Neural networks have become a powerful technique to extract useful knowledge from large amounts of raw, seemingly unrelated data. One of the most suitable tools for implementing neural networks is the Java language. Besides being a very popular programming language, there are many APIs and packages that help in the development, not mentioning its "write once, run everywhere" portability.

This book gives you a complete walkthrough of the process of developing neural networks with Java, from the very basic to the advanced practical examples.

First, you will learn the basics of neural networks and their process of learning. We then focus on perceptrons and their features. Next, you will implement self-organizing maps using the concepts you’ve learned. Furthermore, you will learn about some of the applications such as weather forecasting, disease diagnosis, customer profiling, and optical character recognition (OCR). Finally, you will learn methods to optimize and adapt neural networks in real time.

All the examples generated in the book are provided in the form of illustrative source code, which merges object-oriented programming (OOP) concepts and neural network features to enhance your learning experience.

Who this book is written for
This book is for developers or computing students with just basic Java programming knowledge. No previous knowledge of neural networks is required as this book covers the concepts from scratch.

What you will learn from this book
- Get to grips with the basics of neural networks and what they are used for
- Develop neural networks using hands-on examples
- Explore and code the most widely-used learning algorithms to make your neural network learn from most types of data
- Discover the power of a neural network’s unsupervised learning process to extract the intrinsic knowledge hidden behind the data
- Apply the code generated in practical examples, including weather forecasting and pattern recognition
- Understand how to make the best choice of learning parameters to ensure you have a more effective application
- Select and split data sets into training, test, and validation, and explore validation strategies
- Discover how to improve and optimize your neural network


Fábio M. Soares
Alan M.F. Souza

Unleash the power of neural networks by implementing professional Java code
In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 5 'Forecasting Weather'
- A synopsis of the book’s content
- More information on Neural Network Programming with Java
About the Authors

Fábio M. Soares holds a master's degree in applied computing from UFPA and is currently a PhD candidate at the same university. He has been designing neural network solutions since 2004 and has developed applications with this technique in several fields, ranging from telecommunications to chemistry process modeling, and his research topics cover supervised learning for data-driven modeling.

He is also self-employed, offering services such as IT infrastructure management as well as database administration to a number of small- and medium-sized companies in northern Brazil. In the past, he has worked for big companies such as Albras, one of the most important aluminium smelters in the world, and Eletronorte, a great power supplier in Brazil. He also has experience as a lecturer, having worked at the Federal Rural University of Amazon and as a Faculty of Castanhal, both in the state of Pará, teaching subjects involving programming and artificial intelligence.

He has published a number of works, many of them available in English, all including the topics of artificial intelligence applied to some problem. His publications include conference proceedings, such as the TMS (The Minerals Metals and Materials Society), Light Metals and the Intelligent Data Engineering and Automated Learning. He has also has published two book chapters for Intech.
Alan M.F. Souza is computer engineer from Instituto de Estudos Superiores da Amazônia (IESAM). He holds a post-graduate degree in project management software and a master's degree in industrial processes (applied computing) from Universidade Federal do Pará (UFPA). He has been working with neural networks since 2009 and has worked with IT Brazilian companies developing in Java, PHP, SQL, and other programming languages since 2006. He is passionate about programming and computational intelligence. Currently, he is a professor at Universidade da Amazônia (UNAMA) and a PhD candidate at UFPA.
Preface

The life of a programmer can be described as a continual never-ending learning pathway. A programmer always faces challenges regarding new technology or new approaches. Generally, during our lives, although we become used to repeated things, we are always subjected to learn something new. The process of learning is one of the most interesting topics in science, and there are a number of attempts to describe or reproduce the human learning process.

The writing of this book was guided by the challenge of facing new content and then mastering it. While the name neural networks may appear strange or even give an idea that this book is about neurology, we strived to simplify these nuances by focusing on your reasons for deciding to purchase this book. We intended to build a framework that shows you that neural networks are actually simple and easy to understand, and absolutely no prior knowledge on this topic is required to fully understand the concepts we present here.

So, we encourage you to explore the content of this book to the fullest, beholding the power of neural networks when confronting big problems but always with the point of view of a beginner. Every concept addressed in this book is explained in easy language, and also with a technical background. Our mission in this book is to give you an insight into intelligent applications that can be written using a simple language.

Finally, we would like to thank all those who directly or indirectly have contributed to this book and supported us from the very beginning, right from the Federal University of Pará, which is the university that we graduated from, to the data and component providers INMET (Brazilian Institute of Meteorology), Proben1, and JFreeCharts. We want to give special thanks to our advisor Prof. Roberto Limão, who introduced us to the subject of neural networks and coauthored many papers with us in this field. We also acknowledge the work performed by several authors cited in the references, which gave us a broader vision on neural networks and insights on how to adapt them to the Java language in a didactic way.
We welcome you to have a very pleasurable reading experience and you are encouraged to download the source code and follow the examples presented in this book.

What this book covers

Chapter 1, *Getting Started with Neural Networks*, is an introductory foundation on the neural networks and what they are designed for. You will be presented with the basic concepts involved in this book. A brief review of the Java programming language is provided. As in all subsequent chapters, an implementation of a neural network in Java code is also provided.

Chapter 2, *How Neural Networks Learn*, covers the learning process of neural networks and shows how data is used to that end. The complete structure and design of a learning algorithm is presented here.

Chapter 3, *Handling Perceptrons*, covers the use of perceptrons, which are one of the most commonly used neural network architectures. We present a neural network structure containing layers of neurons and show how they can learn by data in basic problems.

Chapter 4, *Self-Organizing Maps*, shows an unsupervised neural network architecture (the Self-Organising Map), which is applied to finding patterns or clusters in records.

Chapter 5, *Forecasting Weather*, is the first practical chapter showing an interesting application of neural networks in forecasting values, namely weather data.

Chapter 6, *Classifying Disease Diagnostics*, covers another useful task neural networks are very good at—classification. In this chapter, you will be presented with a very didactic but interesting application for disease diagnosis.

Chapter 7, *Clustering Customer Profiles*, talks about how neural networks are able to find patterns in data, and a common application is to group customers that share the same properties of buying.

Chapter 8, *Pattern Recognition (OCR Case)*, talks about a very interesting and amazing capability of recognizing patterns, including optical character recognition, and this chapter explores how this can be done with neural networks in the Java language.

Chapter 9, *Neural Network Optimization and Adaptation*, shows advancements regarding how to optimize and add adaptability to neural networks, thereby strengthening their power.
Forecasting Weather

This chapter presents an application of neural networks to the prediction of future weather data. We are going to walk through the entire process of designing a neural network to be applied to this problem, how to choose the neural architecture, the number of neurons, as well as selecting and preprocessing data. Then, the reader will be presented with a dataset on which our neural network is going to make predictions of weather variables using the Java programming language. The topics covered in this chapter are as follows:

- Neural networks for prediction problems
- Selecting data
  - Input/Output variables
  - Filtering
- Preprocessing
  - Normalization
- Java implementation
  - Adaptations
- Empirical design of neural networks
Neural networks for prediction problems

So far, the reader has been presented with a number of neural network implementations and architectures, so now; it is time to get into more complex cases. The power of neural networks in predictions is really astonishing, since they can perform "learning" from historical data in a fashion in which the neural connections are adapted to produce the same results according to some input data. For example, for a given situation (cause), there is a consequence (result) and this is coded as data; the neural network can be used to learn the nonlinear function that maps the situation to the consequence (or the cause to the result).

Prediction problems are an interesting category to apply neural networks to. Let's take a look at a sample table containing weather data:

<table>
<thead>
<tr>
<th>Date</th>
<th>Avg. temperature</th>
<th>Pressure</th>
<th>Humidity</th>
<th>Precipitation</th>
<th>Wind speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 31</td>
<td>23° C</td>
<td>880 mbar</td>
<td>66%</td>
<td>16 mm</td>
<td>5 m/s</td>
</tr>
<tr>
<td>August 1</td>
<td>22° C</td>
<td>881 mbar</td>
<td>78%</td>
<td>3 mm</td>
<td>3 m/s</td>
</tr>
<tr>
<td>August 2</td>
<td>25° C</td>
<td>884 mbar</td>
<td>65%</td>
<td>0 mm</td>
<td>4 m/s</td>
</tr>
<tr>
<td>August 3</td>
<td>27° C</td>
<td>882 mbar</td>
<td>53%</td>
<td>0 mm</td>
<td>3 m/s</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>December 11</td>
<td>32° C</td>
<td>890 mbar</td>
<td>64%</td>
<td>0 mm</td>
<td>2 m/s</td>
</tr>
</tbody>
</table>

The preceding table depicts five variables containing hypothetical values of weather data collected from a hypothetical city, only for the purpose of this example. Now, let's suppose that each of the variables contains a list of values sequentially taken over time. We can think of each list as a time series. On a time-series chart, one can see how they evolve along with time:
The relationship between these time series denotes a dynamic representation of weather in a certain city, as depicted in the preceding chart. We indeed want the neural network to learn these dynamics; however, it is necessary to understand a little bit more about the phenomena, because we need to structure this data in a way that neural networks can process it.

Only after structuring the data can we structure the neural network, that is, the number of inputs, outputs, and hidden nodes. However, there are many other architectures that may be suitable for prediction problems, such as radial basis functions and feedback networks. In this chapter, we will deal with the feedforward multi layer perceptron with backpropagation learning algorithm, to demonstrate how this architecture can be simply exploited to predict weather variables. Also, this architecture presents very good generalized results with good selected data and there is little complexity involved in the design process.

The overall process for designing neural networks for prediction processes is depicted in the following figure:

1. Selecting and Filtering History Data
2. Data Preprocessing
3. Defining of Neural Network Structure
4. Training Neural Network
5. Validating Neural Network

If the neural network fails to be validated (step 5), then usually, a new structure (step 3) is defined, although sometimes, steps 1 and 2 may be repeated. Each of the steps in the figure will be addressed in the following sections of this chapter.

**No data, no neural net – selecting data**

The first thing to do is to select appropriate relevant data that carries most of the system's dynamics that we want the neural network to reproduce. In our case, we need to select data that is relevant for weather forecasting.
While selecting data, getting an expert opinion about the process and its variables can be really helpful. The expert does help a lot in understanding the relationship between the variables, thus selecting them in an appropriate fashion.

In this chapter, we are going to use the data from the Brazilian Institute of Meteorology (INMET - http://www.inmet.gov.br/ in Portuguese), which is freely available on the Internet and we have the rights to apply it in this book. However, the reader may use any free weather database from the Internet while developing applications. Some examples from the English language sources are listed as follows:

- Wunderground (http://wunderground.com/)
- Open weather map (http://openweathermap.org/api)
- Yahoo weather API (https://developer.yahoo.com/weather/)
- U.S. National Climatic Data Center (http://www.ncdc.noaa.gov/)

**Knowing the problem – weather variables**

Any weather database has almost the same variables:

- Temperature (°C)
- Humidity (%) 
- Pressure (mbar)
- Wind speed (m/s)
- Wind direction (°)
- Precipitation (mm)
- Sunny hours (h)
- Sun energy (W/m²)

This data is usually collected from meteorological stations, satellites, or radars, on an hourly or daily basis.

Depending on the collection frequency, some variables may be summarized with average, minimum, or maximum values.

The data units may also vary from source to source; that's why the units should always be observed.
Choosing input and output variables

Neural networks work as a nonlinear block that may have a predefined number of inputs and outputs, so we have to select the role that each weather variable will play in this application. In other words, we have to choose which variable(s) the neural network is going to predict and by using which input variables.

Regarding time series variables, one can derive new variables by applying historical data. This means that given a certain date, one may consider this date's values and the data collected (and/or summarized) from past dates, therefore extending the number of variables.

While defining a problem to use neural networks on, we need to consider one or more predefined target variables: predict temperature, forecast precipitation, measure insolation, and so on. However, in some cases, one may want to model all the variables and to find the causal relationships between them. To identify a causal relationship, there are a number of tools that can be applied:

- Cross-correlation
- Pearson's coefficient
- Statistical analysis
- Bayesian networks

For the sake of simplicity, we are not going to explore these tools in this chapter; however, the reader is recommended to go to the references [Dowdy & Wearden, 1983; Pearl, 2000; Fortuna et al., 2007] for obtaining more details about these tools. Instead, since we want to demonstrate the power of neural networks in predicting weather, we will choose the average temperature of a given day, based on the other four variables, on the basis of the current technical literature, which is cited in the preceding reference.

Removing insignificant behaviors – Data filtering

Sometimes, some issues are faced while getting data from some source. The common problems are as follows:

- Absence of data in a certain record and variable
- Error in measurement (for example, when a value is badly labeled)
- Outliers (for example, when the value is very far from the usual range)
To handle each of these issues, one needs to perform filtering on the selected data. The neural network will reproduce exactly the same dynamics as those of the data that it will be trained with, so we have to be careful in feeding it with bad data. Usually, records containing bad data are removed from the dataset, ensuring that only "good" data are fed to the network.

To better understand filtering, let's consider the dataset as a big matrix containing $n$ measurements and $m$ variables.

$$
A = \begin{bmatrix}
  a_1(1) & \cdots & a_m(1) \\
  a_1(2) & \cdots & a_m(2) \\
  \vdots & \ddots & \vdots \\
  a_1(n) & \cdots & a_m(n)
\end{bmatrix}
$$

Where $a_j(i)$ denotes the measurement of variable $j$ at moment $i$.

So, our task is to find the bad records and delete them. Mathematically, there are a number of ways of identifying a bad record. For error measurement and outlier detection, the following three-sigma rule is very good:

$$
|d_i| = \left| \frac{x_i - E[X]}{\sigma_X} \right| = \begin{cases} 
> 3 \text{ bad record, remove} \\
\leq 3 \text{ good record, keep it}
\end{cases}
$$

Where $x_i$ denotes the value of the $i^{th}$ measurement, $E[X]$ represents the average value, $\sigma_X$ indicates the standard deviation, and $d_i$ refers to the weighted distance from the average. If the absolute distance of the $i^{th}$ measurement fails to fit in less than three records, the $i^{th}$ measurement will be labeled as a bad measurement, and although the other variables from the same instance (row of the matrix) are good, one should discard the entire row of the dataset.

**Adjusting values – data preprocessing**

Raw data collected from a data source usually presents different particularities, such as data range, sampling, and category. Some variables result from measurements, while the others are a summary or even calculated. Preprocessing means to adapt these variables' values to form neural networks that can handle them properly.
Regarding weather variables, let’s take a look at their range, sampling, and type, shown in the following table:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Unit</th>
<th>Range</th>
<th>Sampling</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean temperature</td>
<td>°C</td>
<td>23.86–29.25</td>
<td>Hourly</td>
<td>Average of hourly measurements</td>
</tr>
<tr>
<td>Precipitation</td>
<td>Mm</td>
<td>0–161.20</td>
<td>Daily</td>
<td>Accumulation of daily rain</td>
</tr>
<tr>
<td>Insolation</td>
<td>h</td>
<td>0–10.40</td>
<td>Daily</td>
<td>Count of hours receiving sun radiation</td>
</tr>
<tr>
<td>Mean humidity</td>
<td>%</td>
<td>65.50–96.00</td>
<td>Hourly</td>
<td>Average of hourly measurements</td>
</tr>
<tr>
<td>Mean wind speed</td>
<td>km/h</td>
<td>0.00–3.27</td>
<td>Hourly</td>
<td>Average of hourly measurements</td>
</tr>
</tbody>
</table>

Except for insolation and precipitation, the variables are all measured and share the same sampling, but if we wanted, for example, to use an hourly dataset, we would have to preprocess all the variables to use the same sample rate. Three of the variables are summarized using daily average values, but if we wanted to, we could use hourly data measurements. However, the range would surely be larger.

**Equalizing data – normalization**

Normalization is the process to get all the variables into the same data range, usually with smaller values, between 0 and 1 or -1 and 1. This helps the neural network to present values within the variable zone in activation functions such as sigmoid or hyperbolic tangent:
Values too high or too low may drive neurons to produce values that are too high or too low as well for the activation functions, therefore leading the derivative for these neurons to be too small, near zero.

The normalization should consider a predefined range of the dataset. It is performed right away:

\[ X_{\text{norm}} = (N_{\text{max}} - N_{\text{min}}) \left( \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \right) + N_{\text{min}} \]

Where \( N_{\text{min}} \) and \( N_{\text{max}} \) represent the normalized minimum and maximum limits, respectively; \( X_{\text{min}} \) and \( X_{\text{max}} \) denote \( X \) variable's minimum and maximum limits, respectively; \( X \) indicates the original value; and \( X_{\text{norm}} \) refers to the normalized value.

If we want the normalization to be between 0 and 1, for example, the equation is simplified as follows:

\[ X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \]

By applying the normalization, a new "normalized" dataset is produced and is fed to the neural network. One should also take into account that a neural network fed with normalized values will be trained to produce normalized values on the output, so the inverse (denormalization) process becomes necessary as well.

\[ X = (X_{\text{max}} - X_{\text{min}}) \left( \frac{X_{\text{norm}} - N_{\text{min}}}{N_{\text{max}} - N_{\text{min}}} \right) + X_{\text{min}} \]

or:

\[ X = (X_{\text{max}} - X_{\text{min}})[X_{\text{norm}}] + X_{\text{min}} \]

For the normalization between 0 and 1.
Java implementation for weather prediction

In order to implement this case in Java, we had to make some adjustments in the already written code. The `NeuralNet` class is updated with a new method called `getNetOutputValues()`, to give some output values given a training input dataset. This method performs almost the same operation as the forward method in the backpropagation phase, except for the fact that it returns a matrix containing the output dataset.

In addition, we had to add two components to the project (package `edu.packt.neuralnet.util`): data and chart.

Plotting charts

Charts can be drawn in Java by using the freely available package **JFreeChart** ([http://www.jfree.org/jfreechart/](http://www.jfree.org/jfreechart/)). This package is attached with this chapter's source code. So, we designed a class called `Chart`. It implements methods basically for plotting data series by making calls to natively implemented methods of the JFreeChart classes. The following table shows a list of methods contained in this class:

<table>
<thead>
<tr>
<th>Class name: Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attributes</strong></td>
</tr>
<tr>
<td>public enum ChartPlotTypeENUM { FULL_DATA, COMPARISON; }</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>public void plotXYData(Object[] vector, String chartTitle, String xAxisLabel, String yAxisLabel)</td>
</tr>
<tr>
<td><strong>Parameters</strong>: Vector with data to plot, chart title, x-axis label, and y-axis label</td>
</tr>
<tr>
<td>public void plotXYData(double[] [] matrix, String chartTitle, String xAxisLabel, String yAxisLabel, ChartPlotTypeENUM chartPlotType)</td>
</tr>
<tr>
<td><strong>Parameters</strong>: Matrix with data to plot, chart title, x-axis label, y-axis label, and plot type</td>
</tr>
<tr>
<td>private String selectComparisonSeriesName(int index)</td>
</tr>
<tr>
<td><strong>Parameters</strong>: Index</td>
</tr>
</tbody>
</table>
Forecasting Weather

<table>
<thead>
<tr>
<th>private String</th>
<th>Method to select temperature series name</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectTemperatureSeriesName(int index)</td>
<td>Parameters: Index</td>
</tr>
<tr>
<td></td>
<td>Returns: Series name</td>
</tr>
</tbody>
</table>

Class Implementation with Java: file Chart.java

Handling data files
To work with data files, we have to implement a class called Data. It currently performs reads from the so-called CSV format, which is suitable for data import and export. This class also performs preprocessing on the data by means of normalization.

| Class name: Data |
| Attributes |
| private String path; | Variable to store the CSV file folder path |
| private String fileName; | Attribute to store the CSV file name (with extension) |

|          | public enum NormalizationTypesENUM { |
|          |   MAX_MIN, MAX_MIN_EQUALIZED; |
|          | } |

| Constructors |
| public Data(String path, String fileName) | Constructor to set path and filename attributes |
| public Data( ) | Empty constructor to create an empty object |

| Methods |
| Note: The getters and setters methods of this attribute were created too. |
| public double[][] rawData2Matrix(Data r) throws IOException | Method to read raw data (CSV file) and convert to a double Java matrix |
| Parameters: Data object |
| Returns: Double matrix with raw data |

| private String defineAbsolutePath(Data r) throws IOException | Method to define the absolute CSV file path |
| Parameters: Data object |
| Returns: String with the absolute CSV file path |
Building a neural network for weather prediction

To forecast weather, we collected daily data from the Brazilian Institute of Meteorology (INMET). The data was measured from a Brazilian city located in the Amazon region.

From the eight variables available at the INMET website, five were selected for use in this project, where the average of the maximum and the minimum temperature became the mean temperature variable. The neural network was trained to forecast the average temperature. So, the structure of the neural network is as shown in the following figure:
Forecasting Weather

We designed a class called `Weather` exclusively for the weather case. It only has a static main method and is solely aimed at reading the weather data files, creating and training a neural network with this data, and plotting the error for validation. Let's take a glance at how the data files are read inside this class:

```java
Data weatherDataInput = new Data( "data", "inmet_13_14_input.csv" );
Data weatherDataOutput = new Data( "data", "inmet_13_14_output.csv" );

//sets the normalisation type
NormalizationTypesENUM NORMALIZATION_TYPE = Data. NormalizationTypesENUM.MAX_MIN_EQUALIZED;

try {
    double[][] matrixInput = weatherDataInput.rawData2Matrix( weatherDataInput );
    double[][] matrixOutput = weatherDataOutput.rawData2Matrix( weatherDataOutput );

    //normalise the data
    double[][] matrixInputNorm = weatherDataInput.normalize( matrixInput, NORMALIZATION_TYPE );
    double[][] matrixOutputNorm = weatherDataOutput.normalize( matrixOutput, NORMALIZATION_TYPE );
}

Then, the main method builds a neural network with four hidden neurons and sets the training dataset, as shown in the following code:

```java
NeuralNet n1 = new NeuralNet();
n1 = n1.initNet(4, 1, 4, 1);

n1.setTrainSet( matrixInputNorm );
n1.setRealMatrixOutputSet( matrixOutputNorm );

n1.setMaxEpochs( 1000 );
n1.setTargetError( 0.00001 );
n1.setLearningRate( 0.5 );
n1.setTrainType( TrainingTypesENUM.BACKPROPAGATION );
n1.setActivationFnc( ActivationFncENUM.SIGLOG );
n1.setActivationFncOutputLayer(ActivationFncENUM.LINEAR);

NeuralNet n1Trained = new NeuralNet();

n1Trained = n1.trainNet( n1 );

System.out.println();
```
Here, the network is trained, and then, the charts of the error are plotted. The following lines show how the chart class is used:

```java
Chart c1 = new Chart();
c1.plotXYData( n1.getListOfMSE().toArray(), "MSE Error", "Epochs", "MSE Value" );

//TRAINING:
double[][] matrixOutputRNA = n1Trained.getNetOutputValues( n1Trained );
double[][] matrixOutputRNADenorm = new Data().denormalize( matrixOutput, matrixOutputRNA, NORMALIZATION_TYPE);

ArrayList<double[][]> listOfArraysToJoin = new ArrayList<double[][]>();
listOfArraysToJoin.add( matrixOutput );
listOfArraysToJoin.add( matrixOutputRNADenorm );

double[][] matrixOutputsJoined = new Data().joinArrays( listOfArraysToJoin );

Chart c2 = new Chart();
c2.plotXYData( matrixOutputsJoined, "Real x Estimated - Training Data", "Weather Data", "Temperature (Celsius)" , Chart.ChartPlotTypeENUM.COMPARISON );
```

In the following graph, it is possible to see the MSE training error plotted. The x-axis represents 1000 points (epochs of training), and the y-axis shows the variation of the MSE values. It is noticed that before the 100th epoch, the MSE value establishes.
Another graph is displayed next. It shows a comparison between the real (red line) and the estimated (blue line) average temperature. Dotted black lines symbolize the margins of error (-1.0 °C and +1.0 °C).

**Empirical design of neural networks**

While using neural networks in regression problems (that include prediction), there is no fixed number of hidden neurons, so usually, the solver chooses an arbitrary number of neurons and then varies it according to the results produced by the networks created. This procedure may be repeated a number of times until a network with a satisfying criterion is found.

**Choosing training and test datasets**

In order to attest the neural network's capability to properly respond to new data, it is useful to have two separate datasets, called training and test datasets. In this application, we worked with two distinct periods, one for each dataset.

<table>
<thead>
<tr>
<th>Period</th>
<th>Begin</th>
<th>End</th>
<th>Type</th>
<th>Number of records</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/01/2013</td>
<td>31/12/2014</td>
<td>Training</td>
<td>730</td>
<td>93.8</td>
</tr>
<tr>
<td>2</td>
<td>30/04/2015</td>
<td>16/06/2015</td>
<td>Test</td>
<td>48</td>
<td>6.2</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>778</td>
<td>100</td>
</tr>
</tbody>
</table>

The recommendation is for the training set to have at least 75% of the overall dataset.
Designing experiments

Experiments can be performed on the same training and test datasets, but by varying the other network parameters, such as the learning rate, normalization, and the number of hidden units. In this case, we performed 12 experiments, whose parameters were chosen as shown in the following table:

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Number of neurons in hidden layer</th>
<th>Learning rate</th>
<th>Data normalization type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.1</td>
<td>MAX_MIN</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>MAX_MIN_EQUALIZED</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0.5</td>
<td>MAX_MIN</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>MAX_MIN_EQUALIZED</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0.9</td>
<td>MAX_MIN</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>MAX_MIN_EQUALIZED</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>0.1</td>
<td>MAX_MIN</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>MAX_MIN_EQUALIZED</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>0.5</td>
<td>MAX_MIN</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>MAX_MIN_EQUALIZED</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0.9</td>
<td>MAX_MIN</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>MAX_MIN_EQUALIZED</td>
</tr>
</tbody>
</table>

The objective is to choose a neural network that presents the best performance from the experiments. The best performance is assigned to the network that presents the lowest MSE error, but an analysis of generalization with the test data is also useful.

While designing experiments, consider starting always from a relatively low number of hidden neurons, since it is desirable to have low computational cost.

Results and simulations

After running the 12 experiments, we found the following MSE errors:

<table>
<thead>
<tr>
<th>Experiment</th>
<th>MSE training error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.6551720491360E-4</td>
</tr>
<tr>
<td>2</td>
<td>0.3034120360203837</td>
</tr>
<tr>
<td>3</td>
<td>3.8543681112765E-4</td>
</tr>
<tr>
<td>4</td>
<td>0.3467096464653794</td>
</tr>
</tbody>
</table>
Forecasting Weather

<table>
<thead>
<tr>
<th>Experiment</th>
<th>MSE training error</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4.6319274448088E-4</td>
</tr>
<tr>
<td>6</td>
<td>0.4610935945738937</td>
</tr>
<tr>
<td>7</td>
<td>2.6604395044000E-4</td>
</tr>
<tr>
<td>8</td>
<td>0.2074979827120087</td>
</tr>
<tr>
<td>9</td>
<td>2.7763926432754E-4</td>
</tr>
<tr>
<td>10</td>
<td>0.2877786584371894</td>
</tr>
<tr>
<td>11</td>
<td>3.4582006086257E-4</td>
</tr>
<tr>
<td>12</td>
<td>0.4610935945709355</td>
</tr>
</tbody>
</table>

The following graph exhibits neural net 5th experiment's comparison between real and estimated values, and the respective margins of error:
The following graph shows that the same results as those discussed in the previous paragraph, but for neural network 10th experiment:

Although experiment 10 has a larger MSE than experiment 5 and 10's chart presents a better generalization behavior. Therefore, we can conclude the following:

- Considering only the final MSE value to decide about the neural net quality is not recommended.
- Estimated value from experiment 10 follows the real value closer than that from experiment 5.
- Neural net obtained in experiment 10 preserves the trending by ascent and descent better than that obtained in 5, as may be viewed between weather data 1 and 17.

Therefore, by viewing the corresponding charts, we chose network 10 to be the most suitable for weather prediction.
Summary

In this chapter, we’ve seen an interesting practical application of neural networks. Weather forecasting has always been a rich research field, and indeed, neural networks are widely used for these tasks. In this chapter, the reader also learned how to prepare similar experiments for prediction problems. The correct application of techniques for data selection and preprocessing can save a considerable amount of time while designing a neural network for the prediction. This chapter also serves as a foundation for the following chapters, since all of them will focus on practical cases, so the concepts learned here will be explored widely in the rest of the book.

In the next chapter we will cover classification tasks, which is another common research field where neural networks can be used. Two case studies will be presented, covering the whole process on how neural networks are built for disease diagnosis.
Where to buy this book
You can buy Neural Network Programming with Java from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.