DEEP LEARNING REPORT

SURYAN PINNOJU – SE21UCSE273

SHREYAS PEYYETI – SE21UCSE204

B. KEERTHI VARDHAN VARMA – SE21UCSE100

TOY PROBLEM:

1. Introduction

This document focuses on the analysis of a custom-built Artificial Neural Network (ANN) algorithm designed to approximate the sine function. The main goal is to explore how varying key parameters such as learning rate, batch size, and network architecture affect the model's performance, measured using the Mean Absolute Percentage Error (MAPE). We will also provide a detailed explanation of the code and its functionality.

2. Algorithm Overview and Code Explanation

The provided ANN model is implemented in Python using NumPy. The core components of the code include forward propagation, backpropagation, and weight updates using momentum.

Key Elements in the Code:

a. Initialization (Weights, Biases, and Activation Functions):

- The ANN class initializes weights using (np.sqrt(2.0 / self.layers[i-1])), which helps in keeping the weights scaled properly, especially for deeper networks.

- The activation function is set during initialization. In this case, tanh was used, which outputs values between -1 and 1, fitting well with the sine function's range.

b. Forward Propagation:

- The forward function computes the output of the ANN for a given input. It iteratively applies the weight and bias transformations followed by the activation function across all layers.

-The network takes input data X. In the sine function it is a one dimensional array representing values between -2π and 2π

-For each layer, the output of the previous layer (vector a) is multiplied by the weights (W) and added to the biases(b): $Z = W.a + b$

-The weighted sum Z is then passed through the activation function tanh allowing nonlinearity.

c. Backpropagation:

- The backward function computes the gradients of the loss with respect to the weights and biases using the chain rule. Momentum is used to speed up the convergence.

- The loss is calculated using MSE.

- The error term is found by a simple subtraction of true value and predicted value. Using this error term the gradients of weights are calculated. These gradients tell us how much the weights should be adjusted to reduce the loss.

-The error at each layer is propagated backward and gradients for each layer are calculated. The weights are then updates using gradient descent with momentum

d. Training:

- In the train function, the model is trained over a set number of epochs. The data is shuffled to avoid overfitting to a specific sequence, and loss is calculated after each epoch to monitor performance.

e. Loss Calculation:

- The loss function includes a Mean Squared Error (MSE) term, which is common in regression tasks where continuous output is expected.

3. Parametric Variations and Their Impact on Output

This section discusses how changes in the model's hyperparameters—specifically learning rate, batch size, and network architecture—affected the sine function approximation results, focusing on MAPE scores.

Hyperparameters Studied:

- Learning Rate: Determines the step size for updating weights during training.

- Batch Size: Controls how many samples are processed before updating the weights.

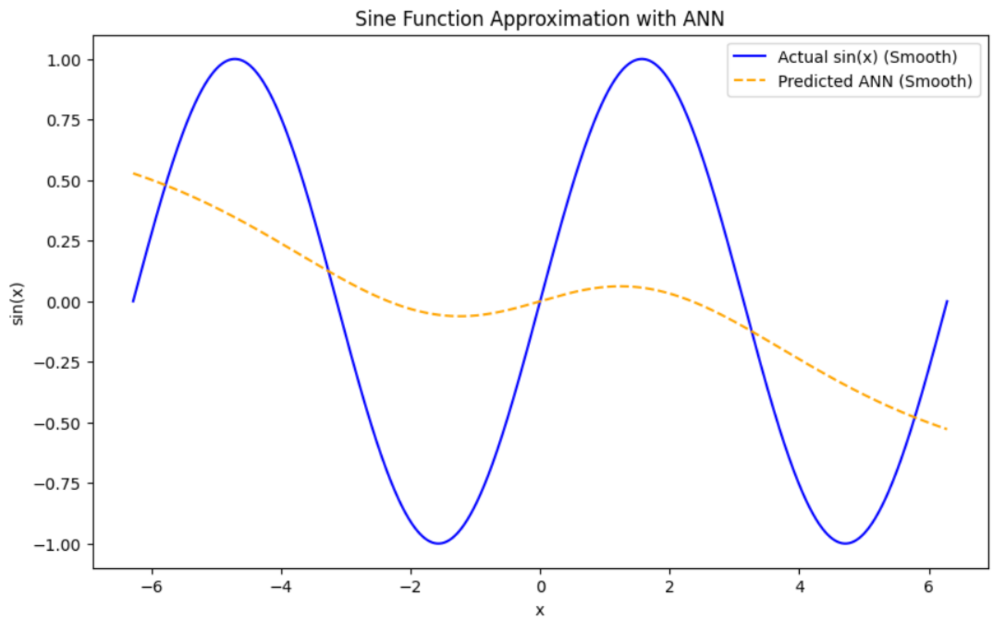
- Network Architecture: Refers to the number of neurons and layers in the network.

Effect of Learning Rate

Three values for the learning rate were tested: 0.001, 0.01, and 0.0001.

- Learning Rate = 0.001: This learning rate served as a baseline. The model converged relatively well, but certain architectures and batch sizes led to suboptimal performance, with MAPE scores ranging from 58% to 146%.

EX: Architecture = [1, 10, 1], Batch size = 256

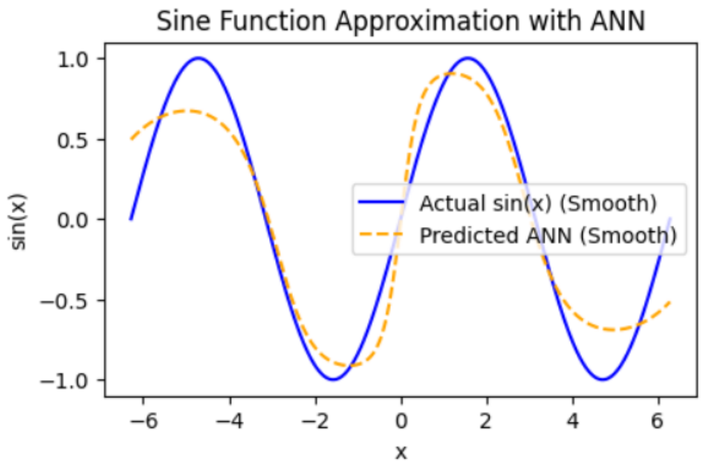


MAPE: 146.99436861096942%

- Learning Rate = 0.01: Increasing the learning rate accelerated convergence, but in some cases, it also led to instability, especially with larger batch sizes. However, smaller batch sizes (1 or 64) performed better, with the lowest MAPE of 47.21%.

EX: Architecture = [1, 7, 7, 7, 1], Batch size = 64

```
Epoch 0 - Loss: 0.4608645408007033
Epoch 100 - Loss: 0.07626033278593482
Epoch 200 - Loss: 0.06346847986696746
Epoch 300 - Loss: 0.06020670333442913
Epoch 400 - Loss: 0.056966092932070334
Epoch 500 - Loss: 0.05384828860125915
Epoch 600 - Loss: 0.05057118538356181
Epoch 700 - Loss: 0.04682106875320967
Epoch 800 - Loss: 0.043668888722929004
Epoch 900 - Loss: 0.04119414771010083
```



MAPE: 74.58551495954492%

- Learning Rate = 0.0001: The lower learning rate made the model converge slower but more steadily. In some architectures (e.g., [1, 7, 7, 7, 1]), it resulted in better approximation, but the MAPE was still higher compared to 0.01 for batch size 1.

Effect of Batch Size

Batch sizes 1, 64, 256, and Full Batch were tested. Smaller batch sizes generally led to better performance for approximating the sine function.

- Batch Size = 1: This led to the best overall performance in most cases. For example, with a learning rate of 0.01 and architecture [1, 7, 7, 7, 1], the MAPE was as low as 47.21%. This is because smaller batch sizes provide more frequent updates, which can be beneficial for complex functions like the sine wave.

- Batch Size = 64: This batch size produced varying results, often showing a balance between computational efficiency and performance. In some cases, such as with a learning rate of 0.01 in Architecture 3, it produced a low MAPE of 50.12%.

- Batch Size = 256 and Full Batch: Larger batch sizes resulted in higher MAPE scores. This may be due to less frequent updates, which prevented the model from capturing finer nuances in the sine wave data.

Effect of Network Architecture

Three architectures were tested:

- Architecture 1: [1, 10, 1]

  - This simple architecture showed moderate performance. While it produced some good results (MAPE = 69.11% with LR = 0.0001, Full Batch), it was outperformed by the deeper networks in most cases.

- Architecture 2: [1, 7, 7, 7, 1]

  - This deeper architecture with multiple hidden layers achieved the best overall performance. With LR = 0.01 and Batch Size = 1, it achieved the lowest MAPE of 47.21%. This demonstrates the importance of network depth for capturing the complexity of the sine function.

- Architecture 3: [1, 22, 22, 1]

  - This architecture, although deeper than Architecture 1, did not perform as well as Architecture 2. With a learning rate of 0.01 and batch size of 64, it achieved a MAPE of 50.12%, but this was still higher than the best-performing architecture.

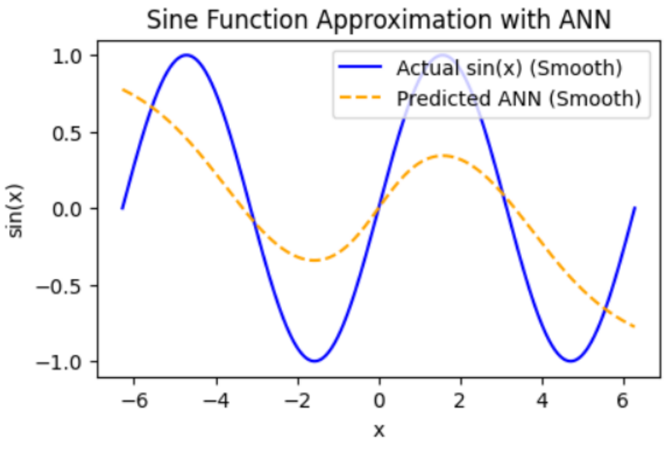
4. Comparative Analysis of Parametric Variations

The following table summarizes the best results from each architecture:

S.NO	Architecture	Learning Rate	Batch Size	MAPE (%)
1	[1, 10, 1]	0.0001	Full Batch	69.11
2	[1, 7, 7, 7, 1]	0.01	1	47.21
3	[1, 22, 22, 1]	0.01	64	50.12

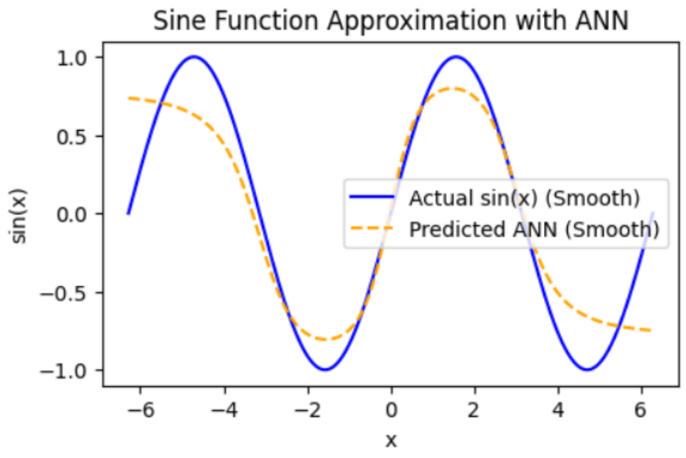
S.no 1:

```
Epoch 0 - Loss: 0.5736136525195926
Epoch 100 - Loss: 0.3799199210486416
Epoch 200 - Loss: 0.31412132565294126
Epoch 300 - Loss: 0.2751702645176859
Epoch 400 - Loss: 0.24937315051381267
Epoch 500 - Loss: 0.23181249937878326
Epoch 600 - Loss: 0.2197539774460401
Epoch 700 - Loss: 0.21136039072193752
Epoch 800 - Loss: 0.2054644648796201
Epoch 900 - Loss: 0.20121274271359377
```



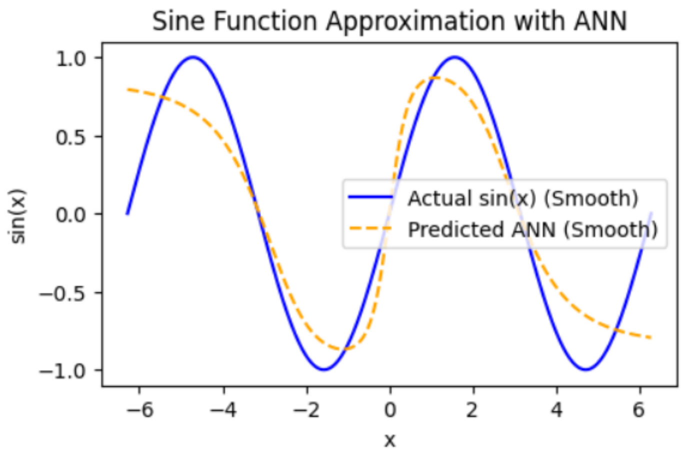
S.no 2:

```
Epoch 0 - Loss: 0.25380221497313904
Epoch 100 - Loss: 0.05000255821364198
Epoch 200 - Loss: 0.048269477726169845
Epoch 300 - Loss: 0.050820572136456625
Epoch 400 - Loss: 0.0498113335193665
Epoch 500 - Loss: 0.051974839621585875
Epoch 600 - Loss: 0.04870919552103064
Epoch 700 - Loss: 0.04850900854507149
Epoch 800 - Loss: 0.051764278730223
Epoch 900 - Loss: 0.04972700096765368
```



S.no 3:

```
Epoch 0 - Loss: 0.5860535866639662
Epoch 100 - Loss: 0.10636732326220256
Epoch 200 - Loss: 0.07874456035357347
Epoch 300 - Loss: 0.07504253821413798
Epoch 400 - Loss: 0.07271556479112003
Epoch 500 - Loss: 0.07108997915766331
Epoch 600 - Loss: 0.06924162321276421
Epoch 700 - Loss: 0.06779248238793535
Epoch 800 - Loss: 0.06639526670368705
Epoch 900 - Loss: 0.06499426436138456
```

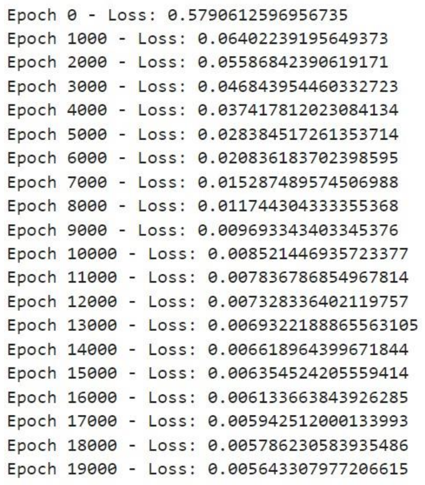


5. Additional Observations

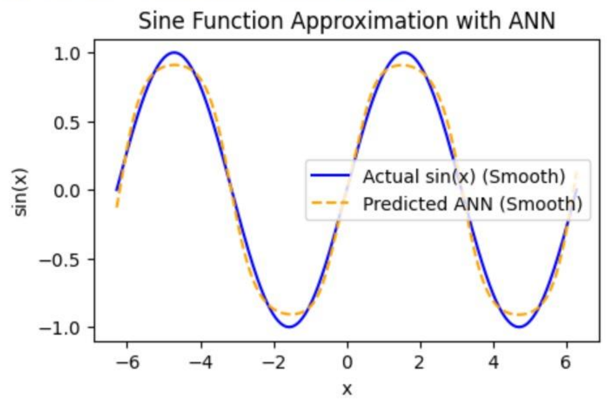
We've attempted to improve the performance by extending the training duration for the best-performing architectures, Architecture 2 ([1, 7, 7, 7, 1]) and Architecture 3 ([1, 22, 22, 1]) and ran them for 20,000 epochs. The goal was to achieve a lower Mean Absolute Percentage Error (MAPE) score.

Results:

Architecture 3 ([1, 22, 22, 1]) showed the most significant improvement, with a MAPE score of 17.21792 achieved using a batch size of 256 and learning rate of 0.01.

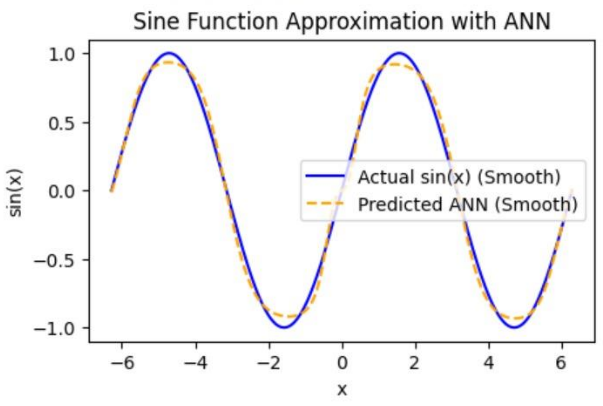


```
Epoch 0 - Loss: 0.5790612596956735
Epoch 1000 - Loss: 0.06402239195649373
Epoch 2000 - Loss: 0.05586842390619171
Epoch 3000 - Loss: 0.046843954460332723
Epoch 4000 - Loss: 0.037417812023084134
Epoch 5000 - Loss: 0.028384517261353714
Epoch 6000 - Loss: 0.020836183702398595
Epoch 7000 - Loss: 0.015287489574506988
Epoch 8000 - Loss: 0.011744304333355368
Epoch 9000 - Loss: 0.009693343403345376
Epoch 10000 - Loss: 0.008521446935723377
Epoch 11000 - Loss: 0.007836786854967814
Epoch 12000 - Loss: 0.007328336402119757
Epoch 13000 - Loss: 0.0069322188865563105
Epoch 14000 - Loss: 0.006618964399671844
Epoch 15000 - Loss: 0.006354524205559414
Epoch 16000 - Loss: 0.006133663843926285
Epoch 17000 - Loss: 0.005942512000133993
Epoch 18000 - Loss: 0.005786230583935486
Epoch 19000 - Loss: 0.005643307977206615
```



Architecture 2 ([1, 22, 22, 1]) also showed significant improvement, with a MAPE score of 29.32121% achieved using a batch size of 256 and learning rate of 0.01.

```
Epoch 0 - Loss: 1.135987270156584
Epoch 1000 - Loss: 0.05434471719144959
Epoch 2000 - Loss: 0.04272821459219059
Epoch 3000 - Loss: 0.029071565791783226
Epoch 4000 - Loss: 0.015520598080636189
Epoch 5000 - Loss: 0.008900868852400382
Epoch 6000 - Loss: 0.007258172289555242
Epoch 7000 - Loss: 0.006474370883209387
Epoch 8000 - Loss: 0.0060511278728593715
Epoch 9000 - Loss: 0.005798464109848458
Epoch 10000 - Loss: 0.0055775951563607476
Epoch 11000 - Loss: 0.005325784672316461
Epoch 12000 - Loss: 0.005046053030889282
Epoch 13000 - Loss: 0.004789075874114148
Epoch 14000 - Loss: 0.004583335844857078
Epoch 15000 - Loss: 0.004429584456854127
Epoch 16000 - Loss: 0.004314976780899577
Epoch 17000 - Loss: 0.004224578121672205
Epoch 18000 - Loss: 0.004145635938608995
Epoch 19000 - Loss: 0.004073621092147597
```



This indicates that extended training, along with an optimized architecture and hyperparameters (particularly a learning rate of 0.01 and smaller batch sizes), allowed the network to generalize better, significantly reducing the prediction error for the sine function. These results highlight the importance of careful hyperparameter tuning and sufficient training epochs for achieving optimal model performance.

6. Further Experimentation with Wider Networks:

In addition to extending the training epochs, we experimented with wider networks, such as the architecture [1, 25, 12, 9, 1], to see if increasing the model's capacity would further improve performance.

Results:

After running this architecture for 20,000 epochs, we achieved a MAPE score of 13.41%, which is the best score achieved so far.

However, further experimentation with this wider architecture, including additional hyperparameter tuning (batch size, learning rate, etc.), did not yield better results. The performance seemed to plateau, suggesting that simply adding more neurons and layers does not always lead to better accuracy.

This experiment underscores that while increasing network width can lead to improvements in some cases, there is a limit beyond which additional complexity might not contribute to better performance. Careful balancing of architecture depth, width, and hyperparameters is crucial for achieving optimal results.

COMBINED CYCLE POWER PLANT DATASET:

1. Introduction

This project aims to analyze the performance of different architectures of Artificial Neural Networks (ANN) on the Combined Cycle Power Plant Dataset. We experimented with multiple activation functions (tanh, ReLU, and sigmoid) and architectures to identify the most suitable model configuration. The results are evaluated using Mean Absolute Percentage Error (MAPE) on validation and test datasets.

2. Dataset Overview

The Combined Cycle Power Plant dataset contains 9,568 data points collected from a power plant under full load. The features include ambient temperature, ambient pressure, relative humidity, and exhaust vacuum, while the target variable is the net hourly electrical energy output (PE).

Input Features (X): Ambient Temperature (AT), Exhaust Vacuum (V), Ambient Pressure (AP), and Relative Humidity (RH).

Target (y): Electrical Energy Output (PE).

We split the dataset into 72% training, 18% validation, and 10% testing.

3. Model Architecture and Setup

We experimented with the following architectures and activation functions:

Activation Functions: Tanh, ReLU, Sigmoid.

Model Layers: Various architectures with different hidden layer configurations.

Optimization Algorithm:

Gradient descent with momentum (β=0.9) and L2 regularization (λ=0.0001) was used for training.

4. Parametric Variations and Results

4.1 Tanh Activation Function:

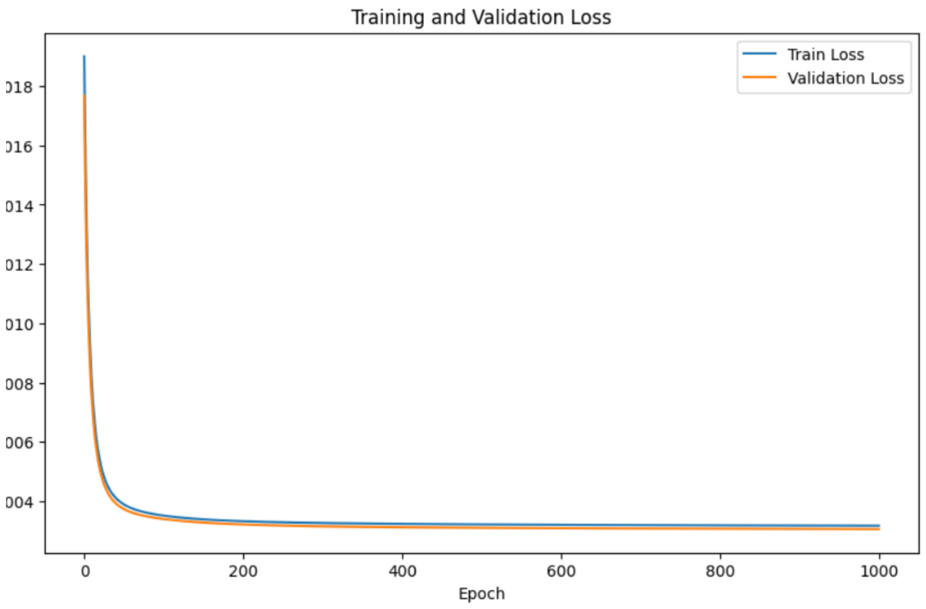
The tanh function was tested with different architectures and batch sizes. The results were as follows:

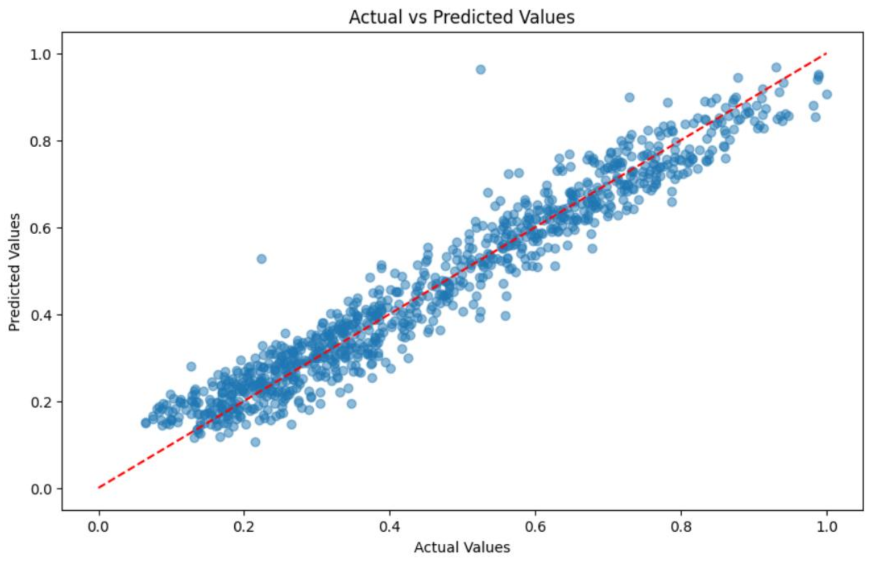
Architecture: (4, 40, 40, 40, 1)

Batch Size: 64

Learning Rate: 0.0005

Performance:





Validation MAPE: 13.86

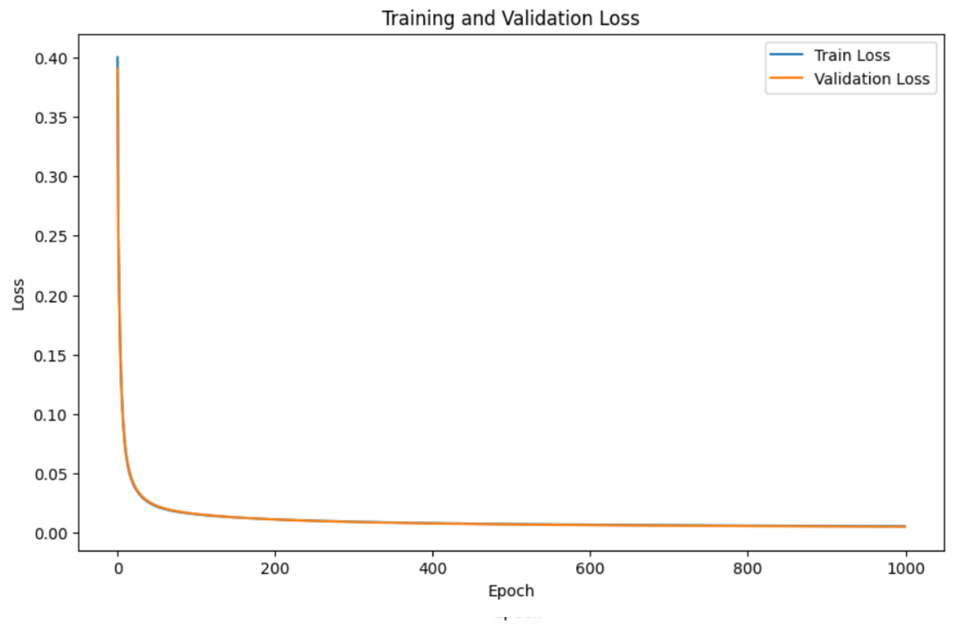
Test MAPE: 13.53

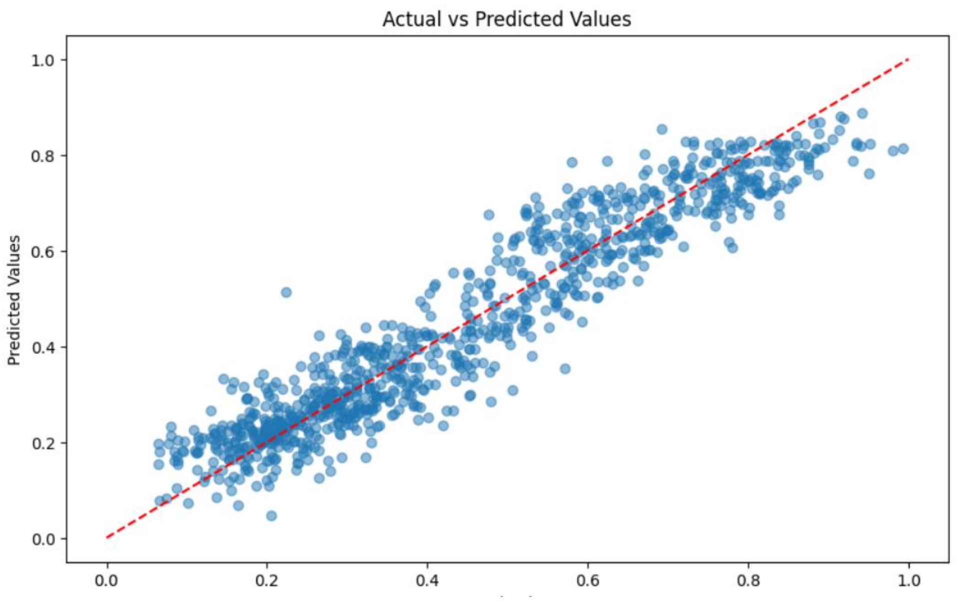
Architecture: (4, 40, 40, 40, 1)

Batch Size: 256

Learning Rate: 0.0001

Performance:





Validation MAPE: 14.91

Test MAPE: 12.77

4.2 ReLU Activation Function

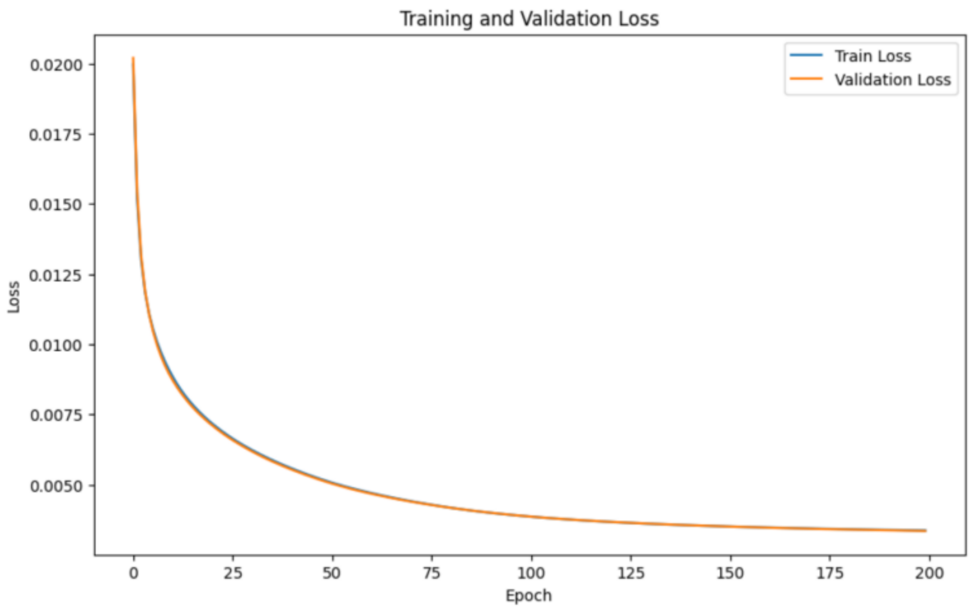
ReLU was also tested with two different architectures, showing a significant variation in performance.

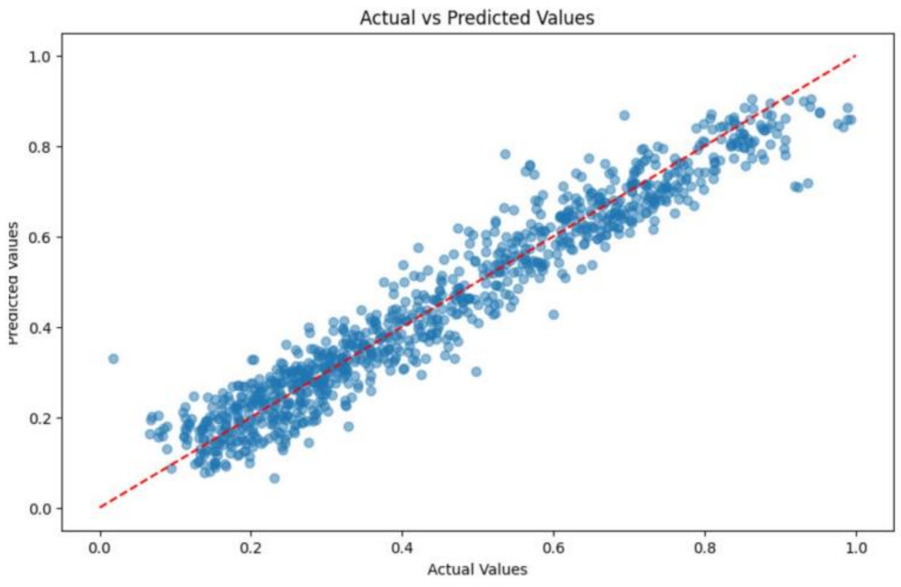
Architecture 1: (4, 64, 32, 16, 1)

Batch Size: 64

Learning Rate: 0.0005

Performance:





Validation MAPE: 14.29

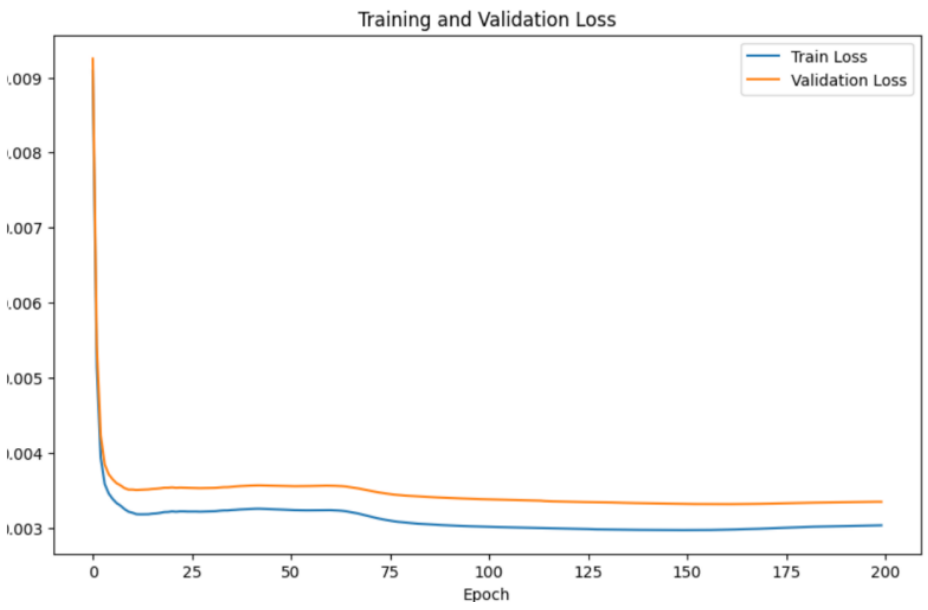
Test MAPE: 16.12

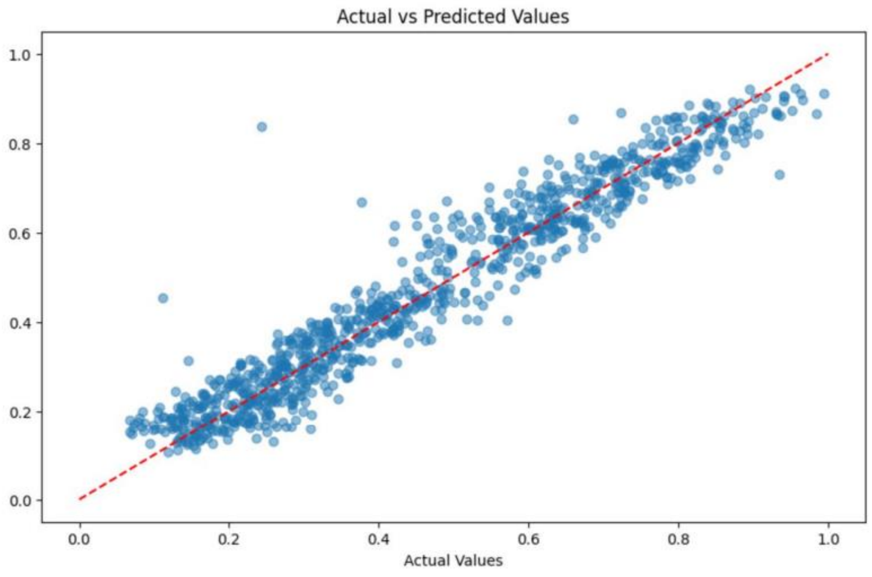
Architecture 2: (4, 16, 16, 1)

Batch Size: 64

Learning Rate: 0.0005

Performance:





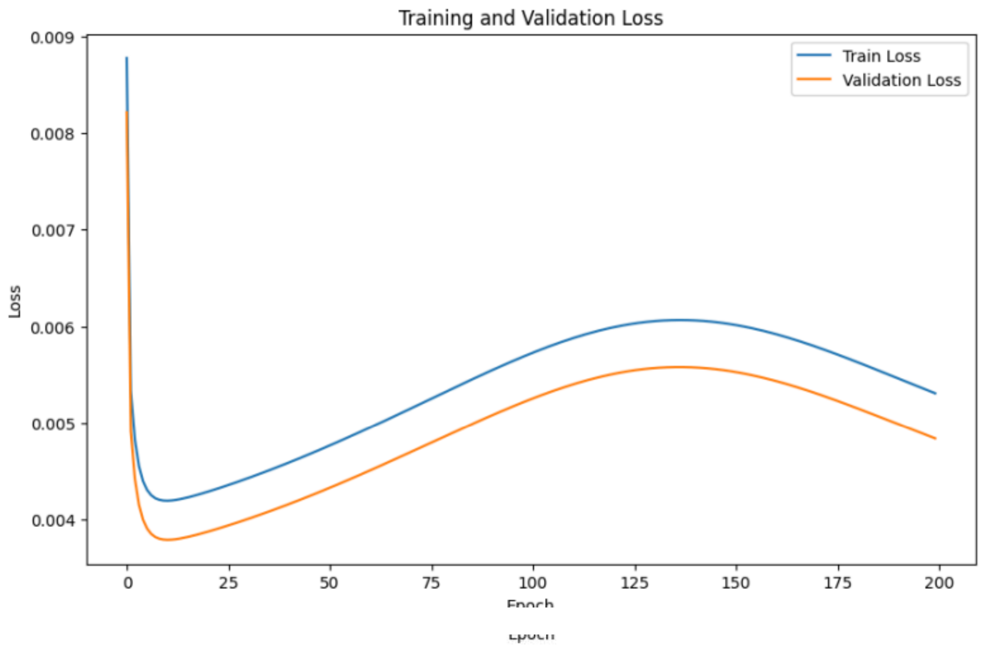
Validation MAPE: 14.25

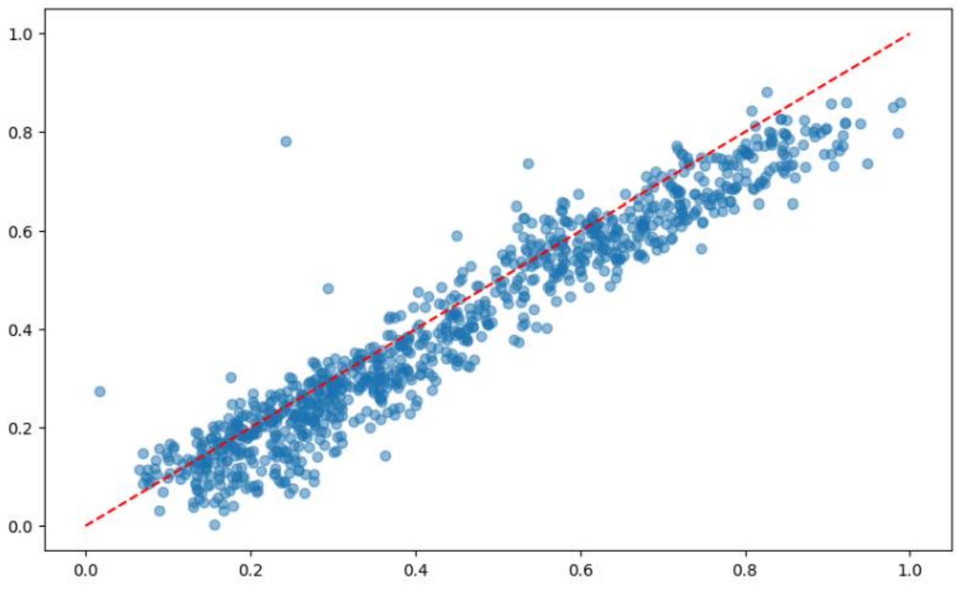
Test MAPE: 14.47

4.3 Sigmoid Activation Function

The sigmoid function was tested with three different architectures, yielding the following results:

Config 1: (4, 100, 1)

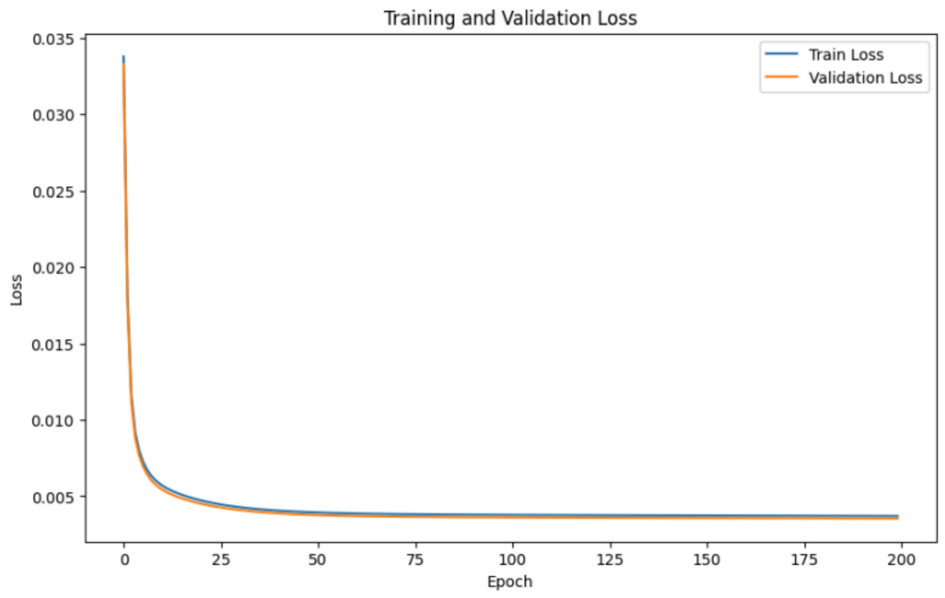


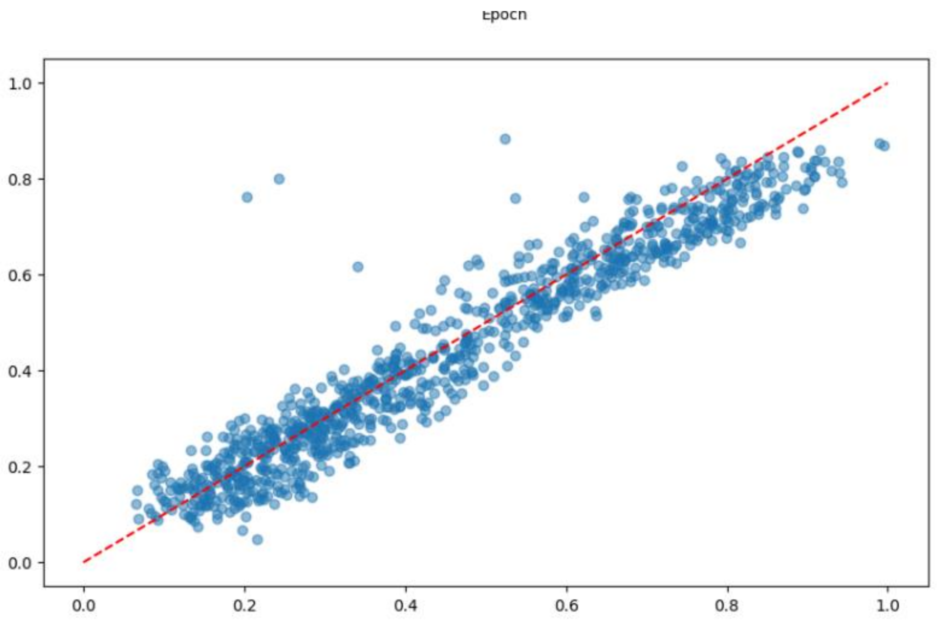


Validation MAPE: 14.31

Test MAPE: 14.27

Config 2: (4, 60, 20, 1)





Validation MAPE: 13.81

Test MAPE: 14.19

5. Comparisons and Observations

5.1 Learning Rate and Batch Size:

For both tanh and ReLU functions, using a batch size of 64 and a learning rate of 0.0005 provided stable results. For sigmoid, the best performance was observed with Config 3 (4, 60, 20, 1), where the validation MAPE was 13.81% and the test MAPE was 14.19%.

5.2 Activation Functions:

ReLU: The ReLU activation function did not yield the best results in this case. The architectures tested did not provide the desired low MAPE values.

Tanh: showed better performance than tanh but still had limitations due to the chosen architecture.

Sigmoid: Sigmoid performed relatively well with Config 3, achieving a test MAPE of 14.19%, which was the lowest error among all configurations.

5.3 Architecture Design:

Deeper architectures (4, 40, 40, 40, 1) for tanh did not show promising results.

Moderate-sized architectures like (4, 60, 20, 1) for sigmoid showed better generalization with the lowest test MAPE.

6. Best Performing Model

Based on the experiments, the best-performing model was (4, 40, 40, 40 1) using the tanh activation function, with batch size 64 and learning rate 0.0005. It achieved:

Validation MAPE: 13.86%

Test MAPE: 13.41%

There was another model with architecture (4, 40, 40, 40, 1) using the tanh activation function, batch size 256 and learning rate 0.0001 that performed well with test mape being 12.77%, but did not perform as well with the validation dataset with the validation mape being 14.91%.

7. Conclusion

In this analysis, we experimented with multiple architectures and activation functions (tanh, ReLU, and sigmoid). The sigmoid activation function with architecture (4, 60, 20, 1) provided the best performance with the lowest test MAPE. The choice of architecture, learning rate, and batch size significantly impacted the model’s performance.

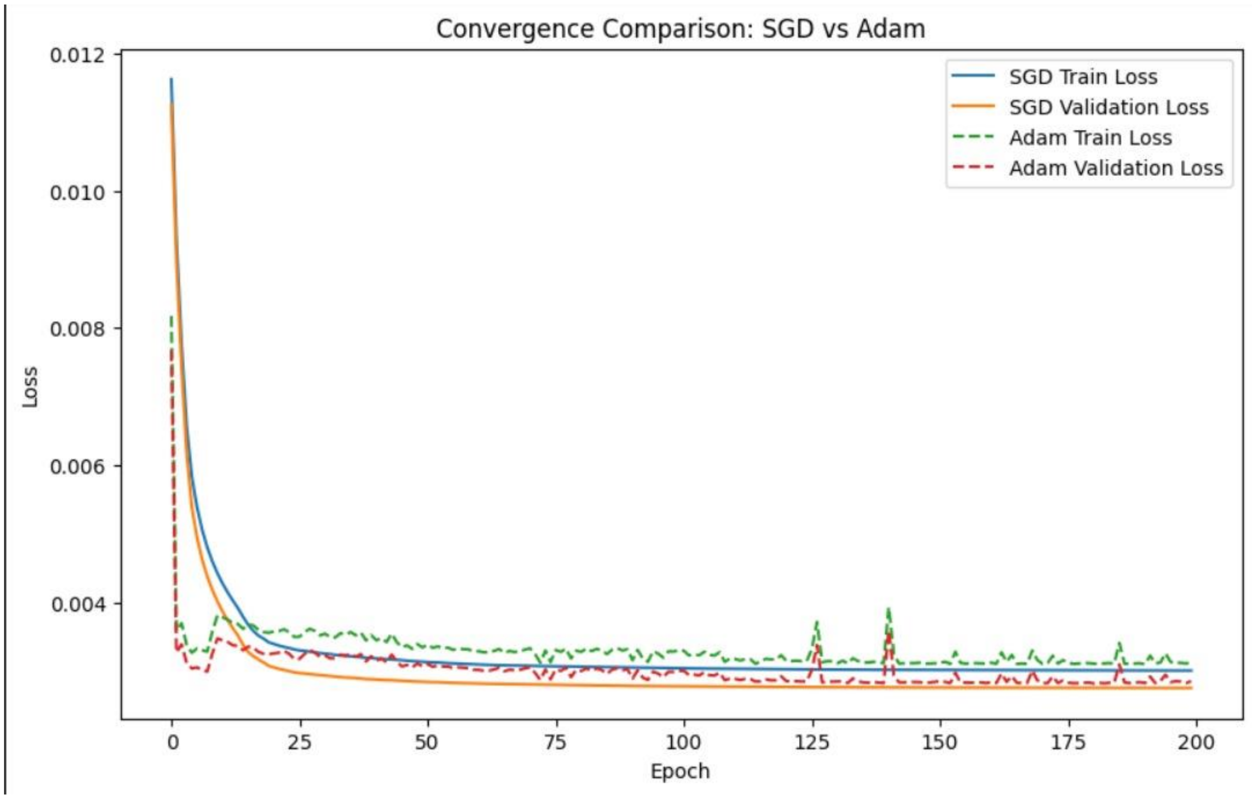
BONUS QUESTION:

SGD with Momentum vs Adam

Objective:

The experiment aimed to compare the performance of two optimizers, SGD with Momentum and Adam, when manually implemented in a neural network. The comparison focused on their effectiveness in minimizing prediction error, evaluated using the Mean Absolute Percentage Error (MAPE) metric.

Results:



SGD with Momentum:

Validation MAPE: 13.43%

Test MAPE: 13.07%

Adam:

Validation MAPE: 13.88%

Test MAPE: 13.39%

Analysis:

SGD with Momentum performed slightly better overall, achieving a lower Test MAPE (13.07%) compared to Adam's Test MAPE (13.39%). This suggests that SGD with momentum provided a more stable convergence and better generalization in this case.

Adam, while typically known for faster convergence, resulted in a Validation MAPE of 13.88%, indicating that it may not have generalized as well as SGD in this scenario.

Conclusion:

Both optimizers delivered comparable results, but SGD with Momentum showed a slight advantage in this experiment. This outcome highlights the importance of experimenting with different optimizers, as the performance can vary based on the specific dataset and model configuration. Further hyperparameter tuning and adjustments could enhance Adam's performance, which is generally favored for its adaptive learning capabilities.