**CS 457 Deep Learning Assignment 1 Report**

Team number 15:

1. Shagufta Anjum (19XJ1A0568)
2. Sriyugesh B (19XJ1A0512)

**Part 1: Toy Problem – Sine Function**

Architecture: [1, 20, 20, 1]

Learning Rate: 0.01  
No. of Epochs: 1000  
Activation Function: Tanh  
Minibatch Size: 64  
Regularization: None  
Optimizer: None

Graphical user interface

Description automatically generated with low confidenceDiagram

Description automatically generated with medium confidence

Learning Rate: 0.01  
No. of Epochs: 5000  
Activation Function: Tanh  
Minibatch Size: 256  
Regularization: None  
Optimizer: None

A picture containing shape

Description automatically generated A picture containing diagram

Description automatically generated

Learning Rate: 0.01  
No. of Epochs: 5000  
Activation Function: Tanh  
Minibatch Size: 1000  
Regularization: None  
Optimizer: Adam

A picture containing square

Description automatically generated A picture containing shape

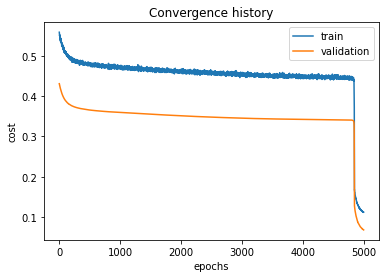
Description automatically generated

Learning Rate: 0.001  
No. of Epochs: 1000  
Activation Function: Tanh  
Minibatch Size: 1  
Regularization: None  
Optimizer: None  
A picture containing graphical user interface

Description automatically generatedA picture containing diagram

Description automatically generated

Learning Rate: 0.001  
No. of Epochs: 5000  
Activation Function: Tanh  
Minibatch Size: 64  
Regularization: L2  
Optimizer: None

 A picture containing logo

Description automatically generated

Learning Rate: 0.001  
No. of Epochs: 1000  
Activation Function: Tanh  
Minibatch Size: 64  
Regularization: L2  
Optimizer: Adam

A picture containing shape

Description automatically generated Diagram

Description automatically generated

Learning Rate: 0.001  
No. of Epochs: 5000  
Activation Function: Tanh  
Minibatch Size: 64  
Regularization: L2  
Optimizer: Adam

**A picture containing shape

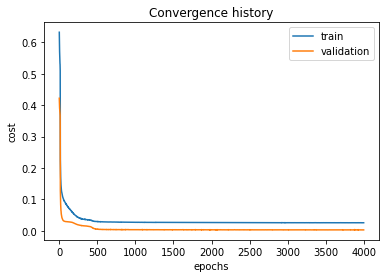
Description automatically generated** **Diagram

Description automatically generated**

**Final model for Sine function data:**

Architecture = [1,20,20,1]  
No. of iterations = 4000  
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: 10.780018695103752

**** **Diagram

Description automatically generated**

R2 value = 0.9958201105769453

**Chart

Description automatically generatedPart 2: Combined Cycle Power Plant Dataset**

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

1. **Comparison of ANN architecture**

Fixed parameters:

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

|  |  |
| --- | --- |
| **Architecture: 4, 5, 1**  Validation MAPE: 62.2584  Testing MAPE: 100.1120 |  |
| **Architecture: 4, 10, 1**  Validation MAPE: 58.5736  Testing MAPE: 90.9472 |  |
| **Architecture: 4, 5, 5, 1**  Validation MAPE: 59.3819  Testing MAPE: 89.9444 |  |
| **Architecture: 4, 10, 10, 1**  Validation MAPE: 54.2401  Testing MAPE: 84.2245 |  |
| **Architecture: 4, 10, 5, 10, 1**  Validation MAPE: 60.5116  Testing MAPE: 91.5711 |  |
| **Architecture: 4, 10, 10, 10, 1**  Validation MAPE: 59.9611  Testing MAPE: 95.7404 |  |

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.

1. **Comparison of mini batch size**

Fixed parameters:

Architecture: 4, 10, 10, 1  
Learning Rate: 0.001  
Activation Function: Tanh

|  |  |
| --- | --- |
| **Mini Batch Size: 1 (SGD)**  No. of Epochs: 100  Validation MAPE: 56.2401  Testing MAPE: 86.2245 |  |
| **Mini Batch Size: 64**  No. of Epochs: 100  Validation MAPE: 101.6112  Testing MAPE: 156.3929 |  |
| **Mini Batch Size: 64**  No. of Epochs: 300  Validation MAPE: 90.4310  Testing MAPE: 139.5851 |  |
| **Mini Batch Size: 64**  No. of Epochs = 10,000  Validation MAPE: 52.8013  Testing MAPE: 78.1437 |  |
| **Mini Batch Size: 256**  No. of Epochs: 100  Validation MAPE:  Testing MAPE: |  |
| **Mini Batch Size: 256**  No. of Epochs: 1000  Validation MAPE: 84.2379  Testing MAPE: 118.2521 |  |
| **Mini Batch Size: 6888 (Full batch)**  No. of Epochs: 1000  Validation MAPE: 110.2652  Testing MAPE: 142.7225 |  |

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.

1. **Comparison of Activation functions**

Fixed parameters:

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: 56.2401  Testing MAPE: 86.2245 |  |
| **Activation function: Tanh**  Mini batch size = 64  No. of Epochs = 2000  Validation MAPE: 56.2401  Testing MAPE: 86.2245 |  |
| **Activation function: Sigmoid**  Mini batch size = 1 (SGD)  Number of epochs = 100  Validation MAPE: 85.5745  Testing MAPE: 96.4624 |  |
| **Activation function: Sigmoid**  Mini batch size = 64  Number of epochs = 2000  Validation MAPE: 91.2648  Testing MAPE: 105.0215 |  |
| **Activation function: Relu**  Mini batch size = 1 (SGD)  Number of epochs = 100  Validation MAPE: 88.3112  Testing MAPE: 99.2806 |  |
| **Activation function: Relu**  Mini batch size = 64  umber of epochs = 2000  Validation MAPE:  Testing MAPE: |  |

Observations:

* Convergence is fastest when the relu activation function is used, both with and without the use of mini batches. Using relu also makes computations faster.
* The convergence graph is less smooth in relu than in other functions.
* The MAPE value is much lower when tanh is used as compared to sigmoid and relu.

1. **Comparison of Learning Rate**

Fixed 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: 54.2555  Testing MAPE: 82.6850 |  |
| **Learning rate: 0.1**  No. of Epochs: 500  Validation MAPE: 54.3445  Testing MAPE: 79.1349 |  |
| **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.

1. **Comparison of L2 Regularization parameter (λ)**

Fixed parameters:

Architecture: 4, 10, 10, 1  
Mini batch size: 32  
No. of Epochs: 500  
Learning rate: 0.001  
Activation Function: Tanh

|  |  |
| --- | --- |
| **λ = 0**  Validation MAPE: 62.9812  Testing MAPE: 89.4093 |  |
| **λ = 0.1**  Validation MAPE: 63.0973  Testing MAPE: 97.3926 |  |
| **λ = 0.4**  Validation MAPE: 54.7606  Testing MAPE: 82.2339 |  |
| **λ = 0.7**  Validation MAPE: 56.2918  Testing MAPE: 78.9496 |  |
| **λ = 0.95**  Validation MAPE: 56.8477  Testing MAPE: 77.3624 |  |

Observations:

* The convergence is significantly slower with the use of regularization. However, the errors do start converging to much smaller values than they did without regularization. This is due to prevention of overfitting, which allows the errors to drop lower without stagnation.
* Hence, for higher values of λ, more iterations are required to reach convergence, since the degree of bias introduced would be higher.
* The MAPE values are slightly lower with the use of regularization but are not affected much by changes in λ.

1. **SGD with momentum**

β = 0.9  
Activation Function: Tanh

|  |  |
| --- | --- |
| Learning Rate: 0.01 No. of Epochs: 100 Minibatch Size: 1  Validation MAPE: 54.0310  Testing MAPE: 79.5138 |  |
| Learning Rate: 0.01 No. of Epochs: 500  Minibatch Size: 64  Validation MAPE: 59.1800  Testing MAPE: 88.9355 |  |
| Learning Rate: 0.001 No. of Epochs: 100  Minibatch Size: 1  Validation MAPE: 53.3872  Testing MAPE: 77.0754 |  |
| Learning Rate: 0.001 No. of Epochs: 500  Minibatch Size: 64  Validation MAPE: 84.2041  Testing MAPE: 132.6824 |  |

1. **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: 57.3343  Testing MAPE: 90.2311 |  |
| Learning Rate: 0.01 No. of Epochs: 500  Minibatch Size: 64  Validation MAPE: 47.8706  Testing MAPE: 74.5961 |  |
| Learning Rate: 0.001 No. of Epochs: 100  Minibatch Size: 1  Validation MAPE: 50.9972  Testing MAPE: 72.0441 |  |
| Learning Rate: 0.001 No. of Epochs: 500  Minibatch Size: 64  Validation MAPE: 51.8346  Testing MAPE: 73.7396 |  |

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: 56.4907 Test MAPE: 86.7422 |  |

R2 score for the model’s predictions = 0.9374672332628895  
Chart, scatter chart

Description automatically generated

**Code structure:**

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',c ost\_func=cost\_func) -- compute the validation cost
* plt.plot(costs) -- plot the training and validation costs