Pseudo-code to implement a Neural Network to fit a curve

a. Draw n = 300 real numbers uniformly at random on [0, 1], call them $x_1, x_2, \ldots, x_n$.
b. Draw n = 300 real numbers uniformly at random on [-1/10,1/10] call them $v_1, v_2, \ldots\ldots, v_n$.
c. Let di = sin(20 * x_i) + 3x_i + v_i for i = 1, . . . , n
d. Plot the points (x_i, d_i), i = 1, . . . , n
e. Let there be 1 neuron in the input layer, 24 neurons in the first hidden layer and 1 neuron in the output layer. we wish to find (24 * 2) weights in the first hidden layer and (25 * 1) weights at the output layer which are including the bias.
f. We will use the 300 input samples that are generated in step 1 and find the output corresponding to each input.
g. The algorithm will be:
   1. Initialize $W_1 \in \mathbb{R}_{(24 \times 2)}$ including the bias, where weights are normally distributed between (1,1) and bias is normally distributed between (0,1)
   2. Initialize $W_2 \in \mathbb{R}_{(1 \times 25)}$ including the bias where weights are normally distributed between (1,1) and bias is normally distributed between (0,1)
   3. Let X = [$x_1, x_2, \ldots, x_n$], n = 300 be the set of input samples
   4. Let D = [$d_1, d_2, \ldots, d_n$], n = 300 be the set of desired output for the given input
   5. Initialize η = 0.01 and ε = 0.001
   6. Initialize epoch = 0
   7. Initialize errors[epoch] = 0 for epoch in 0,1,2....

   8. Do (This loop is where we iterate the algorithm until convergence)

      8.1 for i in 1 to n (300) do (This loop is where we perform the forward and backward propagation)

      ---Forward Propagation---

      8.1.1 Calculate the induced local field v_1 with current input samples and weights W_1. i.e $v_1 = W_1 * [1, x_1] \in \mathbb{R}_{(24 \times 1)}$, where $W_1 \in \mathbb{R}_{(24 \times 2)}$ and $x_i \in \mathbb{R}_{(1 \times 1)}$ is i_{th} training sample

      8.1.2 The neurons in the hidden layer will use the tanh(v) activation function. Let $y_1 = \Phi(v_1)$ where $\Phi(v_1) = \tanh(v_1)$, here y_1 is the output of the neuron in the hidden layer

      8.1.3 Calculate the induced local field v_2 with inputs from first hidden layer and weights W_2. i.e. $v_2 = W_2 * [1, y_1] \in R[1,1]$, where $W_2 \in \mathbb{R}_{(1 \times 25)}$ and y1 $\in \mathbb{R}_{(25 \times 1)}$, where y_1 is the input from the first hidden layer.

      8.1.4 The neurons at the output layer will use the (v) activation function. Let $y_2 = \Phi(v_2)$ where $\Phi(v_2) = v_2$, here y_2 is the final output of the network = f(x,w)

      8.1.5 Define Energy E = 1/2 (d – y_2) , where d is desired response for the same and y_2 is the output obtained by the network

      ---Back Propagation---

      8.1.5 Calculate the signal at the output using $\delta_2 = (D_i - y_2)$ where D_i is the desired output for input x_i and y_2 is the actual output of the network

      8.1.6 Calculate the signal at the output of 1st hidden layer $\delta_1 = (W_2 * \delta_2) * \Phi'(v_1)$ where $\Phi'(v_1)$ is the derivative of the function $\Phi(v_1)$ i.e. $\Phi'(v_1) = \partial\Phi(v_1)/\partial v = (1 - \tanh(v_1))$ ** 2)

      8.1.7 Calculate the partial derivatives of the energy function with respect to the weights

8.1.7.1 $\partial E/\partial W_1 = -\delta_1 * [1,x]T$.

8.1.7.2 $\partial E/\partial W_2 = -\delta_2 * [1,y_1]T$.

--Update Weights---

8.1.8 Update the weights W_1 and W_2 based on the eta provided and the partial derivatives calculated earlier

8.1.8.1 $W_1 \leftarrow W_1 + (\eta * (\delta_1 * [1,x]T))$

8.1.8.2 $W_2 \leftarrow W_2 + (\eta * (\delta_2 * [1,y_1]T))$

8.2 Calculate the Mean Squared Error using the formula MSE = $((D_i - Y2_i) ** 2 / n)$ for i in 1,2,3... n where D is the set of desired values and Y_2 is the set if output values obtained inn 8.1

8.3 The algorithm may not always result in a monotonically decreasing MSE, if you find the MSE has. Increased from the previous iteration i.e. MSE[epoch] > MSE[epoch - 1]

then decrease the learning rate $\eta \leftarrow \eta * 0.9$

8.4 epoch <- epoch + 1

8.5 Loop to 8 until MSE <= ε

---Algorithm Converges---

9. Return the Mean Squared Error at each epoch and the final weights W_1 and W_2

h. Use the weights that were returned from the above algorithm and perform the forward propagation in the network to get the predicted output for each of the input samples

1. Calculate the induced local field v1 with current input samples and weights W1. i.e $v_1 = W_1*[1,x_1] \in \mathbb{R}_{(24 \times 1)}$, where $W_1 \in \mathbb{R}_{(24 \times 2)}$ and $x_i \in \mathbb{R}_{(1 \times 1)}$ is i_{th} training sample
2. The neurons in the hidden layer will use the tanh(v) activation function. Let $y_1 = \Phi(v_1)$ where $\Phi(v_1) = \tanh(v_1)$, here y_1 is the output of the neuron in the hidden layer
3. Calculate the induced local field v_2 with inputs from first hidden layer and weights W_2. i.e $v_2 = W_2*[1,y_1] \in \mathbb{R}_{(1 \times 1)}$, where $W_2 \in \mathbb{R}_{(1 \times 25)}$ and $y_1 \in \mathbb{R}_{(25 \times 1)}$, where y_1 is the input from the first hidden layer.
4. The neurons at the output layer will use the (v_2) activation function. Let $y_2 = \Phi(v_2)$ where $\Phi(v_2) = v_2$, here y_2 is the final output of the network = f(x,w)

i. Plot the points $(x_i, Y2_i)$, i = 1, . . . , n, where x_i is the set of input sample and $Y2_i$ is the predicted from the network

j. The plots in [c] and [i] should be very similiar(it should be a good fit) as we have trained the network using the samples and adjusted the weights

k. Plot the number of epochs vs the MSE in the backpropagation algorithm.