Neural Network for System Identification

– Project 1

Enes Erçin

April 2023

1 Aim

To train a single layer neural network using any adaptive algorithm, with the input of sinusoidal signals expecting model to predict a function that is close to sinusoidal. For this particular project adaptive algorithm chosen is famous back propagation algorithm. It is important to keep in mind there are infinite amount of different parameter combinations that can be tested, for this document only few of them are investigated. The outcome of the algorithm might be differ with different parameters, this document focuses on the results of the parameters chosen which are showed in parameters section. At the end of the experiment model did predict well defined sinusoidal function although the output seems perfect sinusoidal waveform at first maximum, minimum and mean values are different then the sinusoidal values.

2 Features

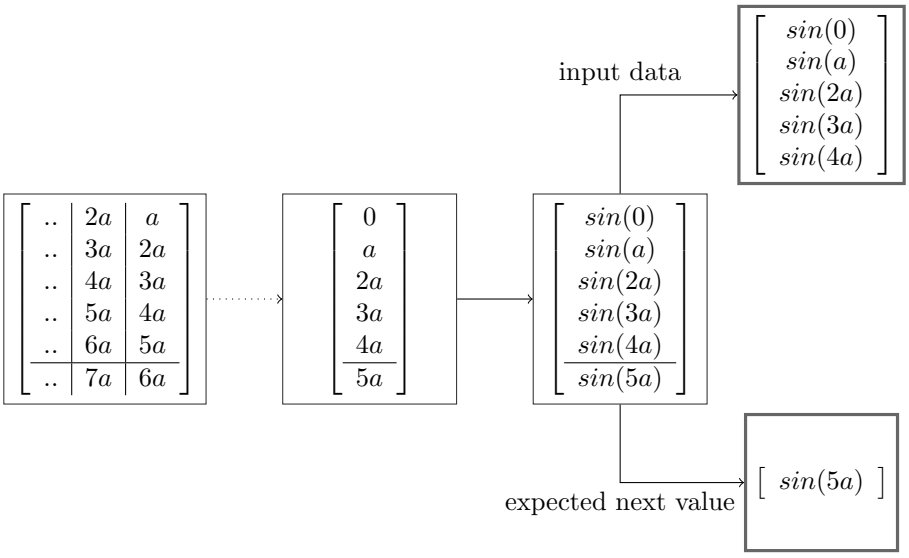
2.1 Parameters

Number of inputs:	5
The frequency of sinusoidal sampling:	100
Adaptive Algorithm:	Back propagation
Non-linear Activation Function:	Sigmoid
Number of Training Size:	2500
Learning Rate:	0.001
Cost Function:	Least Squares

Table 1: Parameters of the Model

2.2 Input

An array is initiated by consecutive numbers with the difference of defined sampling rate a $a = 2\pi/samplefreq$. After that array is used as an input for sinusoidal function $sin(a * t)$. For each training sequence first five of values of the array is chosen as input and the next one namely sixth data is chosen as expected value. Then second training starts with second value of the array and expected value for this case chosen as seventh value of the array.



2.3 Adaptive Algorithm: Back propagation

The concept of adaptive algorithms responsibility is to find the optimal parameters that will reduce the cost function. In this model of basic neural network it is only possible to change weights and biases. The process of defining a change value for bias and weights comes from particular choice of adaptive algorithm. In this project chosen adaptive algorithm is back propagation. Mindset behind the algorithm is simple there is a function that we want to reduce to its minimum value. That function is cost function. For every input we would get a different set of cost function results.

$$J(\theta) = \sum_{n=0}^{n=k} [y(\theta_n) - d)]^2$$

Since only parameter that are possible to change are weights and biases we must find its partial derivative with respect to cost function. To find $dJ(\theta)/dw, dJ(x)/db$ values chain rule must be applied.

$$y_n = y(\theta_n) = \sigma(w_n * \theta_n + b_n)$$

Lets say $= a_n = \sigma(w_n * \theta_n + b_n))$ for ease of representation.

$$\frac{dJ(\theta_n)}{dw_n} = \frac{dJ(\theta_n)}{y_n} \cdot \frac{dy_n}{da_n} \cdot \frac{da_n}{dw_n}$$

Similarly, also noticing that $\frac{da_n}{db_n} = 1$

$$\frac{dJ(\theta_n)}{db_n} = \frac{dJ(\theta_n)}{y_n} \cdot \frac{dy_n}{da_n}$$

It is a common practice to pick a learning rate which is close to 0 so that the most optimum weights and biases can be reached within reasonable speed. Basically smaller learning rates can give higher precision while gives a slower progress. It is also important to keep in mind it is possible to change the learning rate while training. Approach names ad adaptive learning rate. For this practice picked learning rate is 0.001. Thus weight and biases are adjusted with given formula.

$$w_n[1] = w_n[0] - 0.001 \cdot \frac{dJ(\theta_n)}{dw_n}$$

2.3.1 Code

```
function obj = backprop(obj)
    dC_dY = zeros(obj.input_num,1);

    for i = 1:obj.input_num
      dC_dY(i) = -2*(obj.d_cache-obj.y_cache(i)); %Cost Function -> dC/dY
    end

    dY_da = obj.y_cache.*(ones(obj.input_num,1)-obj.y_cache);  %Y -> dY/da
    da_dw = obj.input_cache;            %a -> da/dw
    dw = dC_dY.*dY_da.*da_dw;           %dw -> dC/dw -> dC/dY*dY/da*da/dw
    db = dC_dY.*dY_da;                  %dw -> dC/db -> dC/dY*dY/da*da/db
    obj.w = obj.w -obj.mu*dw;
    obj.b = obj.b -obj.mu*db ;
```

Figure 1: Caption

```
function [obj,exp] = feedforward(obj,x,exp,train)

      assert(length(x)==obj.input_num,"Input number should
      obj.input_cache = x;
      obj.a_cache =   obj.w.*x + obj.b; %Linear Layer
      for i = 1:obj.input_num
          obj.y_cache(i) =   sigmoid(obj.a_cache(i));
      end
      obj.d_cache =   exp;
      temp = 0;
      for i = 1:obj.input_num
          temp = temp + 0.5*(exp-obj.y_cache(i))^2;
      end

      obj.loss =     temp;
      if train == 1

          obj.count =   obj.count + 1;
          obj.cost_result(obj.count) = temp;
          obj.weigth_results(obj.count,:)  = obj.w;
          obj.bias_results(obj.count,:) = obj.b;
      end

end
```

Figure 2: Caption

3 Results

A function close to sinusoidal is estimated at the end of the epochs however maximum and minimum values of the estimated sinusoidal-like function is not -1 and 1 rather its 2-0. Although this value of estimation can change depending on different parameters it is not straight forward to say why it did not give a closer result.

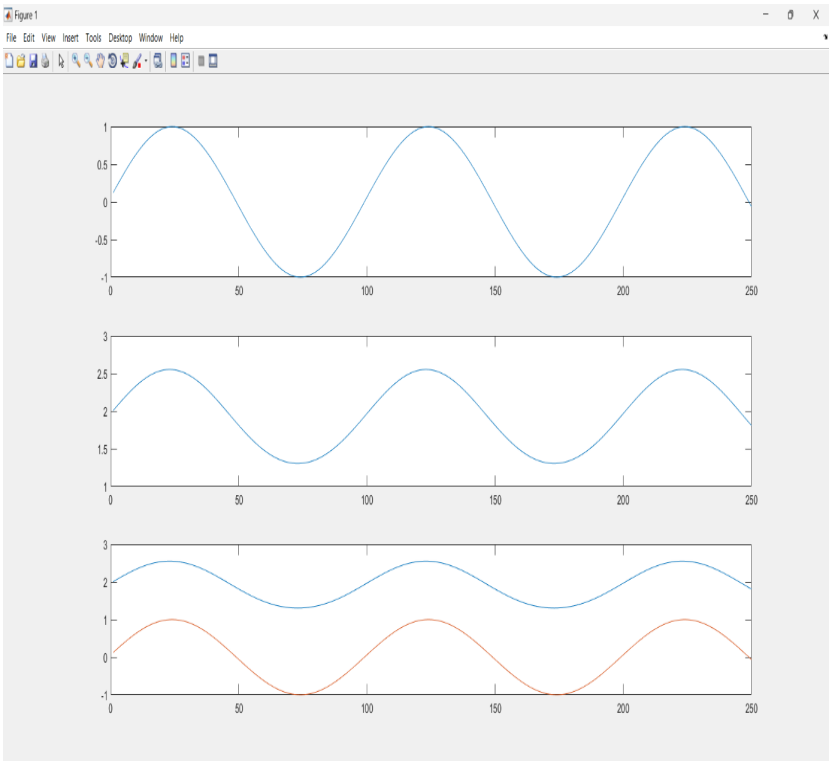
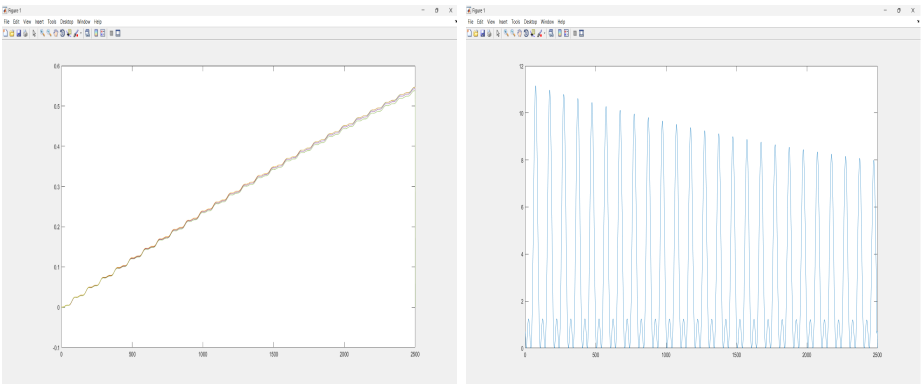


Figure 3: Results with defined parameters. 1-Expected Value 2-Returned Value 3-Return and expected values



(a) Weigths over time

(b) Loss value over time

Figure 4: Changes of the neuron parameters over time

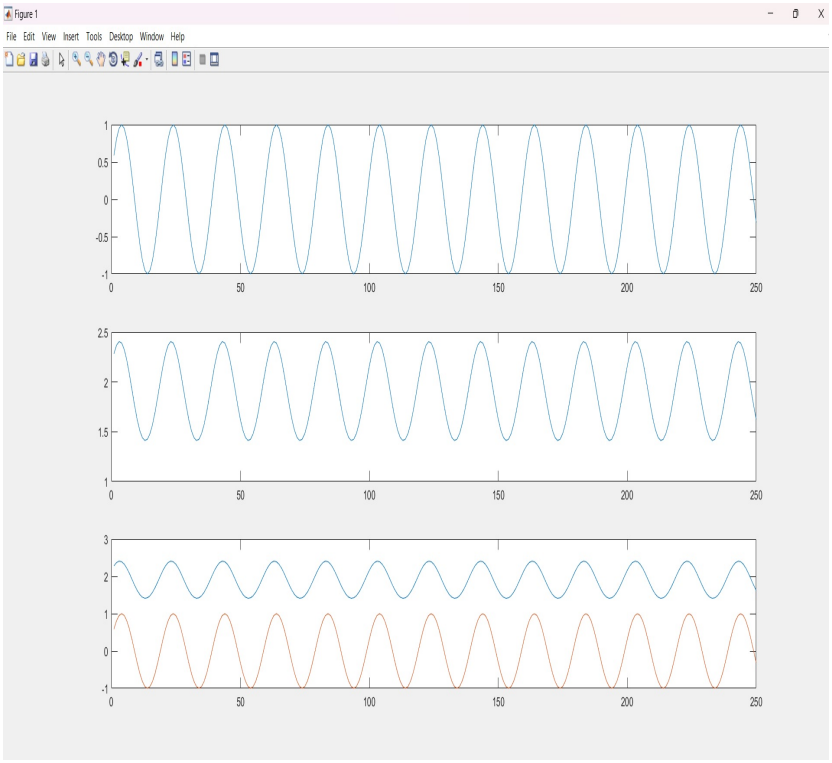


Figure 5: Results with lower sampling rate. 1-Expected Value 2-Returned Value 3-Return and expected values