Open-Source Neural Network and Wavelet Transform Tools for Server Log Analysis

Chunyu Liu
School of Computer Science and Engineering
California State University,
San Bernardino
005398220@coyote.csusb.edu

Tong Lai Yu
School of Computer Science and Engineering
California State University,
San Bernardino

Keywords: Open-source, Neural Network, Wavelet Transform, Server Log Analysis

Abstract

Open-source Java tools are used to implement a server log analysis application, which combines neural network algorithms and wavelet transform techniques to give better predictions. Wavelet transforms are used to denoise the log data, extracting relevant information from the noisy data. The filtered data are then fed into a multi-layer neural network which learns to identify patterns of errors and problematic issues. Experiments conducted on the log datasets of a virtualized cloud platform at CSUSB show that this approach makes better predictions as compared to a stand alone neural network. The application predictions can help server administrators better manage the server platform by taking preventive actions before serious damages have occurred.

1. Introduction

Cloud computing is a computing paradigm that refers to the delivery of on-demand computing power, often involving the access of shared resources and services that can be promptly provisioned with minimal management, typically over the Internet. In recent years, cloud computing has been growing exponentially and has become ubiquitous. This makes its service maintenance highly complex and difficult as servers of a cloud platform could face various internal and external risks that could crash or compromise the system. Analyzing auditing logs is a common effective method to diagnose a large computing system to take preventive actions to mitigate or divert any damages that could result from any threats. However, the various components of such a system might generate a huge amount of log data in real time[18]. It is a formidable task to extract relevant information from a large pool of data this large. Common log-based fault prediction methods include:

1. PCRE (Perl Compatible Regular Expressions) – This is a normalizing technique, which utilizes a library that could handle special separators and regular expressions to extract desired information from unstructured data[13].

2. FFDA (Field Failure Data Analysis) – This method targets specific contents in log messages and groups relevant information together. The main step filters out redundant error entries and irrelevant entries from the log [14]. It then groups the error entries related to the same fault manifestation, and isolates accidental patterns.

3. Neural Network Log Analyzer – This technique uses neural network algorithms to construct an intelligent log analyzer, which can detect known and unknown network intrusions automatically. The log analyzer is trained with multi-layer neural network algorithms for predicting tasks[11].

The above techniques have various limitations. For example, PCRE can only identify isolated problems, but not the correlations among them; FFDA does not have self-learning ability and adaptability. A neural network, which is a mature and effective tool, has many advantages such as having rigorous derivation processes, solid theoretical basis, and strong versatility[16]. It often achieves good results in prediction compared to other methods. However, in the cases that the log time series containing data, which are non-linear, non-stationary, highly noisy and chaotic, a neural network’s training capacity could be sharply reduced and it might yield poor prediction results.

Wavelet transform has become a fast-developing and popular method in signal processing since the 1980s. Because of its powerful feature extraction capability, it is often regarded as a microscope in mathematics [7] and a powerful tool for representing nonlinearities[10]. The concept of wavelet neural network (WNN), which combines the wavelet transform with neural network algorithms was first proposed by Q. Zhang in 1992[17] and in subsequent years, wavelet neural networks have demonstrated remarkable re-
results in prediction, classification, and modeling of different nonlinear signals[6, 5].

Open-source software has been playing a critical role in recent technology developments. A lot of breakthroughs in technology applications such as Watson’s Jeopardy win[4] and the phenomenal 3D movie Avatar[3] are based on open-source software. It is a significant task to explore the usage of available open-source or free tools to develop software applications for research or for commercial use[19, 20, 21, 22, 23, 27]. We report in this paper the use of open-source tools to design and develop a new log analysis application based on wavelet neural network algorithms. It employs wavelet transform to decompose input log data into Approximation Coefficients (AC) and Detail Coefficients (DC). It then performs thresholding on the coefficients, obtaining the denoised data series and feeding them to a neural network. The application is written in Java, an open-source platform, and the Java open-source library JWave[24] is used to perform the wavelet transform in the application. Another Java package, Deeplearning4J[25], which is an open-source distributed learning library for the JVM, is used to implement neural network algorithms. The datasets used in the experiments of this study are produced by a virtualized cloud platform, their features including create-time, host id, program id, and severity.

2. Neural Network

Neural networks are computing systems consisting of interconnected basic processing units (also called neurons or nodes) that vaguely resembles the structure and operation of the biologic nervous system[5]. A neural network can “learn” (i.e. progressively improve performance on) tasks by considering examples, without task-specific programming.

Each neuron is associated with a specific function called activation function, \( f() \), as shown in Figure 1. A weight \( w_i \) is associated with the connection between two neurons, which represents the signal strength passing between them. The output value \( y_i \) of the neuron \( i \) is given by:

\[
net_i = \sum_{j=1}^{n} w_{ij} x_j - b_i \\
y_i = f(net_i)
\]  

(1)

In the equation, the function \( f \) is an activation function. If we treat the negative of the bias \( b_i \) of the neuron shown in Figure 1 as a special input \( x_0 \) with corresponding weight \( w_{i0} \), the equation can be rewritten as

\[
net_i = \sum_{j=0}^{n} w_{ij} x_j \\
y_i = f(net_i)
\]  

(2)

If the net value \( net_i \) of the neuron is positive, the neuron is in the firing state; if it is negative, the neuron is in the inhibitory state.

In a multi-layer neural network, the outputs of one layer could be the inputs of another layer. The middle layers, which do not interact with the external world directly are referred to as hidden layers. Figure 2 shows a simple 3-layer neural network.

![Three Layers Neural Network](image)

In Figure 2, the leftmost and rightmost layers are the input and output layers respectively, and the layer between them is a hidden layer. The number of neurons in an input layer and an output layer is determined by the dimension of the actual input and output requirements[1]. The number of neurons in a hidden layer depends on the complexity of the problems that the system intends to solve. The more complex the problem is, the more neurons the hidden layer needs. In the figure, \( a_i^{(l)} \) represents the activation value of the i-th neuron of layer l. When \( l = 1, a_i^{(1)} = x_i \), the i-th input value. The +1 value inside a circle denotes a bias value.
The symbol $h_{W,b}(X)$ denotes the final output value with a set of weights $W$ and a set $b$ of bias values; it can be computed using formula (2) from the outputs of layer 2. The output of each neuron can be computed from the ‘outputs’ $(x'_i, s)$ of layer 1. This technique is referred to as forward propagation.

A neural network can be initialized with weights of random values. During the training phase, the system compares the final output value with a target value, utilizing the difference to adjust the network parameters. This process is called backpropagation or BP. The BP algorithm derives the difference for each neuron backwardly from hidden layers to the input layer. In this way, each neuron in every layer receives error signals, and adjust its weight accordingly. In the training phase, the weights are adjusted repeatedly until the difference between the network output and the target value is acceptable or the preset training time is up[12].

3. Wavelet Transform

Wavelet transform has been widely used in signal processing. It was first proposed to make up the shortcomings of Fourier Transform and Short Time Fourier Transform[15] for the analysis and processing of non-stationary signals. Wavelet transform[15] can extract the time and frequency information of signals through a series of basic transformation starting from an original wavelet called mother wavelet. Specifically, the time features can be obtained by the translation of a mother wavelet, while frequency features can be obtained by changing the scale of the mother wavelet. Using wavelet transform, we can decompose a signal into a series of wavelets with different scales and positions. These wavelets are dilated and translated forms of a mother wavelet $\psi$:

$$\psi_{s,p}(t) = \frac{1}{\sqrt{s}} \psi\left(\frac{t - p}{s}\right)$$

In equation (3), $s$ is the scale or dilation parameter and $p$ is the shift or translation parameter. The value $\frac{1}{\sqrt{s}}$ is used for normalizing $||\psi_{s,p}(t)|| = 1$. The scale value determines the stretch or compression level of the wavelet[2]. There are several types of wavelet transform: continuous wavelet transform (CWT), discrete wavelet transform (DWT), and wavelet packet transform (WPT). For instance, consider $f(t)$ to be the signal, and $\psi$ to be the shifted wavelet with certain scale and position. CWT calculates the sum of the inner product between signal $f(t)$ and the mother wavelet $\psi$:

$$C_x(s,p) = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} f(t)\psi\left(\frac{t - p}{s}\right)dt, \ s > 0$$

By adjusting the scale and position, CWT can get a series of coefficients, which represent the relationship between the wavelets and the corresponding parts of the signal. Because the continuous variable of scale and translation may cause a computational problem, DWT is introduced. DWT can decompose a signal into different frequency components, and it can also recompose them into a signal. WPT is very similar to DWT; the differences are that DWT only decomposes the approximation coefficients, while WPT decomposes both the approximation and the detail coefficients. Therefore, WPT is more flexible and performs better for complex analysis.

In the project, the main task of wavelet transform is data denoising, removing the noisy part of the signal while keeping the original signal intact as much as possible. A signal can be expressed as:

$$s(t) = f(t) + e(t)$$

where $s(t)$ is the observed signal, consisting of an effective signal component $f(t)$, and a noise component $e(t)$. Extracting $f$ from the noisy signal $s$ is the objective of denoising. Wavelet thresholding denoising[8, 9] is an effective method to remove noise. As noise is random and mostly high-frequency, after a wavelet transform, its coefficients are mostly detail coefficients and are small as compared to those of the effective signal. Therefore, we can filter out the noise by setting any coefficients smaller than a threshold to 0. The reconstructed signal could be mostly free of noise and will be fed into a neural network for training. In our work, we experimented with various kinds of wavelets, including Haar, Morlet, Daubechies, Coiflet, and Symlets wavelets.

4. Wavelet Neural Network Log Analyzer

In this project, we utilize Deeplearning4J to implement the computational basis of the neural network, and JWave to implement wavelet transform and the support of wavelets. The following are the steps of integrating the two packages:

1. Create a multi-layer neural network consisting of LSTM (Long Short-Term Memory) and hidden layers.
2. Create a wavelet transform denoising processor that allows one to select a wavelet with n levels. The processor applies wavelet transform to decompose the noisy data series into a set of approximation and detail coefficients. It calculates a threshold, and applies thresholding to the detail coefficients to remove noises. It then recomposes the coefficients to obtain a denoised data series.
3. Create a connector that takes the outputs of the wavelet transform processor as inputs and feeds the denoised data series into the neural network, which trains, predicts, and selects the best wavelet basis. For a conventional neural network model, which does not involve any wavelet transform, the connector simply takes the
original noisy data and feeds them to the neural network.

Figure 3 shows the architecture of the software application.

5. Experiment Preparation and Model Selection

The experimental platform of this study is a virtualized cloud platform, which serves a variety of applications. The cloud platform consists of a virtualized Linux machine and a Windows machine running in a research lab in CSUSB. Our purpose is to analyze the historical log data of the platform, then make a prediction of the future possible events.

As there are many hosts and executing programs in the virtualized cloud platform we need to first identify interesting features for analysis and when an event occurs, we need to identify its time, location, and event content. To simplify the experiment, we have chosen only to record the event occurring time, its host identifier, program (application) identifier, and severity. Since a neural network needs numerical representations of features, we convert all the qualitative values (text, characters, etc) to quantitative values before feeding them to the neural network. So labeling all the hosts, programs, and severity names with numbers are necessary. We assign each of the three features, the host id, program id, and severity names with specific value domains according to the business requirements. For example, we define severity values from 0 to 19, the higher the value, the more serious the severity. Similarly, based on the business relations, we assign host ids from 0 to 4, and programs from 0 to 50. With the values defined above, we changed the log code in the virtualized cloud platform according to the format we defined. Then the platform ran those applications, serving user requests. After a period of time, we got a log dataset. The following table shows a sample of the experimental data.

<table>
<thead>
<tr>
<th>Time</th>
<th>Host ID</th>
<th>Program ID</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1466536701</td>
<td>3</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>1466536704</td>
<td>3</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>1466536725</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1466536726</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1466536727</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1466536729</td>
<td>3</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>1466536784</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1466536785</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1466536787</td>
<td>1</td>
<td>42</td>
<td>11</td>
</tr>
<tr>
<td>1466536810</td>
<td>1</td>
<td>42</td>
<td>11</td>
</tr>
<tr>
<td>1466536824</td>
<td>3</td>
<td>13</td>
<td>16</td>
</tr>
</tbody>
</table>

In the table, the **Time** column consists of timestamps (time series) of events, which are sequence numbers of numerical data points.

Time series forecasting is the use of a model to predict future values based on previously observed values[26]. RNN (Recurrent Neural Networks) is a kind of neural network for handling time series data forecasting. Not only that it can learn the current information but also take advantage of previous sequence dependence to make better predictions. The Long Short-Term Memory (LSTM) network is a type of RNN that we have used. It is suitable for analyzing and predicting longer time dependencies, and overcomes the drawback called vanishing gradient problem in RNN networks. LSTM is a powerful network that learns the most important previous information and understands whether this information is useful or not for making predictions. We use LSTM as the neural network model and incorporate wavelet transform to make time series log dataset predictions.

The implementation of the neural network is crucial for training and predicting events occurrence. In recent years, many neural network frameworks have been developed for supporting training, such as TensorFlow, Deeplearning4J, Theano, Caffe, and Torch, which are mostly open-source. This study uses Deeplearning4J as the neural network supporting framework, which is a Java-based toolkit, developed by Skymind Company, for building, training and deploying neural networks[25]. The version we have used, 0.8.0 supports the LSTM model we need in this study.

For wavelet transforms, we use the JWave library, which is a Java implementation of a series of algorithms, including Wavelet Packet Transform, Fast Wavelet Transform, and Discrete Fourier Transform, with 1-D, 2-D, and 3-D calculations. The wavelet transform algorithms we used are normalized orthogonal (orthonormal) wavelets including Haar, Coiflet, Daubechies, Symlets, and even some Bi-Orthogonal[24]. As it is also Java-based, it can be easily integrated with Deeplearning4J in our experiments.

As we discussed above, the log data set has four different entities: time, host id, program id, and severity. We treat time as a sequential identifier while using the other three entities for training and testing. Therefore, both the input layer and output layer have 3 nodes. The input log data are first fed into the wavelet transform processor module, where the data series are denoised. The number of hidden layer units is adjusted during the experiment according to the prediction criteria. Between the hidden layer and the output layer, a dense layer is added for the change of dimensions and features extraction.

Based on the model tuning results, the parameters of the LSTM model are set as shown in Table 2. We did experiments with many wavelet transforms and compared their performance. We found that the Haar and Daubechies 3
are the best wavelets, which are also most commonly used in time series dataset. Therefore, our study uses these two wavelets in the wavelet transform processor.

| Table 2 Parameters of the LSTM Model Used in This Study |
|------------------|------------------|
| **Name** | **Value** |
| Learning rate | 0.02 |
| Iterations | 1 |
| Hidden layer 1 nodes | 256 |
| Hidden layer 2 nodes | 256 |
| Dense layer nodes | 32 |
| Backprop type | TruncatedBPTT |
| TruncatedBPTT length | 32 |

### 6. Results and Discussions

We used the datasets created in the preparation stage discussed above as inputs to our program, which splits a dataset into two parts, typically, 60% for training and 40% for testing. We had created three types of neural networks (NN): Haar wavelet NN, Daubechies 3 (DB3) wavelet NN, and conventional NN. These three network models were initialized with the same parameters and trained with the same datasets and the results were compared. Note that the conventional NN is simply a plain neural network, not involving any wavelet transform.

Figure 4 shows the original noisy severity signal (top half) and the denoised signal using Haar wavelet (bottom half). The high noise levels were introduced intentionally or unintentionally for experiment purposes.

One can see that Haar wavelet denoising removes a considerable amount of noise while preserving the sharp features in the signal, meaning that it can preserve important signal features while removing noise. We obtained similar results when applying other wavelets and using other features.

All the LSTM parts of the three models were configured with the same structures and parameters. They were trained with 60 epochs and we obtained prediction results of severity, program id, and host id by Haar, DB3, and Conventional neural network models. Figure 5 shows the severity prediction results using (a) Haar wavelet NN, (b) DB3 wavelet NN, and (c) Conventional NN. One can see that a wavelet NN predicts significantly better than a conventional NN. We obtained similar results for other features, host id and program id.
where \( y'(i) \) is the predicted value and \( y(i) \) is the actual value; \( N \) is the dataset size. Small values of RMSE imply good predictions while large values indicate poor predictions. Table 3 tabulates the RMSE values we obtained for the three neural network models.

![Severity Predictions](image)

**Figure 5.** Severity Predictions

We also computed the Root Mean Squared Error (RMSE) which is given by

\[
RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y'(i) - y(i))^2}
\]  

(6)

<table>
<thead>
<tr>
<th>Neural Network Model</th>
<th>RMSE</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Severity</td>
<td>Program id</td>
<td>Host id</td>
</tr>
<tr>
<td>Haar NN</td>
<td>0.5319</td>
<td>4.0112</td>
<td>0.2031</td>
</tr>
<tr>
<td>DB3 NN</td>
<td>0.4499</td>
<td>4.0420</td>
<td>0.2137</td>
</tr>
<tr>
<td>Conventional NN</td>
<td>0.7533</td>
<td>4.9809</td>
<td>0.2557</td>
</tr>
</tbody>
</table>

Table 3  RMSE Values of Three NN Models

It is clear from the table that while Haar NN and DB3 NN show similar prediction precisions, they predict significantly better than a conventional NN. Take DB3 for example, the MSE values of severity, program id, and host id predictions are 40.3%, 18.9%, and 16.4% better than the corresponding values of the conventional NN respectively. It is evident that a wavelet neural network model performs remarkably better than the conventional neural network model. In the same capacity, it also yields better results in pattern recognition, and convergent speed.

7. Conclusions

We have utilized Java-based open-source tools to develop applications that could be used in virtualized cloud platform management. We conducted a study on the time series log datasets generated by a virtual cloud platform setup in CSUSB for research purposes. We found that by combining a wavelet transform with a neural network, better predictions of the occurrences of events in the servers can be obtained as compared to using a plain neural network. At the final stage of the experiment, we selected three types of neural networks: Haar wavelet NN, DB3 wavelet NN, and conventional NN. They share the same LSTM network architecture and configurations. The three models were fed with the same training and testing datasets. Experimental results reveal that the wavelet NN models produce much better prediction results than a conventional neural network. They could also recognize data patterns significantly better, and their training converges much faster. Therefore, the work demonstrates that combining a wavelet transform with a neural network could create better tools for managing a virtualized cloud platform.

References


