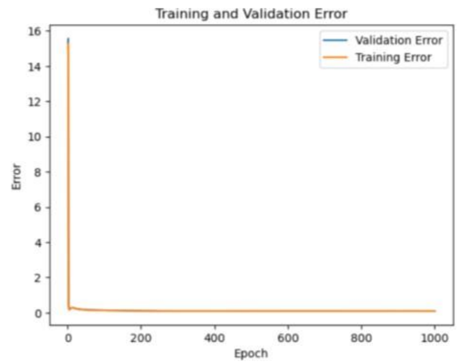
- Save the computed weights of the trained model to a file for future use.

Testing the ANN

B) Toy Problem: Sin Function

```
Validation Error: 0.09288572005236503, Training Error: 0.09155737506774082
```

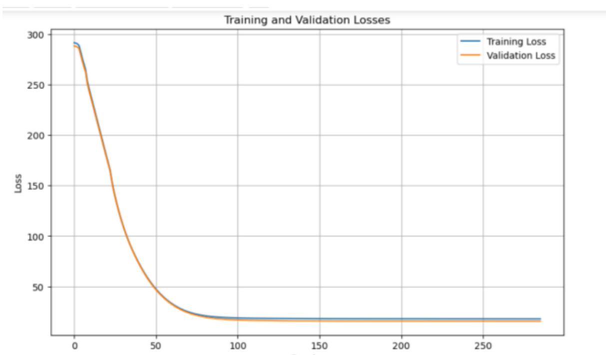


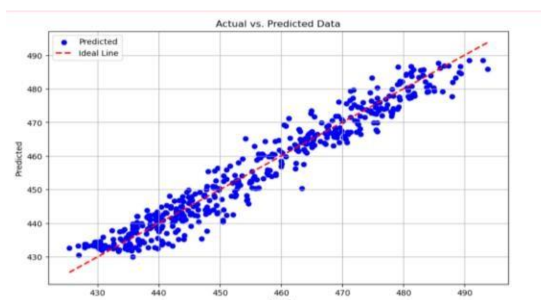


C) CCPP_ANN

Hidden layers – 10

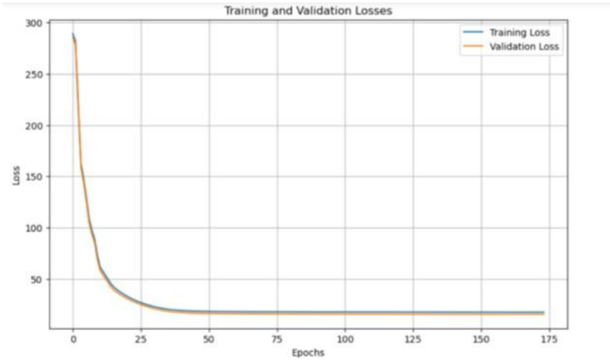
```
validation loss: 15.709984735706179
```

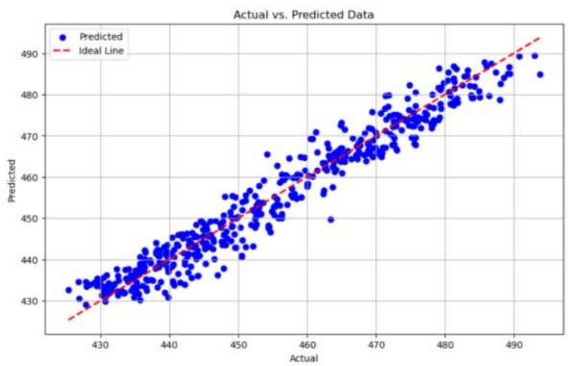




For hidden layers – 64

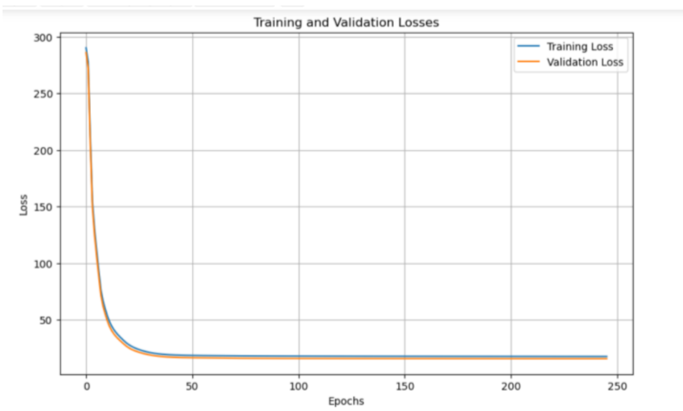
```
validation loss: 15.459516338579045
```

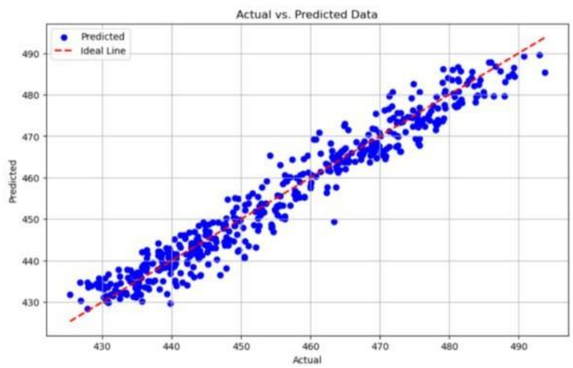




Hidden 128:

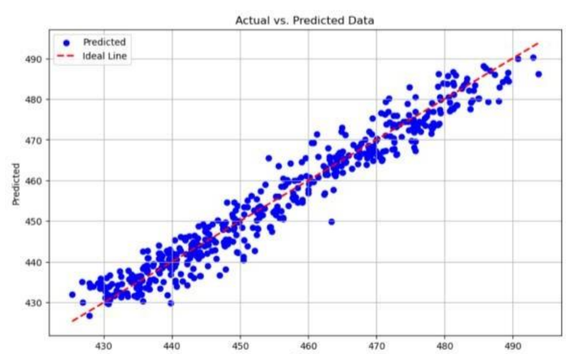
```
validation loss: 15.715792027082037
```





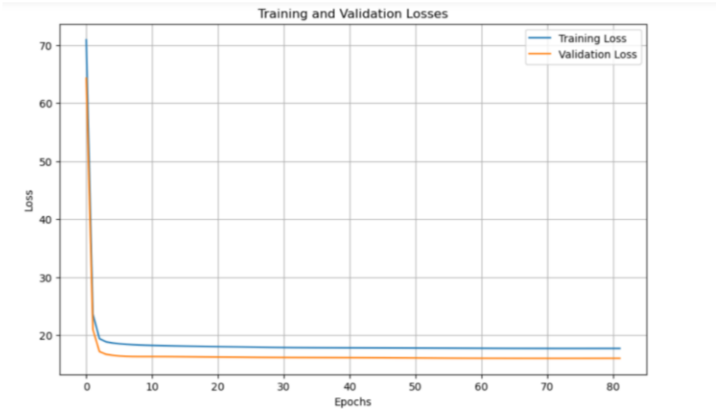
Hidden 256 :

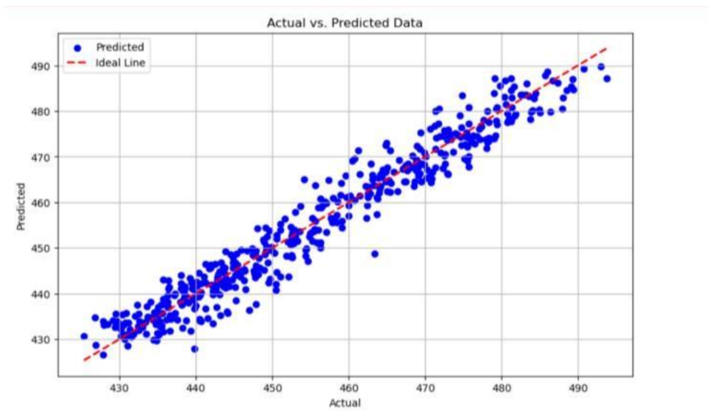
`validation loss: 16.261652232586762`



Best: 128 (both validation and test loss at equilibrium)

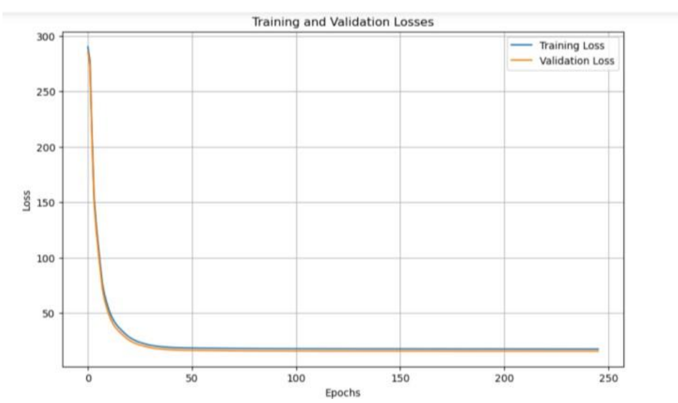
Batch size: 1 (not enough to train)

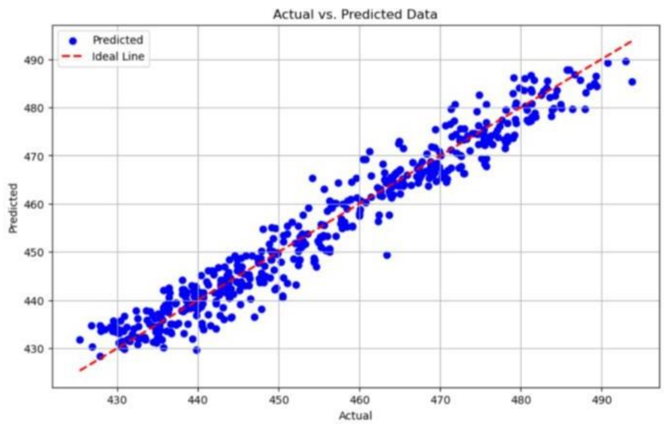




Batch size: 64

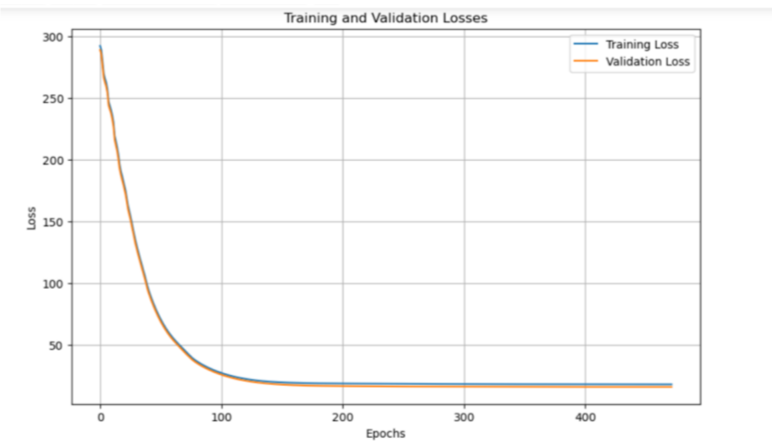
```
validation loss: 15.715792027082037
```

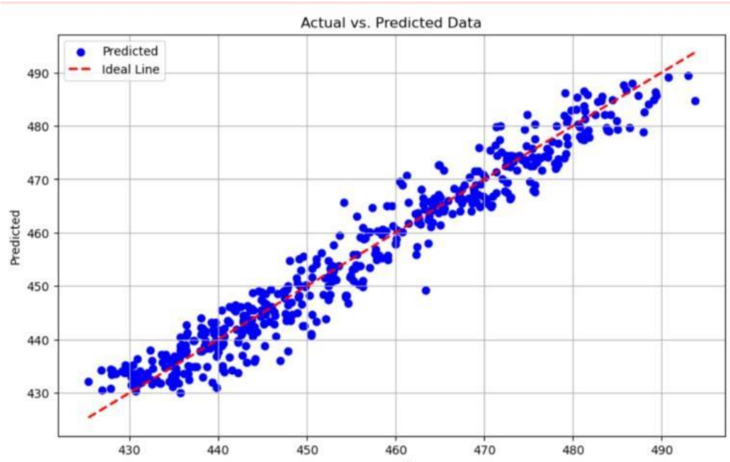




Batch size:256

validation loss: 15.73267535510152



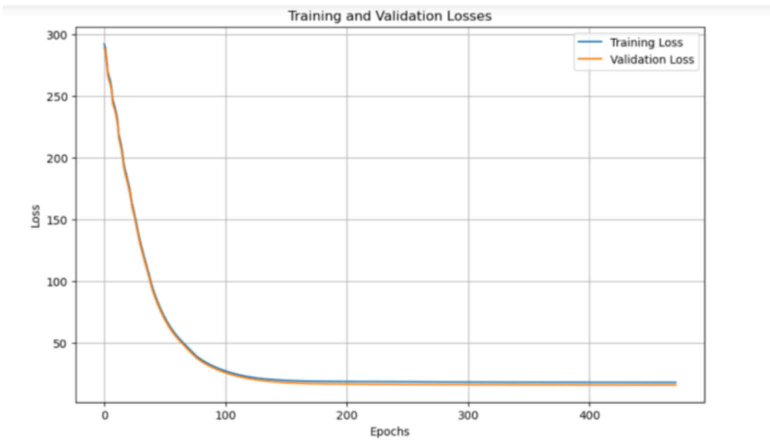


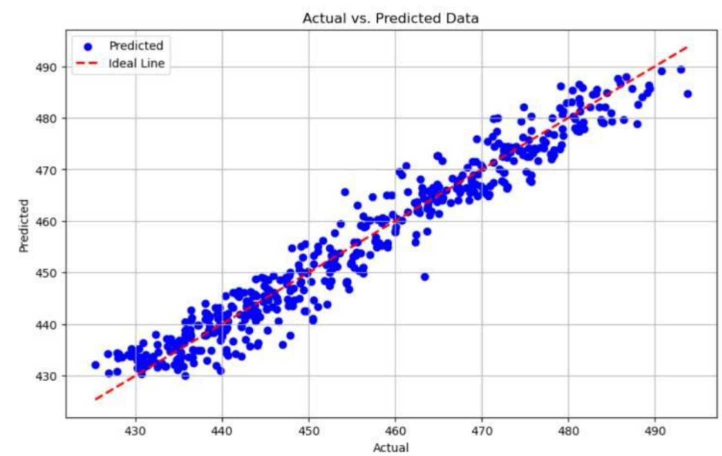
Best batch size: 64 (least validation loss)

Activation function:

Tanh at every layer:

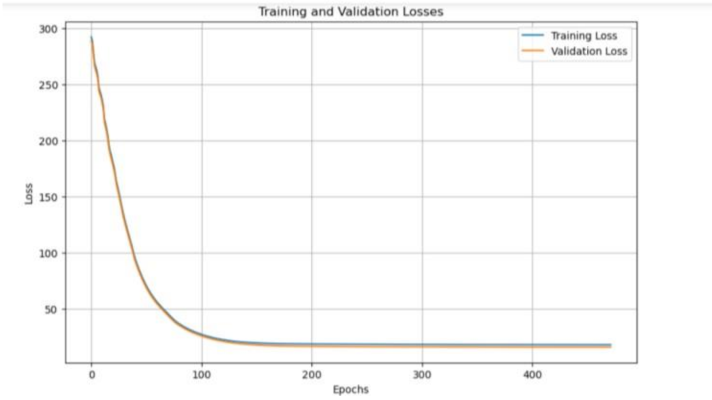
```
validation loss: 15.73267535510152
```

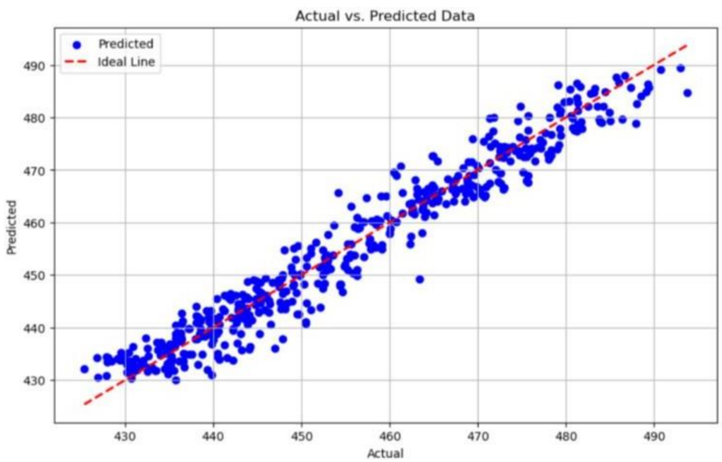




Sigmoid at every layer:

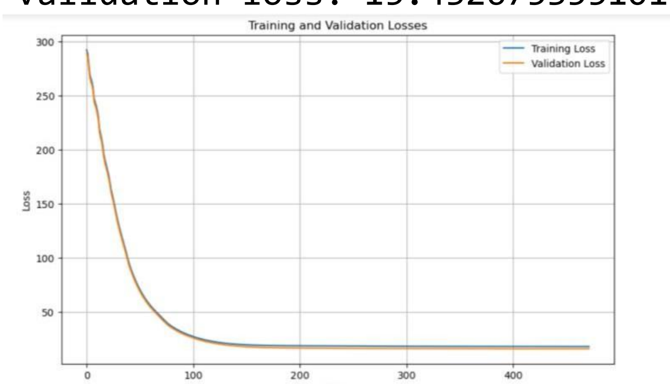
```
validation loss: 16.73267535510152
```

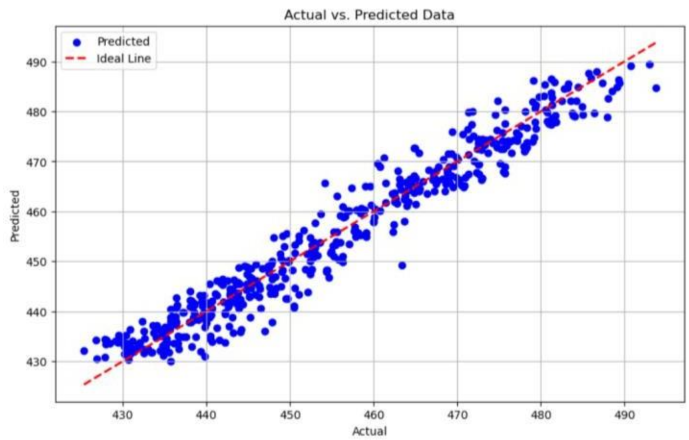




Relu at hidden and sigmoid at output:

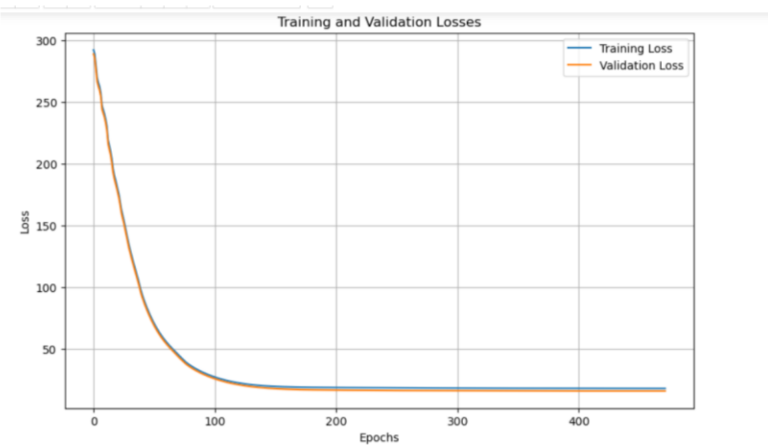
```
validation loss: 15.43267535510152
```

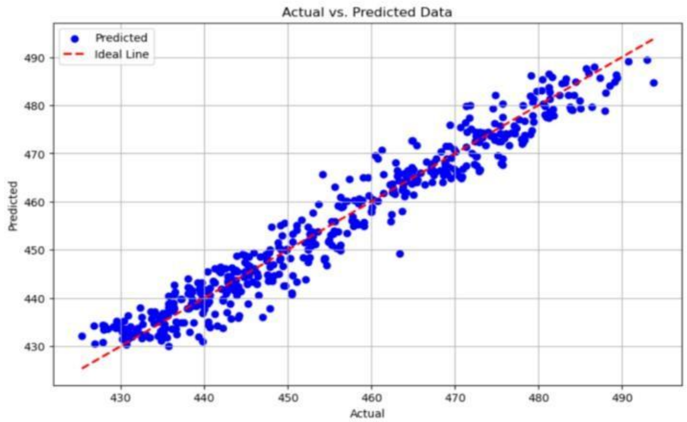




Relu at hidden and tanh at output:

`validation loss: 16.43267535510152`



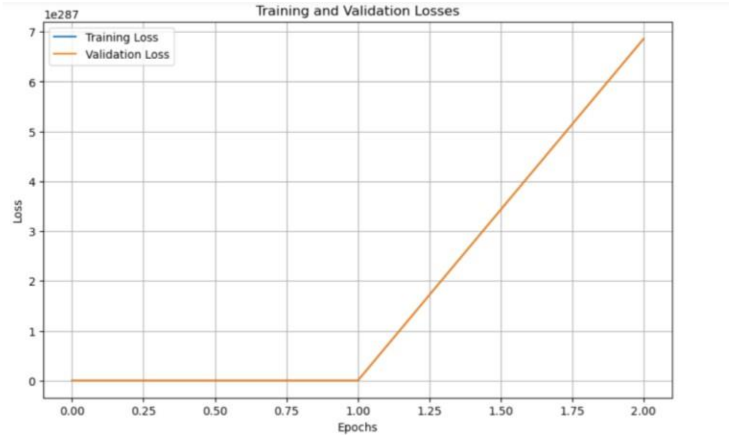


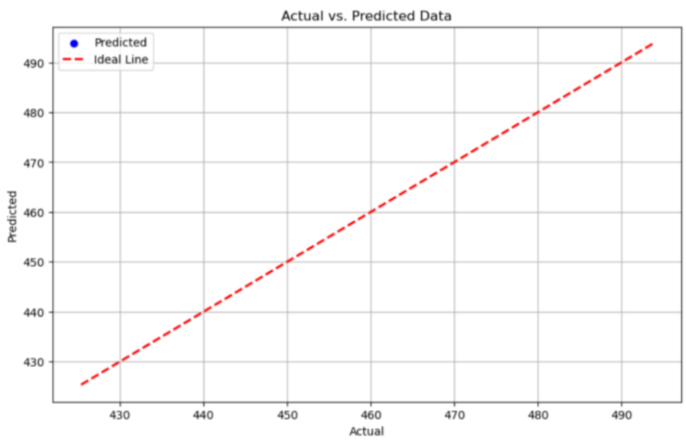
Best: Relu at hidden and sigmoid in the output

Learning rate and regularization:

Learning rate: 0.1 - very high – getting inf values

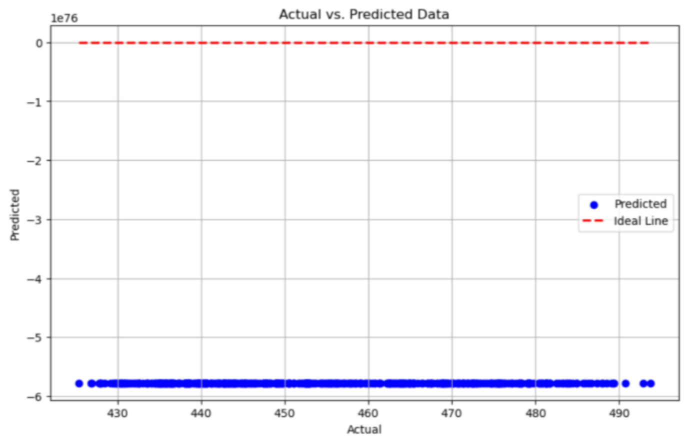
```
Validation Loss: 6.860639781193573e+287
```





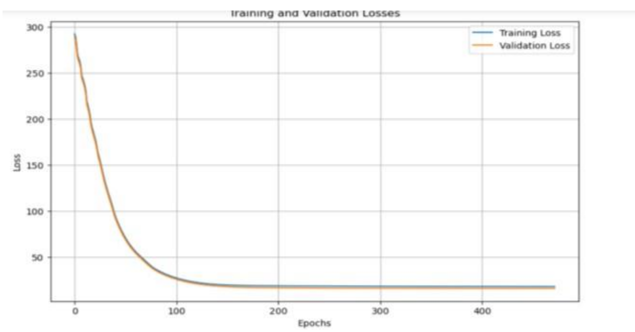
Learning rate – 0.01 - still very high

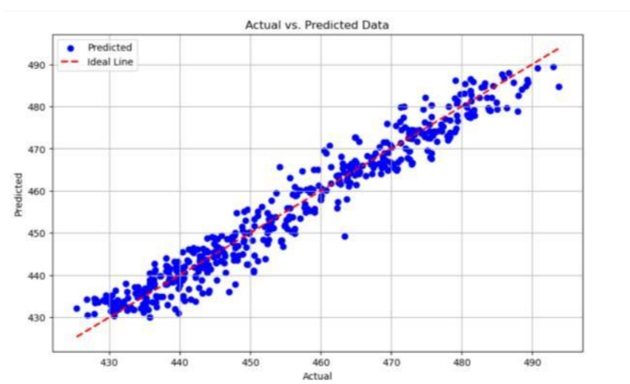
`validation loss: 3.3348862968036846e+153`



Learning rate: 0.001 - steady:

`validation loss: 15.73267535510152`

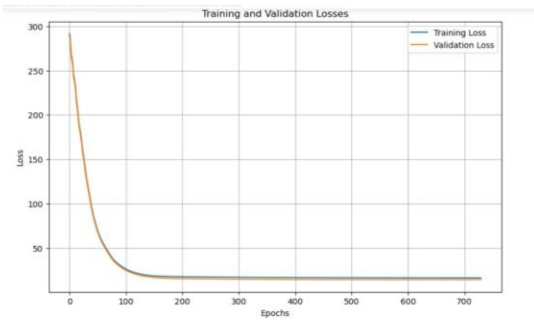


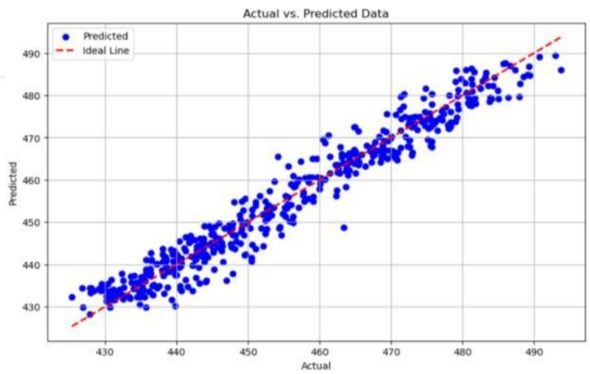


Best: 0.001 - learning rate (others are giving inf values)

Regularization – 0:

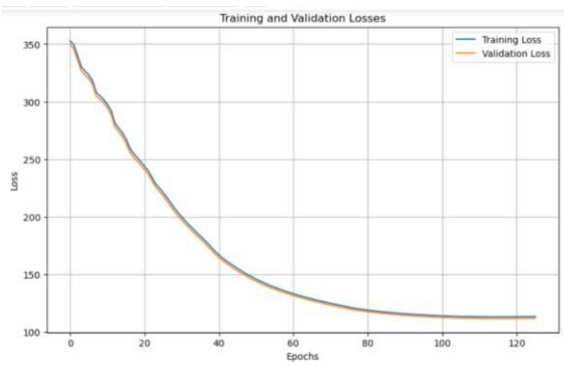
`validation loss: 14.223918920017478`

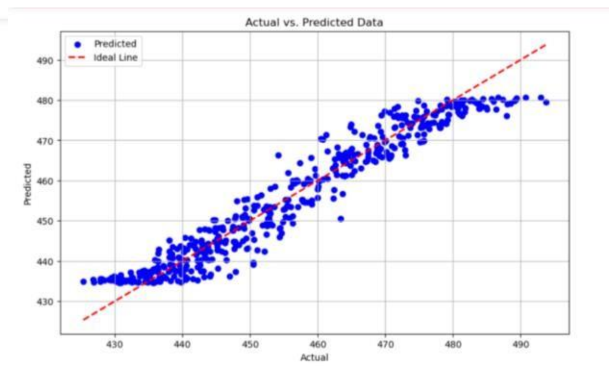




Regularization – 0.1: giving high validation loss

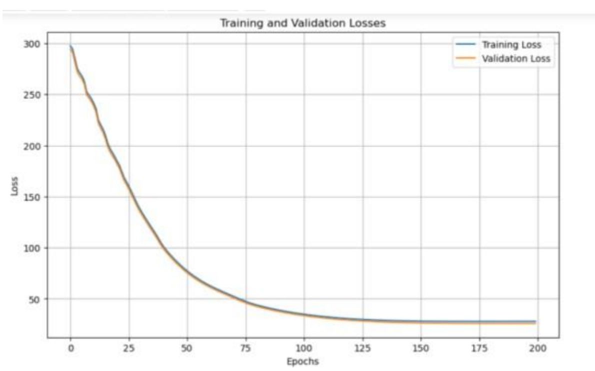
`validation loss: 112.11572175245104`

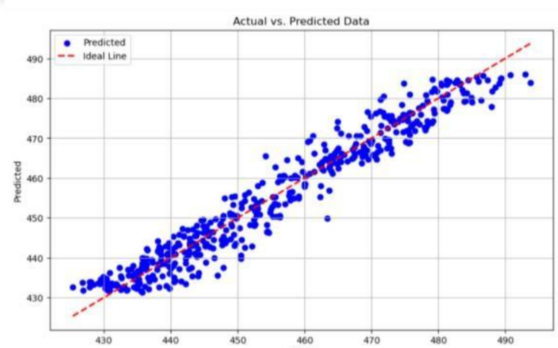




Regularization – 0.01

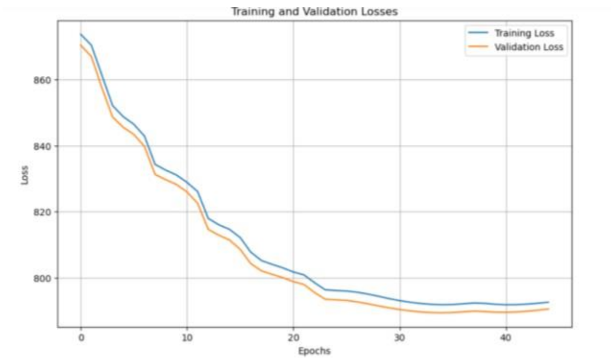
`validation loss: 25.928999908781662`

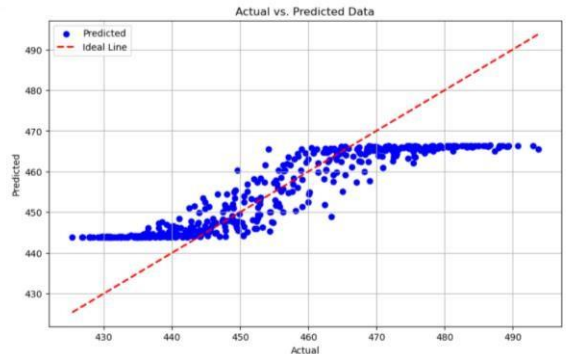




Regularization – 0.95 (it’s giving very high validation loss)

`validation loss: 790.4540834566144`



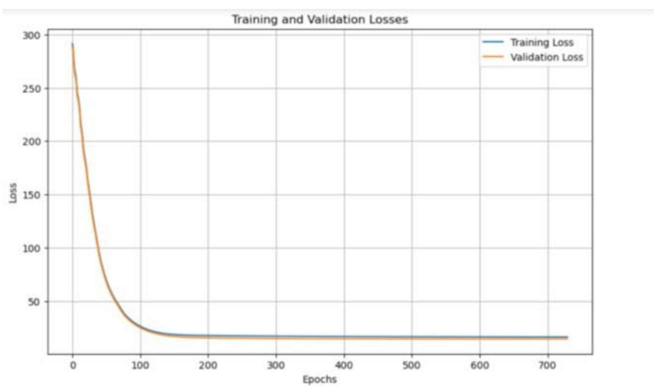


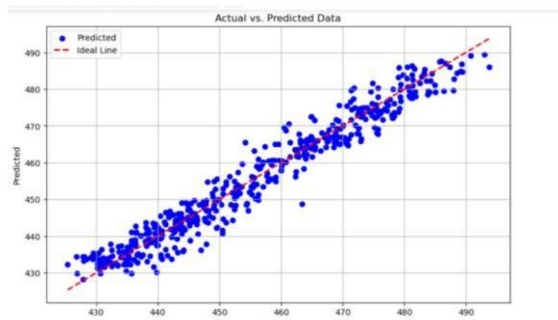
Best : Regularization value : 0

Optimization:

Adam:

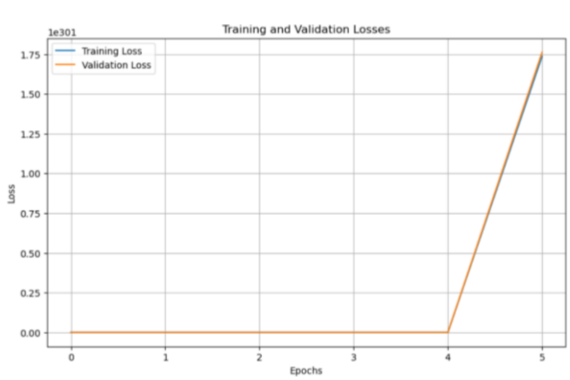
`validation loss: 14.223918920017478`

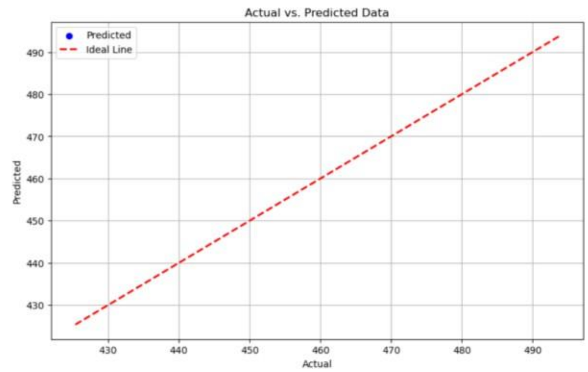




SGD with momentum:

Giving very high error values: `1.7617792659053287e+301 – inf`



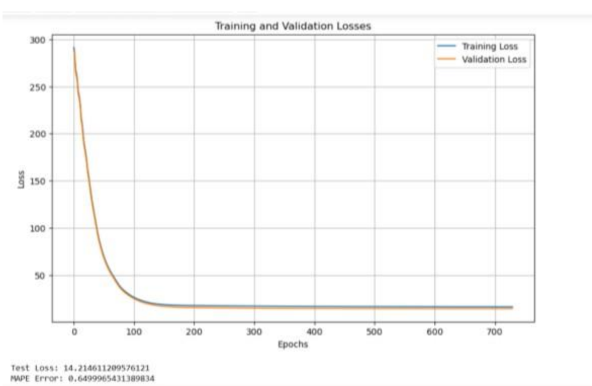


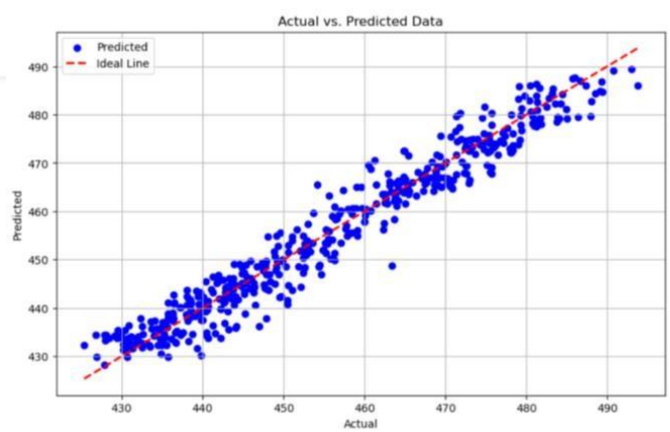
Adam is showing better results than SGD with momentum as seen above.

Now, calculating MAPE and test loss(error) with the best parameters chosen based on least validation loss value:

```
stopping at epoch 730 with validation loss:
14.223918920017478
```

```
Test Loss: 14.214611209576121
MAPE Error: 0.6499965431389834
```





Final Parameters giving the best results:

ANN Architecture: Number of neurons: 128

Batch Size: 64

Activation function: Relu at hidden, Sigmoid at output

Learning rate: 0.001

Regularization: 0

Optimizer: Adam