20th March 2024

Deep Learning Assignment-1

Team :

Abhijit rao	- SE21UARI003	N.Gagan	- SE21UARI094
S.Varshini	- SE21UARI183	Ashritha	- SE21UARI153

Step by step back propagation algorithm:-

L_layer_model_minib() **– trains the model**

- initialize_parameters(layers_dims) **-- initialize the weights**
- initialize_adam(parameters) **-- initialise Adam optimiser**
- random_mini_batches(X, Y, mini_batch_size, seed) **-- create random mini batches**
- For i in mini batches:
  - L_model_forward(X,parameters,activation) **-- forward pass**
    - Loop over the layers and do forward prop on each
      - linear_activation_forward(A_prev, W, b, activation) **-- computes the linear forward and then applies activation function**
        - linear_forward(A, W, b):
          - Z = W.dot(A) + b
        - A, activation_cache = activation(Z)
  - compute_cost(AL, Y,parameters,lambd,regularisation,cost_func='mse') **-- compute the cost**
    - Mse: cost = np.mean(np.square(AL-Y))*0.5
    - Log: cost = (1./m) * np.sum(-np.dot(Y,np.log(AL+epsilon).T) - np.dot(1-Y, np.log(1-AL+epsilon).T))
  - L_model_backward(AL, Y, caches,activation,regularisation,lambd,cost_func) **– backpropagation step**
    - Mse: dAL=(AL-Y)
    - Log: dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))
    - Loop over the layers and do back prop on each
      - linear_activation_backward(dA, cache,regularisation,lambd,activation) **--compute back propation from activation to the weights**
        - dZ = activation_backward(dA, activation_cache)
        - linear_backward(dZ, cache,regularisation,lambd)
          - db = 1./m * np.sum(dZ, axis = 1, keepdims = True)
          - dA_prev = np.dot(W.T,dZ)
  - update_parameters(parameters, grads, learning_rate) **-- update all the weights in the ANN**
- cost_avg = cost_total / batches **-- computes the average cost over the mini batch**
- predicterr(valid_x,valid_y,parameters,lambd,activation=activation,regularisation='none',cost_func=cost_func) **-- compute the validation cost**
- plt.plot(costs) **-- plot the training and validation costs**

These are the formulae and equations used in our code:

$$Z = W \cdot A + b$$

where A is activation from the previous layer

- mean square error — $\frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$
- Logrithmic loss — $\frac{-1}{m} \sum_{i=1}^{m} \left(y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i) \right)$
- mean absolute % error (MAPE) — $\frac{1}{m} \sum_{i=1}^{m} \frac{|y_i - \hat{y}_i|}{y_i} \times 100$

gradiant decent — $\theta = \theta - \alpha \cdot grad(\theta)$

momentum — $v = \beta \cdot v + (1-\beta) \cdot grad(\theta)$

followed by $\theta = \theta - \alpha \cdot v$

Adam — adaphive moment esimation with both 1st and 2nd moment v_1 and v_2

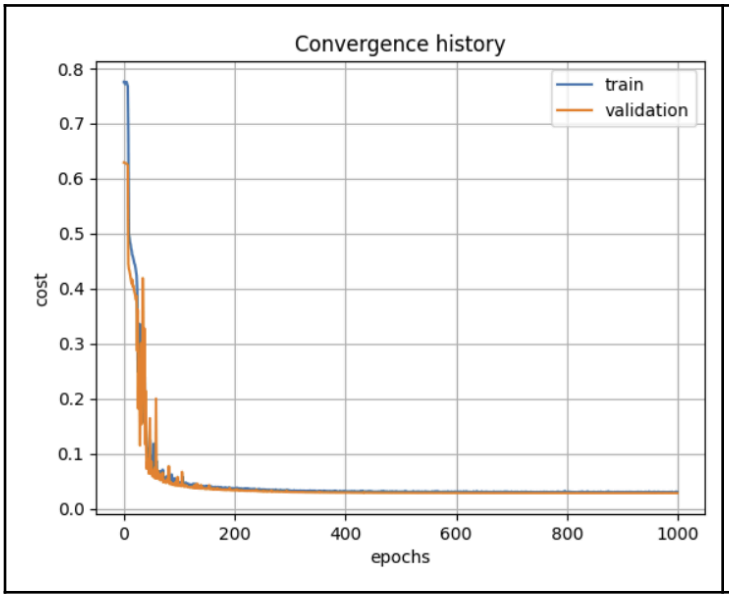
Part 1: Toy Problem - Sine Function

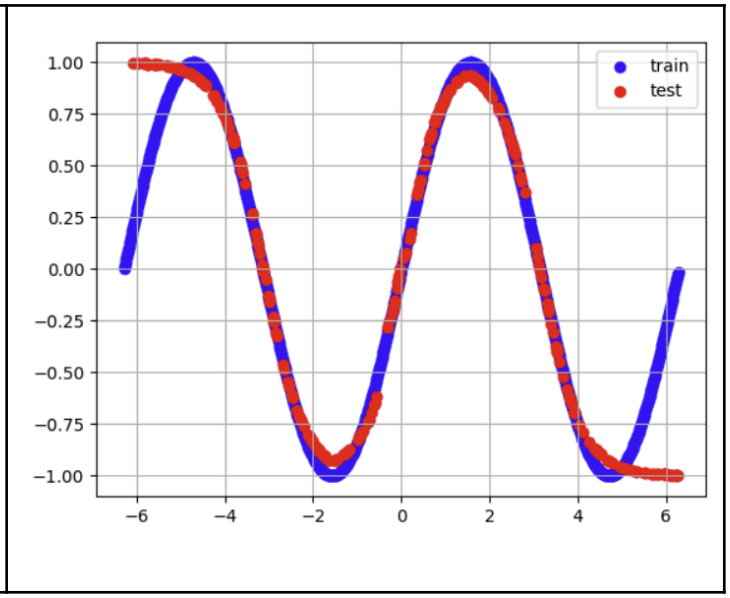
Architecture: [1, 20, 20, 1] **Minibatch Size**: 64

Optimizer: None **Regularization:** None

Learning Rate: 0.01 **Activation Function**: Tanh

No. of Epochs: 1000

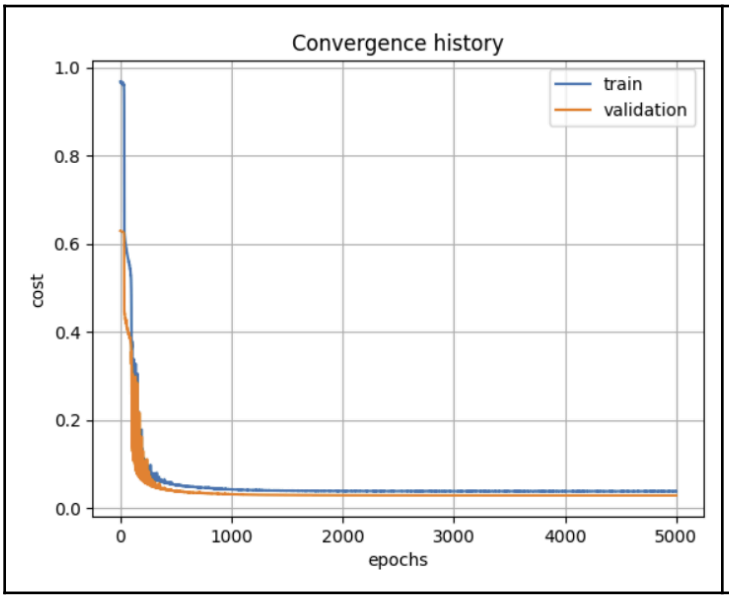


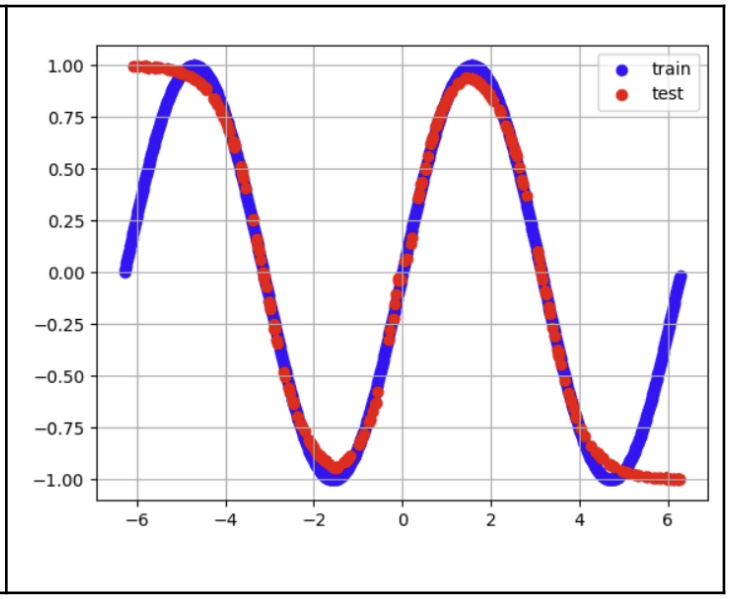


Learning Rate: 0.01 Minibatch Size: 256

Regularization: None No. of Epochs: 5000

Optimizer: None Activation Function: Tanh

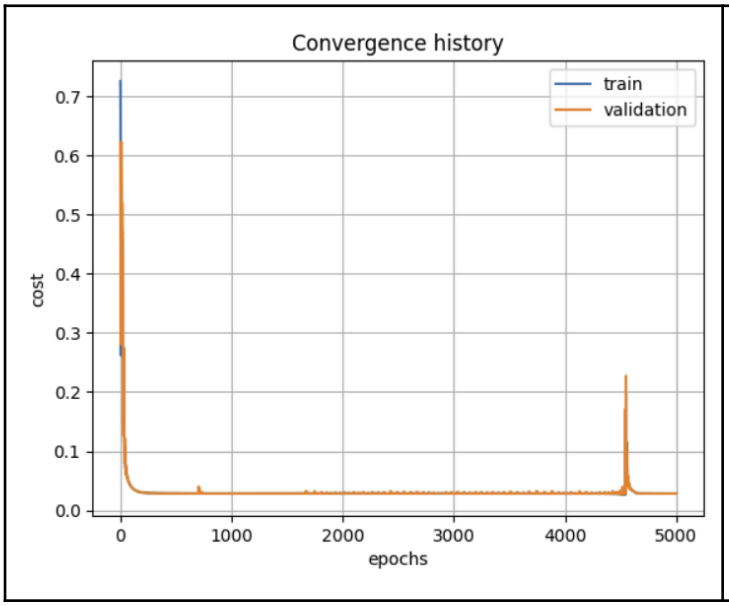


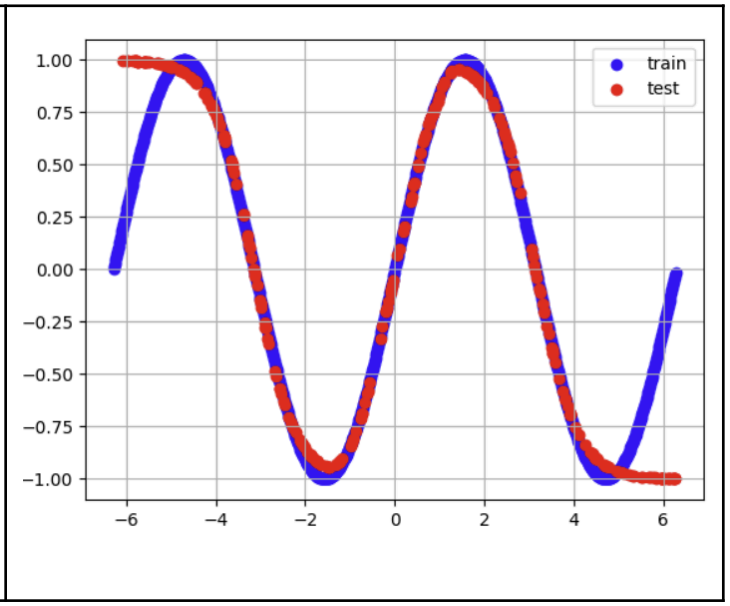


Learning Rate: 0.01 Minibatch Size: 1000

No. of Epochs: 5000 Regularization: None

Activation Function: Tanh Optimizer: Adam

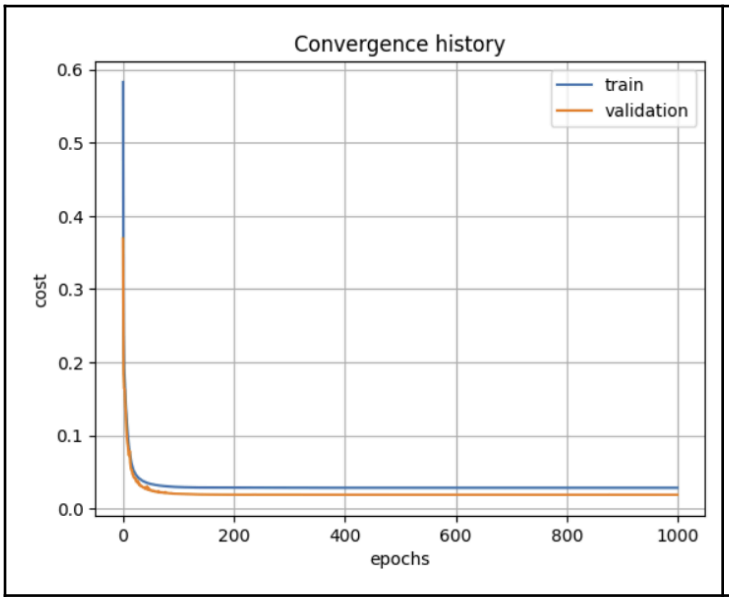


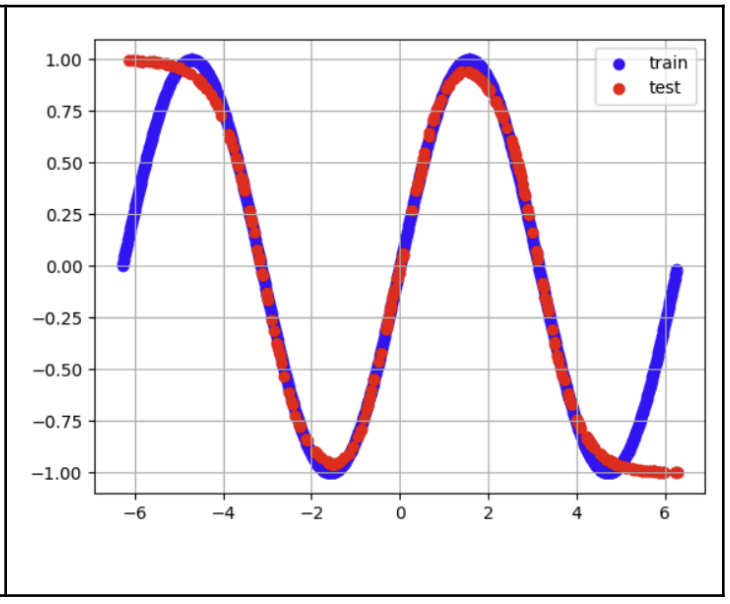


Learning Rate: 0.001 Minibatch Size: 1

No. of Epochs: 1000 Regularization: None

Activation Function: Tanh Optimizer: None

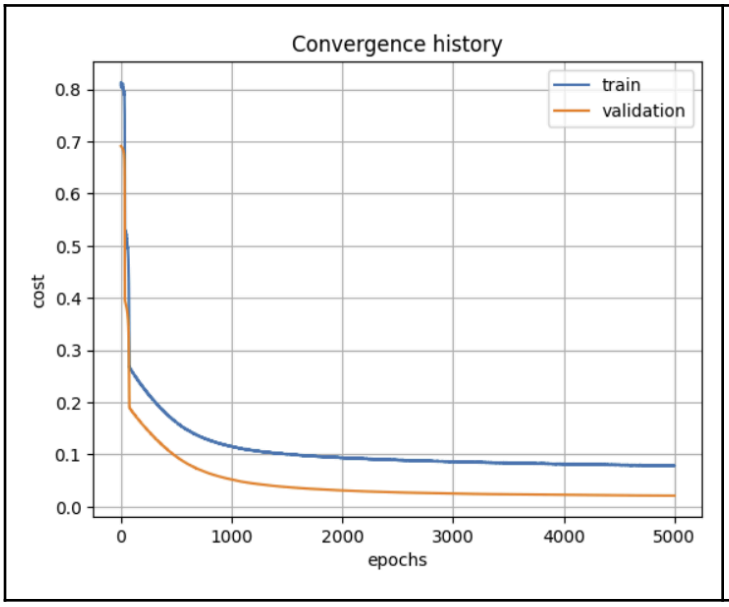


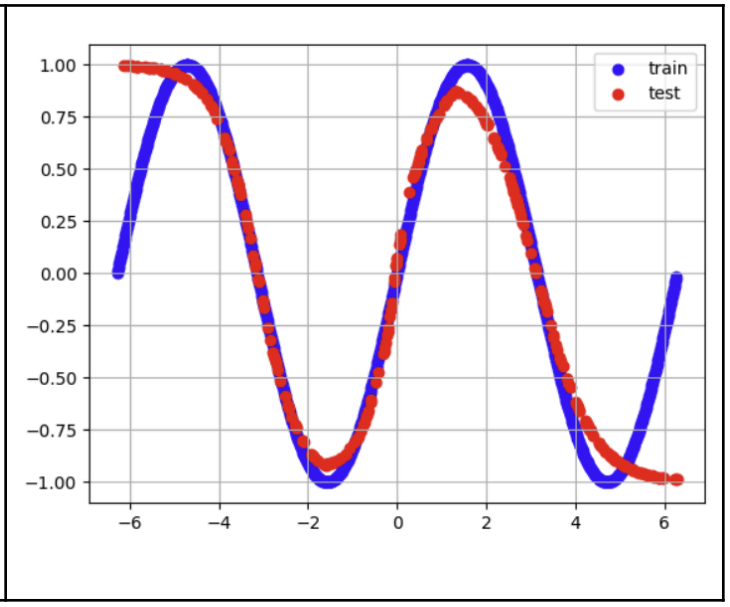


Learning Rate: 0.001 Minibatch Size: 64

No. of Epochs: 5000 Regularization: L2

Activation Function: Tanh Optimizer: None

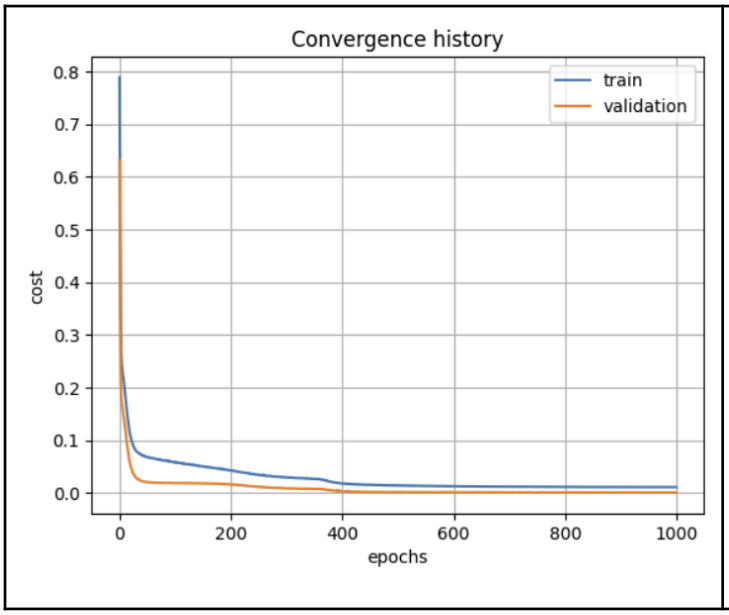


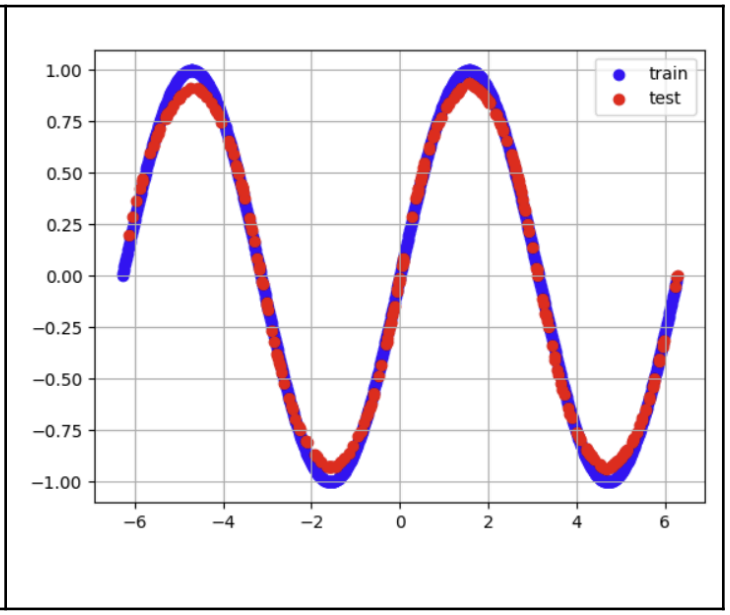


Learning Rate: 0.001 Minibatch Size: 64

No. of Epochs: 1000 Regularization: L2

Activation Function: Tanh Optimizer: Adam

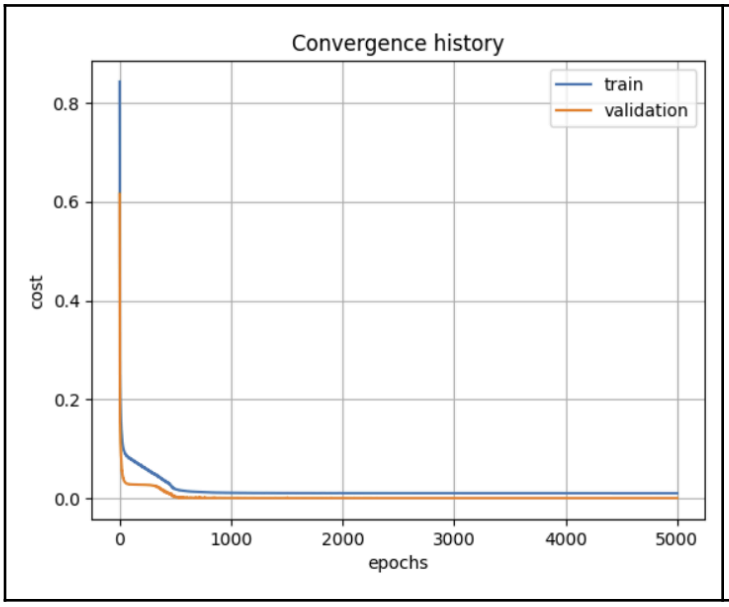


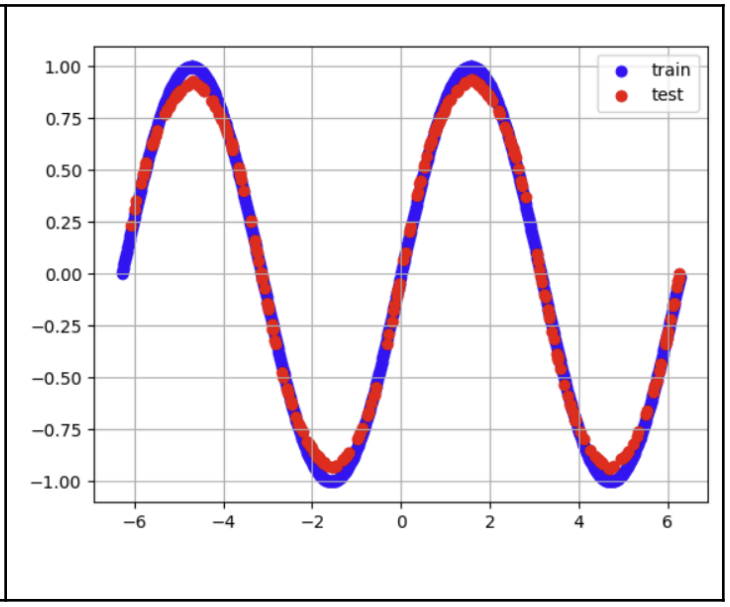


Learning Rate: 0.001 Minibatch Size: 64

No. of Epochs: 5000 Regularization: L2

Activation Function: Tanh Optimizer: Adam





Final model for Sine function data:

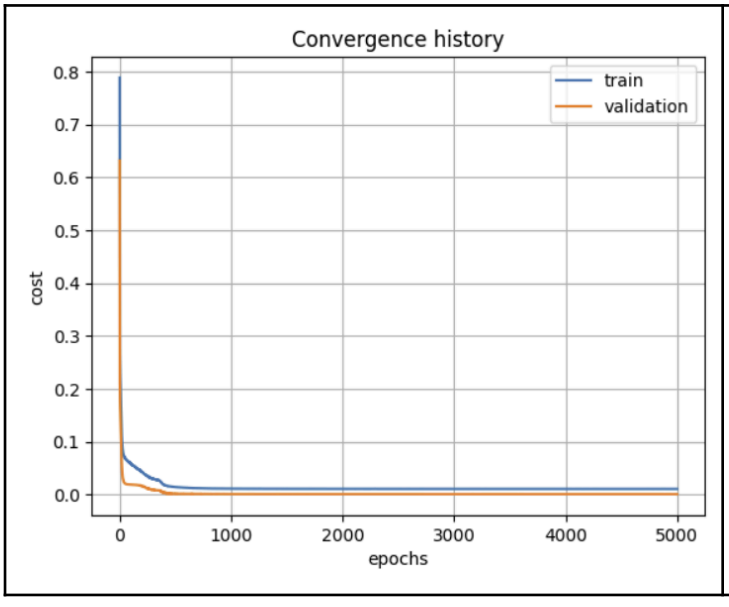
Architecture = [1,20,20,1] Mini batch size = 64

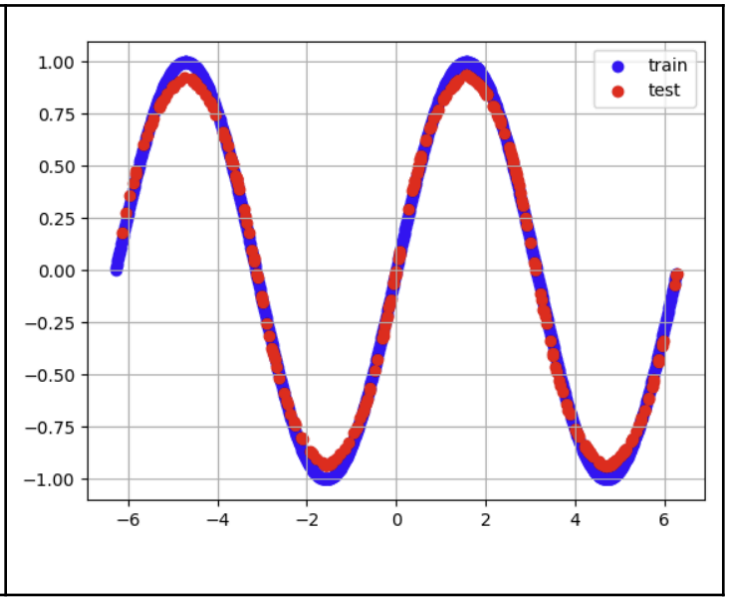
No. of iterations = 5000 Learning rate = 0.001

Activation function = tanh L2 regularization with $\lambda = 0.1$

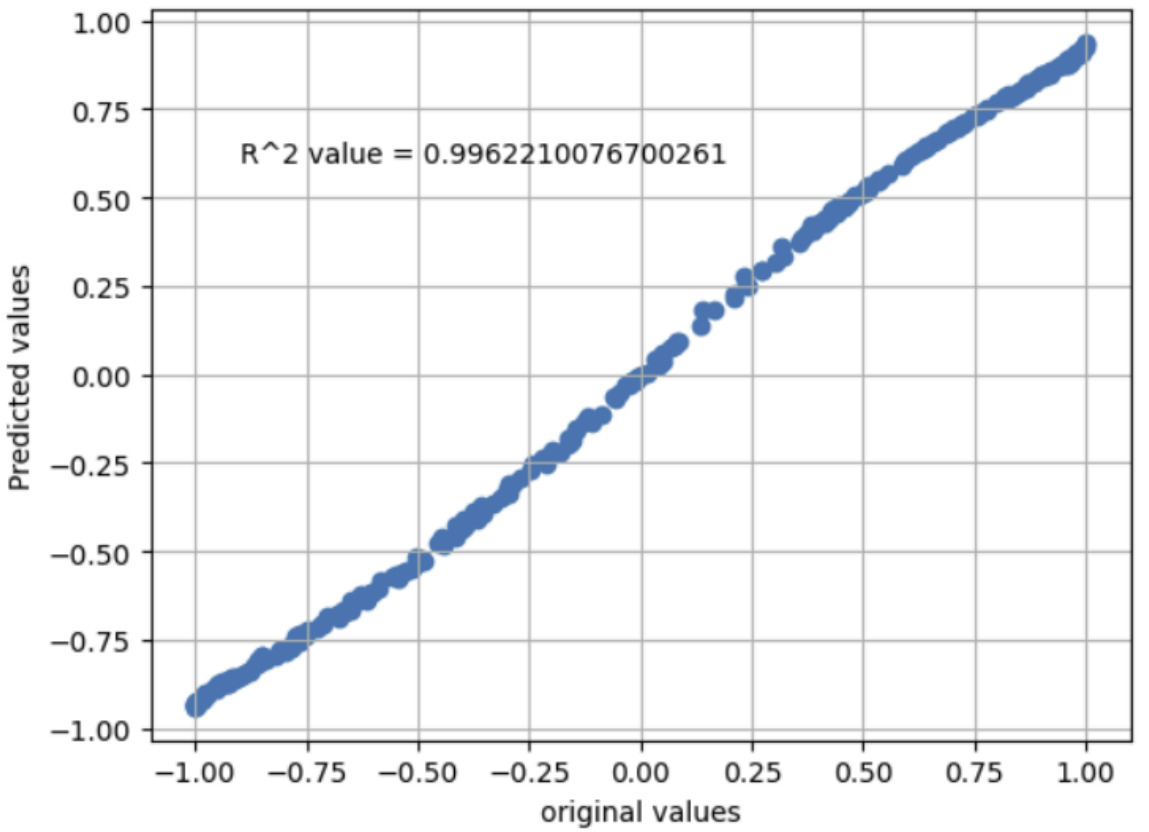
Adam optimizer with $\beta = 0.9$, $\beta 1 = 0.9$, $\beta 2 = 0.99$, epsilon = 1e-8

Validation MAPE: 7.735681529726113





r^2 value = 0.9962210076700261



C) Combined Cycle Power Plant Dataset

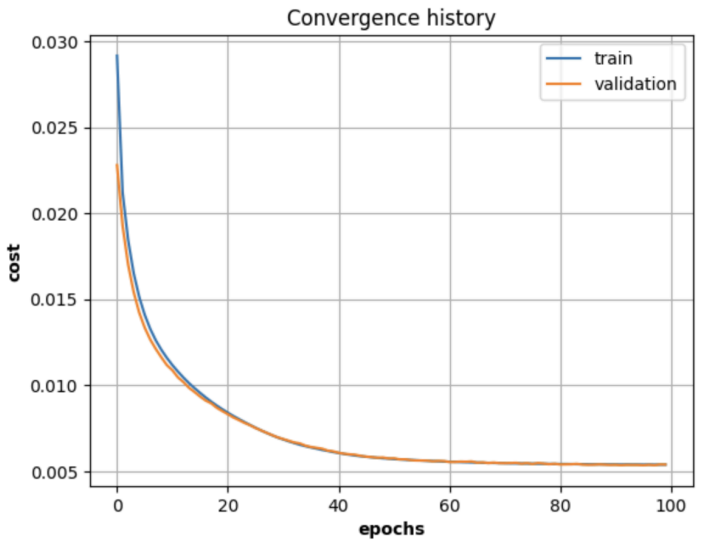
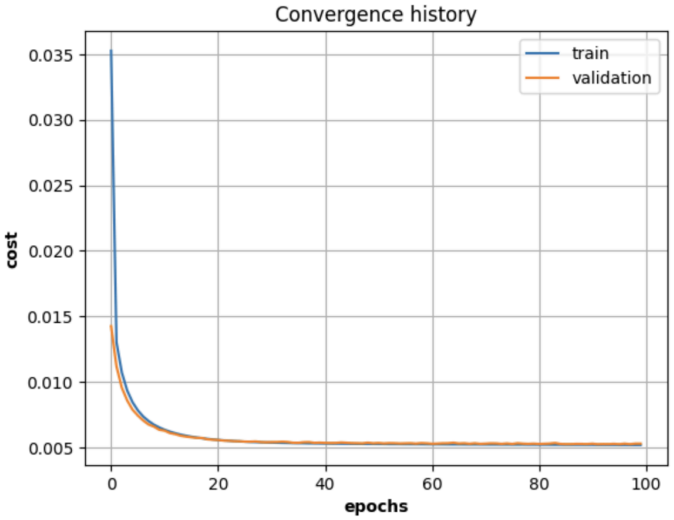
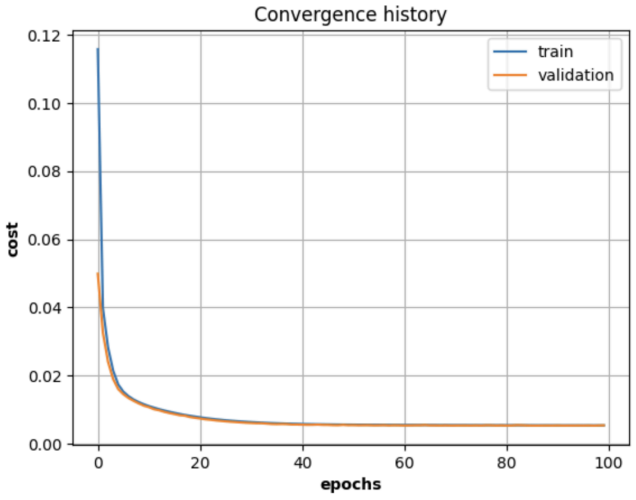
The Combined Cycle Power Plant Dataset is in xlsx so to use it more efficiently lets convert the data set into ccpp.csv file format. Use xlsx2csv.py file.

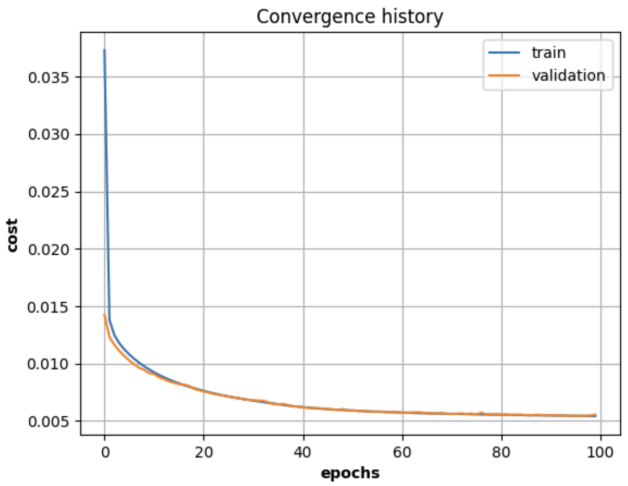
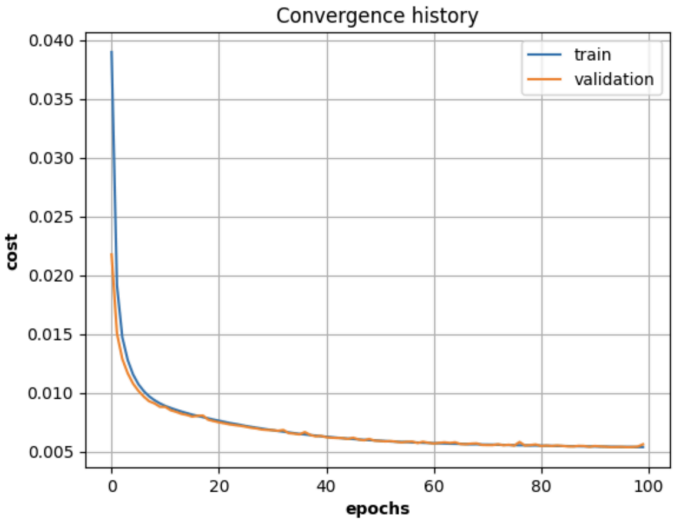
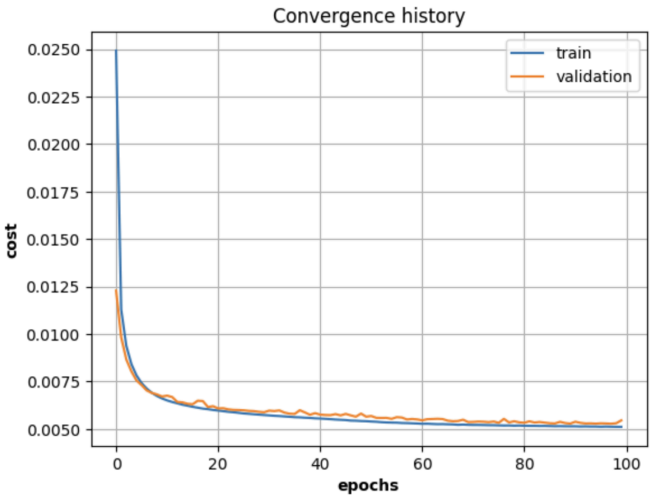
No. of input features: 4
No. of outputs: 1 (regression problem)
No. of samples in training set: 6888
No. of samples in validation set: 1723
No. of samples in testing set: 957

a) Comparison of ANN architecture

Parameters:

Learning Rate: 0.001
Activation Function: Tanh
Minibatch Size: 1 (SGD)
No. of Epochs: 100

Architecture: 4, 5, 1 **Validation MAPE**: 56.39 **Testing MAPE**:89.1312	 Validation mape : 56.39101216724893 Test mape : 89.1312588829463
Architecture: 4, 10, 1 **Validation MAPE**: 60.10395205340254 **Testing MAPE**: 92.55621162835348	 Validation mape : 60.103952053402544 Test mape : 92.55621162835348
Architecture: 4, 5, 5, 1 **Validation MAPE**: 58.85335117522823 **Testing MAPE**: 93.6317318854288	 Validation mape : 58.85335117522823 Test mape : 93.6317318854288

Architecture: 4, 10, 10, 1 **Validation MAPE:** 58.94387833989614 **Testing MAPE**: 88.35821547058174	 Validation mape : 58.94387833989614 Test mape : 88.35821547058174
Architecture: 4, 10, 5, 10, 1 **Validation MAPE**: 57.49552278259554 **Testing MAPE**: 80.59554667386713	 Validation mape : 57.49552278259554 Test mape : 80.59554667386713
Architecture: 4, 10, 10, 10, 1 **Validation MAPE**: 58.427651908906384 **Testing MAPE**: 83.7263864467342	 Validation mape : 58.427651908906384 Test mape : 83.7263864467342

Observations:

- The training error drops to slightly lower levels when there are two hidden layers as compared to one hidden layer. This might be because a single hidden layer with 5 or 10 neurons is less

capable to handle the complexity of the function. In addition, the training error drops slightly more for 10 neurons as compared to 5. If we add more neurons to the single layer, we should be able to capture the function's complexity.

- For two and three hidden layers, the error drops roughly to the same level after 100 epochs. The convergence is a little faster for three hidden layers.
- The MAPE value is lowest for two hidden layers. The [4, 10, 10, 1] models converges slower than the [4, 5, 5, 1] model, but has lower MAPE values for both validation and test.

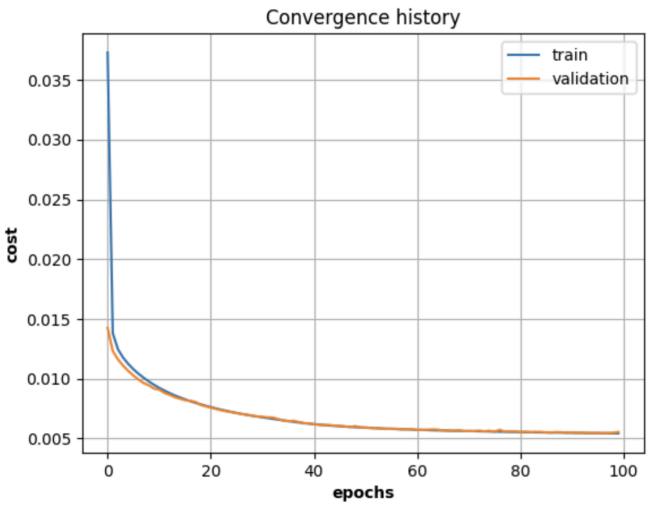
b) Comparison of mini batch size

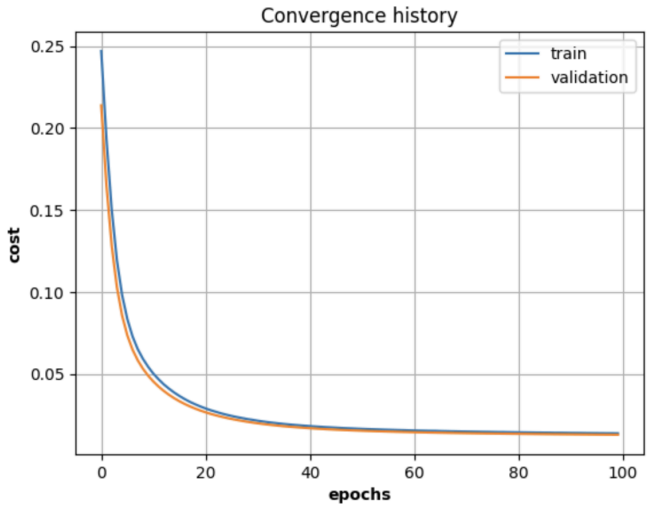
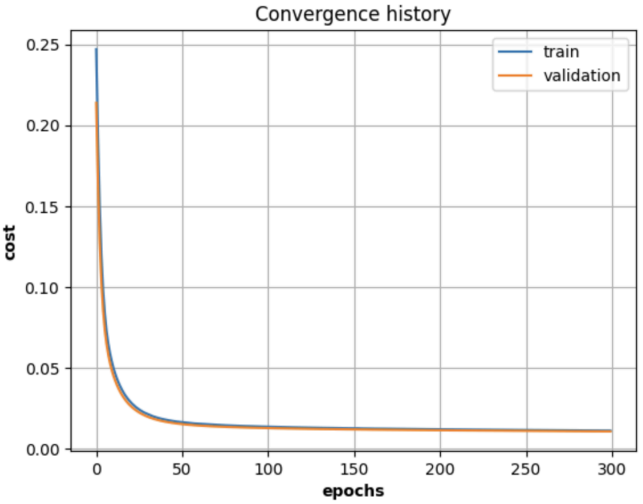
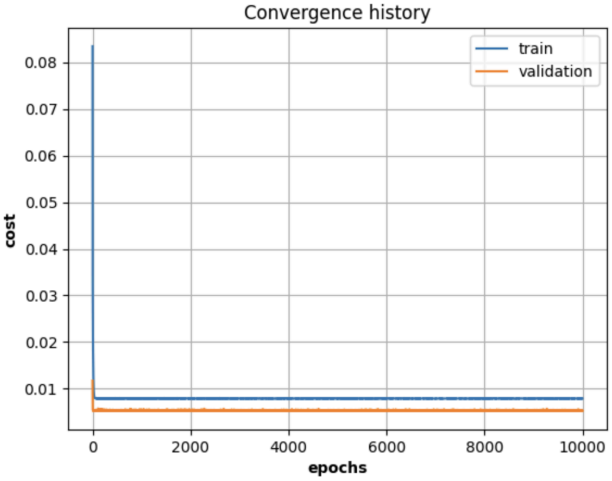
Parameters:

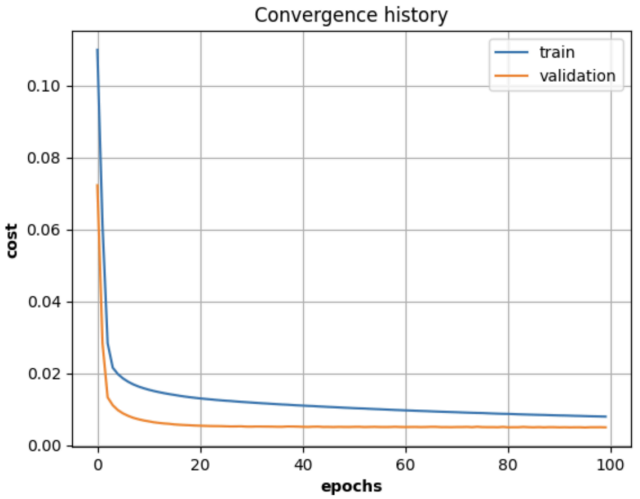
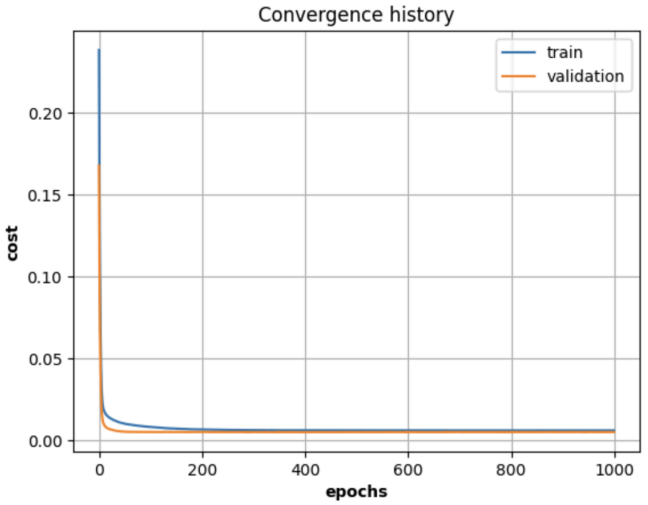
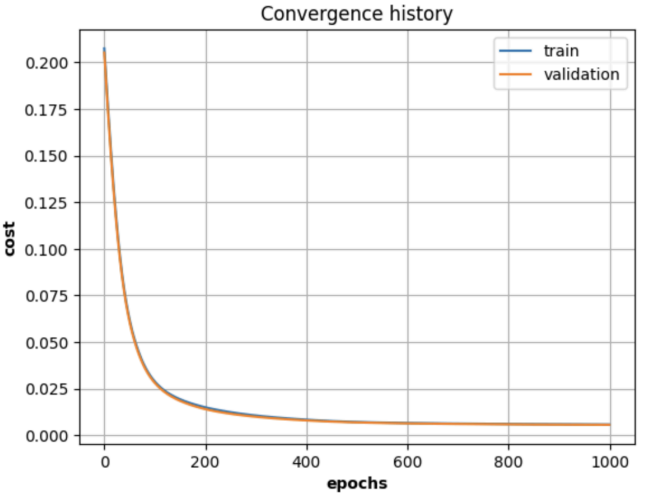
Architecture: 4, 10, 10, 1

Learning Rate: 0.001

Activation Function: Tanh

Mini Batch Size: 1 (SGD) **No. of Epochs**: 100 **Validation MAPE**: 58.94387833989614 **Testing MAPE:** 88.35821547058174	 Validation mape : 58.94387833989614 Test mape : 88.35821547058174

Mini Batch Size: 64 **No. of Epochs**: 100 **Validation MAPE**: 101.6112747486901 **Testing MAPE:** 156.39297294082357	 Validation mape : 101.6112747486901 Test mape : 156.39297294082357
Mini Batch Size: 64 **No. of Epochs**: 300 **Validation MAPE**: 90.4348703859385 **Testing MAPE:** 139.58516360545508	 Validation mape : 90.43487038593851 Test mape : 139.58516360545508
Mini Batch Size: 64 **No. of Epochs**: 10,000 **Validation MAPE**: 59.1157222447164 **Testing MAPE:** 93.4830223958601	 Validation mape : 59.115722244716416 Test mape : 93.4830223958601

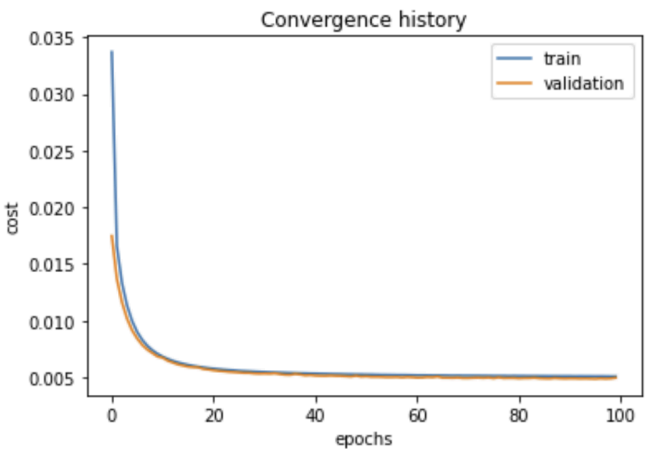
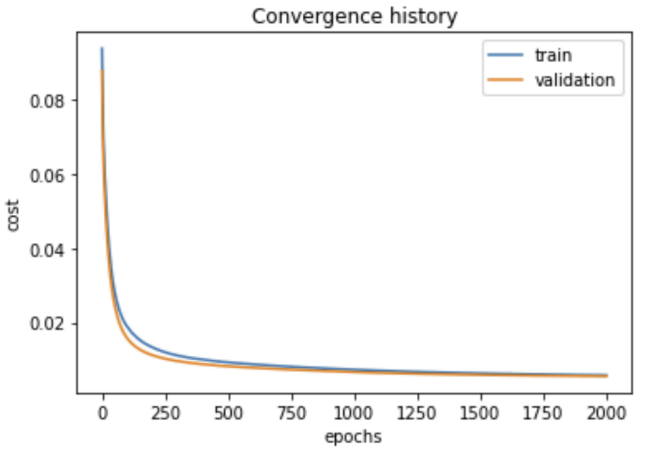
Mini Batch Size: 256 **No. of Epochs**: 100 **Validation MAPE**: 53.74700396113982 **Testing MAPE:** 82.70480958039084	 Validation mape : 53.747003961139825 Test mape : 82.70480958039084
Mini Batch Size: 256 **No. of Epochs**: 1000 **Validation MAPE**: 57.107324805878 **Testing MAPE:** 87.20069828563007	 Validation mape : 57.10732480587858 Test mape : 87.20069828563007
Mini Batch Size: 6888 **No. of Epochs**: 1000 **Validation MAPE**: 62.35794754768 **Testing MAPE:** 89.2523546152497	 Validation mape : 62.357947547680205 Test mape : 89.2523546152497

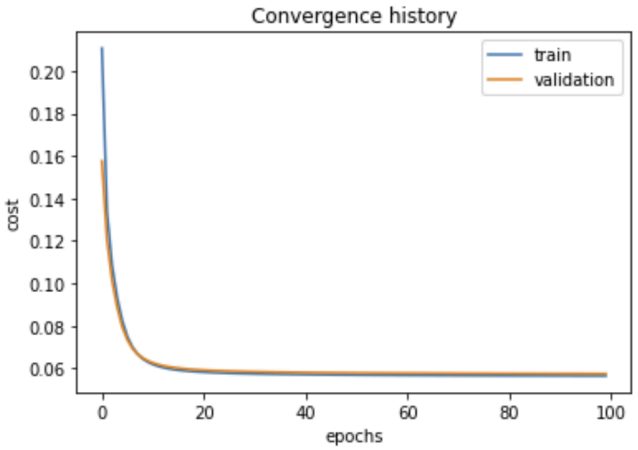
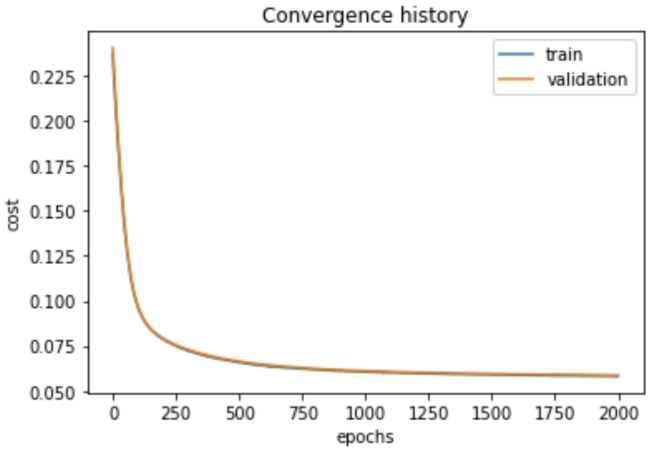
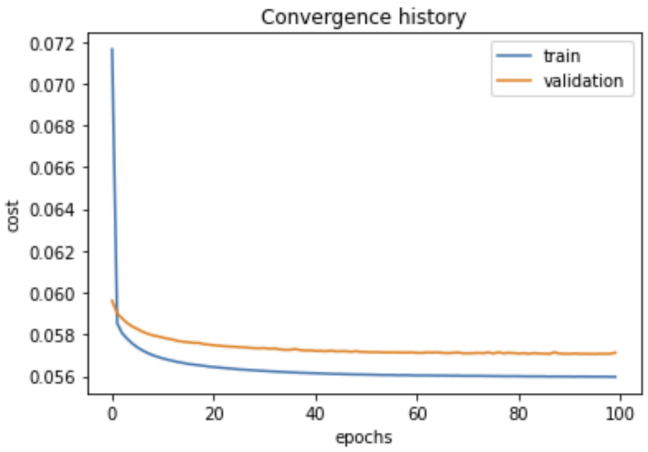
Observations:

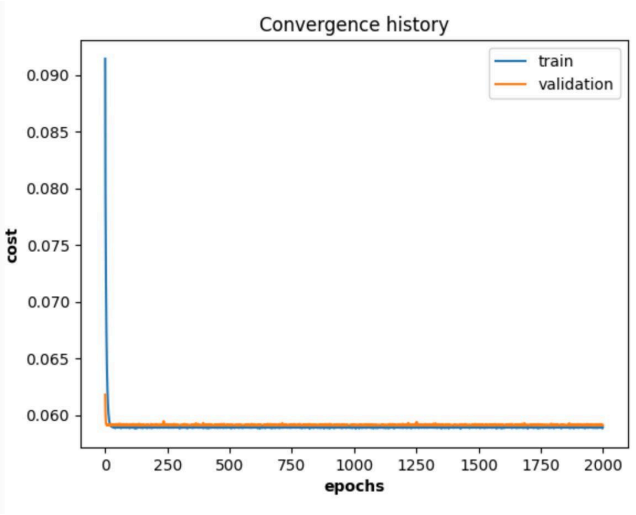
• SGD clearly converges faster than mini batches even when number of iterations are increased. This must be because SGD is less likely to get stuck in local minima than a mini-batch approach.
• As batch size increases, the computations become much faster since average error across all samples in the batch is taken. The learning, however, becomes slower, and more iterations are needed for
convergence.
• As batch size increases, learning per epoch becomes faster. This could be due to the use of vectorisation in computations.
• For the full batch, the convergence is extremely slow.

c) Comparison of Activation functions

Architecture: 4, 10, 10, 1
Learning Rate: 0.001
Testing the tanh, sigmoid and relu function for SGD and mini-batch size of 64.

Activation function: Tanh Mini batch size = 1 (SGD) No. of Epochs = 100 Validation MAPE: 57.3510 Testing MAPE: 82.2375	
Activation function: Tanh Mini batch size = 64 No. of Epochs = 2000 Validation MAPE: 57.3510 Testing MAPE: 82.2385	

Activation function: Sigmoid Mini batch size = 1 (SGD) Number of epochs = 100 Validation MAPE: 86.5935 Testing MAPE: 95.5624	
Activation function: Sigmoid Mini batch size = 64 Number of epochs = 2000 Validation MAPE: 92.2368 Testing MAPE: 103.1215	
Activation function: Relu Mini batch size = 1 (SGD) Number of epochs = 100 Validation MAPE: 87.2116 Testing MAPE: 98.2582	

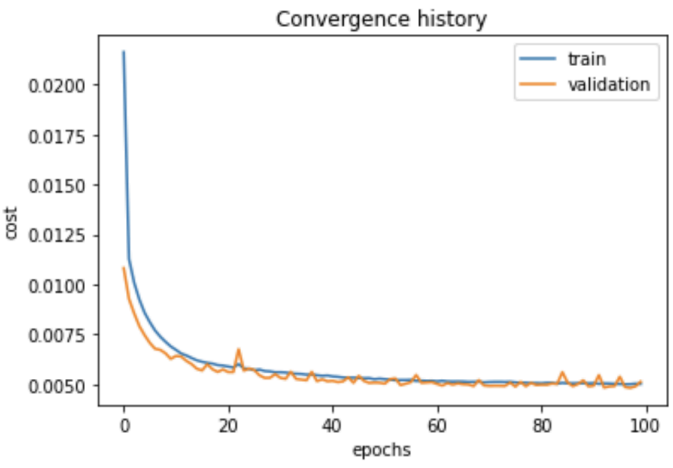
Activation function: Relu Mini batch size = 64 umber of epochs = 2000 Validation MAPE: 90.2026 Testing MAPE:102.2826	

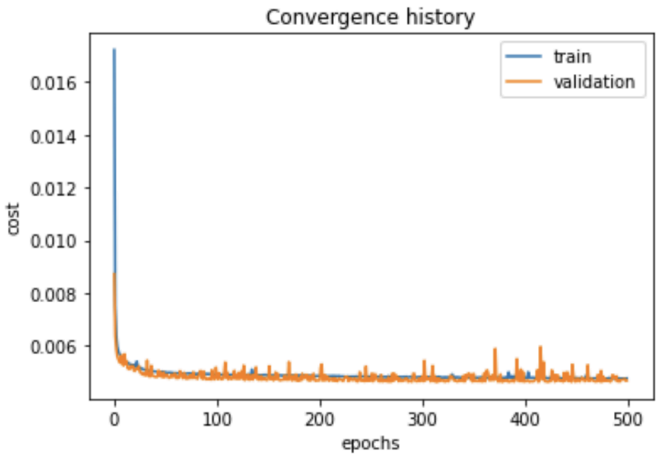
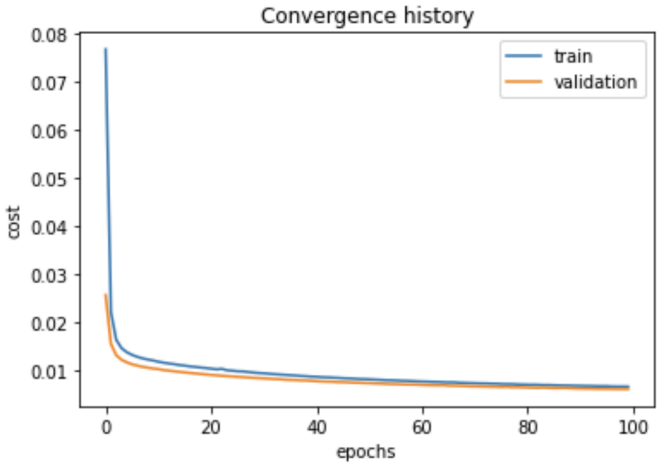
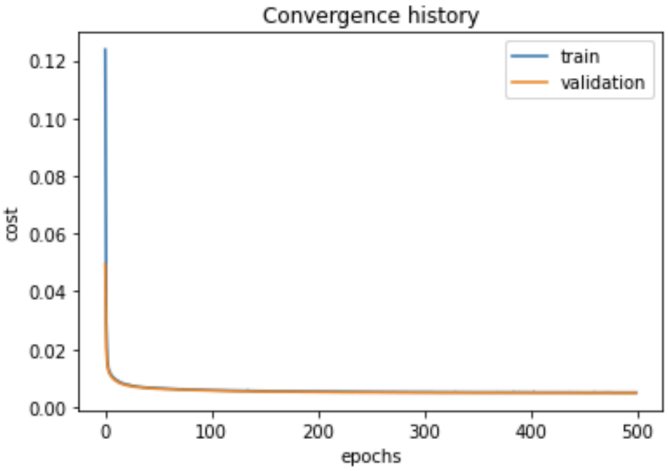
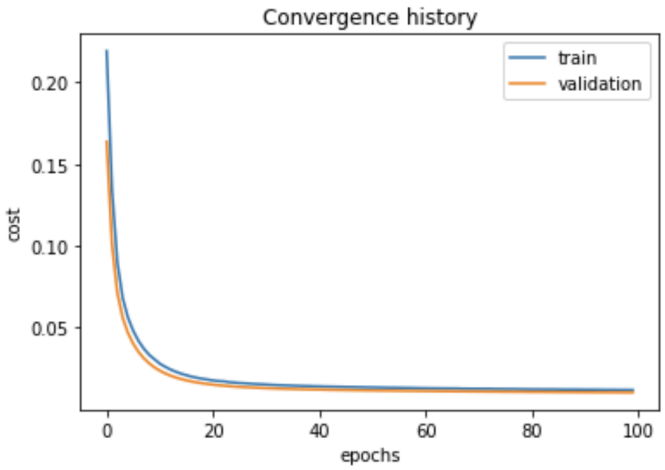
d) Comparison of Learning Rate:

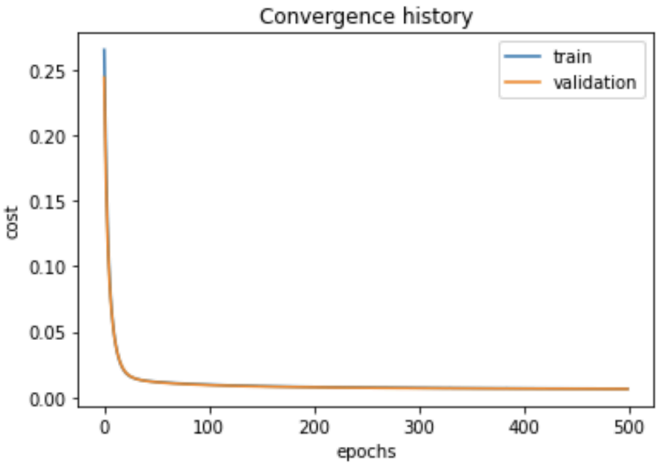
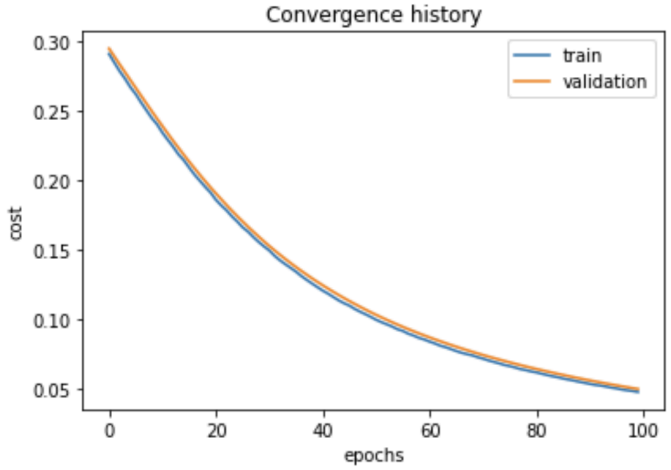
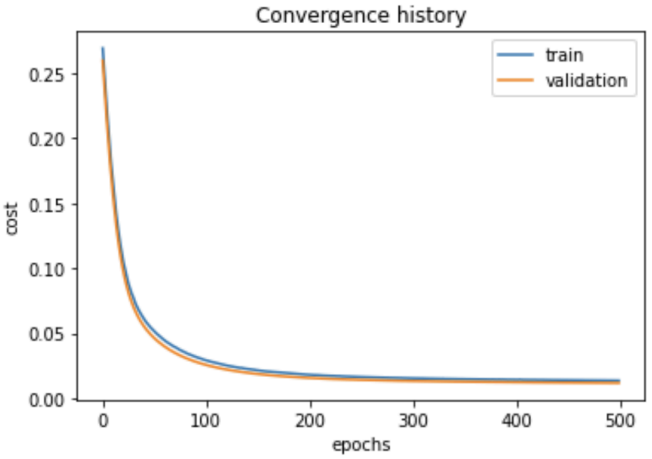
Parameters:

Architecture: 4, 10, 10, 1
Mini batch size: 32
Activation Function: Tanh
Regularization: λ = 0

Plotting convergence graphs for 100 and 500 epochs.

Learning rate: 0.1 **No. of Epochs**: 100 **Validation MAPE**: 54.696 **Testing MAPE:** 84.6634	

Learning rate: 0.1 **No. of Epochs**: 500 **Validation MAPE**: 54.696 **Testing MAPE:** 84.6634	
Learning rate: 0.01 No. of Epochs: 100 Validation MAPE: 65.6389 Testing MAPE: 96.6933	
Learning rate: 0.01 No. of Epochs: 500 Validation MAPE: 54.0433 Testing MAPE: 79.9158	
Learning rate: 0.001 No. of Epochs: 100 Validation MAPE: 94.9266 Testing MAPE: 145.0167	

Learning rate: 0.001 No. of Epochs: 500 Validation MAPE: 108.1817 Testing MAPE: 141.3471	
Learning rate: 0.0001 No. of Epochs: 500 Validation MAPE: 89.3991 Testing MAPE: 141.8189	
Learning rate: 0.0001 No. of Epochs: 500 Validation MAPE: 101.6059 Testing MAPE: 156.4004	

Observations:

- As the learning rate decreases, the gradient descent steps happen in smaller increments, which means the model converges slowly. So, a higher number of iterations are required to reach the minima.
- When the learning rate is 0.1 (large), we can see that the cost function value decreases, but with a lot of fluctuations over the iterations. These fluctuations are a result of choosing a learning rate that is too large – the gradient descent algorithm could easily converge towards a local minima or miss the minima entirely by taking large steps.
- When the learning rate is 0.0001 (very small), the model does not converge well even after 500 iterations. Convergence is reached only near 1000 iterations.

e) Comparison of L2 Regularization parameter (λ)

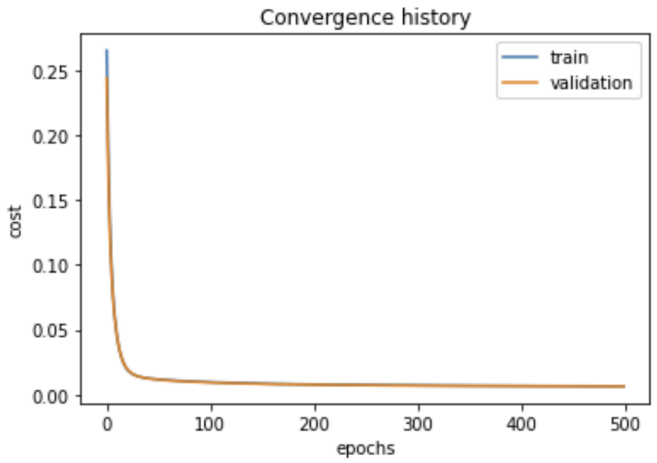
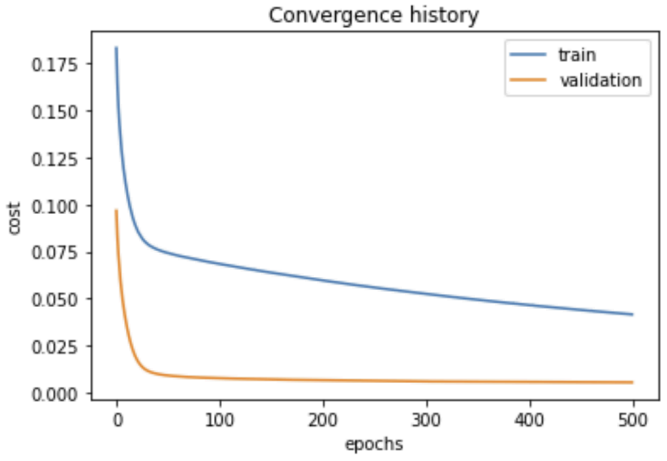
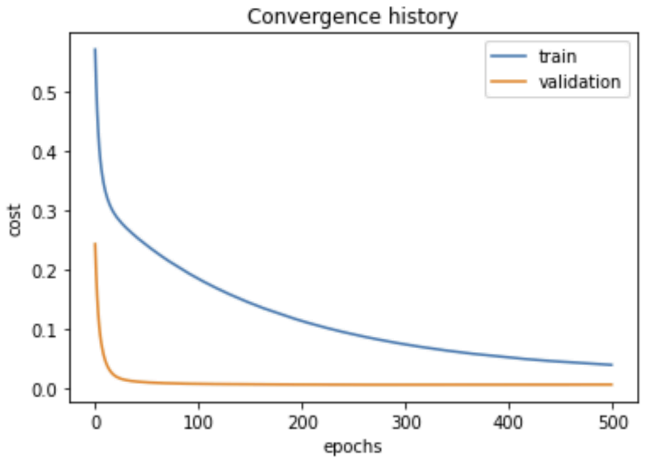
Fixed parameters:

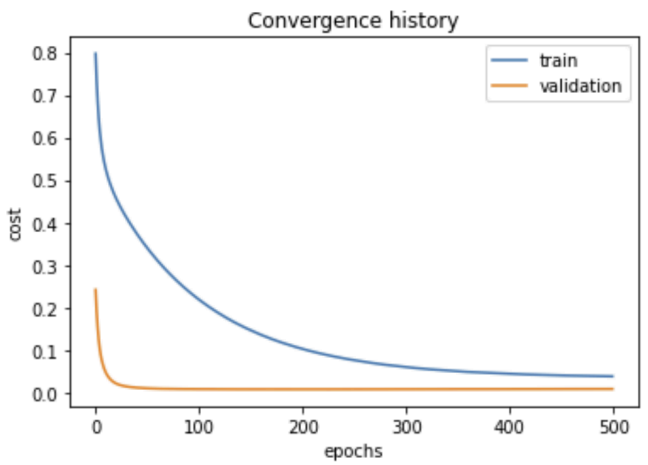
Architecture: 4, 10, 10, 1 Activation Function: Tanh

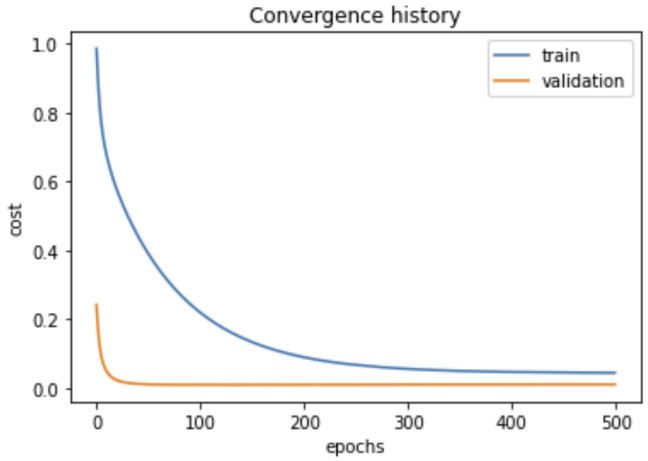
Mini batch size: 32

No. of Epochs: 500

Learning rate: 0.001

λ = 0 Validation MAPE: 63.8945 Testing MAPE: 90.3902	
λ = 0.1 Validation MAPE: 65.1045 Testing MAPE: 98.4157	
λ = 0.4 Validation MAPE: 53.8045 Testing MAPE: 81.3025	

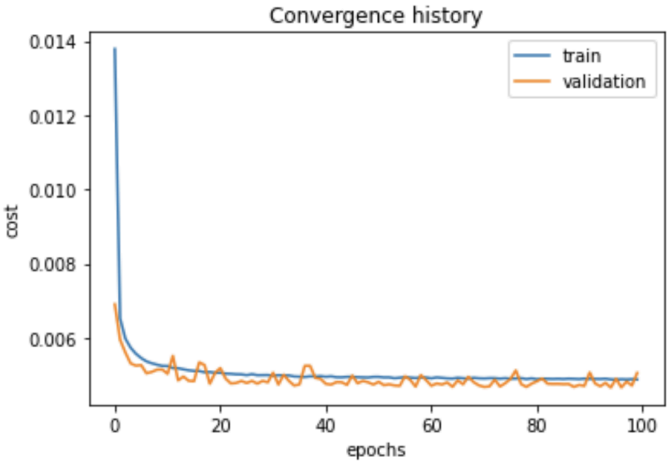
λ = 0.7 Validation MAPE: 55.3045 Testing MAPE: 79.8045	

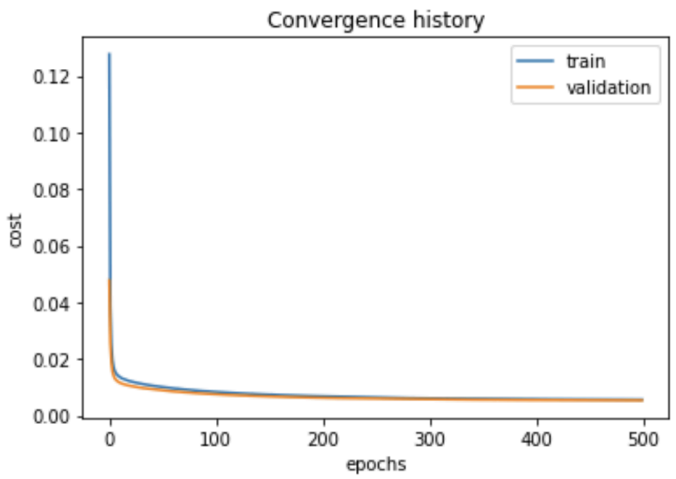
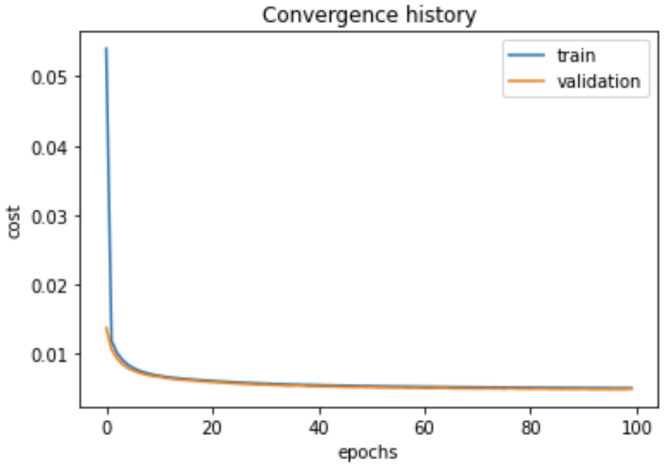
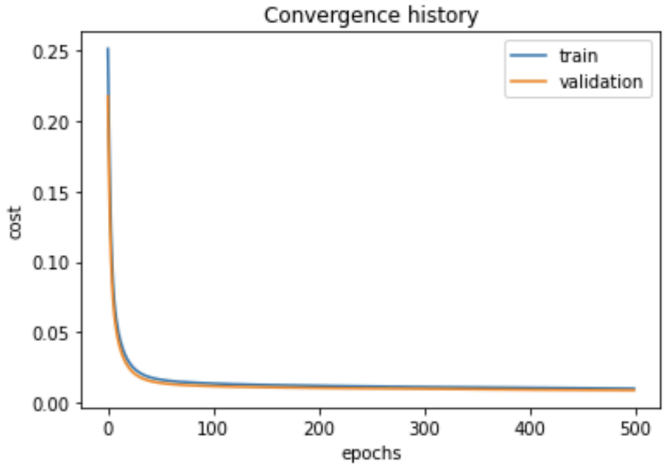
λ = 0.95 Validation MAPE: 55.9345 Testing MAPE: 78.4827	

f) SGD with momentum:

β = 0.9

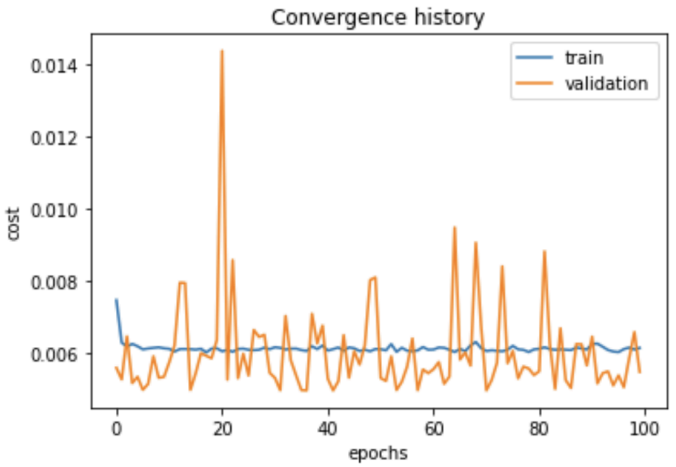
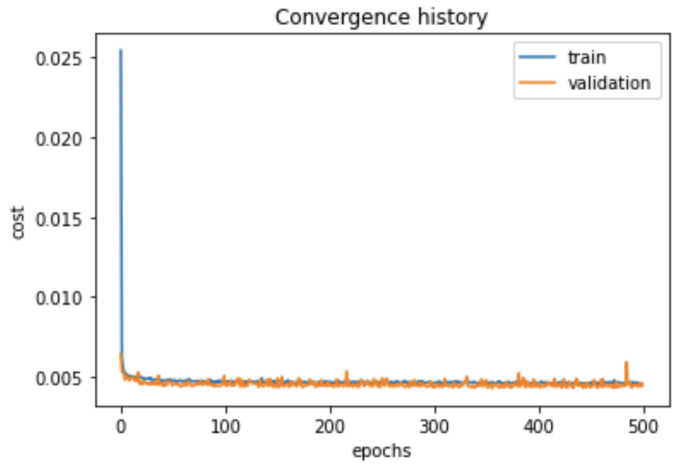
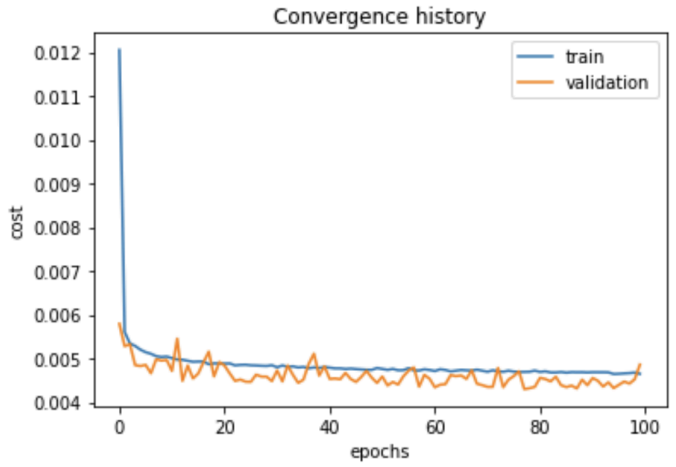
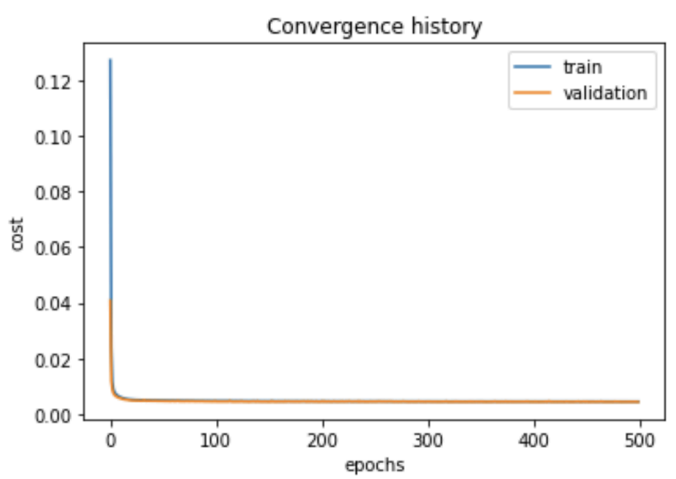
Activation Function: Tanh

Learning Rate: 0.01 No. of Epochs: 100 Minibatch Size: 1 Validation MAPE: 53.1036 Testing MAPE: 80.6925	

Learning Rate: 0.01 No. of Epochs: 500 Minibatch Size: 64 Validation MAPE: 60.1905 Testing MAPE: 89.9501	
Learning Rate: 0.001 No. of Epochs: 100 Minibatch Size: 1 Validation MAPE: 51.2673 Testing MAPE: 78.1974	
Learning Rate: 0.001 No. of Epochs: 500 Minibatch Size: 64 Validation MAPE: 83.2491 Testing MAPE: 130.8573	

g) BONUS: Adam optimization over SGD with momentum

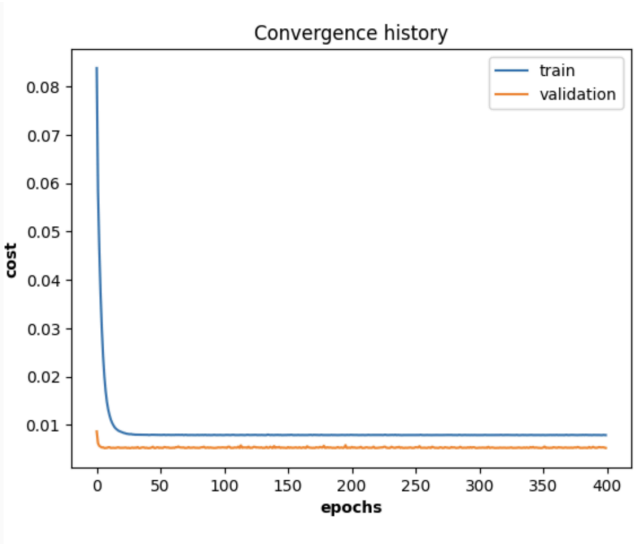
β1 = 0.9
β2 = 0.99
Activation Function: Tanh

Learning Rate: 0.01 No. of Epochs: 100 Minibatch Size: 1 Validation MAPE: 58.3384 Testing MAPE: 91.3311	
Learning Rate: 0.01 No. of Epochs: 500 Minibatch Size: 64 Validation MAPE: 48.9806 Testing MAPE: 75.6091	
Learning Rate: 0.001 No. of Epochs: 100 Minibatch Size: 1 Validation MAPE: 51.9904 Testing MAPE: 71.1443	
Learning Rate: 0.001 No. of Epochs: 500 Minibatch Size: 64 Validation MAPE: 54.9472 Testing MAPE: 72.8302	

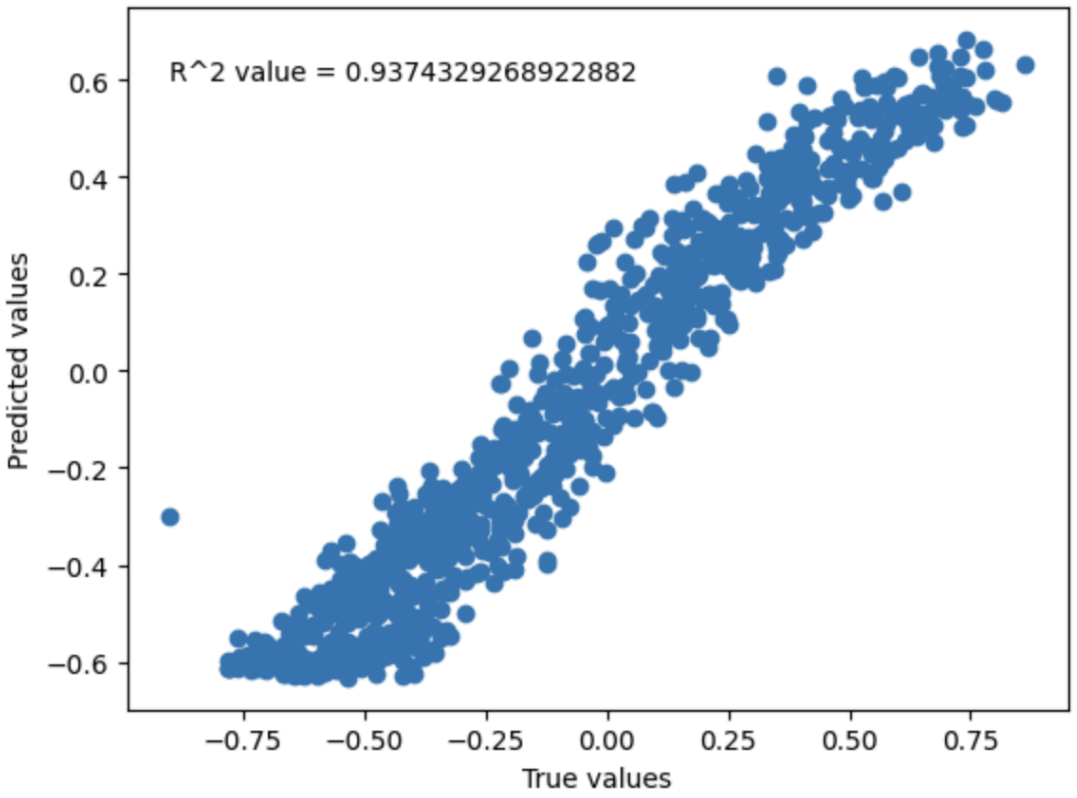
Observations:

• Adam optimization produces the lowest MAPE values. Convergence of training error is much faster than any of the previous models. There is a much higher rate of fluctuation in validation error.
• Both SGD with momentum and Adam converge much more smoothly with a smaller learning rate. Learning rate = 0.0001 would be ideal.
• Using these methods also produces much lower MAPE values.

Final model for CCPP dataset:

Architecture = [4,20,20,1] **No. of iterations** = 400 **Activation function** = tanh **Mini batch size** = 64 **Learning rate** = 0.001 **L2 regularization with** λ = 0.1 **Adam optimizer with** β = 0.9, β1 = 0.9, β2 = 0.99, epsilon = 1e-8 **Validation MAPE**: 59.52579209438036 **Test MAPE**: 94.24365607558214	

```
r2 score for perfect model is 0.9374329268922882
```



Language: Python 3
Dependencies: Numpy, Pandas, Matplotlib

There are two parts of this assignment:

Part 1: toy Sine Function

Hyperparameters and other options can be set in the config_toy.py file.
Main program file: sine.py
To run: python sine.py

Part 2: CCPP Dataset

Hyperparameters and other options can be set in the config_ccpp.py file.
Main program file: ccpp.py
To run: python ccpp.py