A Project Report

on

Building an ANN and Experimenting with Multiple Tools

by

Bhuvana Kanakam – SE21UCSE035
Rachana Vinjamur - SE21UCSE159
Sanjana Kusupudi - SE21UCSE188

Professor

Dr. Arya Kumar Bhattacharya

CS4157: Deep Learning Assignment 1

ECOLE CENTRALE SCHOOL OF ENGINEERING

MAHINDRA UNIVERSITY

Hyderabad, India

Contents

List of Figures

Chapter 1

Step-by-Step Backpropagation Algorithm

Backpropagation is a supervised learning algorithm used for training artificial neural networks (ANNs). It efficiently computes the gradient of the loss function with respect to each weight in the network using the chain rule, allowing for weight adjustments that minimize the error between predicted output and actual target values.

The backpropagation algorithm operates in two main phases: the **forward pass** and the **backward pass**. These steps are repeated over multiple epochs until the network error converges to an acceptable level.

1. **Forward Pass**: Compute the output layer's predictions.

2. **Backward Pass**: Propagate the error backward to update the weights.

3. **Repeat**: Continue this process over many epochs to train the model.

Step-by-Step Process

1.1 Forward Pass

The forward pass computes the output for each neuron by propagating the input through the network.

For each layer l:

$$V^{(l)} = W^{(l)}Y^{(l-1)} + b^{(l)}$$

Where,

- $V^{(l)}$ is the input to layer l,
- $W^{(l)}$ is the weight matrix for layer l,
- $Y^{(l-1)}$ is the activation output from the previous layer, and
- $b^{(l)}$ is the bias term for layer l.

Activation Function:

$$Y^{(l)} = \phi(V^{(l)})$$

Where ϕ is typically a sigmoid, tanh, or ReLU activation function, which produces the activation output for layer l.

1.2 Calculate the Error at the Output Layer

At the output layer, the algorithm compares the predicted output with the actual target values to compute the error.

Error Signal:

$$e_j(n) = d_j(n) - y_j(n)$$

Where

- $d_j(n)$ is the actual target value for neuron j at sample n, and
- $y_j(n)$ is the predicted output for neuron j.

Instantaneous Error Energy:

$$E_j(n) = \frac{1}{2} e_j(n)^2$$

The total error across all output neurons is:

$$E_{\text{total}} = \sum_j E_j(n)$$

1.3 Backward Pass

The backward pass calculates the gradient of the loss function with respect to each weight using gradient descent and updates the weights accordingly.

Step 1: Calculate Local Gradients for the Output Layer

For output neuron j, the local gradient $\delta_j(n)$ is computed as:

$$\delta_j(n) = \phi'(V_j(n)) \cdot e_j(n)$$

Where $\phi'(V_j(n))$ is the derivative of the activation function with respect to the input V_j, and $e_j(n)$ is the error signal at neuron j.

Step 2: Update Weights for the Output Layer

Using the calculated gradients, update the weights connecting the previous layer's output to the output layer:

$$\Delta W_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n)$$

Where

- η is the learning rate, and
- $y_i(n)$ is the output from the previous layer.

Then update the weights:

$$W_{ji}(n+1) = W_{ji}(n) + \Delta W_{ji}(n)$$

1.4 Calculate Local Gradients for the Hidden Layers

For hidden neurons, calculate the gradients recursively, starting from the output layer and moving backward.

For hidden neuron j:

$$\delta_j(n) = \phi'(V_j(n)) \cdot \sum_k \delta_k(n) W_{kj}(n)$$

Where

- $\delta_k(n)$ is the gradient for neuron k in the next layer, and
- $W_{kj}(n)$ is the weight connecting neuron j in the hidden layer to neuron k in the next layer.

Update Weights for Hidden Layers

Similar to the output layer, update the weights for the hidden layers:

$$\Delta W_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n)$$

Then update the weights:

$$W_{ji}(n+1) = W_{ji}(n) + \Delta W_{ji}(n)$$

1.5 Training (Epochs)

An epoch consists of passing all training samples through the network, performing forward and backward passes, and updating the weights.

Shuffle Data: It is good practice to shuffle the training data in every epoch to ensure the model generalizes well.

Chapter 2

Toy Problem - the sin(x) curve

2.1 Problem Statement

In this programming assignment, I developed an Artificial Neural Network (ANN) using backpropagation techniques to learn the function $y = \sin(x)$ over the domain $-2\pi \leq x \leq 2\pi$. I first generated 1000 training pairs of (x, y) values and validated the model using 300 randomly extracted points from the same range. The ANN architecture included a single hidden layer, with the number of neurons adhering to the rule of thumb that the number of unknowns should not exceed half the training samples. I implemented various activation functions, including tanh and logistic functions, and performed data normalization to enhance model accuracy. Additionally, I explored the effects of different mini-batch sizes on training convergence and evaluated the model's performance on a Combined Cycle Power Plant dataset to demonstrate the ANN's ability to capture complex functional relationships. This assignment provided hands-on experience in implementing core machine learning principles and highlighted the importance of effective data management and algorithm optimization in developing neural networks.

2.2 Best Model for the Toy Problem

```
Fixed Parameters:
    - Minibatch Size: 64, No. of Epochs: 4000
    - Cost Function = "mse"
    - Regularisation = "L2", lambda = 0.25
    - Optimizeer = "adam", beta = 0.9, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8
    - Activation Function : "tanh"
    - Architecture: [1,30,30,1]
```

TABLE 2.1: Best model Results

Validation mape	36.48907240711003
r2 score	0.9958321185218095

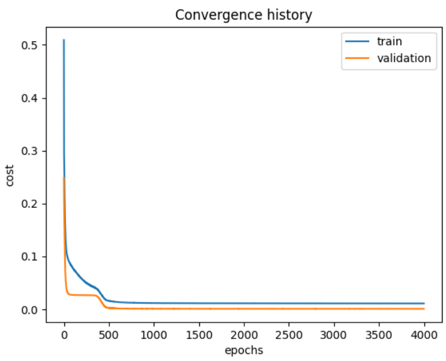

FIGURE 2.1: Convergence History of Best Model

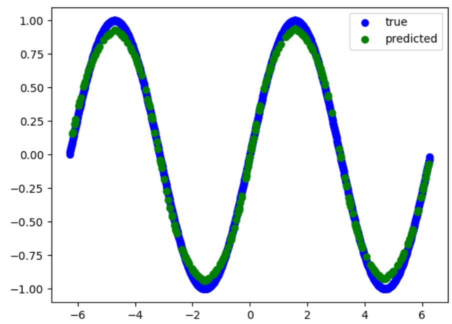

FIGURE 2.2: True vs. Predicted Values of Best Model

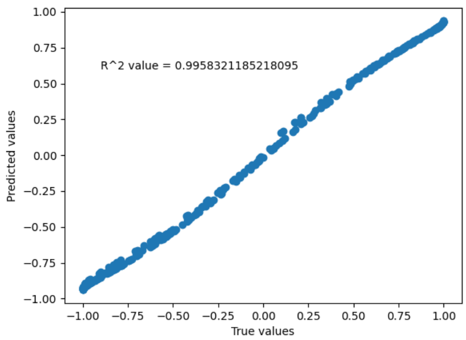

FIGURE 2.3: R2 Values of Best Model

2.3 Comparisons of ANN Architectures

```
Fixed Parameters:
    - Learning Rate: 0.001
    - Minibatch Size = 64
    - No. of Epochs: 1000
    - Activation Function: "tanh".
    - Cost Function = "mse"
```

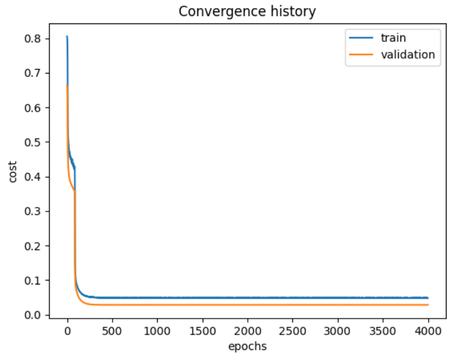

FIGURE 2.4: Architecture: 1-5-1

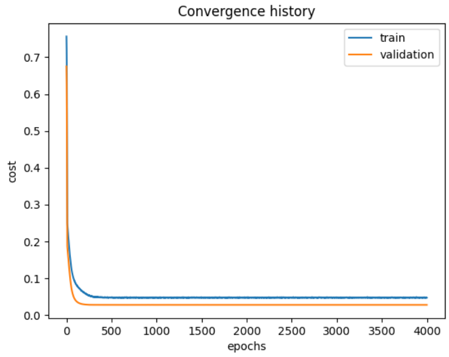

FIGURE 2.5: Architecture: 1-10-1

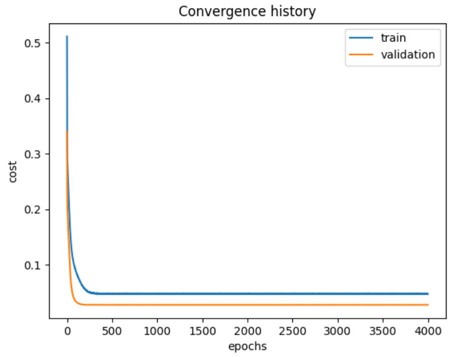

FIGURE 2.6: Architecture: 1-20-1

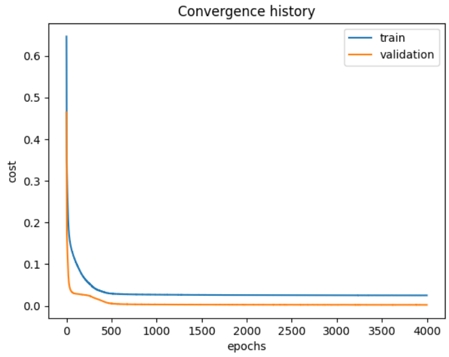

FIGURE 2.7: Architecture: 1-20-20-1

2.4 Granulation of Training Data

```
Fixed Parameters:
    - Learning Rate: 0.001
    - No. of Epochs: 4000
    - Activation Function: "tanh".
    - Cost Function = "mse"
    - Architecture: [1,20,20,1]
```

Table 2.2: Granulation of Training Data

Mini-Batch Size	Validation MAPE
64	204.4099064320096
256	90.53507966836217
1	27.587261759939924
Full Batch (1000)	141.70297947774583

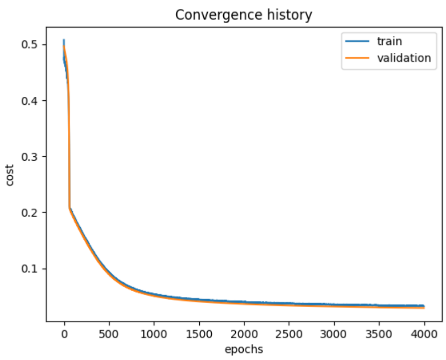

Figure 2.8: Mini Batch Size: 64

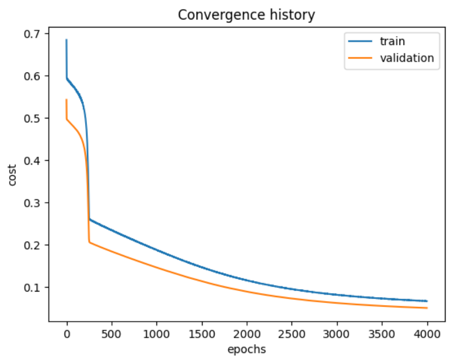

Figure 2.9: Mini Batch Size: 256

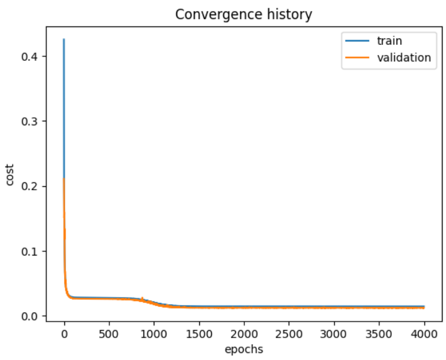

Figure 2.10: Mini Batch Size: 1 (SGD)

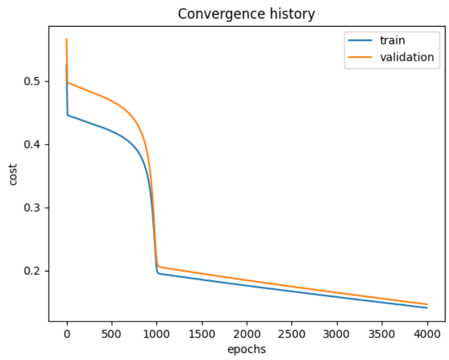

Figure 2.11: Full Batch - 1000

2.5 Activation Functions Comparison

```
Fixed Parameters:
    - Learning Rate: 0.001
    - Minibatch Size: 256, No. of Epochs: 4000
    - Cost Function = "mse"
    - Optimizeer = "adam"
    - Architecture: [1,20,20,1]
```

Table 2.3: Different Activation Functions - sin function

Activation Function	Validation MAPE
Tanh	11.9369811126166
Sigmoid	8.64491633354942
ReLU	108.84627549861669

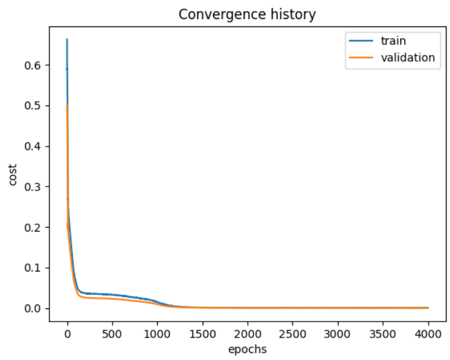

Figure 2.12: Activation Function - Tanh

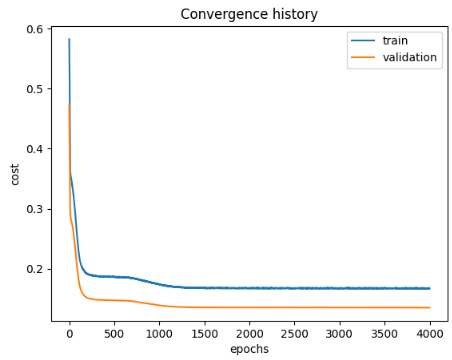

Figure 2.13: Activation Function - Sigmoid

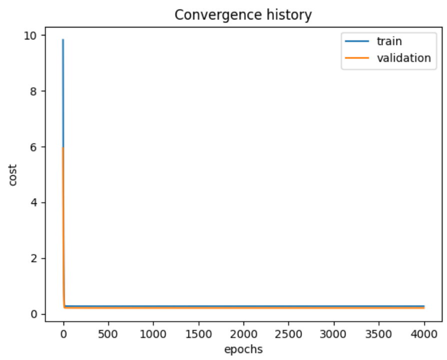

Figure 2.14: Activation Function - ReLU

2.6 Learning Rate Comparison

```
Fixed Parameters:
    - Minibatch Size: 256, No. of Epochs: 4000
    - Cost Function = "mse"
    - Optimizeer = "adam"
    - Activation Function : "tanh"
    - Architecture: [1,20,20,1]
```

TABLE 2.4: Different Learning Rates - Sin Function

Learning Rate	Validation MAPE
0.1	11.93698111261661
0.01	88.64491633354942
0.001	108.84627549861669

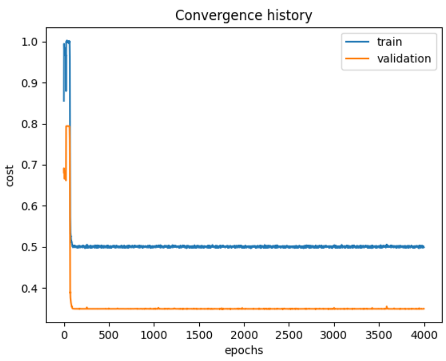

FIGURE 2.15: Learning Rate of 0.1

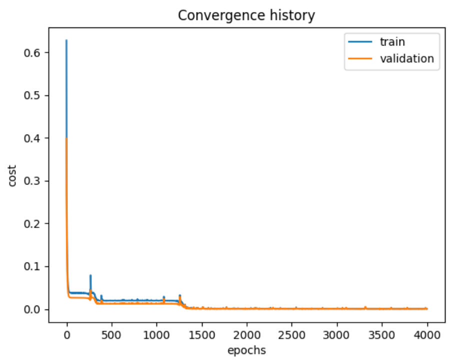

FIGURE 2.16: Learning Rate of 0.01

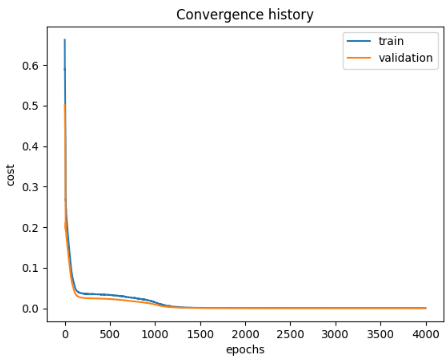

FIGURE 2.17: Learning Rate of 0.001

2.7 Regularization

```
Fixed Parameters:
    - Minibatch Size: 256, No. of Epochs: 4000
    - Cost Function = "mse"
    - Regularisation = "L2", lambda = 0
    - Optimizeer = "adam", beta = 0.9, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8
    - Activation Function : "tanh"
    - Architecture: [1,20,20,1]
```

TABLE 2.5: Different Learning Rates - Sin Function

Learning Rate	Validation MAPE
0.01	44.3362954628574
0.0001	32.20135895213563

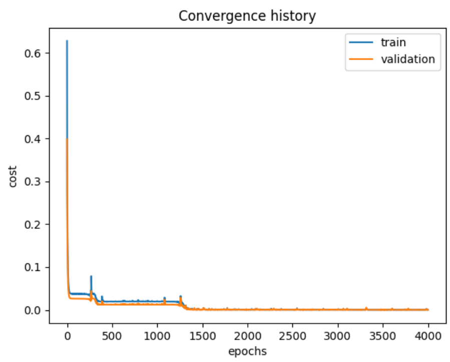

FIGURE 2.18: Learning Rate of 0.01, lambda 0

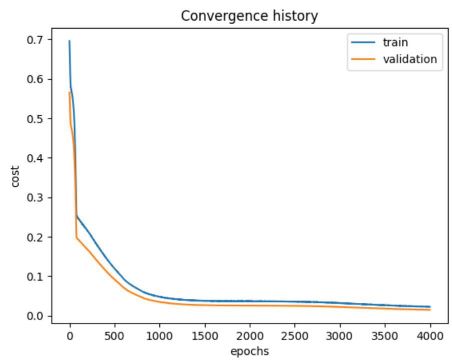

FIGURE 2.19: Learning Rate of 0.0001, lambda 0

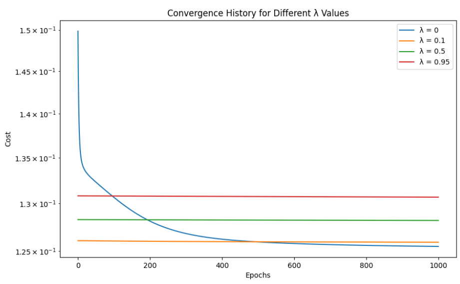

FIGURE 2.20: Learning Rate of 0.001

2.8 SGD-Momemtum

```
Fixed Parameters:
    - Minibatch Size: 64, No. of Epochs: 4000
    - Cost Function = "mse"
    - Regularisation = "L2", lambda = 0.25
    - Optimizeer = "momentum"
    - Activation Function : "tanh"
    - Architecture: [1,30,30,1]
```

TABLE 2.6: SGD Momemtum Results

Validation mape	69.90192566292176
r2 score	0.8899464593922939

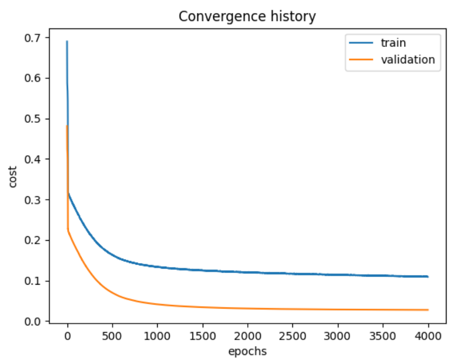

FIGURE 2.21: Convergence History of SGD-Momentum Model

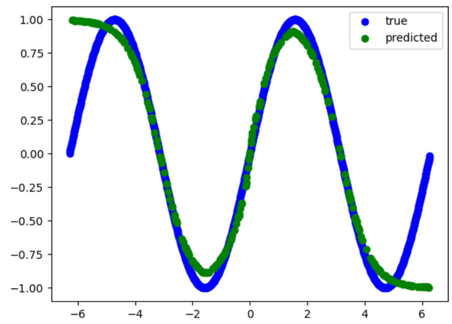

FIGURE 2.22: True vs. Predicted Values of SGD-Momentum Model

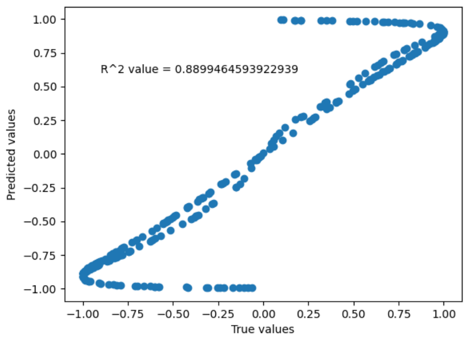

FIGURE 2.23: R2 Values of SGD-Momentum Model

Chapter 3

Combined Cycle Power Plant Dataset

3.1 Problem Statement

In this assignment, we train and validate our Artificial Neural Network (ANN) on the Combined Cycle Power Plant (CCPP) dataset, which consists of more than 9,000 rows of data with four input variables and one output variable. The dataset captures complex functional relationships between various environmental and operational factors, such as temperature, pressure, humidity, and exhaust vacuum, affecting the power output of the plant. The objective was to assess the ANN's ability to accurately model complex functional relationships and predict outputs based on the given inputs.

3.2 Best Model For CCNP Dataset

```
- Architecture = [4,20,20,1]
- Minibatch Size = 256
- Learning Rate = 0.001
- No. of Epochs = 500
- Activation Function = "tanh"
- Cost Function = "mse"
- Optimizer = "adam", beta = 0.9, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8
- Regularisation = "L2", Lambda = 0.1
```

TABLE 3.1: Best Model Results for CCPP

Validation mape	55.765654045999355
Test mape	83.97926361002177
r2 score	0.9413901717959738

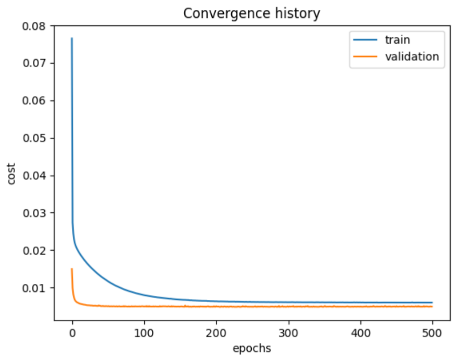

FIGURE 3.1: Convergence History of Best Model

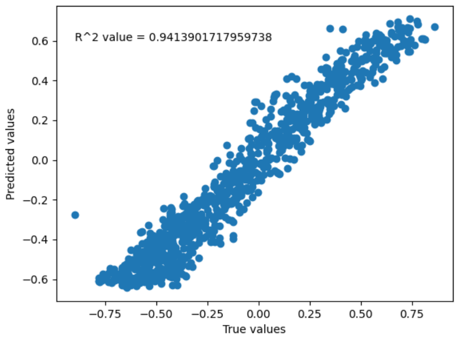

FIGURE 3.2: R2 : True vs. Predicted Values of Best Model

3.3 Comparisons of ANN Architectures

```
Fixed Parameters:
    - Learning Rate: 0.001
    - Minibatch Size = 64, No. of Epochs: 100
    - Activation Function: "tanh", Cost Function = "mse"
```

Table 3.2: Architectural comparisons - CCPP Dataset

Architecture	MAPE - Validation	MAPE - Test	R^2 Score
[4, 5, 1]	108.47	147.62	0.7519
[4, 10, 1]	79.92	123.82	0.8649
[4, 20, 1]	75.26	112.04	0.8998
[4, 10, 10, 1]	92.06	140.24	0.8686

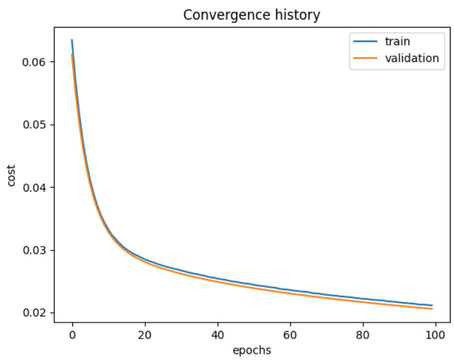

Figure 3.3: Architecture: 4-5-1

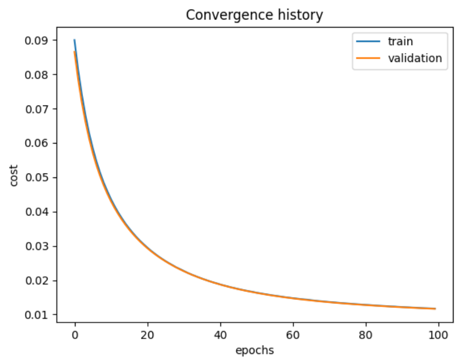

Figure 3.4: Architecture: 4-10-1

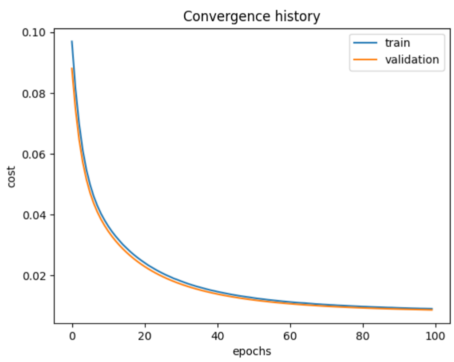

Figure 3.5: Architecture: 4-20-1

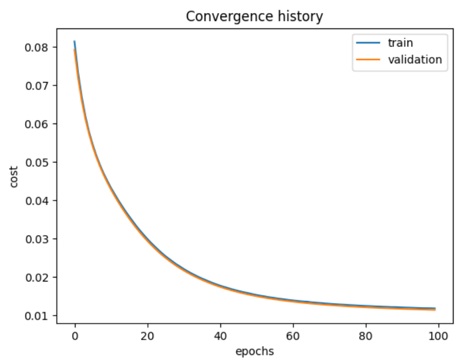

Figure 3.6: Architecture: 4-10-10-1

3.4 Granulation of Training Data

```
Fixed Parameters:
    - Learning Rate: 0.001
    - No. of Epochs: 300
    - Activation Function: "tanh".
    - Cost Function = "mse"
    - Architecture: [4,10,10,1]
```

TABLE 3.3: Granulation of Training Data - CCPP Dataset

Mini Batch Size	**MAPE - Validation**	**MAPE - Test**	**R^2 Score**
64	80.65	130.96	0.8929
256	92.80	138.58	0.8570
1 (SGD)	55.64	84.24	0.9403
Full Batch (6888)	129.37	138.99	0.2018

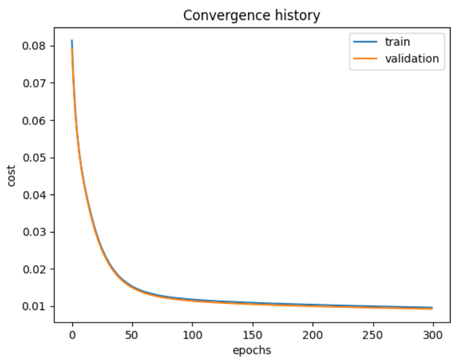

FIGURE 3.7: Batch Size : 64

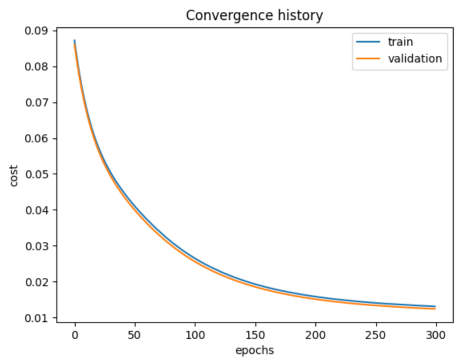

FIGURE 3.8: Batch Size : 256

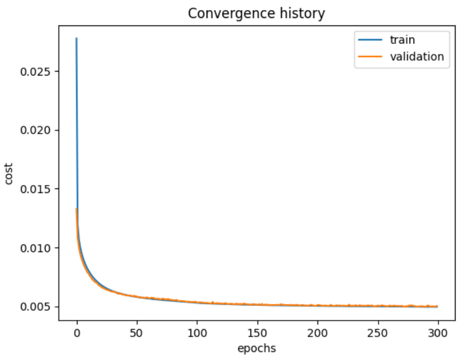

FIGURE 3.9: Batch Size : 1

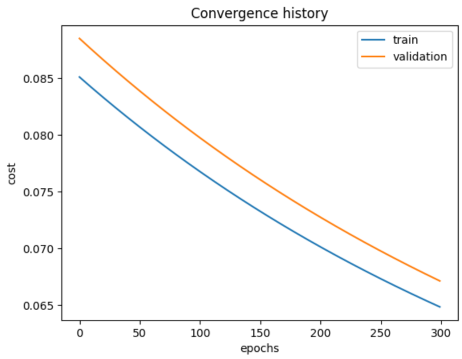

FIGURE 3.10: Full Batch - 6888

3.5 Activation Functions Comparisions

```
Fixed Parameters:
    - Learning Rate: 0.001
    - Minibatch Size: 64
    - No. of Epochs: 100
    - Cost Function = "mse"
    - Architecture: [4,10,10,1]
```

TABLE 3.4: Activation Functions - CCPP Dataset

Activation Function	MAPE - Validation	MAPE - Test	R^2 Score
Tanh	80.65	130.96	0.8929
Sigmoid	135.62	163.28	-4.4952
ReLU	96.99	107.53	0.6474

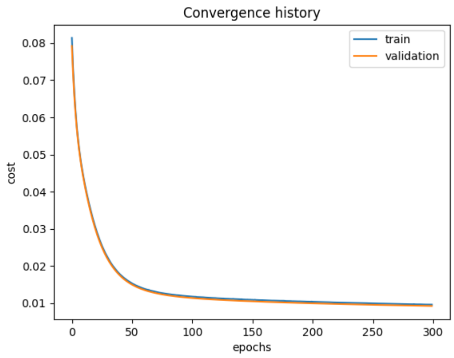

FIGURE 3.11: Activation Function: Tanh

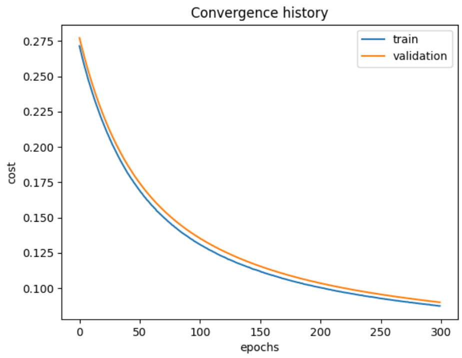

FIGURE 3.12: Activation Function: Sigmoid

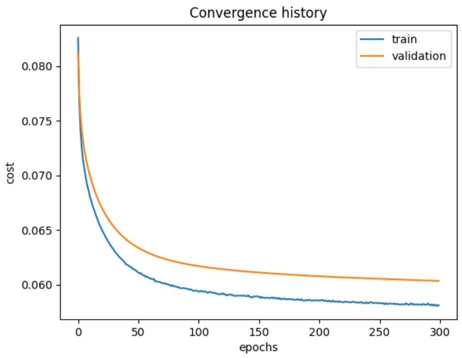

FIGURE 3.13: Activation Function: ReLU

3.6 Learning Rate Comparision

```
Fixed Parameters:
    - Minibatch Size: 64, No. of Epochs: 100
    - Regularisation = "L2", lambda = 0
    - Optimizeer = "adam"
    - Activation Function : "tanh"
    - Architecture: [4,10,10,1]
```

TABLE 3.5: Learning Rates Comparision - CCPP

Learning Rate	MAPE - Validation	MAPE - Test	R^2 Score
0.1	64.01	98.03	0.9161
0.01	49.54	67.62	0.9381
0.001	53.88	82.27	0.9412
0.0001	54.06	84.06	0.9389

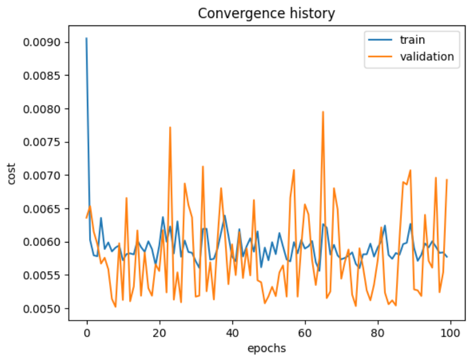

FIGURE 3.14: Learning Rate: 0.1

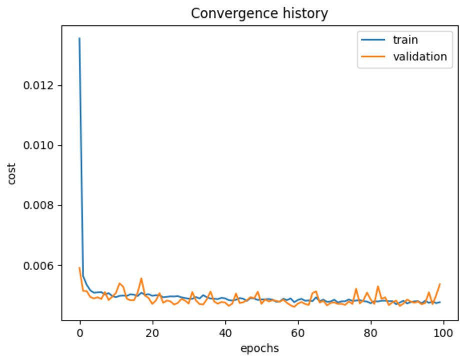

FIGURE 3.15: Learning Rate: 0.01

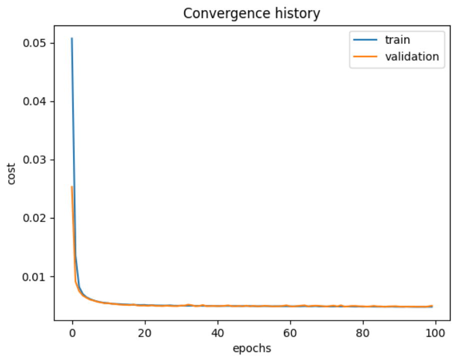

FIGURE 3.16: Learning Rate: 0.001

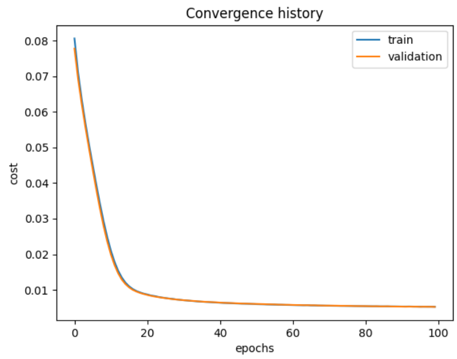

FIGURE 3.17: Learning Rate: 0.0001

3.7 L2 Regularization Parameter Comparisons

```
Fixed Parameters:
    - Minibatch Size: 32, No. of Epochs: 500
    - Regularisation = "L2"
    - Learning Rate : 0.001
    - Activation Function : "tanh"
    - Architecture: [4,10,10,1]
```

TABLE 3.6: Lambda Values - L2 Regularization

Lambda	MAPE - Validation	MAPE - Test	R^2 Score
0	68.44	109.62	0.9161
0.1	66.42	97.54	0.9181
0.5	58.20	83.01	0.9061
0.95	57.44	77.51	0.8738

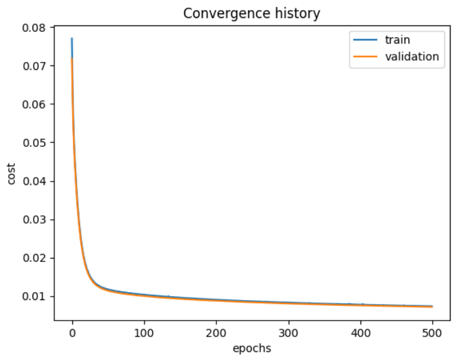

FIGURE 3.18: L2 - Lambda: 0

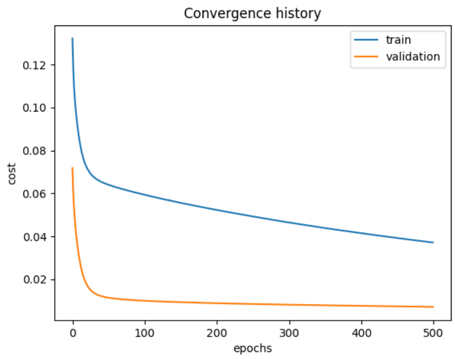

FIGURE 3.19: L2 - Lambda: 0.1

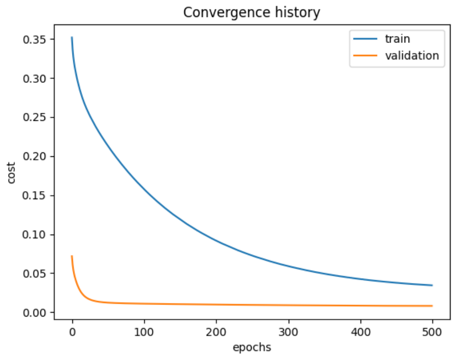

FIGURE 3.20: L2 - Lambda: 0.5

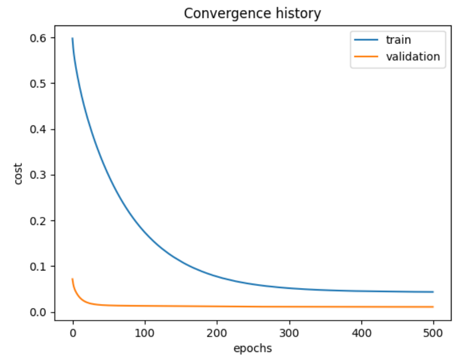

FIGURE 3.21: L2 - Lambda: 0.95

3.8 SGD Momentum

```
Fixed Parameters:
    - Regularisation = "L2" ; lambda = 0.1
    - Learning Rate : 0.01
    - Momentum = 0.9
    - Activation Function : "tanh"
    - Architecture: [4,10,10,1]
```

TABLE 3.7: SGD Momentum Performance

Configuration	MAPE - Val	MAPE - Test
Batch 1 - E100	296.67	372.29
Batch 64 - E100	91.09	138.56

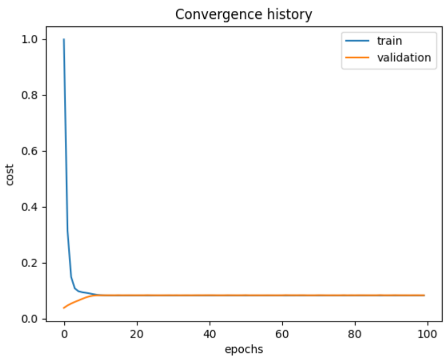

FIGURE 3.22: SGD: Batch 1 - E100 Momentum

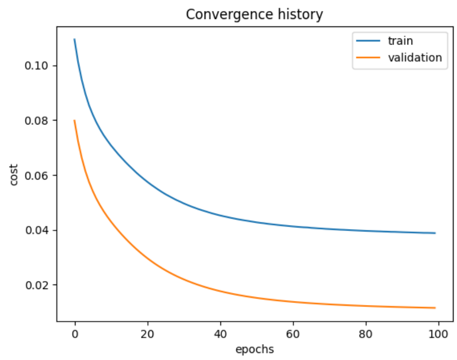

FIGURE 3.23: SGD: Batch 64 - E100 Momentum

3.9 Adam Optimization

```
Fixed Parameters:
    - Regularisation = "L2" ; lambda = 0.1
    - Learning Rate : 0.01
    - beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8
    - Activation Function : "tanh"
    - Architecture: [4,10,10,1]
```

Table 3.8: Adam Optimization Performance

Configuration	MAPE - Val	MAPE - Test
Batch 1 - E100	106.17	112.15
Batch 1 - E500	141.02	168.60
Batch 64 - E100	61.02	93.99
Batch 64 - E500	57.02	90.20

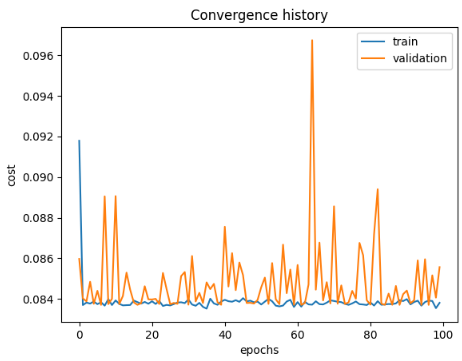

Figure 3.24: B1 - E100

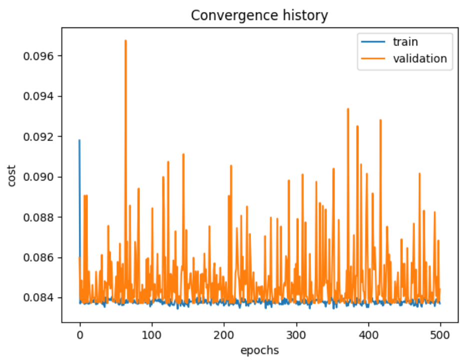

Figure 3.25: B1 - E500

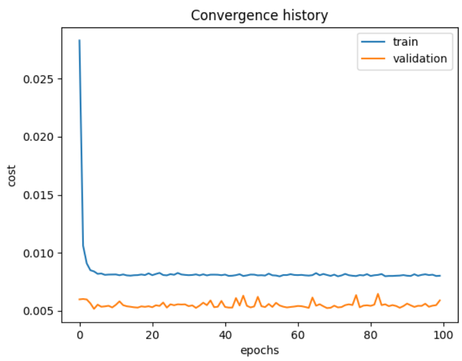

Figure 3.26: B64 - E100

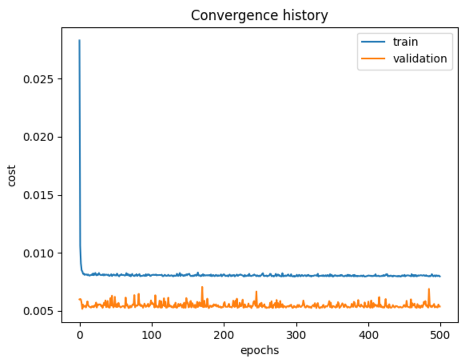

Figure 3.27: B64 - E500