CS 559: Neural Networks

Homework 4

Author:
Gaetano Coppoletta
Email:
gcoppo2@uic.edu

September 29, 2022

1 Introduction

In this exercise we will use a neural network for curve fitting. First we draw n=300 real numbers chosen uniformly at random on [0,1], those are $x_1, ..., x_n$.
Then we draw n real numbers uniformly at random $[-1/10, 1/10]$, those are called $v_1, ..., v_n$. Let $d_i = sin(20x_i) + 3x_i + v_i$, i=1,...,n. We plot the points(x_i, v_i), i=1,...,n.
The result is listed below.

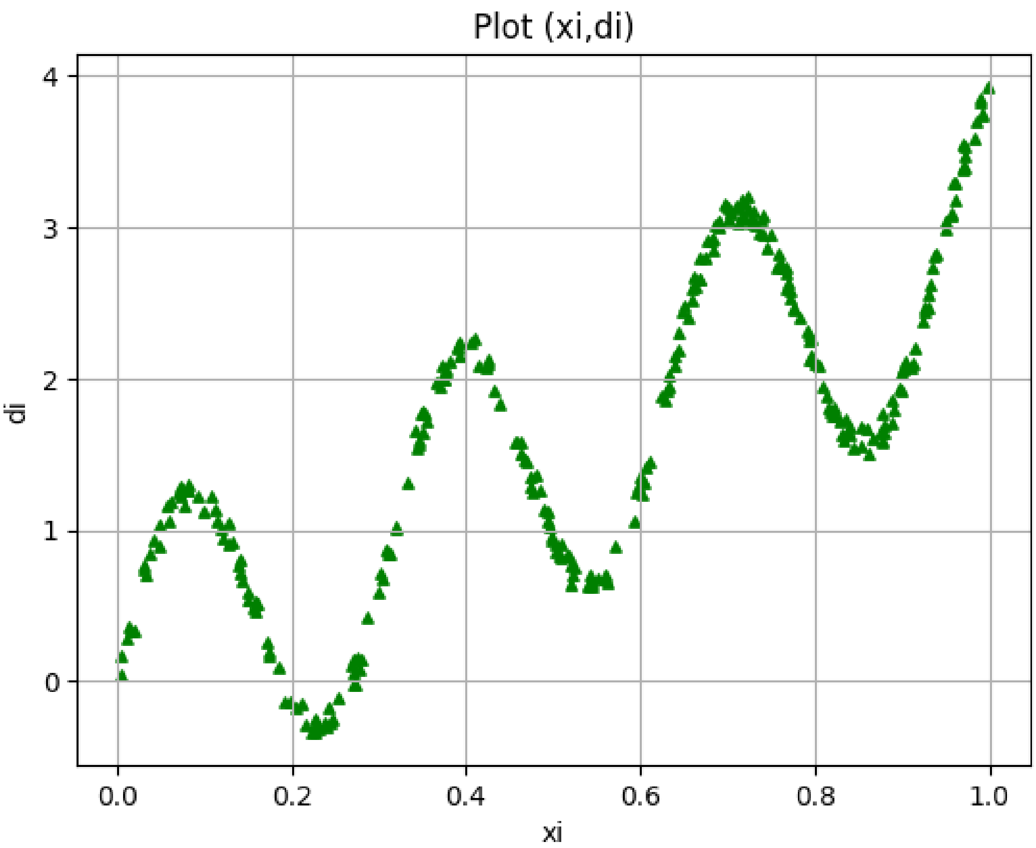

Figure 1: Plot of xi,di

2 The Neural network

We will consider a 1xNx1 neural network, with one input, N=24 hidden neurons and one output neuron. The network will have 3N+1 weights including the biases. Let **w** denote the vector of all of these 3N+1 weights. The output neuron will use $\Phi(v) = v$. All other neurons will use the activation function $\Phi(v) = tanh(v)$. Given input x, we use the notation $f(x, w)$ to represent the network output. Use the backpropagation algorithm with online learning to find the optimal weights/network that minimize the mean-squared error (MSE) $1/n \sum_{i=1}^{n}(d_i - f(x_i, w))^2$.

3 Pseudocode of the training algorithm

Algorithm 1 Training algorithm

$n \leftarrow 300$
$x \leftarrow$ 300 random numbers between $[0, 1]$
$v \leftarrow$ 300 random numbers between $[-1/10, 1/10]$
$d \leftarrow sin(20x) + 3x + v$
$N \leftarrow 24$
$\eta \leftarrow 0.01$
$W1_{Nx2} \leftarrow \begin{bmatrix} bias_1 & weight_1 \\ bias_2 & weight_2 \\ \vdots & \vdots \\ bias_N & weight_N \end{bmatrix}$ chosen randomly
$W2_{1xN+1} \leftarrow \begin{bmatrix} bias_{N+1} & weight_1 & weight_2 & \cdots & weight_N \end{bmatrix}$ chosen randomly
$epochs \leftarrow 10000$
for epoch from 0 to epochs **do**
  for i from 0 to n **do**
    $induced\_local\_field1 \leftarrow W_1 \begin{bmatrix} 1 \\ x_i \end{bmatrix}$
    $y_1 \leftarrow tanh(induced\_local\_field1)$
    $y_2 \leftarrow \begin{bmatrix} 1 \\ y_1 \end{bmatrix}$
    $induced\_local\_field2 \leftarrow W_2 y_2$
    $output_i \leftarrow \Phi(induced\_local\_field2)$
    now we do the backpropagation
    $\delta_L \leftarrow d_i - output_i$
    $\delta_1 \leftarrow \underline{W_2^T \delta_L} \cdot (1 - tanh^2(inducted\_local\_field1))$
    update the weights
    $W_1 \leftarrow W_1 + \eta \delta_1 \begin{bmatrix} 1 \\ x_i \end{bmatrix}^T$
    $W_2 \leftarrow W_2 + \eta \delta_L y2$
  end for
end while

In the pseudocode, $\cdot$ is the elementwise product, while the underlined part is an operator that excludes the first component of the vector or matrix, we need this operator because of the biases, once we backpropagate signal for the biases we don't need these signals anymore and so we destroy them.

4 Results

Now we run the algorithm while the MSE goes under 0.01. We obtain the following results.

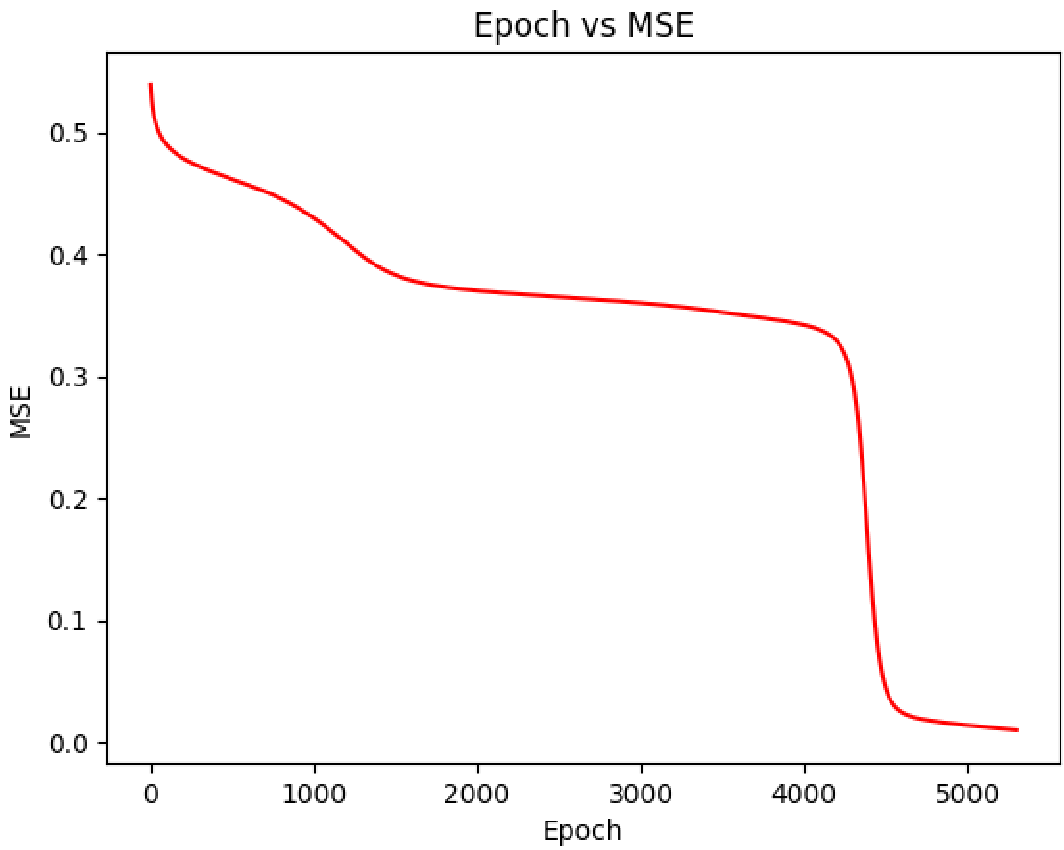

Figure 2: Epochs vs MSE

As we can see the MSE reach a value that is less than 0.01 after 5302 epochs.

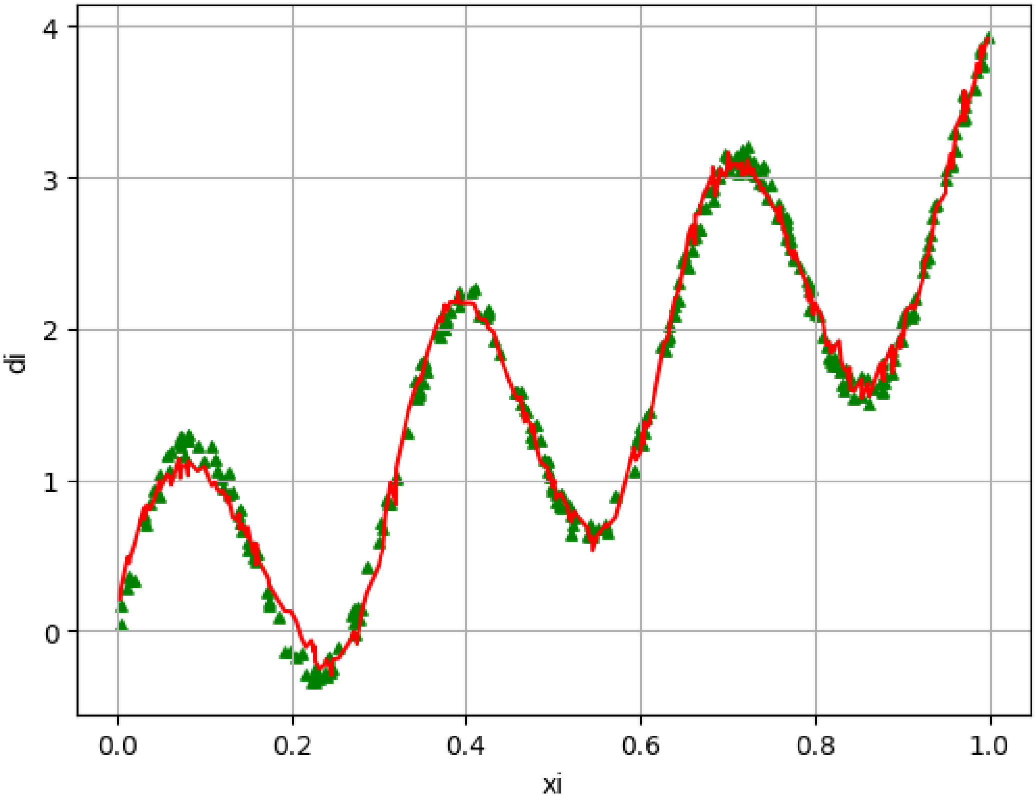

Figure 3: Result

The Figure 3 shows in red the curve obtained after the training of the neural network and in green the desired outputs, as we can see those are similar.

5 Code

```
import numpy as np
import matplotlib.pyplot as plt

def identity_function(x):
    return x

def mean_squared_error(d,f):
    return np.sum((d-f)**2)/300

def tanh_derivative(x):
    return 1- np.tanh(x)**2

if __name__ == '__main__':
    np.random.seed(234)
```

```
    n = 300
    x = np.random.uniform(0,1,(n))
    v = np.random.uniform(-1/10,1/10,(n))
    N=24
    d = np.sin(20*x)+3*x+v

    plt.title("Plot (xi,di)")

    plt.xlabel("xi")
    plt.ylabel("di")
    plt.grid()
    plt.scatter(x,d,color='green',marker='^',s=15)
    plt.show()

    eta=0.01



    W1 = np.random.randn(N,2) # weights + bias vector
    #col 0 = bias
    #col 1 = weights
    W2 = np.random.randn(1,N+1) # last bias + weights
    # | b | w0 | w1 | ... |w_N-1 |

    epochs=10000
    epoch_vs_mse = np.zeros((2,epochs))
    outputs = np.zeros(n)
    index=0
    epoch=0

    while(epoch!=epochs):
        epoch_vs_mse[0,epoch]=epoch
        for i in range(n):
            #induced local field 1
            vl1 = W1 @ [[1],[x[i]]]
            y1 = np.tanh(vl1)
            #induced local field 2
            y2 = np.append([1],y1)
            vl2 = np.dot(W2,y2)
            outputs[index]=identity_function(vl2)

            #backpropagation
            deltaL = d[i]-outputs[index]
            delta1 = np.multiply((np.dot(np.transpose(W2), deltaL.reshape(1,1)))[1:,:],tanh_derivative(vl1))
            index+=1

            #update the weights
            W1 = W1 + (eta * np.dot(delta1, [[1,x[i]]]))
            W2 = W2 + (eta* np.dot(deltaL, y2))


        epoch_vs_mse[1,epoch] = mean_squared_error(d,outputs)
```

```
        # if(epoch != 0 and epoch_vs_mse[1,epoch]>epoch_vs_mse[1,epoch-1]):
        #     eta=0.9*eta
         print(epoch,epoch_vs_mse[1,epoch])
         if(epoch_vs_mse[1,epoch]<0.01):
             print("End at epoch",epoch)
             break
         epoch+=1
         index=0
    #plot
    plt.title("Epoch vs MSE")
    plt.xlabel("Epoch")
    plt.ylabel("MSE")
    plt.plot(epoch_vs_mse[0,1:epoch],epoch_vs_mse[1,1:epoch],'r')
    plt.show()

    xs, ys = zip(*sorted(zip(x, outputs)))
    plt.xlabel("xi")
    plt.ylabel("di")
    plt.grid()
    plt.scatter(x,d,color='green',marker='^',s=15)
    plt.plot(xs,ys,'r')
    plt.show()
```