KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

LABORATORY WORK 2

REPORT

Student: Gurban Shukurov

Lecturer: dr. Germanas Budnikas

Kaunas
2023

Table of Contents

Tasks 1, 2, 3:

Download sunspot.txt and save it in Matlab working directory. File contains historical data on sun plum activity during 1700 - 1950 years. Load contents of the file into Matlab working memory: *load sunspot.txt*. Check whether a corresponding matrix had been loaded– the first column corresponds to years, the second – sun plums.

```
% Load data from text file
sunspot = load('sunspot.txt');
```

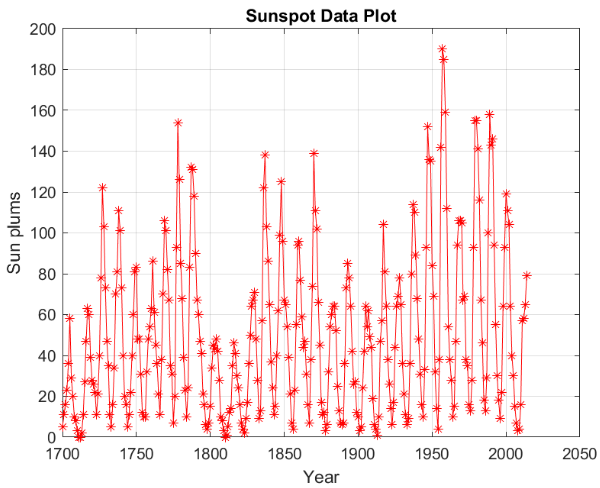
315x2 double

	1	2
1	1700	5
2	1701	11
3	1702	16
4	1703	23
5	1704	36
6	1705	58
7	1706	29
8	1707	20
9	1708	10
10	1709	8
...		
307	2006	15
308	2007	7
309	2008	3
310	2009	4
311	2010	16
312	2011	57
313	2012	58
314	2013	65
315	2014	79

Task 4:

The first task that should implement our program is to draw a diagram of sun plum activity during 1700-1950. Figure has to be fully described (axis and figure titles).

```
% Load data from text file
sunspot = load('sunspot.txt');

% Create figure
figure(1);

% Plot data
plot(sunspot(:, 1), sunspot(:, 2), 'r-*');

% Add title, labels, grid lines
title('Sunspot Data Plot');
xlabel('Year');
ylabel('Sun plums');
grid on;
```



Task 5:

Let us set the order of autoregressive model will be 2 (n=2). That is, we premise that next year's plum forecast is possible based only on two previous years. Then, a neuron will have only two inputs. Supplement the scenario by describing matrices P and T, that contain input (training) data and output data correspondingly. Check content and size of matrices P and T.

```
% Set input data and output data vector (order n=2)
L = length(sunspot); % data length
P = [sunspot(1:L-2,2)' ; % input data matrix
     sunspot(2:L-1,2)'];
T = sunspot(3:L,2)'; % output data vector
disp('Size of P:')
disp(size(P));
disp('Size of T:')
disp(size(T));
```

```
Size of P:
     2   313

Size of T:
     1   313
```

Contents of P:

2x313 double

	1	2	3	...	311	312	313
1	5	11	16		16	57	58
2	11	16	23	...	57	58	65

Contents of T:

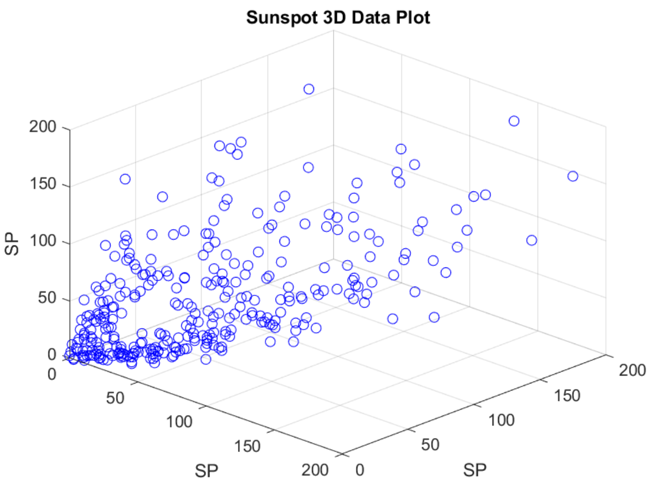
1x313 double

	1	2	3	...	311	312	313
1	16	23	36	...	58	65	79

Task 6:

Plot a diagram in a new graphical window (figure 2) with the following contents – inputs and outputs (P and T correspondingly). Run the scenario, analyze the figure. Add axis and figure titles. What is a graphical interpretation of optimal values of neuron weight coefficients w1, w2, b?

```
figure(2);
plot3(P(1,:),P(2,:),T,'bo');

% Add title, labels, grid lines
title('Sunspot 3D Data Plot');
xlabel('SP');
ylabel('SP');
zlabel('SP');
grid on;
```



By rotating this, figure it can be seen that weight coefficients are the coordinates of the points that are required to draw a 3D plane of the given data.

Task 7:

Let us select from inputs P and outputs T data set fragments of 200 members – so called training data set. Using that set we will calculate optimal values of neuron weight coefficients (parameters of autoregressive model). The rest data will be used for model testing. Then, using existing matrices P and T, let us define two new ones – Pu and Tu.

```
% Set training data sets
Pu = P(:,1:200);
Tu = T(:,1:200);
```

Task 8:

Create an artificial neuron of the structure that had been discussed before. Calculate its weight coefficients using a direct way (function *newlind*). To do so, use matrices of training data Pu and Tu. Name a variable that describes a network by *net*. Add a corresponding command to the scenario.

```
% Create an artificial neuron
net = newlind(Pu, Tu);
```

Task 9:

Display corresponding neuron weight coefficient values.

```
% Display corresponding neuron weight coefficient values
disp('Neuron weight coefficient values:' )
disp(net.IW{1})
disp(net.b{1})
```

```
Neuron weight coefficient values:
   -0.6761    1.3715

   13.4037
```

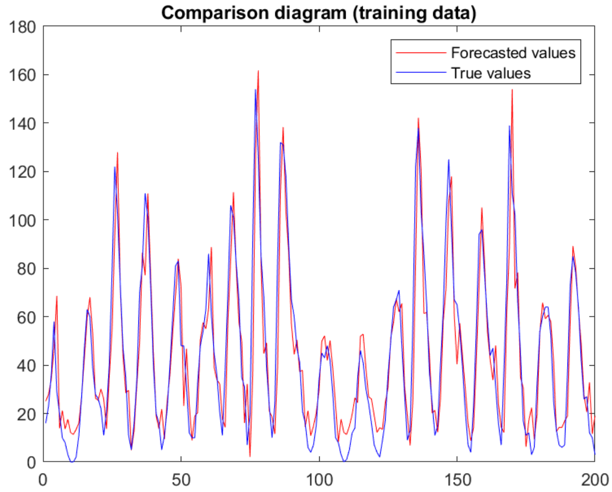
Assign corresponding weight coefficient values to auxiliary variables.

```
% Assign weights to auxiliary variables
w1 = net.IW{1}(1);
w2 = net.IW{1}(2);
b = net.b{1};
```

Task 10:

Perform a testing of the developed model – i.e., check its forecast quality using network simulation. Training data should be used. Compare forecasted values with true values.

```
% Simulate the network
Tsu = sim(net,Pu);

% Comparison diagram 1
figure(3);
plot(Tsu, 'r');
hold on;
plot(Tu, 'b');
title('Comparison diagram (training data)');
legend('Forecasted values', 'True values');
```

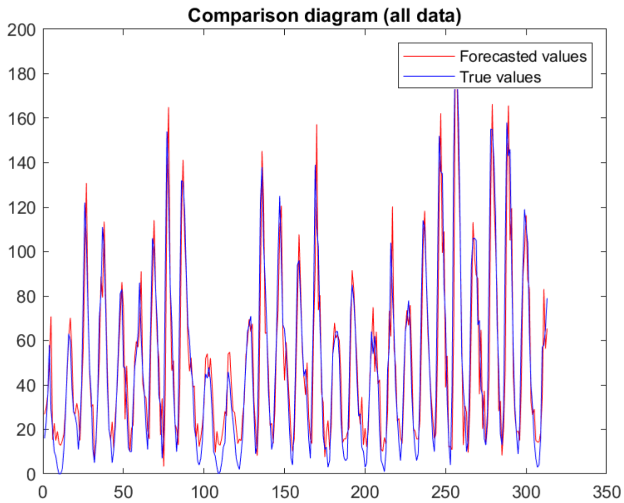


The diagram shows that forecasted values were almost accurate with true values.

Task 11:

Do the same simulation for the rest of the data. Draw a comparison diagram depicting forecasted Ts (neuron outputs) and true values T. Note: you should not recalculate weight coefficients.

```
% Create an artificial neuron
net2 = newlind(P, T);

% Simulate the network
Ts = sim(net2,P);

% Comparison diagram 2
figure(4);
plot(Ts, 'r');
hold on;
plot(T, 'b');
title('Comparison diagram (all data)');
legend('Forecasted values', 'True values');
```

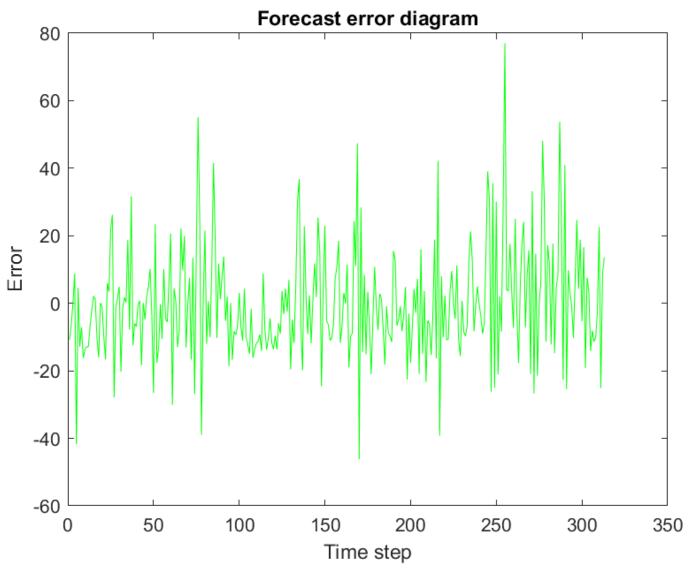


While comparing forecasting of all the data, it can be seen that the situation is the same as with the training data.

Task 12:

Create forecast error vector e. In a new window draw error diagram (plot). Describe axis and chart titles.

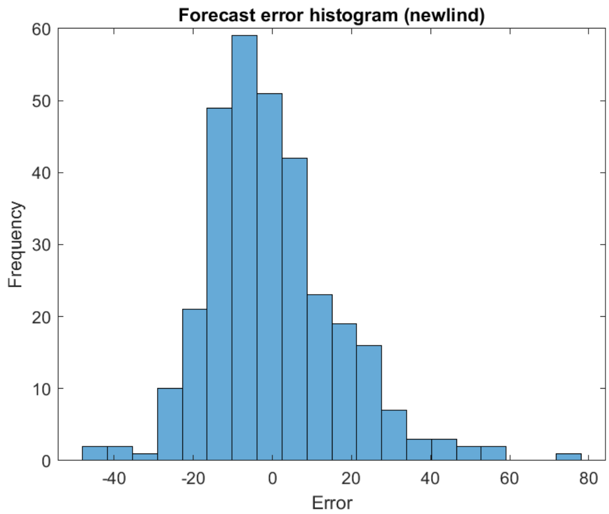
```
% Create forecast error vector
e = T - Ts;

% Error diagram
figure(5);
plot(e, 'g');
title('Forecast error diagram');
xlabel('Time step');
ylabel('Error');
```



Task 13:

Draw forecast error histogram.

```
% Error histogram
figure(6);
histogram(e, 20);
title('Forecast error histogram (newlind)');
xlabel('Error');
ylabel('Frequency');
```



Task 14:

Calculate Mean-Square-Error, MSE and MAD.

```
% Calculate the mean squared error
mse_value = mse(e);

% Calculate the mean absolute error
mae_value = mae(e);

% Display the mean squared error and mean absolute error
fprintf('Mean Squared Error: %.4f\n', mse_value);
fprintf('Mean Absolute Error: %.4f\n', mae_value);
```

```
Mean Squared Error: 274.6448
Mean Absolute Error: 12.5418
```

Task 15:

Save the scenario using different name. Comment lines with commands of direct training (#8).

All the tasks from task 8 until task 15 were done using direct training with in-built Matlab function "newlind". The following tasks (until task 20) were made with iterative method of finding weight coefficients – using neuron training. Code in the task range 1-7 was copied to a separate .m file, where different training method was used. Note: the code is provided including the work with both training data (Pu, Tu) and all data (P, T)

Task 16:

Using function *newlin* create direct neuron. Define function parameters (input delay – 0 and learning rate (lr) – between 0 and 1).

```
% Pu, Tu
% task 16
% Create direct neuron
net = newlin(Pu, Tu, 0, 0.00000082078722469475);
```

```
% P, T
% task 16
% Create direct neuron
net2 = newlin(P, T, 0, 0.00000040452060149784);
```

Task 17:

Define needed learning error goal and number of epochs.

```
% Set the goal and number of epochs
net.trainParam.goal = 100;
net.trainParam.epochs = 30000;
```

```
% Set the goal and number of epochs
net2.trainParam.goal = 100;
net2.trainParam.epochs = 30000;
```

Task 18:

Use function *train* to train a network.

```
% Train and simulate the network
net = train(net,Pu,Tu);
Tsu = sim(net,Pu);
```

```
% Train and simulate the network
net2 = train(net2,P,T);
Ts = sim(net2,P);
```

As can be seen, simulation of the network was also performed. That was needed for forecasting the error.

Task 19:

Save and run the scenario. Answer the questions:

- **What are the new weight values of neural network?**
- **What is the value of mean squared error?**

New weight values and MSE were following:

```
Training data                All data
Weight coefficient values:   Weight coefficient values:
   -0.6596    1.3882            -0.6651    1.4126

   11.1164                      11.3164
MSE (Pu, Tu): 219.0501       MSE (P, T): 279.4974
```

Task 20:

Repeat the procedure with new parameters set in #17. Investigate their impact to learning process and forecasting quality. Which is the maximum value of learning rate lr that enables process convergence?

The procedure was repeated with the same goal but different number of epochs, i.e., 1000, 10000, 30000, 100000. It was found that with 30000 epochs the output of MSE is the most appropriate (considering that only two inputs were used). For finding the maximum value learning rate that enables process convergence, in-built function *maxlinlrl* was used. This is the most optimal value, since if this value is decreased or increased, the resulting MSE would only deteriorate. Found max learning rate was already defined in *newlin* function (refer to task 16). Here is the code and results:

```
maxlr = maxlinlr(Pu, b);                maxlr2 = maxlinlr(P, b2);
MaxlinLr: 0.00000082078722469475        MaxlinLr: 0.00000040452060149784
```

Task 21:

Change the number of network inputs to n=6, n=11. Correspondingly redefine definitions of matrices P and T. Investigate an impact of model structure change on forecast quality (graphically and in a written form).

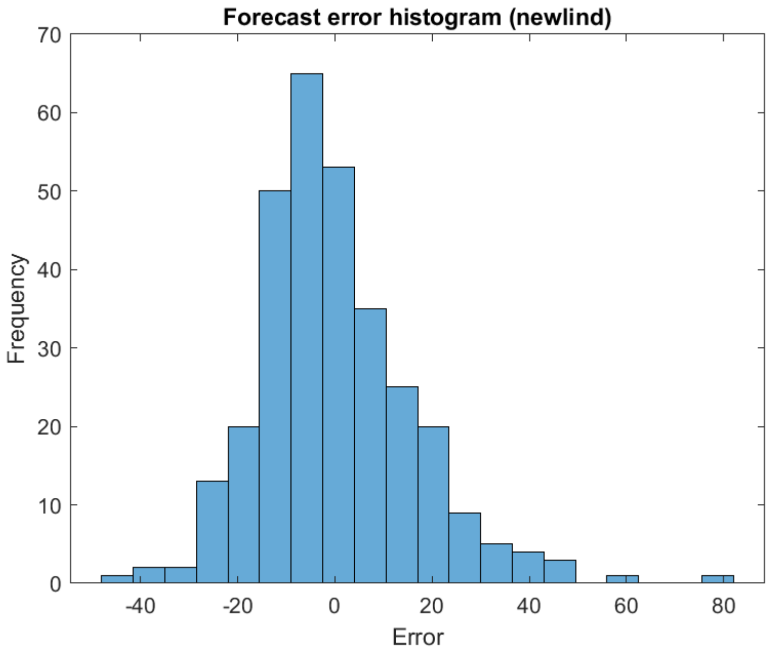
Experiments were conducted using both functions *newlind* and *newlin*. Forecasting of <u>all the data</u> will be considered in this exercise (i.e., excluding training data, however, results of forecasting of training data can be observed in the source Matlab code). For n=6 and n=11 different number of epochs for training was used (i.e., 50000 and 10000 accordingly). Matrices P and T have been redefined for both number of inputs. Changes can be seen below:

<u>**n = 6:**</u>

```
P = [sunspot(1:L-6,2)' ; % input data matrix
     sunspot(2:L-5,2)'
     sunspot(3:L-4,2)'
     sunspot(4:L-3,2)'
     sunspot(5:L-2,2)'
     sunspot(6:L-1,2)'];
T = sunspot(7:L,2)'; % output data vector
```

The following results were obtained using ***newlind***:

```
Neuron weight coefficient values:
    0.1534   -0.2394    0.1257   -0.0309   -0.6424    1.3510

   12.4871

Mean Squared Error: 264.5304
Mean Absolute Error: 12.3017
```

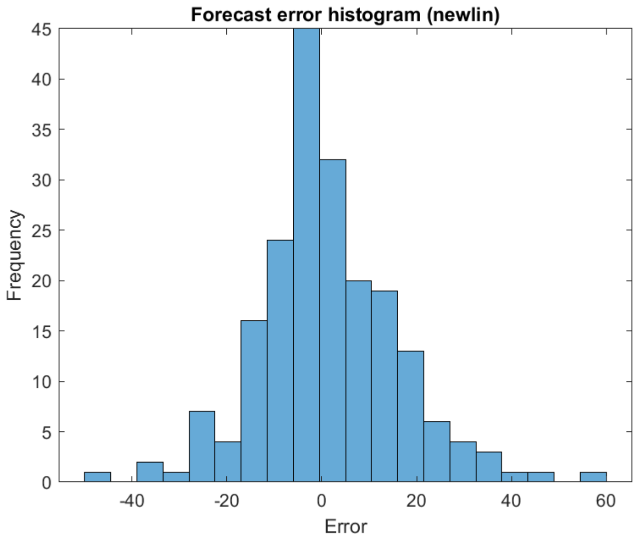


Error histogram here is nearly identical with the one of n=2 inputs. It shows that error values in the range -20:0 occur more frequent.

The following results were obtained using ***newlin***:

```
All data
Weight coefficient values:
    0.2484   -0.2368    0.1394   -0.1410   -0.5117    1.3861

    4.4401

MSE (P, T): 277.1570
```

The results do not differ a lot from the *newlind* function model above, and even MSE became higher than while using *newlind* (considering n=2, this MSE is a bit lower). However, the error tends more to be 0:



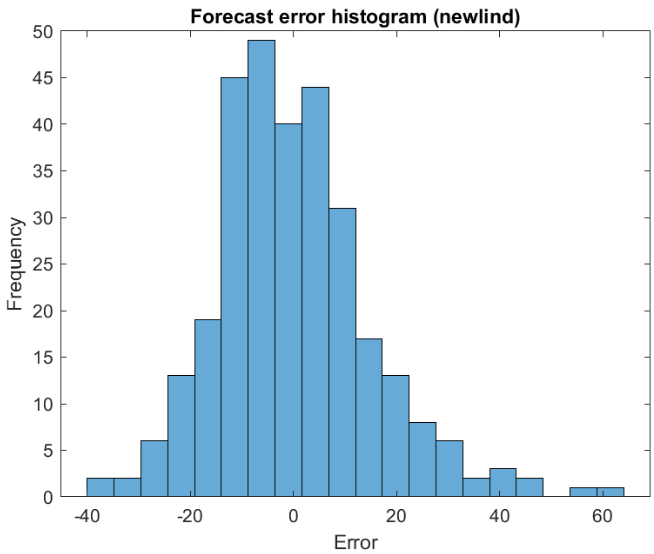
n = 11:

```
P = [sunspot(1:L-11,2)' ; % input data matrix
     sunspot(2:L-10,2)'
     sunspot(3:L-9,2)'
     sunspot(4:L-8,2)'
     sunspot(5:L-7,2)'
     sunspot(6:L-6,2)'
     sunspot(7:L-5,2)'
     sunspot(8:L-4,2)'
     sunspot(9:L-3,2)'
     sunspot(10:L-2,2)'
     sunspot(11:L-1,2)'];
T = sunspot(12:L,2)'; % output data vector
```

The following results were obtained using ***newlind***:

```
Neuron weight coefficient values:
    0.1240   -0.1464    0.1863    0.0406   -0.0498    0.0815   -0.1649    0.1534   -0.0653   -0.5826    1.2636

    7.2693

Mean Squared Error: 224.1177
Mean Absolute Error: 11.3876
```

MSE and bias here have been significantly decreased in comparison with n=6 inputs.

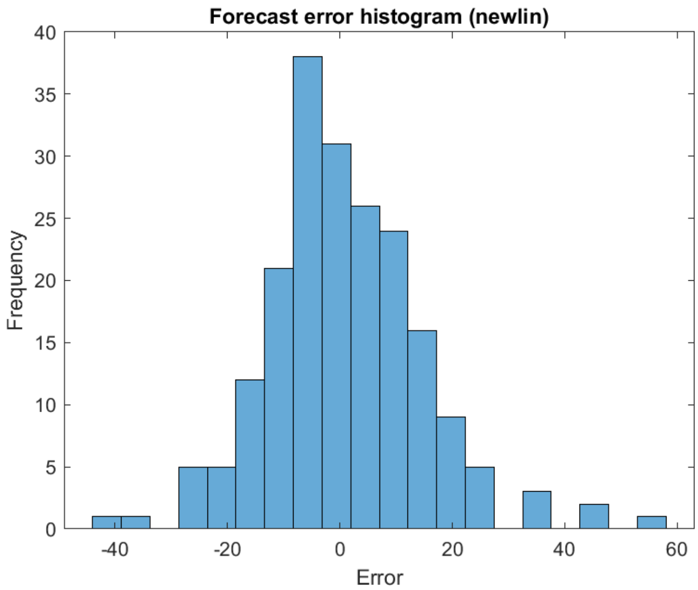


This error histogram is not very different from the n=6 one. It is that the frequency of error values in the range -20:0 is now decreased.

The following results were obtained using ***newlin***:

```
All data
Weight coefficient values:
    0.0367    -0.0427    0.2867    -0.0754    0.0450    0.0300    -0.0799    0.1437    -0.1418    -0.4190    1.1991

    0.2657

MSE (P, T): 229.6223
```

Same is here, MSE has been notably decreased in comparison with n=6 inputs *newlin* function.



Frequency of error values is even more decreased.

Conclusion

Three experiments were conducted with different number of inputs and using two functions, *newlind* and *newlin*. The difference between these neural network creation Matlab functions is not noticeable. *newlin* creates a neural network using linear activation function, and the output of it will be continuous values. But *newlind* function creates a neural network with scaled inputs and outputs. It uses a purelin activation function which produces only binary output values. In a simpler way, *newlind* solves linear equation (built in Matlab) for finding weight coefficients and bias, but *newlin* uses iteration. So, it is hard to say that *newlind* creates a neural network, because it does not, however, *newlin* does. *newlin* is used more for regression problems, and *newlind* for classification problems.

From the experiments with n = 2, 6, 11 number of inputs conclusion will be as follows: the more inputs are provided for training the model, the less value of mean squared error will be obtained and the frequency of error values will be significantly low. The model can learn more about patterns and relationships that exist in the data, therefore, the outputs will be more accurate. In addition, more inputs can prevent overfitting, because the model will have a wider range of information.