EEE 443/543 - Project #4

Q1)

In this project the task was to design and train a neural network in order to do curve fitting. The first task to randomly draw $x_1,\ldots,x_n$ uniformly form [0, 1] and $v_1,\ldots,v_n$ uniformly form [-0.1, 0.1] where n = 300.

In order to get the same result for each run the numpy random seed is fixed:

```
np.random.seed(57)
```

After that, $d_i = sin(20x_i) + 3x_i + v_i$ are calculated and (x_i, d_i) points are plotted where $i = 1,..,n$.

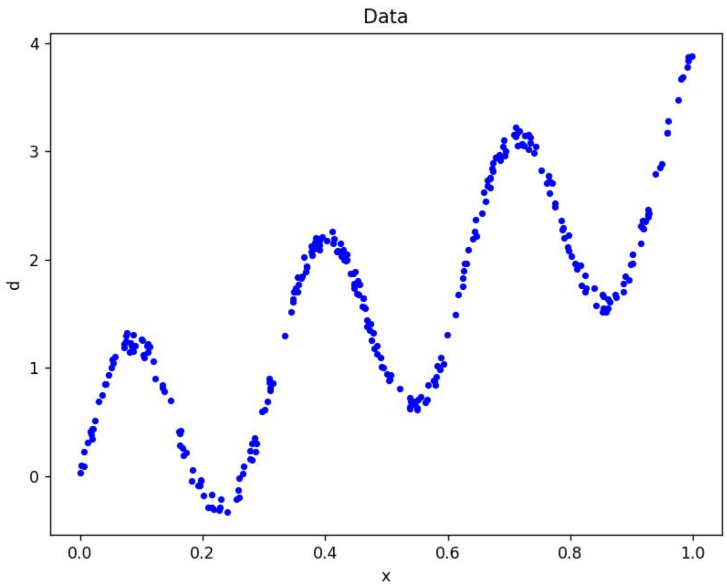

Fig.1 $d_i = sin(20x_i) + 3x_i + v_i$

Psudocode:

1. Initialize:
    - **1.1** Number of neurons in hidden layer (N)
    - **1.2** Learning rate (eta)
    - **1.3** Weights (weights_input, weights_hidden)
    - **1.4** Biases (bias_input, bias_hidden)
2. Define Activation Function:
    - **2.1** tanh(x) = (e^x - e^(-x)) / (e^x + e^(-x))
    - **2.2** tanh_derivative(x) = 1 - tanh(x)^2
3. Training Loop (for epoch in 1 to max_epochs):
    - **3.1** Initialize mean squared error (MSE) to 0
    - **3.2** For each training sample (xi, di):
        - **3.2.1** Forward Pass:
            - **3.2.1.1** Compute hidden layer input: hidden_input = weights_input * xi + bias_input
            - **3.2.1.2** Compute hidden layer output: hidden_output = tanh(hidden_input)

3.2.1.3 Compute final output: output = weights_hidden * hidden_output + bias_hidden
3.2.1.4 Compute error: error = di - output
3.2.1.5 Update MSE: MSE += error^2
3.2.2 Backward Pass (Gradient Descent Updates):
3.2.2.1 Compute gradient at output layer: delta_output = error
3.2.2.2 Compute gradient at hidden layer: delta_hidden = tanh_derivative(hidden_input) * (weights_hidden^T * delta_output)
3.2.2.3 Update output layer weights: weights_hidden += eta * delta_output * hidden_output^T
3.2.2.4 Update output layer bias: bias_hidden += eta * delta_output
3.2.2.5 Update input layer weights: weights_input += eta * delta_hidden * xi^T
3.2.2.6 Update input layer bias: bias_input += eta * delta_hidden
3.3 Compute MSE for the epoch: MSE = MSE / total_samples
3.4 If MSE increases, reduce learning rate: eta = eta * 0.9 (adaptive learning rate)
4. Return the trained weights and biases

The MSE versus number of epochs plot is given below in *Fig.2* , it is clear that as the algorithm runs the MSE goes to zero, which shows that the BP algorithm, in fact, converges to the best fit.

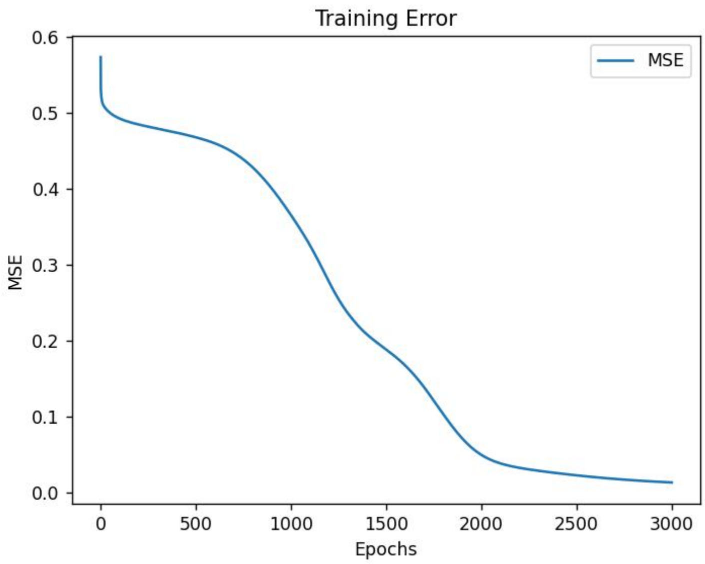

Fig.2 The number of epochs vs the MSE in the backpropagation algorithm

The plot of the curve *(x, f(x,* $\boldsymbol{w_0}$*)), x* $\in$ *[0,1],* where $\boldsymbol{w_0}$ is the weights that the backpropagation algorithm has converged to, is visible in *Fig.3* :

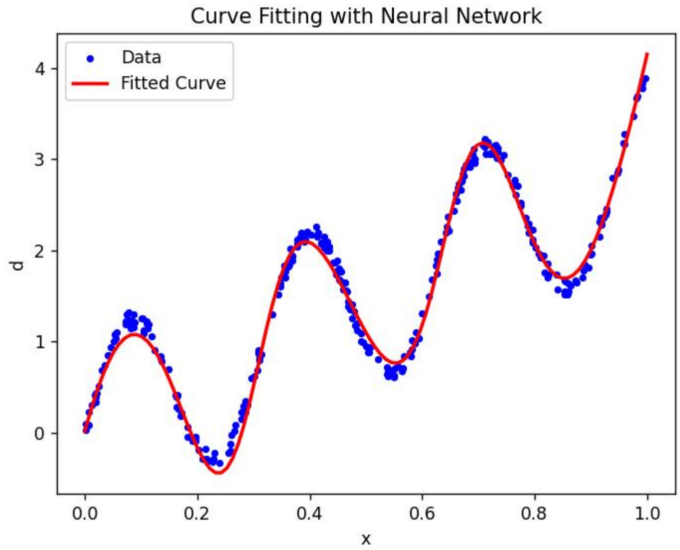

Fig.3 The "best" fit