A Java application for the failure rate prediction using feed forward neural networks

Alexandru Ene
University of Pitesti
Pitesti, Romania
alexandru.ene@upit.ro

Cosmin Stirbu
University of Pitesti
Pitesti, Romania
cosmin.stirbu@upit.ro

Abstract – In this paper we show the possibility to use feed forward neural networks for failure rate prediction, and this can be used for improving predictive maintenance. We use a series of real values that represent the failure rate of a radio-reception system, and describe a Java application that we developed, that simulates a feed forward neural network which is trained to predict, based on the available series of failure rate values, the next failure rate value. We use a one hidden layer neural network that has a single output, the predicted value. The inputs of network are analog, continuous values between 0 and 1, and represent the previous known values of failure rate. Using the software, we used different numbers of input (2, 3, and 5) and the accuracy of prediction is compared for these different numbers.

Keywords: feed forward neural network, predictive maintenance, Java application

I. PREDICTIVE MAINTENANCE

Usually, maintenance in large enterprises [1] is based on a fixed schedule of revisions that is established from the beginning. But, predictive maintenance uses the actual history of faults to update the time interval between two revisions [2], [3].

The faults diagnosis supposes the precise determination of failure blocks in an electronic system. The major approaches from the quantitative diagnosis field of the faults in the electronic systems have been developed starting from 1980.

We consider an electronic system, which from the point of view of diagnosis of failures is composed from N functional blocks (Fig. 1):

![Figure 1](image1.png)

Each functional block is characterized by the parameter \( \lambda_i \), \( \lambda \) meaning the failure rate of the block. The parameter \( \lambda_i \) is based on the average time between two failures the block i and it is calculated with the relation (1):

\[
\lambda_i = \frac{1}{MTBF}
\]

MTBF (Mean time between failures) represents the average time between two failures, expressed in hours. The failure rate is subunitary, and its measure unit is hours\(^{-1}\).

The purpose of using neural networks is that based on the history of the parameters \( \lambda_i \) over a long period of time, there can be estimated the future values of these parameters, and so the model of diagnosis of faults can be improved.

II. FEED FORWARD NEURAL NETWORKS

A feed forward neural network is composed of interconnected processing units (artificial neurons). These neurons are organized in layers. Each connexion has associated a real number, called weight. There is an input layer, one or more middle layers (these are called hidden layers).

The input layer has artificial neurons with a unitary transfer function, their role being to distribute the input to all the neurons from the hidden layer.

The other neurons from the network have a sigmoidal transfer function, as illustrated in the next figure:

![Figure 2](image2.png)
Where:

\[ y = \frac{1}{1 + e^{-a}} \]  

(2)

And

\[ a = \sum_{i=1}^{n} x_i w_i \]  

(3)

In a feed forward neural network each neuron from a layer is totally connected with all the neurons from the next layer. There is no interconnection between neurons of the same layer, as illustrated in Fig. 3, where a 3-2-4 neural network is drawn.

Feed forward neural networks are widely used in pattern recognition applications due to their intrinsic ability to learn from examples, like us. They do not need a mathematical complicated algorithm for pattern recognition, but need only a good set of training examples, and this ability makes them very attractive for the use in pattern recognition and classification tasks. They are also used for the approximation of nonlinear functions, and therefore for the modeling of processes. This ability to approximate nonlinear functions is used in this paper for the estimation of the next value based on a series of previous known values [6], [7].

The inputs of the network could be binary (an input can have only two values: 0 or 1), or analog (the input can take continuous value in a certain interval), or could be a combination of binary and analog. In pattern recognition applications typically there are binary inputs (we have binary images to classify) while in approximation of nonlinear functions, the analog inputs are typically used.

In our application we use only analog inputs, having continuous values.

The feed forward neural networks can be software simulated (using ordinary programming language or special tools for their simulation as in Matlab, neural networks toolbox) or can be hardware implemented based on neural chips.

We used a software simulation of the feed forward neural network, using the Java programming language.

Learning in feed forward neural networks is supervised and is based on a set of examples.

We used in this application a set of 40 examples.

Our training examples are stored in a text file.

The training algorithm used is the classical back propagation algorithm [4], [5]. Its discovery about 30 years ago, led to a huge progress in neural networks domain. The goal of the training algorithm is that the network to learn all the training examples with a certain fixed error \( E \) (in this case is said that the network has converged). Through this algorithm are computed all the weights in the network, following an iterative process. It acts in the following manner:

\[ \text{initiate all weights with random values} \]
\[ \text{while the network didn’t converge execute} \]
\[ \begin{align*}
\text{begin} & \\
\text{for all training patterns execute} & \\
\text{begin} & \\
\text{compute the outputs, for the current pattern} & \\
\text{(forward propagation step of the algorithm)} & \\
\text{compute the learning error for this pattern} & \\
\text{(backpropagation step of the algorithm)} & \\
\text{modify all the weights values using delta rule} & \\
\text{for} & \\
\text{compute total learning error for all patterns} & \\
\text{(the sum of individual errors of each pattern)} & \\
\text{if} (\text{total error} \leq \text{fixed error} \ E) \text{then} & \\
\text{the network has converged} & \\
\text{end} & \\
\text{while} & \\
\end{align*} \]

One iteration of the algorithm through all the training patterns is called a training epoch. Typically a neural network converges in hundreds or thousands of epochs, as in our present Java application.

In [5] there is presented a Java application used to approximate a nonlinear function:

\[ f(x) = x^2 \]
\[ f: (0, 1) \rightarrow (0, 1) \]

But in [5], the authors use a different approach for the approximation of nonlinear functions with feed forward neural network, than in this article. They use a 2-hidden layers network, having one input neuron, 7 neurons in the first hidden layer, 7 hidden neurons in the second hidden layer and one output neuron. Also, the training examples (number of training patterns are NS) are mathematically obtained, following the next algorithm:

\[ \text{for } i=1 \text{ to } \text{NS do} \]
\[ \text{begin} & \\
\text{inputPattern}[i] = i / \text{NS}; & \\
\text{outputPattern}[i] = \text{inputPattern}[i] \times \text{inputPattern}[i]; & \\
\text{end} & \\
\]
III. THE JAVA APPLICATION

We developed a Java application that simulates a feed forward neural network with a single hidden layer.

We have a set of 50 experimental real data (failure rates of a radio reception system). These are stored in a text file, each value on a separate line, like this:

73.14
76.47
77.22
76.86
75.75

Each value multiplied with a constant factor of $10^{-6}$ represents a failure rate.

In the Java application, firstly we scale these data, in order to be brought in a range more adequate to be processed by the neural network. We experimented using three different input ranges: [0.1, 0.9] and [0.2, 0.8]. The best results for the particular data we had, were obtained for the [0.2, 0.8] input range. The minimum value from the text file will correspond to 0.2 and the maximum to 0.8, the other values will be corresponding scaled in the chosen range.

We use from this file a number of NS=40 values for the training (80% from the available data) and the rest of 20% from these scaled values will be used in order to test the network (to show that it can correctly estimate the next failure rate value).

The network has a single output neuron, the estimated value, and it has an experimentally chosen number of hidden neurons.

Considering that there are N input neurons (their number will be variable, for different simulations), the input training data are stored by the software program, in a bidimensional array $xS[NS][N]$, and the corresponding output training data will be stored into an unidimensional array $yS[NS]$.

Using a simplified notation, considering that the available input data for the training are:

$x_1, x_2, x_3, ..., x_N, x_{N+1}, x_{N+2}, ...$

the network is trained in the following manner:

First training example:

input: $x_1, x_2, x_3, ..., x_N$
output: $x_{N+1}$

Second training example:

input: $x_2, x_3, ..., x_N, x_{N+1}$
output: $x_{N+2}$

The third:

input: $x_3, x_4, ..., x_{N+1}, x_{N+2}$
output: $x_{N+3}$

In the same manner the network is trained for all the NS training examples.

In order to test how well the network predicts the next failure rate, let us consider that the last training example used was:

input: $y_1, y_2, y_3, ..., y_N$
output: $y_{N+1}$

We use for the test:

input: $y_2, y_3, ..., y_N, y_{N+1}$

The ideal answer for the network will be $y_{N+2}$, i.e. the next value taken from the text file.

Using the test inputs, we forward propagate these inputs through the network and the real output (the predicted value) is calculated. We compare this real value with the ideal value and then we calculate the error.

IV. SIMULATION RESULTS

Using N=7 input neurons, and 5 hidden neurons (a 7-5-1 neural network), and a total error for the learning of training examples of 4%, the network converges typically in about 700 epochs, and the difference between the predicted value and the actual value is of 5%.

For a 7-7-1 neural network, with the same error of 4%, the network converges typically in about 700 epochs, and the difference between the predicted value and the actual value is of 4%.

If we use a lower training error (3%) the network will converge slower (in about 11000 epochs) but the obtained prediction is better: a lower error of 1.8%. There are no important differences in case we vary the hidden neurons number, but, as expected, the network approximates better when the learning error rate is lower.

We also experimented changing the number of input neurons to 5 and then then 3.

For 5 input neurons, for a total error for the learning of training examples of 4%, the network 5-5-1 converges typically in about 5000 epochs, and the difference between the predicted value and the actual value is of 7%.

For 3 input neurons, for a total error for the learning of training examples of 5%, the network 3-5-1 converges typically in about 400 epochs, and the difference between the predicted value and the actual value is of 19%.

So, the best results were obtained for the 7-5-1 configuration, with a training error of 3%.

Of course, the results can vary slightly, for the same configuration, if we train again the network and use another weights set.

We also tried to use for the estimation of the next value, a network that has two hidden layers, with the same architecture as in [5]:

1-7-7-1

But, the network did not succeed to learn the training pattern in a reasonably amount of time.

V. CONCLUSIONS

In this paper we used software simulated feed forward neural network, to predict the failure rate for a system. The network is trained using a series of failure rate values, stored in a text file.

The best predicted results have been obtained using the 7-7-1 topology for the feed forward neural network, with a training error used by the backpropagation algorithm of 3%. In this case the obtained prediction error is lower than 1.8%.
The prediction could be used to improve the preventive maintenance for this electronic system. Predictive maintenance, generally predicts where, when, and why the failures will occur. Using our neural network, we showed by simulation results, that the network can accurately predict only when the failure is likely to occur.

REFERENCES


