Investigation of ANN Used to Solve MLR Problem

Prof. Ziad A. AlQadi¹; Eng. Ahmad S. AlOthman²; Eng. Mahmoud O. Alleddawi³; Eng. Mohammad Ali Al-Hiary⁴; Eng. Osama T. Ghazal⁵; Eng. Sameh S. Salman⁶

Al-Balqa Applied University, Jordan

dr.ziad.alqadi@bau.edu.jo; alothmanahmad@std.bau.edu.jo; Mahmoud.alleddawi@gmail.com; Eng.hiary@gmail.com; e.osa_gh@yahoo.com; eng.sameh@std.bau.edu.jo

Abstract: Solving multiple linear regression problems requires time and efforts concerning statistical and mathematical operations calculation. In this paper we will introduce ANN as a tool to solve MLR problem. Deferent CFANN and FFANN will be introduces; Deferent ANN architectures will be examined using various activation functions in order to select a most efficient, and accurate ANN.

Keywords: ANN, FCANN, FFANN, MLR, AF, goal, training time

1- Introduction

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable [1]. The goal of multiple linear regressions (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable [2], [3].

MLP can be represented by the following formula:

\[ y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon \]

Where,

\( Y_i \) = dependent variable

\( X_i \) = explanatory variables
\[ \beta_0 = \text{y-intercept (constant term)} \]

\[ \beta_p = \text{slope coefficients for each explanatory variable} \]

\[ \epsilon = \text{the model’s error term} \]

It is very simple to solve the MLR using MATLAB; to do this we have to apply the following steps:

- Input the values of independent variables (usually experimental data), one column for each variable.
- Input the experimental values for the output (related function).
- Create a matrix of columns, one column for each variable, adding a first column of 1’s as shown below:

\[
\begin{pmatrix}
1.0000 & 100.0000 & 50.0000 & 60.0000 \\
1.0000 & 105.0000 & 30.0000 & -20.0000 \\
1.0000 & -20.0000 & 20.0000 & 15.0000 \\
1.0000 & 30.0000 & 100.0000 & 200.0000 \\
1.0000 & 40.0000 & 0 & 100.0000 \\
1.0000 & 70.0000 & -7.0000 & 110.0000 \\
1.0000 & 0 & 8.0000 & 200.0000 \\
1.0000 & 1.5000 & 30.0000 & 0 \\
1.0000 & 105.0000 & 9.0000 & -5.0000 \\
1.0000 & 130.0000 & 10.5000 & 90.5000
\end{pmatrix}
\]

- Divide the matrix by the output to get the coefficients of MLR formula

Here is a sample of MATLAB code to solve MLR using 3 variables:

```matlab
x1=[100 105 -20 30 40 70 0 1.5 105 130]';
x2=[50 30 20 100 0 -7 8 30 9 10.5]';
x3=[60 -20 15 200 100 110 200 0 -5 90.5]';
t=10+2*x1-3*x2+8*x3;
f=[ones(size(x1)) x1 x2 x3];
c=f
t
```

Running this code we obtain the coefficients as follows:

\[
c = 
\begin{align*}
10.0000 \\
2.0000 \\
-3.0000 \\
8.0000
\end{align*}
\]

So the expected output can be represented by the following formula:

\[ y = 10 + 2x_1 - 3x_2 + 8x_3 \]
2- Artificial neural networks

Artificial neural networks (ANN) is a set of fully connected neurons, these neurons are connected via weights and operate in parallel [4], [5],[6], [7]. ANNs have been used in different important life applications such as: solving classification problem, regression analysis, pattern recognition, image and signal processing [8]. The backpropagation neural network (BPNN) is a feed forward multi-layer ANNs, the forward phase is used to calculate neurons output, while the backward phase is used to check the error, if the error is not acceptable then the weights to be updated, these two phases forms a training cycle [8], [9], [10]. BPNNs are very useful only when the network architecture is chosen correctly. Too small network cannot learn the problem well, but too large size will lead to over fitting and poor generalization performance [11]. The backpropagation algorithm (BP) can be used to train the BPNN image compression but its drawback is slow convergence. Many approaches have been carried out to improve the speed of convergence [12].

The mathematical model of the computational neuron is shown in figure 1, and each neuron performs the following tasks:

1) Find the summation, which is equal to sum of products of the inputs and associated weights.
2) According to the selected activation function (AF) calculate the neuron output.

The main activation functions used are [2], [3], [13]:

- Linear AF, here the neuron output will equal the summation.
- Logsig AF, here the neuron output will be calculated by the following formula:
  \[ f = \frac{1}{1 + e^{-\theta}} \]
  And the output is always between 0 and 1.
- Tansig AF, here the neuron output will be calculated by the following formula:
  \[ f = 2 - \frac{1}{1 + e^{-2\theta}} \]
  And the output is always between -1 and 1.
Different types of ANN are available [13], [14], [22], here in this paper we will focus on feed forward ANN (FFANN) and cascade forward ANN (CFANN) [15], [16].

FFANN as shown in figure 2 contains a set of fully connected neurons, these neurons are organized in layer, the neurons of the previous layer are connected to next layer [17],[18].

Figure 2: FFANN

CFANN likes FFANN, but includes (as shown in figure 3) a weight connection from the input to each layer and from each layer to the successive layers. While two-layer feed-forward networks can potentially learn virtually any input-output relationship, feed-forward networks with more layers might learn complex relationships more quickly [23].

Figure 3: CFANN

To maximize the benefits of using ANN as a computational model we have to take the following as in important factors [1], [2], [3]:

- Selecting the input data set carefully and organizing it in a suitable way to match ANN architecture [18].
Selecting the output target carefully and organizing it in a suitable way to match the input data set and ANN architecture [19] [20].

Select the appropriate ANN architecture [2], [3]. Architecture here means: Which ANN to use, how many layers, and how many neurons in each layer, what is the activation function to be used for each layer. Here we have to notice that selecting AF depends on problem understanding and what are the required outputs for the neurons to be achieved.

Initializing the weights (the simple way is to reset all the weights) [4].

Selecting some parameters for ANN, especially the acceptable error (goal which must be closed to zero) [4], number of training cycles (epochs), this number can be adjustable during the process of training [5].

Creating ANN by considering the above factors [22].

Training ANN, this process can be repeated much time if necessary (while the error is not acceptable), some time we need to adjust ANN architecture and select different AFs.

After reaching the goal we can save ANN to be used later as a tool to find the exact output for a selected set of inputs [1], [2], [23].

3- Implementation and experimental results

- Experiment 1

The following experimental data set (see table 1) was processed to find the relationship between the output and the 3 independent variable x1, x2, and x3, solving this MLR problem using the same procedures explained above gave the following MLR formula:

\[ y = 10 + 2x1 - 3x2 + 5x3 \]

FFANN and CFANN were created and tested using the input data set and the target output, for each ANN 3 layer were used with 3, 6 1 and 1 neurons, the selected AFs were logsig, logsig and linear, table 2 shows the results of this experiment:

Table 1: Experimental data for experiment 1

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Experimental output Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1016</td>
<td>0.0174</td>
<td>0.8768</td>
<td>14.5352</td>
</tr>
<tr>
<td>0.0653</td>
<td>0.8194</td>
<td>0.0129</td>
<td>7.7369</td>
</tr>
<tr>
<td>0.2343</td>
<td>0.6211</td>
<td>0.3104</td>
<td>10.1572</td>
</tr>
<tr>
<td>0.9331</td>
<td>0.5602</td>
<td>0.7791</td>
<td>14.0809</td>
</tr>
<tr>
<td>0.0631</td>
<td>0.2440</td>
<td>0.3073</td>
<td>10.9306</td>
</tr>
<tr>
<td>0.2642</td>
<td>0.8220</td>
<td>0.9267</td>
<td>12.6958</td>
</tr>
<tr>
<td>0.9995</td>
<td>0.2632</td>
<td>0.6787</td>
<td>14.6030</td>
</tr>
<tr>
<td>0.2120</td>
<td>0.7536</td>
<td>0.0743</td>
<td>8.5347</td>
</tr>
<tr>
<td>0.4984</td>
<td>0.6596</td>
<td>0.0707</td>
<td>9.3712</td>
</tr>
<tr>
<td>0.2905</td>
<td>0.2141</td>
<td>0.0119</td>
<td>9.9984</td>
</tr>
<tr>
<td>0.6728</td>
<td>0.6021</td>
<td>0.2272</td>
<td>10.6749</td>
</tr>
<tr>
<td>0.9580</td>
<td>0.6049</td>
<td>0.5163</td>
<td>12.6824</td>
</tr>
<tr>
<td>0.7666</td>
<td>0.6595</td>
<td>0.4582</td>
<td>11.8456</td>
</tr>
<tr>
<td>0.6661</td>
<td>0.1834</td>
<td>0.7032</td>
<td>14.2982</td>
</tr>
<tr>
<td>0.1309</td>
<td>0.6365</td>
<td>0.5825</td>
<td>11.2647</td>
</tr>
<tr>
<td>0.0954</td>
<td>0.1703</td>
<td>0.5092</td>
<td>12.2259</td>
</tr>
<tr>
<td>0.0149</td>
<td>0.5396</td>
<td>0.0743</td>
<td>8.7824</td>
</tr>
<tr>
<td>0.2882</td>
<td>0.6234</td>
<td>0.1932</td>
<td>9.6724</td>
</tr>
<tr>
<td>0.8167</td>
<td>0.6859</td>
<td>0.3796</td>
<td>11.4738</td>
</tr>
<tr>
<td>0.9855</td>
<td>0.6773</td>
<td>0.2764</td>
<td>11.3211</td>
</tr>
<tr>
<td>0.1016</td>
<td>0.0174</td>
<td>0.8768</td>
<td>14.5352</td>
</tr>
</tbody>
</table>
From the obtained results of experiment 1 we can see that the calculated outputs of FCANN are more accurate than those obtained by FFANN.

-Experiment 2

In this experiment we will take a MLR problem and use deferent CFANN and FFANN architectures, to see how the selected architecture affects the calculated outputs.

The following data set shown in table 3 was manipulated using the previous mentioned method of MLR problem solving and the obtained regression formula was:

\[ y = 10 + 2x_1 - 3x_2 + 8x_3 \]
Using the above data CFANN was created using the architecture shown in figure 4:

![Figure 4: CFANN architecture (2 layers)](image)

The following matlab code was written to implement this ANN:

```matlab
x1=[100 105 -20 30 40 70 0 1.5 105 130];
x2=[50 30 20 100 0 -7 8 30 9 10.5];
x3=[60 -20 15 200 100 110 200 0 -5 90.5];
data=[x1;x2;x3];
target=10+2*x1-3*x2+8*x3;
net1=newcf(minmax(data),[3 1],{'purelin','purelin'});
net1=init(net1);
net1.trainParam.goal=0;
net1.trainParam.epochs=2000;
net1.trainParam.lr=0.001;
net1=train(net1,data,target);
y1=sim(net1,data);
```

Table 3 shows the implementation results:

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>X2</td>
</tr>
<tr>
<td>100.0000</td>
<td>50.0000</td>
</tr>
<tr>
<td>105.0000</td>
<td>30.0000</td>
</tr>
<tr>
<td>-20.0000</td>
<td>20.0000</td>
</tr>
<tr>
<td>30.0000</td>
<td>100.0000</td>
</tr>
<tr>
<td>40.0000</td>
<td>0</td>
</tr>
<tr>
<td>70.0000</td>
<td>-7.0000</td>
</tr>
<tr>
<td>0</td>
<td>8.0000</td>
</tr>
<tr>
<td>1.5000</td>
<td>30.0000</td>
</tr>
<tr>
<td>105.0000</td>
<td>9.0000</td>
</tr>
<tr>
<td>130.0000</td>
<td>10.5000</td>
</tr>
</tbody>
</table>
Figure 5 shows that the calculated output fits the target:

![Figure 5: Target and CFANN output](image)

2) Building FFANN

The same thing was performed using FFANN with the architecture shown in figure 6 and the same results were obtained, here the calculated output fits the target as shown in figure 7.

![Figure 6: FFANN architecture (2 layers)](image)
The following MATLAB code was used to implement the aboveFFANN:

```matlab
x1=[100 105 -20 30 40 70 0 1.5 105 130];
x2=[50 30 20 100 0 -7 8 30 9 10.5];
x3=[60 -20 15 200 100 110 200 0 -5 90.5];
data=[x1; x2; x3];
target=10+2*x1-3*x2+8*x3;
net1=newff(minmax(data),[3 1],{'purelin','purelin'});
net1=init(net1);
net1.trainParam.goal=0;
net1.trainParam.epochs=2000;
net1.trainParam.lr=0.001;
net1=train(net1,data,target);
y1=sim(net1,data);
```

Now we change the architecture of CFANN as shown in figure 8:

```
Figure 7: Target and FFANN output
```

```
Figure 8: CFANN architecture (3 Layers).
```

```
Figure 9 shows that the calculated output fits the target:
```
The same architecture was used for FFANN as shown in figure 1:

Here, the obtained results show that the calculated output was not close to the target and the error was very big this is shown in table and figure 11:

Table 4: FFANN calculation results

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Output</th>
<th>Target1.0e+03</th>
<th>Calculated1.0e+03</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>X2</td>
<td>X3</td>
<td></td>
</tr>
<tr>
<td>100.0000</td>
<td>50.0000</td>
<td>60.0000</td>
<td>0.5400</td>
</tr>
<tr>
<td>105.0000</td>
<td>30.0000</td>
<td>-20.0000</td>
<td>-0.0300</td>
</tr>
<tr>
<td>-20.0000</td>
<td>20.0000</td>
<td>15.0000</td>
<td>0.0300</td>
</tr>
<tr>
<td>30.0000</td>
<td>100.0000</td>
<td>200.0000</td>
<td>1.3700</td>
</tr>
<tr>
<td>40.0000</td>
<td>0</td>
<td>100.0000</td>
<td>0.8900</td>
</tr>
<tr>
<td>70.0000</td>
<td>-7.0000</td>
<td>110.0000</td>
<td>1.0510</td>
</tr>
<tr>
<td>0</td>
<td>8.0000</td>
<td>200.0000</td>
<td>1.5860</td>
</tr>
<tr>
<td>1.5000</td>
<td>30.0000</td>
<td>0</td>
<td>-0.0770</td>
</tr>
<tr>
<td>105.0000</td>
<td>9.0000</td>
<td>-5.0000</td>
<td>0.1530</td>
</tr>
<tr>
<td>130.0000</td>
<td>10.5000</td>
<td>90.5000</td>
<td>0.9625</td>
</tr>
</tbody>
</table>
Different CFANN, FFANN with various architectures, activation functions were selected and implemented, the error between the calculated outputs and the targets was calculated. Training time was measured; table 4 shows the summarized results of these implementations:

Table 4: Summarized results

<table>
<thead>
<tr>
<th>ANN layers</th>
<th>Activation functions</th>
<th>CFANN parameters</th>
<th>FFANN</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3 1</td>
<td>Loxsig, linear</td>
<td>Error 0</td>
<td>Small error 0.079</td>
<td>Training time 17.002</td>
</tr>
<tr>
<td>3 1</td>
<td>Tansig, linear</td>
<td>Error 0.046</td>
<td>high 0.081 0.073</td>
<td></td>
</tr>
<tr>
<td>3 1</td>
<td>Linear, linear</td>
<td>Error 0.740</td>
<td>high 2.168 0.073</td>
<td></td>
</tr>
<tr>
<td>3 6 1</td>
<td>Loxsig, logsig, linear</td>
<td>Error 0.114</td>
<td>high 0.159 0.073</td>
<td></td>
</tr>
<tr>
<td>3 6 1</td>
<td>Tansig, logsig, linear</td>
<td>Error 0.123</td>
<td>high 0.088 0.073</td>
<td></td>
</tr>
<tr>
<td>3 6 1</td>
<td>Loxsig, tansig, linear</td>
<td>Error 0.737</td>
<td>high 2.168 0.073</td>
<td></td>
</tr>
<tr>
<td>3 6 1</td>
<td>Tansig, tansig, linear</td>
<td>Error 0.125</td>
<td>high 0.159 0.073</td>
<td></td>
</tr>
<tr>
<td>3 6 1</td>
<td>Linear, linear, linear</td>
<td>Error 0.247</td>
<td>0 0.073</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: FFANN output does not match the target.
Conclusion

CFANN and FFANN were introduced to solve MLR problem. Both ANN are suitable to solve MLR problem but with various efficiency and accuracy. It was shown that FFANN neural network some time gives a very high error, so here we must be very carefully when selecting ANN architecture and activation function, for MLR it is better to use linear activation functions for all the layers.

Using CFANN is more flexible and accurate, and it always give a zero error using any number of layers and any activation functions for the input and the hiding layers (the AF for the output layer must be linear for our problem.

References


