WEIGHSCORE,

The

Neural Network Server

to automate centralized decision making
in any field of business
suitable for online training
for an immediate adjustment to volatile real world situations

and

The

Neural Network
Command Line Tool

to convert an implicit experience
hidden in available statistical data
by weighing the influence of different factors
and factor combinations
and storing numeric influence representations
in an X.M.L. based neural scorecard.

The User Manual.

Contents

Introduction

This is a common reference manual for the command line and the server versions of Weighscore. If you don't use the Neural Server, skip section `nns:` The Neural Server.

For some, it might be useful to start reading the reference manual from the last section, which is entitled the Neural network developer reference, where you can find information about how to develop and train neural networks with our tools.

Terms

This manual uses some terms that may differ from those adopted in academic circles. Here are some of them:

Question—a set of input values.

Answer—a set of output values. Often referred to as *response*.

Ask—the neural network querying process. You give the network a question, and it produces the answer.

Teach—in this reference manual—a training action. You teach the network by providing it with a question coupled with the correct answer. The network is taught to give that answer, to that question. The teaching action affects the synapses' weights and neurons' biases.

Test—in this reference manual—a probing action. The probing action provides the network with a question and an answer as while teaching; he result is the error—the difference between the answer provided and the answer that the network could give at that time. Testing does not affect the network's weights and biases.

Case—a teaching example, which consists of a question and an answer.

Threshold—bias of the neuron, the value added to weighed inputs before using the activation function.

Typefaces

Important notices

Important notices appear in the document like this:

This is an Important Notice

Please pay close attention to the document sections marked as important notices.

Enterprise feature notices

If the described feature only belongs to the enterprise version of Weighscore, the description will start like this:

This is a feature of the Enterprise Version

File, program, class and other names, protocol commands

File and other names appear in the text in the following style:
    `filename.jar`

Console screenshots

Console screenshots look like this:

```
>java -jar weighscore-clt.jar -h
The Weighscore Neural Network training, probing and requesting command line
tool capable to connect to the JDBC databases.
Weighscore Neural Network Toolkit. (c) Vsetech 2005-2006

Usage: nnt [-n <name>] [-a] [-u] [-p] [-r N] [-t <double>] [-c <filename>]
           [-S <name>] [-D<configname=value>] [-v] [-h]
           [q1 [q2 [q3] ...] [a1 [a2] ...]]
.......
```

Text file and code listings

Text file listings and code in programming examples look like the following:

```
    public double execute(double x) {
        return beta*x;
    }
```

Product Features

Weighscore Neural Network Toolkit has the following features and benefits:

- Free workstation command line tool for the initial neural network training for the server and for educational or scientific purposes;
- platform independent, written in java—it may be run on Solaris, Windows, Linux etc;
- uses JDBC data source as a training case set—it may get training data from any source if there is a JDBC driver for it;
- converts (*translates*) the simple string reference entries to neurons' numeric values—it requires minimum preliminary data transformation;
- stores the neural network in a simple XML file—editable with any text editor.

The main advantage to Weighscore Neural Server is that is has the ability to be trained online, simultaneously with its querying. When new data that characterizes the response that was given earlier arrives, the network may be slightly updated using the new data without stopping.

Moreover, there are the following advantages:

- thread safe simultaneous processing—suitable for distributed calling;
- easily called with a simple protocol—see examples for java and winsock in the reference manual;
- callable as an Axis webservice— easy to call from almost any client platform.

Prerequisites

The Weighscore neural server and command line tool are written in Java, so they may be run on any computer system that already has installed:

- JRE 1.4 or higher
- 128 mb RAM minimum; larger neural networks require more memory to be queried and trained.

The Server and clients need a TCP/IP network interface to communicate with each other.

nnt: Command Line Tool

Installation, running and uninstallation

To install the program, extract the file `weighscore-clt.jar` from the downloaded archive to any directory convenient for you.

Set the path to the Java runtime executable in the PATH environment variable.

You may run the tool directly from the jar file, like this:

```
>java –jar weighscore-clt.jar
```

Or, you may explicitly call the `nnt` executable class, using the following standard syntax:

```
>java -classpath weighscore-pntk.jar com.weighscore.neuro.nnt
```

To uninstall, just delete the `weighscore-clt.jar`.

`nnt.config`: Configuration reference

The nnt tool has a configuration file that defines the variables needed to perform network training and requesting (teaching and asking).

Config file has the standard Java properties format, so you may set variables like “name=value”, and use “#” sign for disabling settings or for comments.

All “db” related variables may be omitted if you pass data from the command line.

The config file defines the following variables:

network

This is an optional variable.

The name of the neural network to teach, test or ask.

In the free version, nnt tool will work only with the network named `NeuralNetwork.xml`***, and this variable will be ignored.***

dbdriver

Name of the JDBC driver class. To access some ODBC data sources, try the JDBC-ODBC driver, like this:

```
dbdriver=sun.jdbc.odbc.JdbcOdbcDriver
```

To access the database server, it’s advised to use the driver that is designed specifically for that particular database.

dburl

JDBC URL to the database.

For example, to access the MS Access database via JDBC-ODBC driver, use

```
dburl=jdbc:odbc:Driver=Microsoft Access Driver (*.mdb);DBQ=C:/testing.mdb
```

See your database’s JDBC driver manual for the syntax of other database’s URLs.

dbuser, dbpassword

These are optional variables.

If the DB server requires it, you may pass the username and password through these variables

dbtable

The name of the table in the database, which is the source for the neural network teaching, and possibly the destination for the results of asking.

This setting is used to construct a SQL “update” statement; if the `dbquery` is omitted, this is also the table used in the “select” statement.

dbquery

This is an optional variable.

The SQL query to use as the data source for teaching. You may use this variable to limit the training set, or to specify the appropriate order of teaching cases to be presented to the network, or anything else.

If you name the fields in the query explicitly (not with an asterisk sign), be sure to include all the fields defined in the neural network’s translator. Include the ID fields, if you wish to update the table and set the `dbidfields` variable.

If you omit this setting, the query will be constructed from the `dbtable` variable as “select * from {dbtable}”

The system doesn’t check if the table names in the `dbquery` and `dbtable` are the same.

dbidfields

This is an optional variable.

If you wish to record the network’s answers to the database table, specify the names of the fields, whose values uniquely identify every record, separated by commas, like this:

```
dbidfields=x1,x2
```

If you omit this setting, the system will construct the SQL update statement using all the input field names and values of the neural network. This will overwrite more than one record in the data source, if those records have the same values in the network input fields.

Operation instructions

You may keep the configuration file in the file with a name other than `nnt.config`. Use the –c switch followed by the configuration file name.

To increase the amount of output information, use the –v switch.

In the Enterprise verstion of nnt, you may override the default network specified in `nnt.config`, using the –n switch followed by the network name.

Asking command line

To ask the network from the command line, use the –a switch.

Add the input values after the last option switch or value, separated by spaces.

The nnt program will output the result in the console.

```
>java –jar weighscore-clt.jar -a 0 1
0.7993303880674504
```

Teaching and testing command line

To teach the network, pass the question together with the correct answer with space separated command line arguments to the nnt tool, without any switches. The error (the difference between the correct answer and the answer that the network could give at that time) will be printed in the console. The network's weights and biases will be updated to possibly minimize the error.

```
>java –jar weighscore-clt.jar 0 1 1
E: 0: -0.20066961193254962
```

To test the network, use the –p switch. Testing the network is computing the error without adjusting the weights and biases values.

Bulk JDBC data source teaching and testing

If you don't pass command line arguments, the nnt tool will take data to teach or to test from the JDBC data source specified in the configuration file.

The teaching (or testing) procedure will be repeated for every record in the data source, using the record as the network's input and output values.

To make nnt not teach but to just test the network, use the –p switch ("p" means "probe").

If it is teaching, the whole record set will be run multiple times. You may limit the number of times the record set will run by setting the –r switch followed by the maximum quantity of runs.

While it is teaching, if everything proceeds as planned, the error must decrease. You may stop nnt when the maximum error value of the network's response becomes less then the value you specify using the –t switch ("t" stands for "target"):

```
>java -jar weighscore-clt.jar -t 0.01
Run: 1034 Case: 4 Net: null  E: 0: -0.0063773810570814105 AE: 0.011108613337232487O8
Target 0.01 met.

1034 cases run.
```

"E" are the current errors of all the output neurons; "AE" are the average errors for the last N cases, where N is the size of the teaching case set.

Bulk JDBC data source update with answers

The nnt tool is capable of updating the data source with the network answers. Use the –u switch to do this. **Important: Be sure to save the copy of training examples set. The update action will overwrite the old information in the specified data source; you won’t be able to use it for training purposes anymore.**

Asking, teaching and testing the server-side neural network

This is a feature of the Enterprise version of nnt.

To ask, to test or to teach the neural network that resides on the server, you can specify the name of the network as follows:

```
neuro://hostname[:port]/networkname
```

The `hostname` is the host where the Weighscore Neural Server is running; the `port` is the port where it is listening. The `networkname` is the name of the neural network on that server.

You can specify the name of the network in this way as in `nnt.config`, or in the command line -n switch argument.

Command line options reference

-n <name> (network)

The name of the neural network to use instead of that which is specified in the configuration file.

-a (ask)

Don't train the network. Run the case set once only.

-u (update)

If the JDBC source is used, don't train the network, update the datasource with the network's responses; run the case set once only. See the Bulk JDBC data source update with answers section.

-p (probe)

Compute errors (probe the network) without training; run the case set once only.

-r <N> (runs)

The maximum quantity of runs of the training set taken from the JDBC datasource; it is ignored when processing command line arguments, or when probing the network, or when it is updating a JDBC set. See the Bulk JDBC data source teaching and testing section.

-t <double> (target)

The minimum target error. See the Bulk JDBC data source teaching and testing section.

-c <filename> (config)

The nnt configuration file instead of the default `nnt.config`.

-D<configname=value> (detail)

Overrides the configuration value in the config file (can set values without spaces only).

-v (verbose)

Outputs the data set with answers and errors, each case in a separate line.

-h (help)

Outputs information about switches.

`neural.config`: Basic configuration of the Enterprise version

This section relates only to the Enterprise version of the Command Line Tool and to the Neural Server.

The free version works only with one neural network and its translator, definitions of which are stored in the files `NeuralNetwork.xml` and `Translator.xml`.

In the Enterprise version, there is an option to determine where all the neural networks are stored.

The basic configuration of the underlying neural library is kept in the file named `neural.config`. In the current version it holds just two variables which define the origin of the neural networks and translators definitions.

neuralNetwork.origin

This is an optional variable, which may take only one value in the current version, `com.weighscore.neuro.XmlFileOrigin`; this is the default. This is the name of the class that helps to serialize neural networks to xml streams.

neuralNetwork.origin.xml.path

The path to the directory where the neural networks and translators xml definition files are stored.

The path must end with the slash character. Use forward slashes ('/') even on the Windows platform.

Example:

```
neuralNetwork.origin=com.weighscore.neuro.XmlFileOrigin
neuralNetwork.origin.xml.path=F:/users/steve/work/neuro/neuralnetworks/
```

This section describes the Neural Server, which is not free.

Installation, launching, stopping and uninstall

To install the server, extract the file `weighscore-server.jar` from the downloaded archive to any directory convenient for you.

To uninstall, just delete the `weighscore-server.jar`.

Set the path to the Java runtime executable in the PATH environment variable.

You may launch the server directly from the command line like this:

```
>java –jar weighscore-server.jar
```

In the Windows platform, in order to avoid the command prompt window interfering with normal viewing on the screen, you may use `javaw` instead of `java`, like this:

```
>javaw –jar weighscore-server.jar
```

In Unix-like systems, use “&” sign to release the shell:

```
>java –jar weighscore-server.jar &
```

Important: Don’t launch two instances of the server that have the same neural network origin path. This may cause the neural network to have file corruption or data loss.

To stop the server, launch the same executable java file but with the –S switch:

```
>java –jar weighscore-server.jar -S
```

If you launched the server on a different port than the default, specify the port number when stopping as well.

Important: Avoid killing the “java” or “javaw” process using the Windows task manager or the Unix kill command. This may cause the neural network file corruption (if it is killed while autosaving the network) or data loss (the network data being taught wouldn’t be saved to a file).

Command line options reference

-p PORT

Specify the PORT to listen to; the default is 1133.

-g

This allows retrieving the neural network XML definitions. If the neural network is used to make financial decisions and/or is nonpublic, don’t run the server with this option.

-l LOGFILE

Set the file to get output messages instead of the console.

-S

Shutdown the server. If the server was launched with the –p switch, add the –p switch and specify that port as well.

-h

Output the command line options information.

Call from client

The Neural Server is intended to be used together with other software that will query the neural networks and help to make the right business decisions. Additionally, there is a possibility to query the server over the network programmatically, asking questions and getting answers.

Test call using `telnet`

The simplest way to test the connection between the client and server machines is to connect to the server from the client machine using a `telnet` program. The server has simple text protocol so the user can query the server through telnet with just a keyboard.

Run `telnet` giving it the server name and port as parameters:

```
>telnet serverhost 1133
```

Then print the server command and its parameters, separated by tabs, as described in Protocol reference section below. For example,

```
ask     Xor.xml       1       1
```

and it will return an answer:

```
OK      0.87775815287206
```

Calling server from client programs

java

In Java programs, you may call the server as easily as follows:

```
import java.net.*;
import java.io.*;

.....

    private static String[] commandExample(String host,
                                           int port,
                                           String networkName,
                                           String command,
                                           String arguments){
```

```
        Socket s = null;
        String[] ret;
        try {
            s = new Socket(host, port);
            s.setSoTimeout(10000);

            PrintWriter out = new PrintWriter(s.getOutputStream(), true);
            BufferedReader in =
                new BufferedReader(new InputStreamReader(s.getInputStream()));

            out.println(command + '\t' + networkName + '\t' + arguments);
            out.flush();

            try {
                String reply = in.readLine();
                String[] replyArr = reply.split("\t" ,2);
                if(replyArr[0].equals("OK")){
                    try {
                        ret = replyArr[1].split("\t");
                    } catch (ArrayIndexOutOfBoundsException e) {
                        ret = new String[0];
                    }
                }
                else{
                    throw new RuntimeException("Server error: " + replyArr[1]);
                }
            } catch (SocketTimeoutException ex2) {
                throw new RuntimeException("Server does not respond on port " +
                                           port, ex2);
            }
            out.close();
            in.close();
            s.close();
        } catch (IOException ex2) {
            throw new RuntimeException(ex2);
        }
        return ret;
    }
```

This example method receives the commands described in the Protocol reference section of this document, accompanied by the neural network name and the arguments as one string containing tab separated values. The Server host and port need to be specified as well.

The String’s `split` method is a feature of JDK 1.4 or higher. You may use `StringTokenizer` to extract the “OK”/”ER” flag and other tab separated parts of response.

Windows Sockets (including a Visual Basic Winsock example)

Using Microsoft’s Winsock, or some third party library, you may connect to the Neural Server from your Windows programs with C++, Visual Basic, Delphi etc.

You may connect to the Server from MS Office programs like Excel or Access, if you add a Visual Basic for Applications module that will communicate with the Neural Server.

To connect to the Server through the socket, you may use Winsock library (`ws2_32.dll`, or formerly `wsock32.dll`). There are some third party libraries that provide a less difficult way to call the server. Below is a brief example of how to interact with the server using Winsock. See the Winsock reference manual (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/winsock_reference.asp) for details.

To gain access to the dll’s functions, you should declare them like this:

```
Public Declare Function WSAStartup Lib "ws2_32.dll" (ByVal _
    wVR As Long, lpWSAD As WSA_Data) As Long
```

```
Public Const WINSOCK_VERSION = 1

Public Const WSADESCRIPTION_LEN = 257
Public Const WSASYS_STATUS_LEN = 129

Public Type WSA_Data
    wVersion       As Integer
    wHighVersion   As Integer
    szDescription  As String * WSADESCRIPTION_LEN
    szSystemStatus As String * WSASYS_STATUS_LEN
    iMaxSockets    As Integer
    iMaxUdpDg      As Integer
    lpVendorInfo   As Long
End Type

Public Declare Function socket Lib "ws2_32.dll" (ByVal _
    af As Long, ByVal s_type As Long, ByVal protocol _
    As Long) As Long

Public Enum ADDRESS_FAMILIES
  AF_INET = 2
  AF_NS = 6
  AF_IPX = AF_NS
  PF_INET = 2
End Enum

Public Enum SOCKET_TYPES
  SOCK_STREAM = 1
  SOCK_DGRAM = 2
End Enum

Public Enum PROTOCOLS
   IPPROTO_TCP = 6
   IPPROTO_IP = 0
End Enum

Type sockaddr
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type

Public Declare Function connect Lib "ws2_32.dll" _
    (ByVal s As Long, addr As sockaddr, addrlen As Long) As Long

Public Const QUEUE_SIZE = 5

Public Declare Function recv Lib "ws2_32.dll" _
    (ByVal s As Long, buf As Any, ByVal buflen As Long, ByVal flags _
    As Long) As Long

Public Declare Function send Lib "ws2_32.dll" _
    (ByVal s As Long, buf As Any, ByVal buflen As Long, ByVal _
    flags As Long) As Long

Public Declare Function htons _
    Lib "ws2_32.dll" (ByVal hostshort As Integer) As Integer

Public Declare Function inet_addr _
    Lib "ws2_32.dll" (ByVal cp As String) As Long

Public Declare Function closesocket _
    Lib "ws2_32.dll" (ByVal s As Long) As Long

Sub test()
```

```
    commandExample "127.0.0.1", 1133, "Xor.xml", "ask", "1" & vbTab & "1"
End Sub
```

This is an example of how the server call function may be written.

```
Sub commandExample(host As String, _
        port As Integer, _
        networkName As String, _
        command As String, _
        arguments As String)

    Const MAX_BUFFER_LENGTH As Long = 8192

    Dim wsaData As WSA_Data
    Dim s As Long

    Dim arrBuffer(1 To MAX_BUFFER_LENGTH)   As Byte
    Dim arrSendBuffer()                     As Byte
    Dim lngBytesReceived                    As Long
    Dim strTempBuffer                       As String
    Dim strBuffer                           As String

    If (WSAStartup(WINSOCK_VERSION, wsaData)) Then
        MsgBox "Can't init"
        Exit Sub
    Else
        s = socket(PF_INET, SOCK_STREAM, 0)

        If (s = 0) Then
            MsgBox "Error create socket"
            Exit Sub
        End If

        Dim socketaddr As sockaddr

        socketaddr.sin_family = AF_INET
        socketaddr.sin_addr = inet_addr(host)
        socketaddr.sin_port = htons(port)


        If (connect(s, socketaddr, Len(socketaddr)) <> 0) Then
            MsgBox "Bad connect"
            Exit Sub
        End If

        strBuffer = command & vbTab & networkName & vbTab & arguments & vbLf
        arrSendBuffer = StrConv(strBuffer, vbFromUnicode)

        Call send(s, arrSendBuffer(0), Len(strBuffer), 0)
        strBuffer = ""
        Do
            lngBytesReceived = recv(s, arrBuffer(1), MAX_BUFFER_LENGTH, 0)
            strTempBuffer = StrConv(arrBuffer, vbUnicode)
            strBuffer = strBuffer & Left$(strTempBuffer, lngBytesReceived)
        Loop While lngBytesReceived > 0

        MsgBox strBuffer

        closesocket (s)
    End If
End Sub
```

Protocol reference

The commands described below are sent to the server together with arguments, separated by tabs, with the newline character at the end. The answer is returned as a line with a set of values separated by tabs as well; the first value in most cases is either “OK” if success or “ER” if an error occurred. The “OK” string is not returned only when xml stream is a result (when requesting translator or neural network definitions).

In most cases the first argument is the neural network or translator name. The neural network name can be skipped if the session is in “persistent mode”. The persistent mode session has the default neural network.

persistent networkname

Switch the session to the persistent mode with the specified neural network as the default neural network of the session. The server will not close the connection after this and next commands until the “quit” command comes.

quit

Stop the persistent session, close the connection.

ask [networkname] question

Ask the question to the network. The questions are tab separated values. The results are tab separated answers.

teach [networkname] question answer

Teach the network. The question and answer are tab separated values. The results are tab separated errors.

test [networkname] question answer

Test the network. The question and answer are tab separated values. The results are tab separated errors.

getAnswerFieldName [networkname | translatorname] index

Gets the field name of the network translator’s output field number index. If the network doesn’t have a translator, this returns an empty string.

getAnswerSize [networkname | translatorname]

Returns the quantity of the output fields in the network translator. If the network doesn’t have a translator, this returns the number of output neurons.

getAskFieldName [networkname | translatorname] index

Gets the field name of the network translator’s input field number index. If the network doesn’t have a translator, this returns an empty string.

getAskSize [networkname | translatorname]

Returns the quantity of the input fields in the network translator. If the network doesn’t have a translator, this returns the number of input neurons.

getFieldNames [networkname | translatorname]

Returns tab separated field names of the network’s translator. If the network doesn’t have a translator, this returns tab separated empty strings.

outputDefinition [networkname | translatorname]

Gets the xml definition stream of the specified neural network or translator, whichever name is specified. If it is in persistent mode, the neural network name may be omitted.

outputTranslatorDefinition [networkname | translatorname]

Gets the xml definition stream of the translator from the specified neural network. If the translator name is provided, it returns the translator xml definition. If it is in persistent mode, the neural networks name may be omitted.

Neural network developer reference

The Neural Network structure is defined by a XML file that has the special structure described below.

The name of the neural network is the same name of the file defining it, including the ".xml" ending.

There is one more entity in the neural system—the translator (see Translator definition below). This object holds the information about the neural network's data source—like field names—and the correspondence of the data source fields with the neural network's input neurons. While the neural network alone can get only a set of floating point numeric values as its input, and then returns a set of the same type values as the output, using the translator lets it use non-numeric values as input and output. These values can be the dictionary entries or IDs; the translator will convert it to numeric values sent to one or more neurons.

Neural network structure definition

General information

File format

The neural network definition is a text XML file which has a predefined structure described in `NeuralNetwork.dtd` file.

Below is the explanation of all neural network tags and their attributes.

Header

The file should start from this set of tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE neuralNetwork SYSTEM "NeuralNetwork.dtd">
```

Neural Network

The main document tag for the neural network file is the `<neuralNetwork>`.

For compatibility reasons, it should always have the attribute `xmlns:xlink` set to the value `"http://www.w3.org/1999/xlink"`, as shown below:

```
<neuralNetwork xmlns:xlink="http://www.w3.org/1999/xlink" translator="Translator.xml">
...
```

Translator specifying

The translator is specified in the `translator` attribute. If the translator is not specified, the network will not be able to accept and return non-numeric values.

Teacher specifying

The `<teacher>` tag is a child tag for `<neuralNetwork>` tag.

```
<teacher class="com.weighscore.neuro.SimpleTeacher"/>
```

See Teacher section of this reference manual for more information.

You may skip this definition; and if so, then the system will use the default `SimpleTeacher`.

Neuron

The neural network’s main object is a neuron. Neurons are described by `<neuron>` tags, which are children to `<neuralNetwork>` tag.

```
  <neuron input="true" threshold="0.26003716863883036">
    ....
  </neuron>
```

Threshold

Bias (that is called *threshold* in the system) is set by an attribute `threshold` of the `<neuron>` tag. The threshold is a number that is added to the weighed neuron’s input before the application of an activation function.

You may skip this definition; and if so, the system will initialize the threshold with a random value.

Activation

The activation function is defined by the `<activation>` tag which is a child tag for `<neuron>`.

See the Activation section of this reference manual for more information.

You may skip this definition; and if so, the system will use the default activation `Sigmoid`.

```
<neuron input="true" threshold="0.3196715423362544">
  <activation class="com.weighscore.neuro.Sigmoid">
    <parameter name="beta">1.0</parameter>
  </activation>
...
```

Set the network’s input and output neurons

You define which neurons will be input or output for the network by setting the `<neuron>` tag’s attributes `input` or `output` to “true” value.

Statistic

You may define a class of statistic object through which all the values will pass, by the `<statistic>` tag which is a child tag for `<neuron>`.

```
<statistic class="com.weighscore.neuro.FullStatistic"/>
```

See the Statistic section of this reference manual to get more information about the statistic definition.

You may skip this definition; and if so, the system will use the default statistic.

Axon and synapses

The neuron output links are defined by its `<axon>` child tag. The `<axon>` tag may have multiple `<synapse>` children tags.

See the next section for more information about defining synapses.

Synapse

Weight

You may set the weight of a synapse by adding the attribute `weight` to the `<synapse>` tag, like this:

```
<synapse weight="0.2963048183041408">
```

You may skip this definition; and if so, the system will initialize the weight with a random value.

Synapse's output neuron

The `<synapse>` tag must have two attributes that point to the synapse's output neuron. The first is the `xlink:type` attribute which must have the value "locator". The second is a hyper link `xlink:href` attribute in a form of a XPointer that points to the output neuron.

The XPointer link should always look like "#xpointer(/neuralNetwork/neuron[*number*])", where the number is a sequential number of the neuron in this document starting from 1.

The whole example looks like this:

```
  <neuron>
    <axon>
      <synapse xlink:type="locator" xlink:href="#xpointer(/neuralNetwork/neuron[3])"/>
      <synapse xlink:type="locator" xlink:href="#xpointer(/neuralNetwork/neuron[4])"/>
    </axon>
  </neuron>
```

This means that this neuron is connected to the third and fourth neuron as defined in this xml document.

The system is not able to resolve XPointer links that have a form that differs from the described above.

This way of defining synapses was chosen to avoid senseless IDs in the document; we chose addressing neurons only by their position in the document, because of the following reasons. 1. While editing the XML definition file, the user would have to invent new IDs for new neurons; while now he or she may just add neurons to the end to keep numbers and links. 2. The senseless (surrogate) IDs are not stored in memory, so the system would have to re-generate them every time when serializing the network to XML stream. 3. Using IDs require the obligatory accompanying xml file with the dtd or schema file, while relative pointers themselves have a meaning.

But, this method may change in the next version, giving way to being more convenient for a human user to edit the network definition.

Statistic

You may define the class of a statistic object through which all the values will pass, by the `<statistic>` tag which is a child tag for `<synapse>`.

```
<statistic class="com.weighscore.neuro.FullStatistic"/>
```

See the Statistic section of this reference manual to get more information about the statistic definition.

You may skip this definition; if so, the system will use the default statistic.

Neural network special objects

Activation

Every neuron should have an activation function (“activation”). This function takes the sum of weighed inputs and the bias of the neuron as an argument. There is a couple of different activation functions, so you can choose the class that defines the activation function appropriate for your task.

Set the `class` attribute of the `<activation>` tag to the appropriate class name. Activations may have parameters that control their behavior. Activation and parameters are set like this:

```
<activation class="com.weighscore.neuro.Sigmoid">
  <parameter name="beta">1.0</parameter>
</activation>
```

Sigmoid

This is the most popular activation for neural networks.

To use it, set the `class` attribute of the `<activation>` tag to the value "com.weighscore.neuro.Sigmoid".

It takes one parameter, named “beta”.

The function is a hyperbolic tangent and is defined as follows:

```
y = (exp(beta * x) - exp(beta * x * -1)) / (exp(beta * x) + exp(beta * x * -1))
```

The function converges to 1 for big positive arguments and to –1 for big negative ones; the curve rapidly crosses the zero point when the argument changes from negative to positive. The beta parameter controls the angle by which the curve crosses the zero coordinate.

See `http://en.wikipedia.org/wiki/Hyperbolic_function` for more information.

Linear

To use the Linear activation, set the `class` attribute of the `<activation>` tag to the value "com.weighscore.neuro.Linear".

It takes one parameter, named “beta”.

The function is defined as follows:

```
y = beta * x
```

The function’s curve is a straight line. The beta parameter controls its inclination to the x-axis.

Teacher

The network’s teacher is a special object that is responsible for the teaching process. It is instantiated and kept in the neural network, and it is called when teaching is needed. Different teachers may have different teaching procedures.

The teachers depend on neurons' and synapses' statistics because they use values stored in them (for example, the last weight change to compute the momentum).

SimpleTeacher

To use the `SimpleTeacher` activation, set the `class` attribute of the `<teacher>` tag to the value "com.weighscore.neuro.SimpleTeacher".

This teacher is a simple back propagation teacher. Though it is quite simple, today it is the best choice to teach the neural networks online, right at the working time. The network can modify its behavior in response to new events happening and new teaching cases that are presented to it. You may send teaching commands to the network if new data, that characterizes the previously taken decision, comes. Teaching online, you make the network adjust its weights and next time a similar question is asked, a better answer will be given.

The `SimpleTeacher` requires all synapses and neurons to have `MomentumStatistic` or it's descendants.

The `SimpleTeacher` has two parameters, `momentumCoefficient` and `learnRate`. The first is a multiplier to the last weight changes; the latter is the multiplier to the computed teaching values. Please refer to the special literature on the error back propagation method of neural network training.

```
<teacher class="com.weighscore.neuro.SimpleTeacher">
  <parameter name="momentumCoefficient">0.2</parameter>
  <parameter name="learnRate">0.4</parameter>
</teacher>
```

EmpiricTeacher

To use the `EmpiricTeacher` activation, set the `class` attribute of the `<teacher>` tag to the value "com.weighscore.neuro.EmpiricTeacher".

This teacher is an experimental implementation of a modified back propagation teacher.

This teacher does the following:

- automatically decreases the learn rate while teaching;
- if the error doesn't decrease, jogs the selected weights and biases of the network, increases the learn rate and continues training.

The `EmpiricTeacher` has the same parameters as the `SimpleTeacher`; and unlike the latter, it changes the `learnRate` parameter while teaching.

To use the `EmpiricTeacher`, all the synapses and neurons should have statistics of the `LastErrorStatistic` class.

Statistic

While asking, testing or teaching, the neuron passes through itself and outputs some values. There are special classes of objects that can record those values and compute some statistical figures that can be used for teaching as well as for network's performance analyzing.

Below are the descriptions of the statistic classes available in Weighscore. They are ordered by their inheritance: last classes are the descendants of the first and have all the parameters that the previous classes have.

Note that using the complicated statistics in many neurons and synapses slows down the performance of the system.

When starting the network design, define the class of the statistic without setting its parameters, like this:

```
    <axon>
      <synapse xlink:type="locator" xlink:href="#xpointer(/neuralNetwork/neuron[7])">
        <statistic class="com.weighscore.neuro.AverageGradientStatistic"/>
      </synapse>
....
```

The statistical values will be added as parameters automatically.

MomentumStatistic

To use the `MomentumStatistic` statistic, set the `class` attribute of the `<statistic>` tag to the value "com.weighscore.neuro.MomentumStatistic".

This statistic has the following parameters:

- `lastCorrection`: holds the last change of the weight or threshold;
- `teachCnt`: the teaching signals counter.

AverageGradientStatistic

To use the `AverageGradientStatistic` statistic, set the `class` attribute of `<statistic>` tag to the value "com.weighscore.neuro.AverageGradientStatistic".

This statistic is intended to be used in the future for a conjugate gradient teacher.

This statistic has all parameters of `MomentumStatistic` plus:

- `epochCasesCnt`: counts the number of epochs (the whole case sets);
- `epochCurAgvGradient`: the average gradient member for the current epoch;
- `epochLastAgvGradient`: the average gradient member for the previous epoch.

LastErrorStatistic

To use the `LastErrorStatistic` statistic, set the `class` attribute of the `<statistic>` tag to the value "com.weighscore.neuro.LastErrorStatistic".

When using `EmpiricTeacher`, all the synapses and neurons must have statistics of this class.

This statistic has all the parameters of `AverageGradientStatistic` plus:

- `lastErrCnt`: the size of an array of the last errors of this synapse or neuron;
- `lastAvgErr`: the average value of the last `lastErrCnt` errors of this neuron or synapse;
- `lastAvgErrAbs`: the average value of the last `lastErrCnt` absolute values of errors of this neuron or synapse;
- `lastAvgErrDev`: the average difference between the last `lastErrCnt` errors and the last lastErrCnt average errors;
- `lastAvgErrDevAbs`: the average absolute difference between the last `lastErrCnt` errors and the last lastErrCnt average errors.

FullStatistic

To use the `FullStatistic` statistic, set the `class` attribute of the `<statistic>` tag to the value "com.weighscore.neuro.FullStatistic".

This statistic has all parameters of `LastErrorStatistic` plus:

- `askCnt`: total count of asks;
- `askAvg`: average value input of this neuron or synapse;
- `askAvgPosDev`: average positive difference between input and `askAvg` of this neuron or synapse;
- `askAvgNegDev`: average negative difference between input and `askAvg` of this neuron or synapse;
- `askPosDevCnt`: count of the cases when the difference between the input and `askAvg` of this neuron or synapse was positive;
- `errCnt`: testing the actions count; note that teaching includes testing;
- `errAvg`: average error of this neuron or synapse;
- `errAvgPosDev`: average positive difference between the error and `errAvg` of this neuron or synapse;
- `errAvgNegDev`: average negative difference between the error and `errAvg` of this neuron or synapse;
- `errPosDevCnt`: count of cases when the difference between the error and `errAvg` of this neuron or synapse was positive.

Translator definition

General information

File format

The translator definition is a text XML file which has the predefined structure described in Translator.dtd file.

Below is the explanation of all neural network tags and their attributes.

Header and root tag

The file should have the following header and root tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE translator SYSTEM "Trabslator.dtd">
<translator>
      .....
</translator>
```

Field

The only tag allowed between the `<translator>` tags is the `<field>` tag. Every `<field>` tag corresponds to one field of the data source. It has the `name` attribute that should match the datasource field name.

The `<field>` tag has the `type` attribute that says if the field is input (a part of the question) or output (a part of the answer). It takes the value “ask” or “answer”, respectively.

The fields may be of two kinds, passing or translating.

The passing field means that the value in the datasource field just is passed to (or from) the neuron, possibly divided by some value. The passing field has the `<pass>` tag as its child.

The translating field sets the correspondence between the string value of the datasource field and numeric value of one or more neurons. The translating field has the `<translate>` tag as its child.

Pass

The `<pass>` tag may have the `dividor` attribute holding a divisor by which the datasource numeric value is divided.

The corresponding neuron is set by the child `<activator>` tag.

The following example means that the value of input field named “age” is passed to the first input neuron, divided by 200.

```
  <field name="age" type="ask">
    <pass dividor="200">
      <activator index="1"/>
    </pass>
  </field>
```

Translate

The `<translate>` tag may have one or more child `<item>` tags corresponding to the dictionary entries of the datasource field. In other words, every possible value of the datasource field must have the corresponding `<item>` tag.

Item

The `<item>` tag has the `value` attribute that holds the value of the field’s dictionary entry.

The neuron corresponding to the field and the field’s value is set by the child `<activator>` tag.

In this example, if the “gender” field has value “m”, the second input neuron gets a 0.5; if it is “f” then the second input neuron gets the value of –0.5.

```
  <field name="gender" type="ask">
    <translate>
      <item value="m">
        <activator index="2">
          <range value="0.5" min="0"/>
        </activator>
      </item>
      <item value="f">
        <activator index="2">
          <range value="-0.5" max="0"/>
        </activator>
      </item>
    </translate>
  </field>
```

It is possible to link one datasource field to more than one neuron, selecting which neuron the signal is sent to depending on the field's value.

In the following example, if the “education” field’s value is “higher”, the fourth neuron gets the input value of 0.5, if “high”—then the fifth, if “low”—the sixth. Other neurons get 0 as the input values.

```
  <field name="education" type="ask">
    <translate>
      <item value="higher">
        <activator index="4">
          <range value="0.5" min="0.01"/>
        </activator>
      </item>
      <item value="high">
        <activator index="5">
          <range value="0.5" min="0.01"/>
        </activator>
      </item>
      <item value="low">
        <activator index="6">
          <range value="0.5" min="0.01"/>
        </activator>
      </item>
    </translate>
  </field>
```

Activator

The `<activator>` tag denotes the neuron to which the value will be passed while asking. The neuron index is set by the `index` attribute. Note that the indices numbers start from 1 for output neurons, so that the first output (answer) neuron will have index 1.

Activators of the translating fields should have the `<range>` child tag.

Range

The `<range>` tag sets the value which will be passed to the neuron denoted in the parent `<activator>` tag, in its `value` attribute.

The `<range>` tag may also have attributes `min` and `max`. These are used to help to translate back the neuron’s numeric signal to the field string entry.

Neural network designing and training hints

Network size

At this level of scientific research, developing a neural network structure is a non-trivial task and is a sort of an art. A developer needs to have an initial experience in building networks. Here are some hints; though they may not help, if the statistical data available is not excessive or doesn’t have any inner dependencies.

If the training error never becomes less than the target (for example, it stops somewhere around 0.5 or 0.3), decrease the learn rate and momentum coefficient.

If this doesn't help, add more neurons. Consider reading industry specific literature about the latest research to decide whether to add layers to the network or to add neurons to some layer.

When the target error is met, save the trained network to a backup file and attempt to train the network with a reduced number of neurons. This will help to avoid the common *overfitting* problem.

Training algorithm

The optimal training algorithm requirements state that it should have separate training datasource and testing datasource. The training datasource should have enough training cases to cover all the dependencies between the individual factors. The actual testing datasource may be times shorter than the training one

The network designer trains the network with the first training datasource. If the target error was not met, the training must be repeated with different starting weight values (if the initial xml network definition file doesn't have any set weights and thresholds, they will automatically be initialized with random values. Keep a separate backup copy of the initial network definition file).

When the error is less than the target error, we should run the testing case set again in the testing mode, and check the results to see if the error rate is comparable or slightly higher than the target. This helps to find out if the network was overfitted in the first training set. If overfitting happens, try to reduce the overall network size.