Joone Documentation

1. About

1.1. Joone - Java Object Oriented Neural Engine

1.1.1. News

- `May 08, 2006` - **JoonePad 0.5.0** is out (thanks to H.Fiorletta). Try now the new GUI Editor that will substitute the old one. read more...
- `Sep 08, 2005` - Joone 1.2.1 is out! This is mainly a bug fix release. read more...
- `Feb 03, 2005` - Joone 1.2.0 is out. In this version: new components, plugins and a lot of bug fixes. read more...
- `Jan 19, 2005` - The **Joone's Wiki** is open! You can find FAQs, Books Reviews, Discussion Forums, and more... Join the Joone Community!
- `Sep 19, 2004` - The new Distributed Training Environment 0.8.0 released! read more...

1.1.2. About Joone

Joone is a FREE Neural Network framework to create, train and test artificial neural networks. The aim is to create a powerful environment both for enthusiastic and professional users, based on the newest Java technologies.

Joone is composed by a central engine that is the fulcrum of all applications that are developed with Joone. Joone's neural networks can be built on a local machine, be trained on a distributed environment and run on whatever device.

Everyone can write new modules to implement new algorithms or new architectures starting from the simple components distributed with the core engine. The main idea is to create the basis to promote a zillion of AI applications that revolve around the core framework.

Download it, and enjoy!

1.1.3. Some features...

- Supervised learning:
  - Feed Forward neural networks (FFNN)

  - Recursive neural networks (Elman, Jordan, ...)
  - Time Delay neural networks (TDNN)
  - Standard Back-prop (Gradient Descent, on-line and batch)
  - Resilient Back-prop (RPROP)
- Unsupervised learning:
  - Kohonen SOMs (with WTA or Gaussian output maps)
  - Principal Component Analysis (PCA)
- Modular Neural Networks (i.e. possibility to mix all the above architectures)
- Powerful built-in data pre-processing mechanism
- Scripting capability (JavaScript) in order to add custom behaviour to the NNs
- more...

1.1.4. The Framework

Joone is a Java framework to build and run AI applications based on neural networks.

Joone consists of a modular architecture based on linkable components that can be extended to build new learning algorithms and neural networks architectures. All the components have specific features, like persistence, multithreading, serialization and parameterisation that guarantee scalability, reliability and expansibility, all mandatory features to make Joone suitable for commercial applications and to reach the final goal to represent the future standard of the AI world.

Joone applications are built out of components. Components are pluggable, reusable, and persistent code modules. Components are written by developers. AI experts and designers can build applications by gluing together components with a graphical editor, and controlling the logic with scripts.

All the modules and applications written with Joone will be based on the above reusable components. Joone can be used to build Custom Systems, adopted in an Embedded manner to enhance an existing application, or employed to build applications on Mobile Systems.

Joone has an own GUI Editor to visually create and test any neural network, and a Distributed Training Environment to train in parallel mode many neural networks to find the fittest one for a given problem.

If you want learn more on Joone, you can read the available documentation

1.1.5. Resources

- The Joone Complete Guide (pdf)
- The Core Engine
- The Java API cookbook

- The JOONE GUI Editor
- A sample GUI, that learns the XOR problem
- **A selection of the best Neural Networks books**

1.2. News

1.2.1. JoonePad 0.5.0 is out

The new neural network GUI editor - **JoonePad** - is out! Try it now by launching it directly from the web (You need JRE 1.5 and WebStart installed). Some of the new features:

- Added a wizard to easily build several kinds of neural networks
- Added a java code generator in order to reproduce in a custom java program the same network currently displayed in the editor
- Now it's possible to import a neural network exported by the old GUI editor
- The TODO-list now is an active list: click an item in order to highlight the corresponding component containing the error
- Improved the user interface with docking panels and enhanced property sheets, in order to make it simpler to use the editor

This is a pre-view beta release, so please be patient if you discover some bug...

I want to thank **Huascar Fiorletta** for the great support and contribution he gave to make all that possible

Posted: May 08, 2006

1.2.2. Joone 1.2.1 released

This is mainly bug fix release. Some of the new features:

- Added the support for the Groovy scripting language
- Added the LogarithmicPlugin to apply a logarithmic transformation (base e) to input data
- Added the save as XML in the GUI Editor

Several bugs were fixed:

- Fixed a problem that prevented the SangerSynapse from learning when in training mode
- Now the inspection panel doesn't show the biases for Layers for which this doesn't make sense
- Fixed the lacking of the first column when copied in the clipboard the inspected values

Read all the changes...

Posted: September 08, 2005

1.2.3. Joone 1.2.0 released

A new version of Joone is out! Among the new features:

- Added many new components (SineLayer, JDBCOutputSynapse, InputConnector,...) and plug-ins (ShufflerPlugin, DeltaNormPlugin, ...)
- Revisited many internal mechanisms in order to make joone more expandible (a new Weights Initializer, a new Learner mechanism,...)
- Fixed a lot of bugs
- Improved the documentation

Read all the changes...

Posted: February 03, 2005

1.2.4. The new Distributed Training Environment 0.8.0 released

The main features of this powerful framework for distributed processing of neural networks:

- The training process scale almost linearly adding more machines
- Possibility to add or remove machines dynamically during the training process
- The overall process controlled by XML parameters
- Latest SUN's Jini 2.0 compliant
- Easily expandible with new features/algorithms/control modules, etc.

Read more...

Posted: September 19, 2004

1.2.5. Joone 1.1.1 released

This is a minor bug fix release. Several bugs were fixed: resolved the 'No matching patterns' problem, added the momentum in the BatchLearner, etc.

Changes

Posted: September 13, 2004

1.2.6. Joone 1.1.0 released

A new version of Joone is out! Great new features in this release:

- Added the **Resilient Backprop** algorithm (RPROP), a really performant new training algorithm (thanks to Boris Jansen)
- Improved the **core engine's speed** of about **35-40%**, thanks to a deep review of the

threads synchronization mechanism.

On an Athlon 2200Mhz the XORMemory example took 4.4 sec to execute 10,000 epochs with the old 1.0.1 version; now it executes the same elaboration in **only 2.4 seconds!**

Now the Joone's performances are nearly comparable to a C/C++ framework!

- Added the setting of a **http proxy** to access to Internet also in presence of a firewall proxy
- ...plus several bug fixes and a lot of small enhancements

Changes

Posted: August 18, 2004

1.2.7. New site Look&Feel

Thanks to the Forrest project, we have changed the look&feel of our web site. Now all the content is written in XML, and the HTML pages are obtained by applying a XSL transformation. By separating the content from the layout, we can easily and quickly add and modify the content, without affecting the graphical layout. Moreover, now all the pages are easily reachable from the menu on the left, and for each page is also available the content in PDF format.

Posted: July 16, 2004

1.2.8. Joone 1.0.1 released

The final release 1.0.1 is out! This is a minor bug fix release. Several bugs in the NestedNeuralLayer were fixed and more examples were added to the Joone Complete Guide PDF paper

Changes

Posted: June 13, 2004

1.2.9. Joone 1.0.0 final released!

The final release 1.0.0 is out! In the past months many bugs have been fixed, and now the version 1.0 can be declared stable. Download the auto-installer program and enjoy!

Changes

Posted: April 13, 2004

1.2.10. Joone on .NET!

Thanks to Casey Marshall, Joone now runs also on the .NET platform.

Download

Posted: January 13, 2004

1.3. Features

1.3.1. Architecture

- The Joone's framework is built with a modular architecture: the 'core engine' is separated from the visual interface and permits easily to implement any new application based on it.
- Joone is portable, being written in 100% pure Java. It can run in any environment, from big multiprocessor machines to small palmtop devices.

1.3.2. Neural Network's usability and transportation

- The neural networks based on Joone are usable stand-alone (separated from the framework that has created or trained them).
- The Joone's based neural networks can be transported using common protocols (like http or ftp) to run on remote machines

1.3.3. Framework expandability

- The framework is expandable with more components to implement new learning algorithms or new architectures.
- With Joone it's possible to implement any kind of optimization; there are two main methods to find the best solution to a given problem (i.e. to find the best neural network): local optimization and global optimization techniques. The local optimization is obtained applying some 'internal' mechanism (the most famous is the momentum), the global optimization, instead, try to find the best solution applying some external technique to select the best performing NN among a predefined group of NNs (like genetic algorithms). Both are implemented with Joone, and many new optimization techniques can be experimented thanks to its expansibility.

1.3.4. Multithreading and scalability

- Joone's core engine is based on a multithreaded engine, capable to scale using all the computing resources available.
- Joone provides the professional users with a distributed environment to train many neural networks in parallel on several machines.

1.3.5. Licensing

- Joone is freely usable. Its license is the Lesser General Public License (LGPL).
- You're encouraged to try it and use it for whatever (both commercial and academic) application!

1.3.6. Resources

- The Joone Complete Guide (pdf)
- The Core Engine
- The Java API cookbook
- The JOONE GUI Editor
- A sample GUI, that learns the XOR problem
- **A selection of the best Neural Networks books**

1.4. Downloads

1.4.1. The GUI Editor

Do you like the idea of a community-developed Open Source Neural Network framework? If you think Joone will help you in any way, consider **DONATING** to the project, so we can continue to enhance the features, fix the bugs and pay the hosting service. Your donation will return to the Open Source Community in the form of a even more powerful and stable free neural network engine. Thank you in advance for your contribution

Table 1: Donate to Joone

To run the joone's visual editor, all you need to do is to download and launch one of the following installers. Choose it according your platform:

Linux	Windows	All platforms
Without JVM (9.0 MB)	Without JVM (8.9 MB)	Without JVM (8.0 MB)
Includes JVM (40.5 MB)	Includes JVM (24.2 MB)	N/A

Table 2: GUI Editor downloads - Last released version: 1.2.1 (Sep 08, 2005)

Note:

- Due to a problem reported for the MacOSX installer, the Mac users have to choose now the platform independent version
- The packages without JVM require a JRE 1.4.2 or above installed

1.4.1.1. Installation Instructions

Linux

After downloading open a shell and, cd to the directory where you downloaded the installer. At the prompt type: sh ./JooneEditorx.y.z.bin

If you do not have a Java virtual machine installed, be sure to download the package above which includes one. Otherwise you may need to download one from Sun's Java web site or contact your OS manufacturer.

Windows

After downloading, double-click JooneEditorx.y.z.exe. If you do not have a Java virtual machine installed, be sure to download the package above which includes one.

All Platforms

After downloading, expand JooneEditorx.y.z-All.zip into a directory

After that, cd into that directory and launch ./RunEditor.sh (Linux/Unix/MacOSX) or RunEditor.bat (Windows)

1.4.2. The Core Engine

You can download the compiled classes or the source code of the core engine. To compile and runnit you need to download the External Libraries, too.

1.4.3. The Distributed Training Environment (DTE)

Download the compiled classes or the source code of the DTE. Download the DTE Complete Guide to read how to install and use it.

1.4.4. Portings

A subset of the core engine can be also used on the .NET platform. Download here the excellent work made by Casey Marshall (contains a porting of the version 1.0.0).

1.4.5. Resources

- The Joone Complete Guide (pdf)
- The Core Engine
- The Java API cookbook
- The JOONE GUI Editor
- A sample GUI, that learns the XOR problem
- **A selection of the best Neural Networks books**

1.5. Support

1.5.1. How to obtain support

The support is mainly provided trough two public forums available on SourceForge:

- **Help**: Use this forum if you need help about Joone, its core engine or the GUI Editor. It's the best way to obtain support directly by the developers of Joone.
- **Open Discussion**: This forum is to post suggestions, proposals, or to ask for the opinion of other users about whatever aspect of Joone.

Please don't write directly via email to the author of the project (not even to its developers), because he has limited free time, and you will probably never get an answer. Moreover, by posting on the Forums, you make public your problem, so the suggested solution could be useful for someone else. Thanks.

1.5.2. How to use the Forums

The following are some general "common-sense" guidelines for using the forums: Most people are already following these guidelines. To those people, I thank you. For those of you who are not...please read carefully.

1. Please be patient. We dedicate to Joone our free time, hence we cannot use it only to respond to the questions. Anyway we monitor the forum each day, hence sooner or later we'll give a response to any post.
2. Please refrain from using words and phrases like URGENT, IMPORTANT, MY WIFE WILL LEAVE ME IF YOU DO NOT RESPOND, etc, in the subject line of your postings. Or even in the body of the message for that matter. You may think that things which are URGENT, or IMPORTANT to you, are just as urgent and important to everyone else reading this forums. You'd be surprised at just how wrong an assumption like that can be. As a matter of fact, I personally tend to read all such postings last and give them a low priority.
3. Please use meaningful subject lines. "Newbie Question", is not meaningful what-so-ever.
4. Please provide necessary information, where applicable. Statements like, "I recieve a NullPointerException when using Joone, what's up with that?", are not very helpful, add unnecessary traffic on the forum, and will probably never get an answer.
5. As a follow on to #3, when submitting examples, please try to submit the smallest possible example which demonstrates the problem. Short examples are easier to follow and will get quicker responses from developers or other users than long examples with lots of extraneous code/data.
6. Please read the documentation before posting. One of the biggest problems is that there is not a lot of documentation to Joone. At the same time, because it doesn't take long, everyone should read what is written before posting.

7. Please check the forum archives before posting. Many questions on this forums have already been answered before.
8. Please refrain from posting confidential information. Once a message is sent to the forum, it goes out to all subscribers, including archive engines. If you are unsure about what is confidential or not, check with your manager, supervisor, or the person in charge at your company.
9. Please be kind and respectful to your fellow Joone users. Remember, except for the author, you were all newbies at one point.

1.5.3. Resources

- The Joone Complete Guide (pdf)
- The Java API cookbook
- The JOONE GUI Editor
- A sample GUI, that learns the XOR problem
- *A selection of the best Neural Networks books*

1.6. History of Changes

RSS

1.6.1. Version 1.2.1 (09/08/2005)

- Added the FahlmanTeacherSynapse to implement the Fahlman criterion (by Boris Jansen)()
- Added the support for the Groovy scripting language (by Yan Cheng Cheok)()
- Added the LogarithmicPlugin to apply a logarithmic transformation (base e) to input data (by Yan Cheng Cheok)()
- Added the save as XML in the GUI Editor (needs xstream.jar in the classpath) (by Paolo Marrone)()
- Added the AbstractTeacherSynapse in order to permit to add the calculus of whatever new cost function (by Boris Jansen)()
- Fixed the ChartOutputSynapse: small bug in values Y axis labels (bug #1143858) (by Jan Boonen)()
- Fixed the lacking of the first column when copied the inspected values (bug #1158597) (by Paolo Marrone)()
- Fixed a problem about the data not reloaded after calling StreamInputSynapse.resetInput (#1164061) (by Paolo Marrone)()
- Fixed resetInput not propagated through the InputConnector (bug #1164064) (by Paolo Marrone)()
- Resolved an ArrayIndexOutOfBoundsException into the ToBinaryPlugin (by Yan Cheng Cheok)()

- Fixed a ArrayIndexOutOfBoundsException within the Monitor.fireNetStopped method (by Jonathan Love)()
- Fixed a problem that prevented the SangerSynapse from learning when in training mode (by Paolo Marrone)()

- GUI Editor: Added code in the GUI Control Panel to refresh the display on the last epoch (by Firestrand)()
- Engine: Fixed netStopped wont be invoked in recall mode (bug #1047774)()
- Engine: Fixed a NullPointerException when ispected a DirectSynapse (bug #1038525)()
- Engine: Fixed Joone cannot run within an applet (bug #1110154)()
- Engine: Resolved a problem about unbuffered StreamInputSynapses when a big input file is read()

1.6.3. Version 1.1.1 (09/13/2004)

- Engine: Fixed the 'No matching patterns' problem (bug #1024973)()
- Engine: Fixed the globalError set to 0.0 for a cloned neural network (bug #1013956)()
- Engine: Fixed the missing use of the momentum parameter in the BatchLearner (bug #1024960)()
- Engine: Now the KohonenSynapse registers itself correctly as NeuralNetListener when deserialized (bug #1025177)()
- Engine: Fixed a NullPointerException in the YahooFinanceInputSynapse when no data present (bug #1026110)()

1.6.4. Version 1.1.0 (08/18/2004)

- Engine: Added the Resilient Backprop (RPROP) algorithm (by Boris Jansen)()
- Engine: Added the method join to the NeuralNet object in order to implement a mechanism to wait for all the running threads to finish()
- Engine: Added the 'param' property to the Monitor object in order to permit to store whatever new parameter without the need to change the java code()
- GUI Editor: Added the learningMode property in the Control Panel to choose the learning algorithm()
- GUI Editor: Added the 'http_proxy...' tag to the xml property file in order to permit to access to Internet also when in presence of a firewall proxy()
- Engine: Revisited the threads synchronization to improve the training speed: **improved more than 35%!**()
- Engine: Revisited the learners handling mechanism in order to be able to add/choose dynamically the Learner components()
- Engine: Fixed a bug in batchmode (bug #991335) (by Boris Jansen)()
- Engine: Fixed a wrong calculus of the output neighbourhood radius of the GaussianLayer component when in recall mode()
- Engine: Fixed a bug in Layer.isRunning method (bug #1007838) (by Jerry R. Vos)()
- GUI Editor: Fixed the file chooser, that didn't appeare when the .cfg file was corrupted()

1.6.5. Version 1.0.1 (06/13/2004)

- GUI Editor: Resolved the rmse not displayed when the net runs for few epochs (> 50)()

- Engine: Fixed a wrong initialization of the parent Monitor within the NestedNeuralLayer class()
- Engine: Added the netStoppedError notification to the internal NN of the NestedNeuralLayer class()
- Engine: The NestedNeuralLayer's randomize/addNoise method calls now have effect only when in learning mode()
- Engine: Resolved a wrong calculus of the validation error after consecutive runs (bug #957664)()
- Engine: Fixed the wrong validation error calculus when preLearning > 0 (bug #949353)()
- Engine: Eliminated an useless warning raised by the TeacherSynapse when stopped()

1.6.6. Version 1.0.0 final (04/13/2004)

- GUI Editor: Added the 'Page Setup' menu item to setup the printer's options()
- GUI Editor: Added the 'Paste' button in the inspection frame()
- GUI Editor: Added the maxBufSize property to the input synapses based on the StreamInputTokenizer()
- GUI Editor: Now the main window remembers its last position and size()
- GUI Editor: Added the initial splash screen()
- GUI Editor: Inverted the In/Out dataMin/Max properties' description on the NormalizerPlugin()
- Engine: Added the setComponent method to the Inspection interface and its inherited classes()
- Engine: Added the maxBufSize property to the input synapses based on the StreamInputTokenizer()
- Engine: Decoupled the engine from the logging libraries to be independent from the Log4j library()
- Engine: The StreamInputSynapse checks now if its input patterns number is lesser than the Monitor's input patterns setting()
- Engine: The FileInputSynapse opens the input stream only when requested()
- Engine: Resolved the IOException 'too many open files' on FileOutputSynapse object()
- Engine: Changed the tokenizer handling into all the classes inheriting StreamInputSynapse()
- Engine: The NestedNeuralLayer now handles correctly the attached I/O components()
- Engine: The XLSInputSynapse now handles correctly sheets containing empty columns/rows()
- Engine: The XLSInputSynapse now handles correctly the sheet name property()
- Engine: The Monitor object now handles correctly the nested Monitor, if present()

1.7. Todo List

1.7.1. high

- **[DTE]** Complete the rewriting of the DTE to be compliant to the Jini 2.0 specifications. # PM
- **[engine]** Add the XML serialization of a neural network # PM
- **[GUI Editor]** Rewriting the new GUI Editor by using the JGraph library and also by completing and adopting the SwiXAT XUL MVC framework in order to be able to write the new editor's interface by using XML and JavaScript. # PM

1.7.2. medium

- **[engine]** Adding of a new output synapse in order to write the results of a neural network in a form suitable to be visualized with GNUPLOT. # PM
- **[engine]** Add the possibility to calculate the quantization error (distance between the input vector and the best-matching unit BMU) of a SOM network. # PM

2. Documentation

2.1. Documentation

2.1.1. Books

If you want to learn more on Neural Networks, reading a good book is the first and most useful thing to do. Look at the titles (over 15000 books!) available at Amazon.com:

A variety of books available on Amazon that might be interesting to users of Joone. Every book you order through the links below helps to fund our project. Thanks in advance.

- The **must have**: our Neural Network preferred books
- The Neural Network books page at Amazon.com

Books containing chapters about JOONE:

- Introduction to Neural Networks with Java - by Jeff Heaton
- "Wicked Cool" Java - by Heinz M. Kabutz

2.1.2. Internal Documentation about Joone

Visit our new Community web page. You can find Tutorials, FAQs, Discussion Forums, Books Review, etc.

- **About Joone**
- **The Core Engine**
    - The Joone Complete Guide (pdf) - All you need to know about Joone (technical details, code examples, etc.)
    - A description of the Core Engine

  - How to build a neural network using java code
- **The GUI Editor**
  - A description of the GUI Editor
  - A sample GUI that learns the XOR problem
- **The Distributed Training Environment**
  - A description of the DTE
  - The DTE Complete Guide (pdf)

2.1.3. Articles

- Using Joone for Artificial Intelligence Programming by Jeff Heaton
- Programming Neural Networks in Java by Jeff Heaton

2.1.4. Projects based on Joone

This is a list of projects that use the Joone's libraries. Feel free to contact us to add here a link to your own project/application

- **Software Development Effort Estimations Through Neural Networks** (by Sebastian Donatti) A very interesting work about the development of a neural network model to solve the software development effort estimation problem with a more complex feed-forward architecture and using a large software development metrics historical database
- A complete **character recognition program** (by Yan Cheng)
- **JOONEGAP** (by Richard Kennard) A Neural Network evolutionary environment

2.1.5. Joone used in the Academic World (write us to list here your work)

Neural Networks and Evolutionary Algorithms (by Huascar Fiorletta) Huascar is one of the most active developers of Joone, and he published his PhD Thesis here in order to share the document with the users of Joone and receive comments, suggestions, etc. If you want, you can write him to huascar80[AT]infinito[DOT]it

- 3D Freeform Surfaces from Planar Sketches using Neural Networks (by Usman Khan, Abdelaziz Terchi, Sungwoo Lim, David Wright, Sheng-Feng Qin)
- Using Java™ Technology-Based Neural Networks to Predict Trauma Mortality (pdf slides presented at JavaOne 2006)
- A Neural Networked-based Approach to a Dead Reckoning Predictor for Haptic Interfaces (pdf)
- Post-processing Analysis of Grid Monitoring data (pdf)
- Real-time Adaptive Control of Modal Synthesis (pdf)
- Layered Learning in RoboCup Rescue Simulation (pdf)
- Ontology Mapping - An Integrated Approach (pdf)
- XPOD - A human activity and emotion aware mobile music player (pdf)

- Modeling and verification of digital logic circuit using neural networks (pdf)
- Hierarchical Confidence Based Clustering (pdf)
- Hardware Implementation of Distributed Artificial Neural Network (DANN) Models of Neural Functioning (pdf)
- Implementation of self-organizing map (SOM) on distributed artificial neural network (DANN) (pdf)
- Developing a Data Communication Model for Distributed Artificial Neural Network (DANN) Remote Training Environments (pdf)
- Using Distributed Network Systems To Model Perception With an Artificial Neural Network (pdf)

2.2. The Core Engine

2.2.1. The Core Engine

2.2.1.1. Features

The core engine is the fulcrum around which are built all the other modules and applications based on Joone. It exposes a Java API providing the developer with the following features:

- Components to create any neural net architecture (feed forward or recurrent)
- Several supervised training algorithms:
  - On-line backprop
  - Batch backprop
  - Resilient Backprop (RPROP)
- Components to build unsupervised neural networks (Kohonen SOMs and Principal Component Analysis)
- Components to build modular neural networks
- A serialization mechanism to save and restore a nnet to/from a file system or to send/receive a nnet on remote locations (via HTTP, RMI, etc.)
- I/O components to read patterns from:
  - Ascii comma delimited files from file system or HTTP
  - Excel files
  - RDBMS using JDBC
  - Stock price time series from Yahoo Finance
- Components to implement both the supervised and unsupervised learning control
- Components to control the behavior of the neural net (start/stop, recall/learn) and its parameters (learning rate, momentum, etc.)
- Plug-ins to preprocess the input data (Normalization, Moving Average, etc.) and to control dynamically the training parameters (Annealing)
- A complete event notification mechanism, along with a scripting capability to control the

behaviour of a neural network at the happening of some events

2.2.1.2. Packages

The engine is composed by a set of components subdivided into the following packages:

- **org.joone.engine**

  All the classes of the core engine wich represent the bricks to build the neural nets
- **org.joone.engine.learning**

  All the components to train the neural net
- **org.joone.io**

  All the I/O components to read and write patterns from/to external sources
- **org.joone.net**

  The shell components of the neural net. They are useful to manage a neural net as a indivisible entity
- **org.joone.util**

  Some utility classes to perform several tasks (input normalization, input scaling, scripting, etc.)
- **org.joone.log**

  The classes to write the output messages to a log file (it can use either Log4J or an internal logger)
- **org.joone.script**

  The classes to define and execute BeanShell scripts (called also Macro)

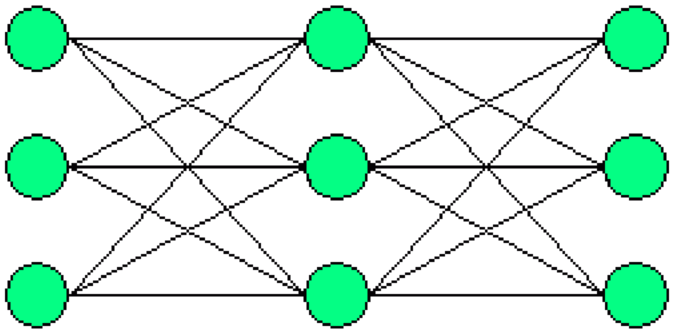
2.2.1.3. Resources

- The Joone Complete Guide (pdf)
- The Java API cookbook
- The Netbeans Homepage (We use it to develop Joone)
- The JOONE GUI Editor
- A sample GUI, that learns the XOR problem
- *A selection of the best Neural Networks books*

2.2.2. The Java API cookbook

2.2.2.1. How to build a NN using java code

The fulcrum of the engine is the Layer object. It is composed by N neurons(that can be set by the attribute 'rows'). Imagine a feed-forward neural net composed by three layers like the following:


a very simple neural network

To build this net with joone, we must create three Layer objects and two Synapse objects:

```
      SigmoidLayer layer1 = new SigmoidLayer();
      SigmoidLayer layer2 = new SigmoidLayer();
      SigmoidLayer layer3 = new SygmoidLayer();
      FullSynapse synapse1 = new FullSynapse();
      FullSynapse synapse2 = new FullSynapse();
```

Then we complete the net connecting the three layers with the synapses:

```
      layer1.addOutputSynapse(synapse1);
      layer2.addInputSynapse(synapse1);
      layer2.addOutputSynapse(synapse2);
      layer3.addInputSynapse(synapse2);
```

Here you can see, each synapse is the output synapse of a layer and the input synapse of the next layer in the net. This simple net is ready, but it can't do any useful job, because there aren't the components to read/write the data that the net must elaborate. Look at the next example to learn how to build a real net, which can be trained and used for a real problem.

2.2.2.2. A real implementation: The XOR Problem

Suppose we must build a net to teach on the classical XOR problem. In this example, the net must learn the following XOR truth table:

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1

1	1	0

Table 1: The XOR truth table

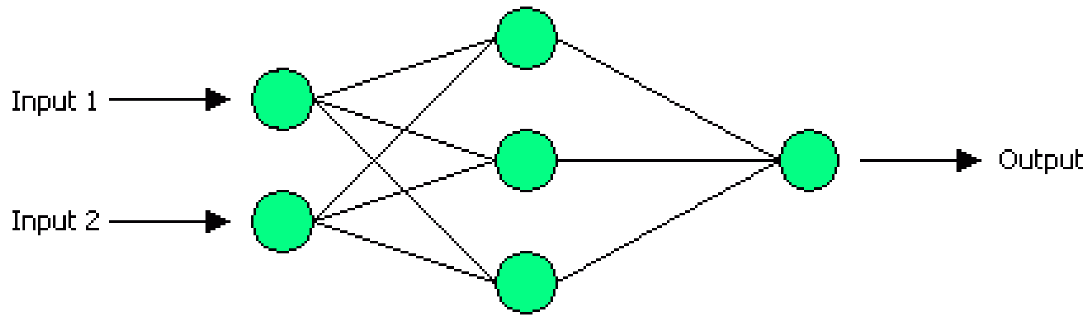
So, we must create a file containing this values:

```
        0.0;0.0;0.0
        0.0;1.0;1.0
        1.0;0.0;1.0
        1.0;1.0;0.0
```

Each column must be separated by a semicolon; the decimal point is not mandatory if the numbers are integer. Write this file with a text editor and save it on the file system (for instance c:\joone\xor.txt in a Windows environment). Now we'll build the neural net that, as the literature says, must have three layers:

1. An input layer with 2 neurons, to map the two inputs of the XOR function.
2. A hidden layer with 3 neurons, a good value to speed up the net's convergence.
3. An output layer with 1 neuron, to represent the XOR function's output.

As shown by the following figure:



sample XOR network

First, we create the three layers (two of them use the sigmoid transfer function):

```
    LinearLayer input = new LinearLayer();
    SigmoidLayer hidden = new SigmoidLayer();
    SigmoidLayer output = new SygmoidLayer();
```

Set their dimensions:

```
    input.setRows(2);
    hidden.setRows(3);
    output.setRows(1);
```

Now we build the neural net connecting the layers, so we create the two synapses; we use the FullSynapse, that connects all the neurons on its input with all the neurons on its output (see the above figure):

```
    FullSynapse synapse_IH = new FullSynapse(); /* Input  -> Hidden conn. */
    FullSynapse synapse_HO = new FullSynapse(); /* Hidden -> Output conn. */
```

First, we connect the input layer with the hidden layer:

```
    input.addOutputSynapse(synapse_IH);
    hidden.addInputSynapse(synapse_IH);
```

And then, the hidden layer with the output layer:

```
    hidden.addOutputSynapse(synapse_HO);
    output.addInputSynapse(synapse_HO);
```

Now we must create a Monitor object to provide the net with all the parameters needed for its work: We create the Monitor object and set its parameter:

```
    Monitor monitor = new Monitor();
    monitor.setLearningRate(0.7);
    monitor.setMomentum(0.5);
```

Give a reference of that Monitor to the layers:

```
    input.setMonitor(monitor);
    hidden.setMonitor(monitor);
    output.setMonitor(monitor);
```

The application registers itself as a monitor's listener, so it can receive the notifications of termination from the net. To do this, the application must implement the org.joone.engine.NeuralNetListener interface.

```
    monitor.addNeuralNetListener(this);
```

Now we must define an input for the net, then we create a org.joone.io.FileInputStream and give it all the parameters:

```
    FileInputSynapse inputStream = new FileInputSynapse();
    /* The first two columns contain the input values */
    inputStream.setAdvancedColumnSelector("1,2");
    /* This is the file that contains the input data */
    inputStream.setFileName("c:\\joone\\XOR.txt");
```

We add the input synapse to the first layer. The input synapse extends the Synapse object, then it can be attached to a layer like a synapse; so the layer doesn't deal with the kind of its input objects.

```
    input.addInputSynapse(inputStream);
```

A neural net can learn from examples, so we must provide to it with the right responses. For each input, in fact, the net must be provided with the difference between the desired response and the effective response gave from the net; the org.joone.engine.learning.TeachingSynapse is the object that has this task:

```
    TeachingSynapse trainer = new TeachingSynapse();
    /* Setting of the file containing the desired responses, provided by a FileInputSyn
    FileInputSynapse samples = new FileInputSynapse();
    samples.setFileName("c:\\joone\\XOR.txt");
    trainer.setDesired(samples);
    /* The output values are on the third column of the file */
    samples.setAdvancedColumnSelector("3");
    /* We give it the monitor's reference */
    trainer.setMonitor(monitor);
```

The TeacherSynapse object extends the Synapse object, then we can add it as the output of the last layer of the net.

```
    output.addOutputSynapse(trainer);
```

Now all the layers must be activated invoking their method start; the layers implement the java.lang.Runnable interface, thereby they are instanziated on separated threads.

```
    input.start();
    hidden.start();
    output.start();
```

We set all the training parameters of the net:

```
    monitor.setTrainingPatterns(4); /* # of rows contained in the input file */
    monitor.setTotCicles(2000); /* How many times the net must be trained on the input
    monitor.setLearning(true); /* The net must be trained */
    monitor.Go(); /* The net starts the training job */
```

(You can find the source code in the CVS repository into the org.joone.example package)

If this example seems too complex (it's only a small net with only 5 neurons!), remember that this is a low-level approach; here we have used only the core engine coding in java. You can build neural nets with joone in the following three ways (ordered by decreasing difficulty):

1. Like the above example, writing java code that uses the Core Engine
2. Inserting the components into the toolbar of a java IDE (i.e. Eclipse, NetBeans , etc.) to use them within the visual tools of the development environment; in fact each public component of joone is a JavaBean object and has a BeanInfo class that describes all its property
3. Using the GUI editor provided with joone (go to here to see the XOR problem built with the editor)

2.2.2.3. Resources

- The Joone Complete Guide (pdf)
- The JOONE Core Engine

- The Netbeans Homepage (We use it to develop Joone)
- The JOONE GUI Editor
- A sample GUI, that learns the XOR problem
- **A selection of the best Neural Networks books**

2.3. The GUI Editor

2.3.1. The GUI Editor

2.3.1.1. Features

The GUI editor is a graphical user interface to visually create, modify and train a neural net. It's based on a graphical framework called **JHotDraw** , an other SourceForge project (Thanks to the authors of that package).

Its main features are:

- Creation of any architecture, according to the nature of the core engine
- Neural net save and restore from a file system
- Setting of all parameters for each component of the neural network
- Checking for the correctness of the neural network
- Use of pre-processing and control plug-ins
- A complete macro editor to define and control the scripting code
- A control centre to train the neural network
- A charting component to visualize the output values of the neural network
- A flexible environment based on XML files to add new layers, synapses, etc. to the editor's palette
- Exporting of the embedded neural network to distribute and run it on external applications

2.3.1.2. Screenshots

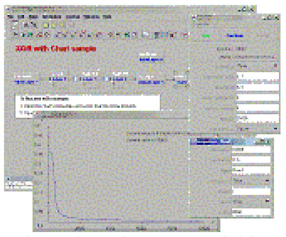
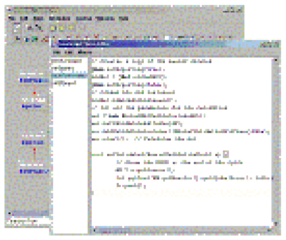
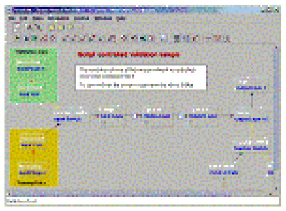
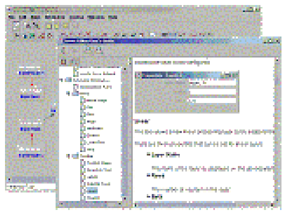
The charting capability	The Macro Editor	The drawing capability	The Help on Line
 The charting capability	 The Macro Editor	 The drawing capability	 The Help on Line

Table 1: Click the images for full size

2.3.1.3. Resources

- A sample GUI, that learns the XOR problem
- The Joone Complete Guide (pdf)
- The JOONE Core Engine
- **A selection of the best Neural Networks books**

2.3.2. A GUI Example

2.3.2.1. The XOR example

Here we'll illustrate an example of how to build a neural net with the GUI editor. Suppose we must build a net to teach on the classical XOR problem. In this example, the net must learn the following XOR truth table:

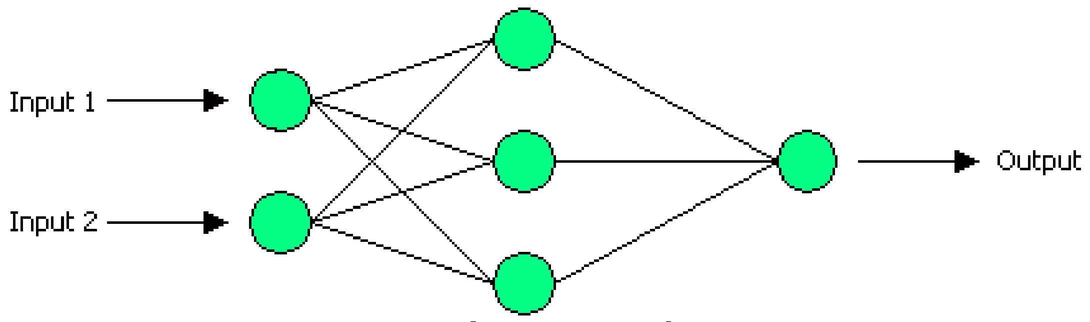
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: The XOR truth table

So, we must create a file containing this values:

```
0.0;0.0;0.0
0.0;1.0;1.0
1.0;0.0;1.0
1.0;1.0;0.0
```

Each column must be separated by a semicolon; the decimal point is not mandatory if the numbers are integer. Write this file with a text editor and save it on the file system (for instance `c:\joone\xor.txt` in a Windows environment). Now we'll build a neural net like this:

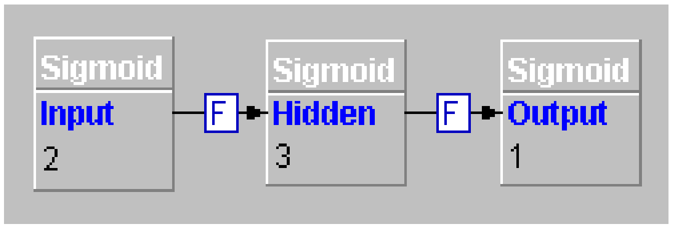


sample XOR network

Run the editor, and execute these following steps:

1. Add a new sigmoid layer and set its layerName to 'Input' and the rows parameter to 2
2. Add a new sigmoid layer, and set its layerName to 'Hidden' and the rows parameter to 3
3. Add a new sigmoid layer, and set its layerName to 'Output', living the rows parameter to 1
4. Connect the input layer to the hidden layer dragging a line from the little circle on the right hand side of the input layer, releasing the mouse button when the arrow is on the hidden layer
5. Repeat the above step connecting the hidden layer to the output layer

   At this stage the screen should look similar to this:



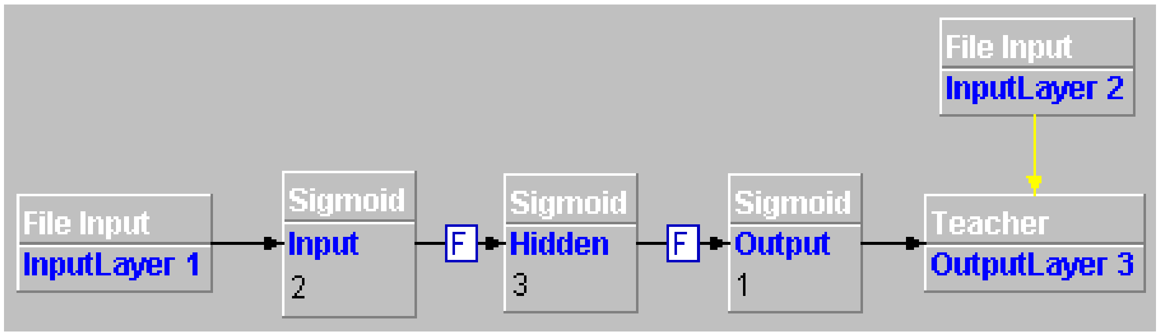
Three XOR layers

6. Insert a File Input layer to the left of the input layer, then click on it to display the properties window:
   - Set the AdvancedColumnSelector parameter to "1,2"
   - Enter c:\joone\xor.txt in the fileName parameter
   - Leave the firstRow as 1 and the lastRow as 0 so that the input layer will read all the rows in the file
7. Connect the File Input to the input layer
8. Insert a Teacher layer to the right of the output layer
9. Connect the output layer to the Teacher layer

   Now we must provide to the teacher the desired data (the last column of the file xor.txt) to train the net:

10. Insert a File Input layer on top of the Teacher layer, then click on it to display the properties window:
    - Set the AdvancedColumnSelector parameter to 3
    - Enter c:\joone\xor.txt in the fileName parameter
    - Leave the firstRow as 1 and the lastRow as 0 so that the input layer will read all the rows in the file
11. Connect the Teacher layer to that last File Input layer dragging a line from the little red box on the top side of the Teacher layer, releasing the mouse button when the yellow arrow is on the last inserted File Input layer.

    At this stage the screen should look similar to this:



Three XOR layers

12. Click on the 'Net->Control Panel' menu item to display the control panel. Insert the following:
    - Set the `epochs` parameter to 10,000. This will process the file 10,000 times
    - Set the `training patterns` parameter to 4. This sets the number of example rows to read
    - Set the `learningRate` parameter to 0.8 and the `momentum` parameter to 0.3
    - Set the `learning` parameter to TRUE, as the net must be trained
13. Click the **START** button, and you'll see the training process starting

The Control Panel shows the cycles remaining and the current error of the net. At the end of the last cycle, the error would be very small (less than 0.1), otherwise click on the 'Net->Randomize' menu item (to add a noise to the weights of the net) and click again on the START button.

If you want, you can save the net with the 'File->Save as' menu item, so you can use again the net later loading it from the file system.

2.3.2.2. Testing the XOR example

To test the trained net:

1. Add an Output File layer on the right of the output layer, click on it and insert into the properties window:
   - "c:\temp\xorout.txt" on the fileName parameter.
2. Connect the output layer to the File Output layer
3. Select the line that connects the output layer to the Teacher layer and click on the 'Edit->Delete' to disconnect the Teacher from the neural net
4. On the Control Panel change the following:
   - Set the epochs parameter to 1. This will process the input file once
   - Set the learning parameter to FALSE, as the net is not being trained
5. Click on the **START** button
6. Open the xorout.txt file with an editor, and you'll see a result like this (the values can change from a run to another, depending on the initial random values of the weights):

```
          0.02592995527027603
          0.9664584492704527
          0.9648193164739925
          0.03994103766843536
```

This result shows that the neural net has learned the XOR problem, providing the correct results:

- a value near zero when the input columns are equal to `[0, 0]` and `[1, 1]`
- a value near one when the input columns are equal to `[0, 1]` and `[1, 0]`

2.3.2.3. Resources

- The JOONE GUI Editor
- The Joone Complete Guide (pdf)
- The JOONE Core Engine
- **A selection of the best Neural Networks books**

2.4. The DTE.

2.4.1. The Distributed Training Environment (DTE)

2.4.1.1. Introduction

The idea that underlies Joone is to build a professional environment to create, train, distribute and use neural networks in a simple but powerful way. To accomplish this final goal, we have built a distributed training environment to train in parallel many neural networks.

Our dream is to provide a professional framework to permit everyone to build neural networks that could be trained on a lot of common problems of the real life suitable to be resolved by the neural networks (spoken and handwriting recognition, stock quotes forecasting, credit loan assessment, etc.), and then distribute these trained nets to interested users (final users, sales force, etc.) that can use the trained neural networks on their devices (desktops, palmtops, PDA, etc.).

Thanks to this framework it would be possible to originate a 'Neural Network Farm' that could fabricate neural networks by training them in parallel on several machines, and finally distribute them to the final users (e.g. for free, or as prepaid service), to *'train once, run anywhere'* a neural network without being worried about the HW or SW platform owned by the user.

2.4.1.2. Why a distributed environment?

When we want to use the neural network technology to resolve a complex job, training only one neural network, often, doesn't resolve the problem, because the net can fall onto a local minima without finding the best results; thereby what must be developed is a mechanism to train many neural nets in parallel on the same problem, on several machines, governing the whole process from a central point.

Joone has an own complete and professional framework where it's possible to create a neural net (with the GUI editor), and then teach several copies of it on many machines where some agents are running (they also based on joone's engine, of course). Those agents (or Workers) reveive a copy of the neural net and train it, sending back the outcome (i.e. the trained net) to the starting machine where the user, or an automatic script, can control all the results of the distrubuted training process, being so able to choose, at the end of the process, the best net, by applying some discriminating mechanism: e.g. choosing the net with the lowest error, applying a genetic algorithm, etc.

2.4.1.3. Possible Applications

Maybe most of you will have already thought about some useful application suitable to be resolved by the DTE, but I think that it could be useful to give some ideas about possible applications of a distributed training environment like this one. Although the global optimization techniques illustrates here can be applied to any problem suitable to be resolved using neural networks, I want to use the most famous and, maybe, most appreciated example: global optimization for **Financial Forecasting**.

Input Selection

If you're thinking to use neural networks to make financial forecasting, maybe you need the DTE, because you could train several networks on several stocks, in order to find the most predictable one (i.e. that one having less noise – why waste time and money on a noisily stock?). Moreover, you could train in parallel several networks having as input different financial indicators in order to find the best combination of them to obtain good performances. The message is: do not choose yourself the input patterns to use, but instead let Joone DTE do it for you.

Parameter Selection

You could also use the DTE to train different neural networks architectures, in order to verify the influence of a different number of hidden neurons, a different weights initialization, a different initial learning rate, a different input temporal window' size, or... a whatever combination of them.

Committee of Experts

Moreover, you could train several neural networks on the same time series and, after that, make predictions by averaging the results of all the trained neural networks (or only using the fittest ones). If a single network could generate bad predictions, maybe a committee of experts could make better predictions. You could also decide to not trade in presence of conflicting outputs, avoiding in this case to make potentially losing trades by entering in the market when it is not very predictable.

2.4.1.4. Features

- Console based interface
- Centralized control
- The Final results are logged into a XML file to permit to analyze the results from a custom program/script
- The training process scale almost linearly adding more machines
- No manual configuration needed to add or remove machines
- Possibility to add or remove machines dynamically during the training process
- The overall process controlled by XML parameters
- Latest SUN's Jini 2.0 compliant
- Easily expandible with new features/algorithms/control modules, etc.

2.4.1.5. How does it work?

The Jini Technology come to our aid, providing a mechanism to implement distributed parallel processing. It's name is the JavaSpaces technology. The core engine has been conceived with many features that make it suitable to be used easily in a distributed environment like Jini/JavaSpaces, in fact:

- All the classes of the joone engine are serializable (the JavaSpaces transport mechanism is based on serialization)

- The input synapses have a buffering mechanism to store input data, so a neural network can be transported along with its training data
- The possibility to use a script to control the behaviour of a neural network represents a powerful mechanism to control remotely its training and validation phases (i.e. a neural network is transported along with its own control logic)

The underlying concept is very simple: there are mainly three components that take part in the whole process:

1. The Jini based services: JavaSpaces, the Transaction Manager and a Lookup Service
2. Several Workers running on different machines connected to a LAN
3. A Master controller, or Job runner

Both the Workers and the Master use the Lookup Service to discover the JavaSpaces and the Transaction Manager services, and register themself as listeners of the JavaSpaces in order to be notified when a neural network is available on it to be elaborated.

The Master, by reading a parameter file written in XML, generates all the tasks to perform (i.e. the neural networks to train); the generated tasks are sent to the JavaSpaces to make them available to all the running Workers. After that the Master waits for a JavaSpaces event to be notified about the availability of trained neural networks.

The Workers are notified about the presence of a new neural network to elaborate. At the above notification event, one of the ready Workers 'takes' that neural network, train it on the remote machine where it lives, and at the end sends back the results to the JavaSpaces. After that the Worker returns in a 'suspended' state waiting for a new notification

When a trained network is available on the JavaSpaces, the Master is notified, so it can 'take' that network from the JavaSpaces and store it on the file system, writing also the results to a log file in XML format

To learn more, read the DTE Complete Guide, and download the binary packages.

2.4.1.6. Credits

The Joone DTE uses the following external packages:

- SUN Jini Network Technology
- The Computefarm Framework (by Tom White)
- The Spring Framework (by Rod Johnson, Juergen Hoeller)

We want to thank all the authors and contributors of the above packages. Please read the respective licenses contained in the distribution package.

2.4.1.7. A robust alternative to Jini: GreenTea

If you want to try an alternative to the Jini/JavaSpaces based DTE, download the GTJoone package, a porting of the previous version of the DTE built on GreenTea, a pure Java Peer-to-Peer (P2P) based Grid OS platform.

2.4.1.8. Resources

- The DTE Complete Guide (pdf)
- The Jini Technology
- The JavaSpaces Technology
- The Joone Complete Guide (pdf)
- The Java API cookbook
- The JOONE GUI Editor
- *A selection of the best Neural Networks books*

2.5. Whole site

2.5.1.

2.5.2.

3. Community

3.1.

3.2.

3.3.