



# DAta Mining & Exploration Program



Dipartimento di Scienze Fisiche  
Università di Napoli "Federico II"



ISTITUTO NAZIONALE di ASTROFISICA  
OSSERVATORIO ASTRONOMICo di CAPODIMONTE



CALTECH



*Multi Layer Perceptron  
trained by  
Quasi Newton Algorithm  
or  
Levenberg-Marquardt Optimization  
Network*

*MLPQNA/LEMON User Manual*

DAME-MAN-NA-0015

Issue: 1.3  
Author: M. Brescia, S. Riccardi

Doc. : MLPQNA\_UserManual\_DAME-MAN-NA-0015-Rel1.3

DAMEWARE MLPQNA + MLPLEMON Model User Manual

This document contains proprietary information of DAME project Board. All Rights Reserved.



# DAta Mining & Exploration Program



## Index

1	Introduction .....	4
2	MLPQNA Model Theoretical Overview.....	5
3	Use of the web application model .....	8
3.1	Use Cases .....	8
3.2	Input .....	9
3.3	Output .....	10
3.4	TRAIN Use case .....	11
3.4.1	Regression with MLPQNA – Train Parameter Specifications .....	11
3.4.2	Classification with MLPQNA – Train Parameter Specifications .....	14
3.5	TEST Use case .....	18
3.5.1	Regression with MLPQNA – Test Parameter Specifications .....	18
3.5.2	Classification with MLPQNA – Test Parameter Specifications .....	20
3.6	Run Use case.....	22
3.6.1	Regression with MLPQNA – Run Parameter Specifications .....	22
3.6.2	Classification with MLPQNA – Run Parameter Specifications .....	24
4	Examples .....	26
4.1	Regression XOR problem .....	26
4.1.1	Regression MLPQNA – Train use case.....	26
4.1.2	Regression MLPQNA – Test use case.....	28
5	Appendix – References and Acronyms .....	31

## TABLE INDEX

<i>Tab. 1 – output file list in case of regression type experiments.....</i>	<i>10</i>
<i>Tab. 2 – output file list in case of classification type experiments .....</i>	<i>11</i>
<i>Tab. 3 – Abbreviations and acronyms.....</i>	<i>31</i>
<i>Tab. 4 – Reference Documents.....</i>	<i>32</i>
<i>Tab. 5 – Applicable Documents.....</i>	<i>33</i>



# DAta Mining & Exploration Program

## FIGURE INDEX

<i>Fig. 1 – MLP architecture</i> .....	5
<i>Fig. 2 – The learning mechanism of QNA rule, using the error function</i> .....	6
<i>Fig. 3 – The content of the xor.csv file used as input for training/test use cases</i> .....	9
<i>Fig. 4 – The content of the xor_run.csv file used as input for Run use case</i> .....	9
<i>Fig. 5 – The setup tab for regression + MLPQNA train use case</i> .....	12
<i>Fig. 6 – The setup tab for classification + MLPQNA train use case</i> .....	15
<i>Fig. 7 – The setup tab for regression + MLPQNA test use case</i> .....	18
<i>Fig. 8 – The setup tab for classification + MLPQNA test use case</i> .....	20
<i>Fig. 9 – The setup tab for regression + MLPQNA run use case</i> .....	22
<i>Fig. 10 – The setup tab for classification + MLPQNA run use case</i> .....	24
<i>Fig. 11 – The starting point, with a Workspace (mlpqnaExp) created and two data files uploaded</i> .....	26
<i>Fig. 12 – The xorTrain experiment configuration tab</i> .....	27
<i>Fig. 13 – The xorTrain experiment status after submission</i> .....	27
<i>Fig. 14 – The xorTrain experiment output files</i> .....	28
<i>Fig. 15 – The files error (left) and weights (right) output of the xorTrain experiment</i> .....	28
<i>Fig. 16 – The file “weights” and “frozen_train_net” copied in the WS input file area for next purposes</i> .....	29
<i>Fig. 17 – The xorTest experiment configuration tab (note “weights” file and frozen_train_net file inserted)</i> .....	29
<i>Fig. 18 – The xorTest experiment output files</i> .....	30



# DAta Mining & Exploration Program

## 1 Introduction

**T**he present document is the user guide of the data mining models MLPQNA (Multi Layer Perceptron with Quasi Newton) and MLP-LEMON (Multi Layer Perceptron with Levenberg Marquardt), as implemented and integrated into the DAMEWARE web application. They are models that can be used to execute scientific experiments for both classification (crispy or two-class mode) and regression (single output mode) on massive data sets, formatted in one of the supported types: ASCII (columns separated by spaces), CSV (comma separated values), FITS-Table (numerical columns embedded into the fits file) or VOTable.

More scientific and technical information about models and their use in the astrophysical context are available in [R9, R10, R11, R12].

This manual is one of the specific guides (one for each data mining model available in the webapp) having the main scope to help user to understand theoretical aspects of the model, to make decisions about its practical use in problem solving cases and to use it to perform experiments through the webapp, by also being able to select the right functionality associated to the models, based upon the specific problem and related data to be explored, to select the use cases, to configure internal parameters, to launch experiments and to evaluate results.

**The documentation package consists also of a general reference manual on the webapp (useful also to understand what we intend for association between functionality and data mining model) and a GUI user guide, providing detailed description on how to use all GUI features and options.**

**So far, we strongly suggest to read these two manuals and to take a little bit of practical experience with the webapp interface before to explore specific model features, by reading this and the other model guides.**

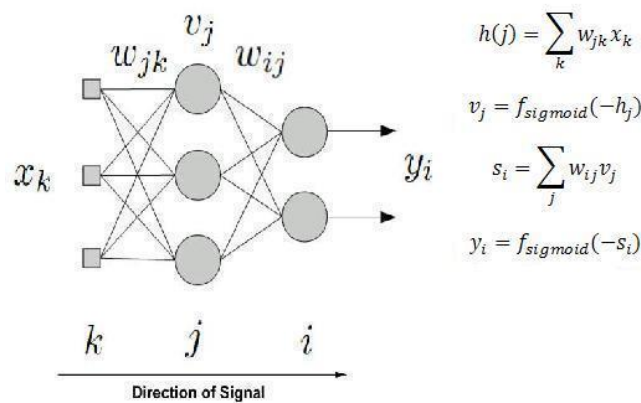
**All the cited documentation package is available from the address <http://dame.dsf.unina.it/dameware.html>, where there is also the direct gateway to the beta webapp.**

As general suggestion, the only effort required to the end user is to have a bit of faith in Artificial Intelligence and a little amount of patience to learn basic principles of its models and strategies.

By merging for fun two famous commercial taglines we say: *“Think different, Just do it!”*  
(casually this is an example of *data (text) mining...!*)

## 2 MLPQNA Model Theoretical Overview

The MLP architecture is one of the most typical *feed-forward* neural network model. The term feed-forward is used to identify basic behavior of such neural models, in which the impulse is propagated always in the same direction, e.g. from neuron input layer towards output layer, through one or more hidden layers (the network brain), by combining weighted sum of *weights associated to all neurons* (except the input layer). As easy to understand, the neurons are organized in layers, with proper own role. The input signal, simply propagated throughout the neurons of the input layer, is used to stimulate next hidden and output neuron layers. The output of each neuron is obtained by means of an *activation function*, applied to the weighted sum of its inputs. Different shape of this **activation function** can be applied, from the simplest *linear* one up to *sigmoid*. The number of hidden layers represents the degree of the complexity achieved for the energy solution space in which the network output moves looking for the best solution. As an example, in a typical classification problem, the number of hidden layers indicates the number of hyper-planes used to split the parameter space (i.e. number of possible classes) in order to classify each input pattern. What is different in such a neural network architecture is typically the learning algorithm used to train the network. It exists a dichotomy between *supervised* and *unsupervised* learning methods.



**Fig. 1 – MLP architecture**

In the first case, the network must be firstly trained (*training phase*), in which the input patterns are submitted to the network as couples (input, desired known output). The feed-forward algorithm is then achieved and at the end of the input submission, the network output is compared with the corresponding desired output in order to quantify the learning quote. It is possible to perform the comparison in a *batch* way (after an entire input pattern set submission) or *incremental* (the comparison is done after each input pattern submission) and also the *metric* used for the *distance* measure between desired and obtained outputs, can be chosen accordingly problem specific requirements (the simplest is **MSE**, Mean Square Error).

After each comparison and until a desired error distance is unreached (typically the error tolerance is a pre-calculated value or a constant imposed by the user), the weights of hidden layers must be changed accordingly to a particular law or learning technique.

After the training phase is finished (or arbitrarily stopped), the network should be able not only to recognize correct output for each input already used as training set, but also to achieve a certain degree of *generalization*, i.e. to give correct output for those inputs never used before to train it. The degree of generalization varies, as obvious, depending on how “good” has been the learning phase. This important feature is realized because the network doesn’t associates a single input to the output, but it discovers the relationship present behind their association. After training, such a neural network can be seen as a black box able to perform a particular function (input-output correlation) whose analytical shape is a priori not known.



# DAta Mining & Exploration Program

In order to gain the best training, it must be as much homogeneous as possible and able to describe a great variety of samples. Bigger the training set, higher will be the network generalization capability.

Despite of these considerations, it should always take into account that neural networks application field should be usually referred to problems where it is needed high flexibility (quantitative result) more than high precision (qualitative results).

Concerning the **hidden layer choice**, there is the possibility to define zero hidden layers (SLP, Single Layer Perceptron, able to solve only linear separation of the parameter space), 1 or 2 hidden layers, depending on the complexity the user wants to introduce in the not linear problem solving experiment.

Second learning type (unsupervised) is basically referred to neural models able to classify/cluster patterns onto several categories, based on their common features, by submitting training inputs without related desired outputs. This is not the learning case approached with the MLP architecture, so it is not important to add more information in this document.

The Newton method is the general basis for a whole family of so called **Quasi-Newton** methods. One of those methods, implemented here is the L-BFGS algorithm. More rigorously, the QNA is an optimization of learning rule, also because, as described below, the implementation is based on a statistical approximation of the Hessian by cyclic gradient calculation, that, as said in the previous section, is at the base of BP method.

As known, the classical Newton method uses the Hessian of a function. The step of the method is defined as a product of an inverse Hessian matrix and a function gradient. If the function is a positive definite quadratic form, we can reach the function minimum in one step. In case of an indefinite quadratic form (which has no minimum), we will reach the maximum or saddle point. In short, the method finds the stationary point of a quadratic form.

In practice, we usually have functions which are not quadratic forms. If such a function is smooth, it is sufficiently good described by a quadratic form in the minimum neighborhood. However, the Newton method can converge both to a minimum and a maximum (taking a step into the direction of a function increasing).

Quasi-Newton methods solve this problem as follows: they use a positive definite approximation instead of a Hessian. If Hessian is positive definite, we make the step using the Newton method. If Hessian is indefinite, we modify it to make it positive definite, and then perform a step using the Newton method. The step is always performed in the direction of the function decrement. In case of a positive definite Hessian, we use it to generate a quadratic surface approximation. This should make the convergence better. If Hessian is indefinite, we just move to where function decreases.

$$\min_w E(w) = \frac{1}{2P} \sum_{p=1}^P E_p(w)$$

$$w^{k+1} = w^k + \alpha^k d^k$$

$d^k \in R^N$  DIRECTION OF RESEARCH

$\alpha^k \in R$  STEP

$d^k = -\nabla E(w^k)$	Descent gradient (BP)
$d^k = \dots$	⋮
$\nabla^2 E(w^k) d^k = -\nabla E(w^k)$	Hessian approx. (QNA)

Fig. 2 – The learning mechanism of QNA rule, using the error function



# DAta Mining & Exploration Program

Some modifications of Quasi-Newton methods perform a precise linear minimum search along the indicated line, but it is proved that it's enough to sufficiently decrease the function value, and not necessary to find a precise minimum value. The L-BFGS algorithm tries to perform a step using the Newton method. If it does not lead to a function value decreasing, it lessens the step length to find a lesser function value.

Up to here it seems quite simple...but it is not!

The Hessian of a function isn't always available and in many cases is too much complicated; more often we can only calculate the function gradient.

Therefore, the following operation is used: the Hessian of a function is generated on the basis of the  $N$  consequent gradient calculations, and the quasi-Newton step is performed. There is a special formulas which allows to iteratively get a Hessian approximation. On each step approximation, the matrix remains positive definite. The algorithm uses the **L-BFGS** update scheme. BFGS stands for Broyden-Fletcher-Goldfarb-Shanno (more precisely, this scheme generates not the Hessian, but its inverse matrix, so we don't have to waste time inverting a Hessian).

The L letter in the scheme name comes from the words "Limited memory". In case of big dimensions, the amount of memory required to store a Hessian ( $N^2$ ) is too big, along with the machine time required to process it. Therefore, instead of using  $N$  gradient values to generate a Hessian we can use a smaller number of values, which requires a memory capacity of order of  $N \cdot M$ . In practice,  $M$  is usually chosen from 3 to 7, in difficult cases it is reasonable to increase this constant to 20. Of course, as a result we'll get not the Hessian but its approximation. On the one hand, the convergence slows down. On the other hand, the performance could even grow up. At first sight, this statement is paradoxical. But it contains no contradictions: the convergence is measured by a number of iterations, whereas the performance depends on the number of processor's time units spent to calculate the result.

As a matter of fact, this method was designed to optimize the functions of a number of arguments (hundreds and thousands), because in this case it is worth having an increasing iteration number due to the lower approximation precision because the overheads become much lower. This is particularly useful in astrophysical data mining problems, where usually the parameter space is dimensionally huge and confused by a low signal-to-noise ratio. But we can use these methods for small dimension problems too. The main advantage of the method is scalability, because it provides high performance when solving high dimensionality problems, and it allows to solve small dimension problems too.

From the implementation point of view, in the MLPQNA case the following features are available for the end user:

- only batch learning mode is available;
- Strict separation between classification and regression functionality modes;
- For classification mode, the Cross Entropy method is available to compare output and target network values. It is possible to alternatively use standard MSE rule, that is mandatory for regression mode;
- K-fold cross validation method to improve training performances and to avoid overfitting problems;
- Resume training from past experiments, by using the weights stored in an external file at the end of the training phase;
- Confusion matrix calculated and stored in an external file for both classification and regression modes (in the last case an adapted version is provided). It is useful after training and test sessions to evaluate model performances.

The MLP-LEMON is identical to MLP-QNA, with the only exception of:

- The Hessian of the MLP error is not approximated, but exactly calculated, using the rule inspired to the Levenberg-Marquardt optimization strategy;
- MLP-LEMON has same parameters of MLPQNA, but internally it uses only restarts and decay values;



# DAta Mining & Exploration Program

## 3 Use of the web application model

The MLPQNA is a neural network based on the self-adaptive machine learning in supervised mode (i.e. trained by input/output samples), used in many application fields and specialized to work on massive data sets, i.e. data sets formed by thousands of rows (patterns), composed by tenth of columns (pattern features).

It is especially related to classification/regression problems, and in DAME it is designed to be associated with such functionality domains. The description of these functionalities is reported in the Reference Manual [A18], available from webapp GUI (Graphical User Interface) menu or from the intro web page (see address in the introduction).

In the following are described practical information to configure the network architecture and the learning algorithm in order to launch and execute science cases and experiments.

There are two functionalities available in the webapp for this model. The functionalities extracted have been plugged into the DAMEWARE by using an internal software integration wrapping system and available as two methods from the GUI.

The methods are:

- Classification\_MLPQNA or Classification\_MLPLEMON: use this method to perform classification experiments, in which the model is able to learn and perform generalization for the prediction of input patterns as belonging to one of possible classes or categories (for example star/galaxy separation);
- Regression\_MLPQNA or Regression\_MLPLEMON: use this method to perform nonlinear regression experiments, i.e. one-dimension approximation or interpolation function, where its analytical expression is not known.

### 3.1 Use Cases

For the user the MLPQNA system offers four use cases:

- *Train*
- *Test*
- *Run*

As described in [A19] a supervised machine learning model like MLPQNA or MLPLEMON requires different use cases, well ordered in terms of setup and execution sequence. A typical complete experiment consists of the following steps:

1. **Train** the network with a dataset as input, containing both input and target features; then store as output the final weight matrix (best configuration of trained network weights);
2. **Test** the trained network, in order to verify training quality (it is also available a validation procedure, based on statistical analysis, named k-fold cross validation). The same training dataset or a mix with new patterns can be used as input;
3. **Run** the trained and tested network with datasets containing ONLY input features (without target ones). In this case new or different input data are encouraged, because the Run use case implies to simply execute the model, like a generic static function.





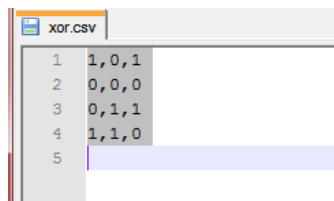
# DAta Mining & Exploration Program

## 3.2 Input

We also remark that massive datasets to be used in the various use cases are (and sometimes must be) different in terms of internal file content representation. Remind that in all DAME models it is possible to use one of the following data types:

- ASCII (extension .dat or .txt): simple text file containing rows (patterns) and columns (features) separated by spaces, normally without header;
- CSV (extension .csv): Comma Separated Values files, where columns are separated by commas;
- FITS (extension .fits): fits files containing tables;
- VOTABLE (extension .votable): formatted files containing special fields separated by keywords coming from XML language, with more special keywords defined by VO data standards;

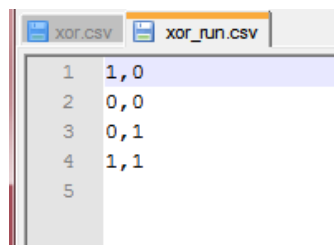
For training and test cases a correct dataset file must contain both input and target features (columns), with input type as the first group and target type as the final group.



	1	2	3
1	1	0	1
2	0	0	0
3	0	1	1
4	1	1	0
5			

**Fig. 3 – The content of the xor.csv file used as input for training/test use cases**

As shown in Fig. 3, the xor.csv file for training/test uses cases has 4 patterns (rows) of 2 input features (first two columns) and one target feature (third column). The target feature is not an input information but the desired output to be used in the comparison (calculation of the error) with the model output during a training/test experiment.



	1	2
1	1	0
2	0	0
3	0	1
4	1	1
5		

**Fig. 4 – The content of the xor\_run.csv file used as input for Run use case**

In Fig. 4, the xor\_run.csv file is shown, valid only for Run use case experiments. It is the same of xor.csv except for the target column that is not present. This file can be also generated by the user starting from the xor.csv. As detailed in the GUI user Guide [A19], the user may in fact use the Dataset Editor options of the webapp to manipulate and build datasets starting from uploaded data files.



# DAta Mining & Exploration Program

## 3.3 Output

In terms of output, different files are obtained, depending on the specific use case of the experiment. In the case of **regression** functionality, the following output files are obtained in all use cases:

TRAIN	DESCRIPTION	REMARKS
mlpqna_TRAIN.log	Experiment status report log	
mlpqna_TRAIN_error.txt	Error trend during training	
mlpqna_TRAIN_output.txt	Output of training with input dataset in append	
<input dataset name>_mlpqna_TRAIN_TrainTestOutLog.txt	training Output vs target table	
mlpqna_TRAIN_errorPlot.jpeg	Error trend plot	
MLPQNA_Train_params.xml	Experiment configuration	
<input dataset name>_mlpqna_TRAIN_frozen_net.txt	Model frozen setup after training	Must be moved to File Manager tab to be used for test and run use cases
mlpqna_TRAIN_weights.txt	Network trained weights	Must be moved to File Manager tab to be used for test and run use cases
TEST	DESCRIPTION	REMARKS
mlpqna_TEST.log	Experiment report log	
mlpqna_TEST_output.txt	Output of test with input dataset in append	
mlpqna_TEST_outputPlot.jpeg	Output plot	
MLPQNA_Test_params.xml	Experiment configuration	
RUN	DESCRIPTION	REMARKS
mlpqna_RUN.log	Experiment report log	
mlpqna_RUN_output.txt	Output of run with input dataset in append	
MLPQNA_Run_params.xml	Experiment configuration	

**Tab. 1 – output file list in case of regression type experiments (in case of LEMON model the prefix of the files may be changed accordingly)**

In the case of **classification** functionality, the following output files are obtained in all use cases:

TRAIN	DESCRIPTION	REMARKS
mlpqna_TRAIN.log	Experiment report log	
mlpqna_TRAIN_error.txt	Error trend during training	
mlpqna_TRAIN_output.txt	Output of training with input dataset in append	
mlpqna_TRAIN_trainTestOutLog.txt	Output vs target	
mlpqna_TRAIN_errorPlot.jpeg	Error trend plot	
mlpqna_TRAIN_ConfMatrix.txt	Output confusion matrix	
MLPQNA_Train_params.xml	Experiment configuration	
mlpqna_TRAIN_frozen_net.txt	Model setup after training	Must be moved to File Manager tab to be used for test and run use cases
mlpqna_TRAIN_weights.txt	Network trained weights	Must be moved to File Manager tab to be used for test and run use cases
TEST	DESCRIPTION	REMARKS
mlpqna_TEST.log	Experiment report log	
mlpqna_TEST_output.txt	Output of test with input dataset in append	



# DAta Mining & Exploration Program

mlpqna_TEST_TestOutLog.txt	Output vs target	
mlpqna_TEST_testConfMatrix.txt	Output confusion matrix	
MLPQNA_Test_params.xml	Experiment configuration	
<b>RUN</b>	<b>DESCRIPTION</b>	<b>REMARKS</b>
mlpqna_RUN.log	Experiment report log	
mlpqna_RUN_output.txt	Output of run with input dataset in append	
MLPQNA_Run_params.xml	Experiment configuration	

**Tab. 2 – output file list in case of classification type experiments (in case of LEMON model the prefix of the files may be changed accordingly)**

### 3.4 TRAIN Use case

In the use case named “**Train**”, the software provides the possibility to train the MLPQNA or MLPLEMON. In the following we refer to MLPQNA what is considered fully applicable also to MLPLEMON. The user will be able to use new or existing (already trained) weight configurations, adjust parameters, set training parameters, set training dataset, manipulate the training dataset and execute the training experiments.

There are several parameters to be set to achieve training, specific for network topology and learning algorithm setup. In the experiment configuration there is also the Help button, redirecting to a web page dedicated to support the user with deep information about all parameters and their default values.

We remark that all parameters labeled by an asterisk are considered as required. In all other cases the fields can be left empty (default values are used and shown in the help web pages).

#### 3.4.1 Regression with MLPQNA – Train Parameter Specifications

The setup of train use case for MLPQNA model, used for regression problems, is shown in Fig. 5.



# DAta Mining & Exploration Program

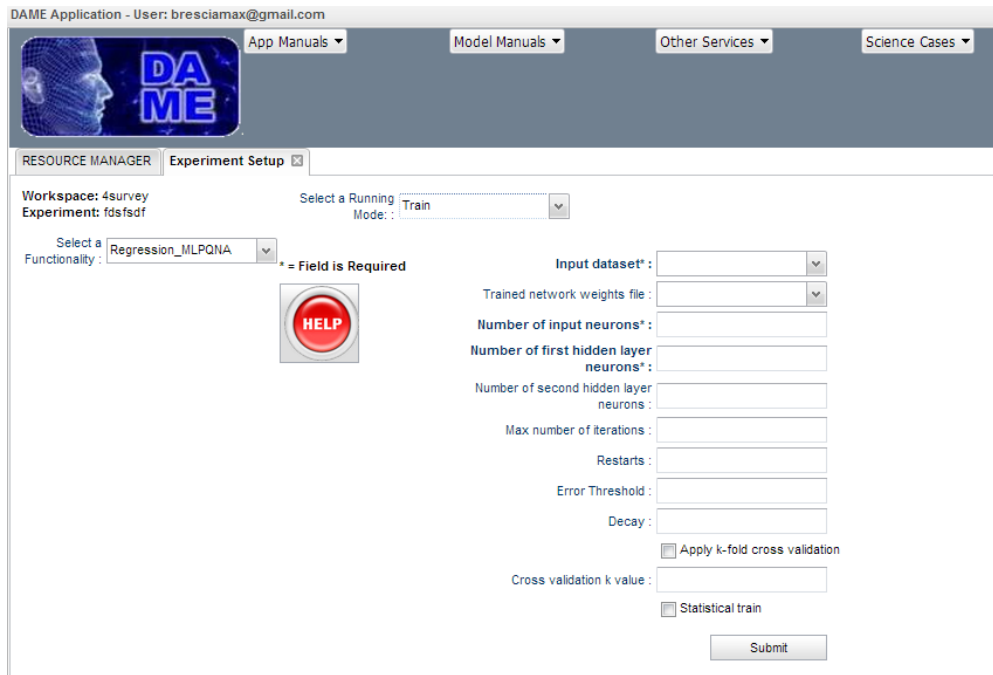


Fig. 5 – The setup tab for regression + MLPQNA train use case

In the case of Regression\_MLPQNA with Train use case, the help page is at the address:  
[http://dame.ds.unina.it/mlpqna\\_help.html#regr\\_train](http://dame.ds.unina.it/mlpqna_help.html#regr_train)

- **Input dataset**

**this parameter is a field required!**

This is the dataset file to be used as input for the learning phase of the model.

The file must be already uploaded by user and available in the current workspace of the experiment

It typically must include both input and target columns, where each row is an entire pattern (or sample of data). The format (hence its extension) must be one of the types allowed by the application (ASCII, FITS-table, CSV, VOTABLE). **More specifically, take in mind the following simple rule: the sum of input and output nodes MUST be equal to the total number of the columns in this file!**

- **Trained network weights file**

It is a file generated by the model during training phase. It contains the resulting network weights, associated to neurons, as stored at the end of a previous training session. Usually this file should not be edited or modified by users, just to preserve its content as generated by the model itself.

The canonical use of this file in this use case is to resume a previous training phase, in order to try to improve it. If users leaves empty this parameter field, by default the current training session starts from scratch, (i.e. the initial weights are randomly generated).

- **Number of input neurons**



# DAta Mining & Exploration Program

**this parameter is a field required!**

It is the number of neurons at the first (input) layer of the network. **It must exactly correspond to the number of input columns in the dataset input file (Training File field), except the target columns.**

- **Number of first hidden layer neurons**

**this parameter is a field required!**

It is the number of neurons of the first hidden layer of the network. As suggestion this should be selected in a range between a minimum of 1.5 times the number of input nodes and a maximum of 2 times + 1 the number of input nodes.

- **Number of second hidden layer neurons**

It is the number of neurons of the second hidden layer of the network. As suggestion this should be selected smaller than the previous layer.

For most experiments, this layer is considered redundant, except for particular complex cases. So use it only if really needed. Moreover some theoretical issues have demonstrated that one hidden layer only is sufficient to solve usual non-linear regression problems.

By default the second hidden layer is empty (not used)

- **Max number of iterations**

One of the internal model parameters. It indicates the number of algorithm iterations, i.e. the maximum number of iterations for each approximation step of the Hessian inverse matrix. It is one of the stopping criteria.

By default the value is 1500

- **Restarts**

One of the internal model parameters. It indicates the number of restarts for each approximation step of the Hessian inverse matrix.

By default the value is 20

- **Error Threshold**

One of the internal model parameters. It indicates the minimum weight error at each iteration step. Except for those problems particularly difficult to solve, in which a value of 0.0001 should be used, a value of 0.01 is usually considered sufficient.

By default the value is 0.01

- **Decay**



# DAta Mining & Exploration Program

One of the internal model parameters. It indicates the weight regularization decay. If accurately chosen, the effect is an important improvement of the generalization error of the trained neural network, with also an acceleration of training.

This parameter is particularly important. In case of unknown value to choose, one can try with values within the range of 0.001 (weak case) up to 100 (very strong regularization). It should be noted that if the chosen value is too small (less than 0.001), it will be automatically increased up to the allowed minimum.

By default the value is 0.001

- **Apply k-fold cross validation**

This is a checkboxed parameter. If selected the validation of the training is activated, by following the k-fold method, i.e. based on an automatic procedure that splits in different subsets the training dataset, applying a k step cycle in which the training error is evaluated and its performances are validated.

If you select this option, you should specify the k value in the next parameter below. Take into account that, if selected, this option will statistically improve the training but the execution time will dramatically grow up.

By default the value is unselected.

- **Cross validation k value**

k-fold cross validation parameter that specify the value of k. Use it in combination with the previous checkbox.

By default the value is 10, but it is not used if the previous k-fold checkbox is unselected.

- **Statistical train**

This is a checkboxed parameter. If selected it engages a very time expensive cycle, in which the training set is split into a growing subset iteratively submitted to the model. This is useful only in case user wants to evaluate the training performances for several dimensions of input patterns.

By default the value is unselected. Use it very carefully and only if really required.

## 3.4.2 Classification with MLPQNA – Train Parameter Specifications

The setup of train use case for MLPQNA model, used for classification problems, is shown in Fig. 6.



# DAta Mining & Exploration Program

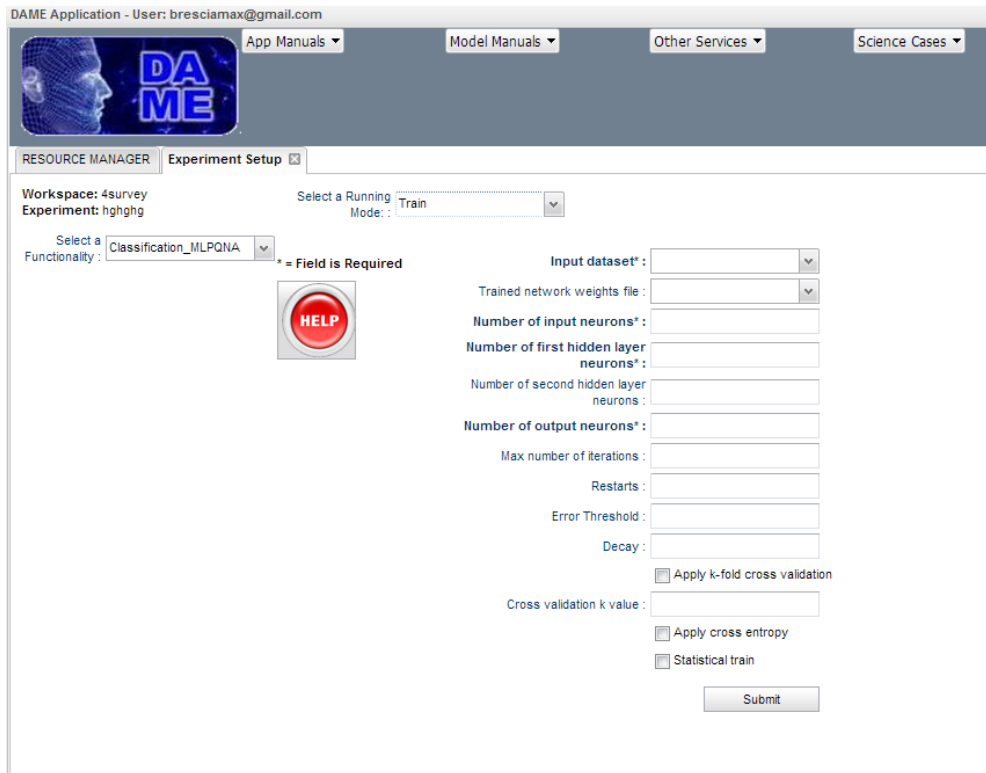


Fig. 6 – The setup tab for classification + MLPQNA train use case

In the case of Classification\_MLPQNA with Train use case, the help page is at the address:  
[http://dame.dsf.unina.it/mlpqna\\_help.html#class\\_train](http://dame.dsf.unina.it/mlpqna_help.html#class_train)

- **Input dataset**

**this parameter is a field required!**

This is the dataset file to be used as input for the learning phase of the model.

The file must be already uploaded by user and available in the current workspace of the experiment

It typically must include both input and target columns, where each row is an entire pattern (or sample of data). The format (hence its extension) must be one of the types allowed by the application (ASCII, FITS-table, CSV, VOTABLE). **More specifically, take in mind the following simple rule: the sum of input and output nodes MUST be equal to the total number of the columns in this file!**

- **Trained network weights file**

It is a file generated by the model during training phase. It contains the resulting network weights, associated to neurons, as stored at the end of a previous training session. Usually this file should not be edited or modified by users, just to preserve its content as generated by the model itself.



# DAta Mining & Exploration Program

The canonical use of this file in this use case is to resume a previous training phase, in order to try to improve it. If users leaves empty this parameter field, by default the current training session starts from scratch, (i.e. the initial weights are randomly generated).

- **Number of input neurons**

**this parameter is a field required!**

It is the number of neurons at the first (input) layer of the network. **It must exactly correspond to the number of input columns in the dataset input file (Training File field), except the target columns.**

- **Number of first hidden layer neurons**

**this parameter is a field required!**

It is the number of neurons of the first hidden layer of the network. As suggestion this should be selected in a range between a minimum of 1.5 times the number of input nodes and a maximum of 2 times + 1 the number of input nodes.

- **Number of second hidden layer neurons**

It is the number of neurons of the second hidden layer of the network. As suggestion this should be selected smaller than the previous layer.

For most experiments, this layer is considered redundant, except for particular complex cases. So use it only if really needed. Moreover some theoretical issues have demonstrated that one hidden layer only is sufficient to solve usual non linear classification problems.

By default the second hidden layer is empty (not used)

- **Number of output neurons**

**this parameter is a field required!**

It is the number of neurons in the output layer of the network. It must correspond to the number of target columns as contained in the dataset input file (filed Training File).

- **Max number of iterations**

One of the internal model parameters. It indicates the number of algorithm iterations, i.e. the maximum number of iterations for each approximation step of the Hessian inverse matrix. It is one of the stopping criteria.

By default the value is 1500

- **Restarts**

One of the internal model parameters. It indicates the number of restarts for each approximation step of the Hessian inverse matrix.





# DAta Mining & Exploration Program

By default the value is 20

- **Error Threshold**

One of the internal model parameters. It indicates the minimum weight error at each iteration step. Except for those problems particularly difficult to solve, in which a value of 0.0001 should be used, a value of 0.01 is usually considered sufficient.

By default the value is 0.01

- **Decay**

One of the internal model parameters. It indicates the weight regularization decay. If accurately chosen, the effect is an important improvement of the generalization error of the trained neural network, with also an acceleration of training.

This parameter is particularly important. In case of unknown value to choose, one can try with values within the range of 0.001 (weak case) up to 100 (very strong regularization). It should be noted that if the chosen value is too small (less than 0.001), it will be automatically increased up to the allowed minimum.

By default the value is 0.001

- **Apply k-fold cross validation**

This is a checkboxed parameter. If selected the validation of the training is activated, by following the k-fold method, i.e. based on an automatic procedure that splits in different subsets the training dataset, applying a k step cycle in which the training error is evaluated and its performances are validated.

If you select this option, you should specify the k value in the next parameter below. Take into account that, if selected, this option will statistically improve the training but the execution time will dramatically grow up.

By default the value is unselected.

- **Cross validation k value**

k-fold cross validation parameter that specify the value of k. Use it in combination with the previous checkbox.

By default the value is 10, but it is not used if the previous k-fold checkbox is unselected.

- **Apply cross entropy**

This is a checkboxed parameter. If selected it integrates a statistical optimization process in the classification training, also taking into account the linear softmax activation function associated to output neurons. Generally for complex problems, like classification in astrophysical contexts, it is strongly suggested to use such error minimization strategy.



# DAta Mining & Exploration Program

Please, consult the MLPQNA model manual for more information.

By default the value is unselected.

- **Statistical train**

This is a checkboxed parameter. If selected it engages a very time expensive cycle, in which the training set is split into a growing subset iteratively submitted to the model. This is useful only in case user wants to evaluate the training performances for several dimensions of input patterns.

By default the value is unselected. Use it very carefully and only if really required.

## 3.5 TEST Use case

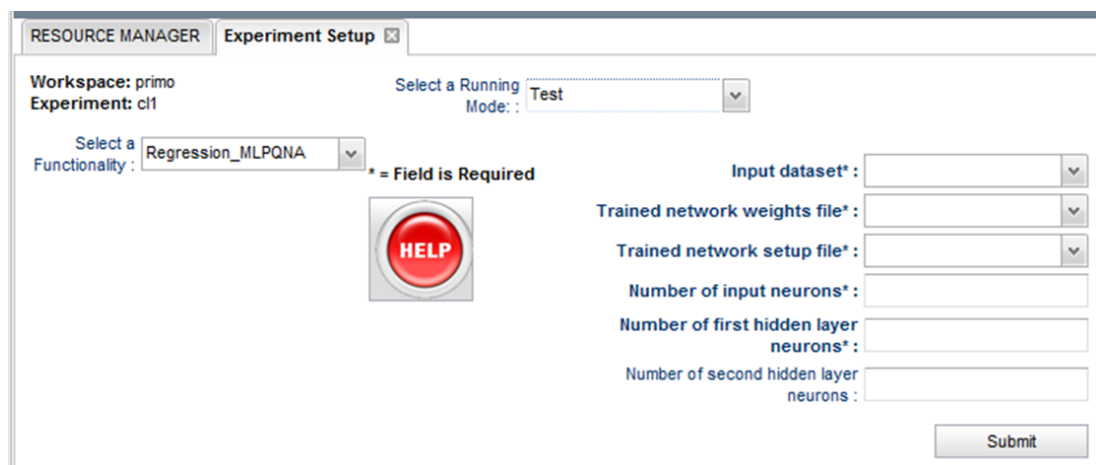
In the use case named “**Test**”, the software provides the possibility to test the MLPQNA. The user will be able to use already trained MLPQNA models, their weight configurations to execute the testing experiments.

In the experiment configuration there is also the Help button, redirecting to a web page dedicated to support the user with deep information about all parameters and their default values.

We remark that all parameters labeled by an asterisk are considered required. In all other cases the fields can be left empty (default values are used).

### 3.5.1 Regression with MLPQNA – Test Parameter Specifications

The setup of test use case for MLPQNA model, used for regression problems, is shown in Fig. 7.



**Fig. 7 – The setup tab for regression + MLPQNA test use case**



# DAta Mining & Exploration Program

In the case of Regression\_MLPQNA with Test use case, the help page is at the address:  
[http://dame.dsf.unina.it/mlpqna\\_help.html#regr\\_test](http://dame.dsf.unina.it/mlpqna_help.html#regr_test)

- **Input dataset**

**this parameter is a field required!**

This is the dataset file to be used as input for the test phase of the model.

The file must be already uploaded by user and available in the current workspace of the experiment. It can be the same input dataset already submitted during training phase.

It typically must include both input and target columns, where each row is an entire pattern (or sample of data). The format (hence its extension) must be one of the types allowed by the application (ASCII, FITS-table, CSV, VOTABLE). **More specifically, take in mind the following simple rule: the sum of input and output nodes MUST be equal to the total number of the columns in this file!**

- **Trained network weights file**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network weights, associated to neurons, as stored at the end of a previous training session. Usually this file should not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Trained network setup file**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network setup information, associated to neurons and other important internal parameters, as stored at the end of a previous training session. This file must not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Number of input neurons**

**this parameter is a field required!**

It is the number of neurons at the first (input) layer of the network. **It must exactly correspond to the number of input columns in the dataset input file (Training File field), except the target columns.**

- **Number of first hidden layer neurons**

**this parameter is a field required!**

It is the number of neurons of the first hidden layer of the network. **In the test phase the number must correspond to the exact number used in the training step. Otherwise the experiment will crash..**



# DAta Mining & Exploration Program

- **Number of second hidden layer neurons**

It is the number of neurons of the second hidden layer of the network. As suggestion this should be selected smaller than the previous layer.

For most experiments, this layer is considered redundant, except for particular complex cases. So use it only if really needed. Moreover some theoretical issues have demonstrated that one hidden layer only is sufficient to solve usual non linear regression problems.

**If used, in the test phase the number must correspond to the exact number used in the training step. Otherwise the experiment will crash.**

By default the second hidden layer is empty (not used)

## 3.5.2 Classification with MLPQNA – Test Parameter Specifications

The setup of test use case for MLPQNA model, used for classification problems, is shown in Fig. 8.

The screenshot shows the 'Experiment Setup' window. At the top, it says 'RESOURCE MANAGER Experiment Setup'. Below that, 'Workspace: primo' and 'Experiment: cl1'. There's a dropdown for 'Select a Running Mode:' set to 'Test'. Another dropdown for 'Select a Functionality:' is set to 'Classification\_MLPQNA'. A red 'HELP' button is in the center. To the right, there are several input fields: 'Input dataset\*' (with a dropdown arrow), 'Trained network setup file\*' (with a dropdown arrow), 'Trained network weights file\*' (with a dropdown arrow), 'Input neurons\*', 'Number of first hidden layer neurons\*', 'Number of second hidden layer neurons', and 'Number of output neurons\*'. A 'Submit' button is at the bottom right.

Fig. 8 – The setup tab for classification + MLPQNA test use case

In the case of Classification\_MLPQNA with Test use case, the help page is at the address:

[http://dame.dsf.unina.it/mlpqna\\_help.html#class\\_test](http://dame.dsf.unina.it/mlpqna_help.html#class_test)

- **Input dataset**

**this parameter is a field required!**

This is the dataset file to be used as input for the test phase of the model.

The file must be already uploaded by user and available in the current workspace of the experiment. It can be the same input dataset already submitted during training phase.

It typically must include both input and target columns, where each row is an entire pattern (or sample of data). The format (hence its extension) must be one of the types allowed by the application



# DAta Mining & Exploration Program

(ASCII, FITS-table, CSV, VOTABLE). **More specifically, take in mind the following simple rule: the sum of input and output nodes MUST be equal to the total number of the columns in this file!**

- **Trained network setup file**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network setup information, associated to neurons and other important internal parameters, as stored at the end of a previous training session. This file must not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Trained network weights file**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network weights, associated to neurons, as stored at the end of a previous training session. Usually this file should not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Number of input neurons**

**this parameter is a field required!**

It is the number of neurons at the first (input) layer of the network. **It must exactly correspond to the number of input columns in the dataset input file (Training File field), except the target columns.**

- **Number of first hidden layer neurons**

**this parameter is a field required!**

It is the number of neurons of the first hidden layer of the network. **In the test phase the number must correspond to the exact number used in the training step. Otherwise the experiment will crash..**

- **Number of second hidden layer neurons**

It is the number of neurons of the second hidden layer of the network. As suggestion this should be selected smaller than the previous layer.

For most experiments, this layer is considered redundant, except for particular complex cases. So use it only if really needed. Moreover some theoretical issues have demonstrated that one hidden layer only is sufficient to solve usual non linear classification problems.

**If used, in the test phase the number must correspond to the exact number used in the training step. Otherwise the experiment will crash.**

By default the second hidden layer is empty (not used)



# DAta Mining & Exploration Program

- Number of output neurons

**this parameter is a field required!**

It is the number of neurons in the output layer of the network. **In the test phase the number must correspond to the exact number used in the training step. Otherwise the experiment will crash.**

## 3.6 Run Use case

In the use case named “**Run**”, the software provides the possibility to run the MLPQNA. The user will be able to use already trained and tested MLPQNA models, their weight configurations, to execute the normal experiments on new datasets.

In the experiment configuration there is also the Help button, redirecting to a web page dedicated to support the user with deep information about all parameters and their default values.

We remark that all parameters labeled by an asterisk are considered required. In all other cases the fields can be left empty (default values are used).

### 3.6.1 Regression with MLPQNA – Run Parameter Specifications

The setup of run use case for MLPQNA model, used for regression problems, is shown in Fig. 9.

The screenshot shows the 'Experiment Setup' tab in the software. It features a 'Workspace: primo' and 'Experiment: cl1' header. Below this, there are two dropdown menus: 'Select a Running Mode' (set to 'Run') and 'Select a Functionality' (set to 'Regression\_MLPQNA'). A red 'HELP' button is prominently displayed. To the right, several input fields are listed, each with an asterisk indicating it is required: 'Trained network setup file\*', 'Trained network weights file\*', 'Input dataset\*', 'Number of input neurons\*', 'Number of first hidden layer neurons\*', and 'Number of second hidden layer neurons'. A 'Submit' button is located at the bottom right of the form.

**Fig. 9 – The setup tab for regression + MLPQNA run use case**

In the case of Regression\_MLPQNA with Run use case, the help page is at the address:  
[http://dame.dsf.unina.it/mlpqna\\_help.html#regr\\_run](http://dame.dsf.unina.it/mlpqna_help.html#regr_run)

- Trained network setup file

**this parameter is a field required!**



# DAta Mining & Exploration Program

It is a file generated by the model during training phase. It contains the resulting network setup information, associated to neurons and other important internal parameters, as stored at the end of a previous training session. This file must not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Trained network weights file**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network weights, associated to neurons, as stored at the end of a previous training session. Usually this file should not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Input dataset**

**this parameter is a field required!**

This is the dataset file to be used as input for the run phase of the model.

The file must be already uploaded by user and available in the current workspace of the experiment.

The format (hence its extension) must be one of the types allowed by the application (ASCII, FITS-table, CSV, VOTABLE).

- **Number of input neurons**

**this parameter is a field required!**

It is the number of neurons at the first (input) layer of the network. **It must exactly correspond to the number of input columns in the dataset input file.**

- **Number of first hidden layer neurons**

**this parameter is a field required!**

It is the number of neurons of the first hidden layer of the network. **In the run phase the number must correspond to the exact number used in the training/test step. Otherwise the experiment will crash..**

- **Number of second hidden layer neurons**

It is the number of neurons of the second hidden layer of the network. As suggestion this should be selected smaller than the previous layer.

For most experiments, this layer is considered redundant, except for particular complex cases. So use it only if really needed. Moreover some theoretical issues have demonstrated that one hidden layer only is sufficient to solve usual non linear regression problems.

**If used, in the run phase the number must correspond to the exact number used in the training/test step. Otherwise the experiment will crash.**



# DAta Mining & Exploration Program

By default the second hidden layer is empty (not used)

## 3.6.2 Classification with MLPQNA – Run Parameter Specifications

The setup of run use case for MLPQNA model, used for classification problems, is shown in Fig. 10.

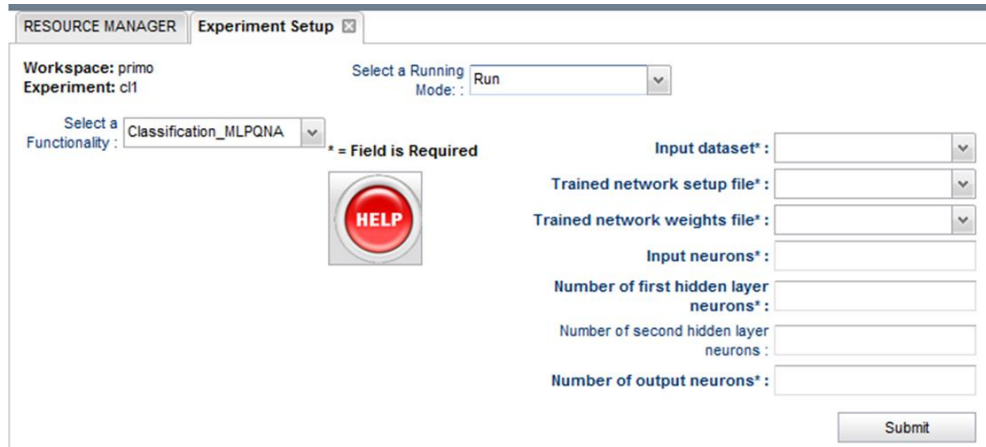


Fig. 10 – The setup tab for classification + MLPQNA run use case

In the case of Classification\_MLPQNA with Run use case, the help page is at the address:  
[http://dame.dsf.unina.it/mlpqna\\_help.html#class\\_run](http://dame.dsf.unina.it/mlpqna_help.html#class_run)

- **Input dataset**

**this parameter is a field required!**

This is the dataset file to be used as input for the run phase of the model.

The file must be already uploaded by user and available in the current workspace of the experiment.

The format (hence its extension) must be one of the types allowed by the application (ASCII, FITS-table, CSV, VOTABLE).

- **Trained network setup file**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network setup information, associated to neurons and other important internal parameters, as stored at the end of a previous training session. This file must not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Trained network weights file**

**this parameter is a field required!**





# DAta Mining & Exploration Program

It is a file generated by the model during training phase. It contains the resulting network weights, associated to neurons, as stored at the end of a previous training session. Usually this file should not be edited or modified by users, just to preserve its content as generated by the model itself.

- **Number of input neurons**

**this parameter is a field required!**

It is the number of neurons at the first (input) layer of the network. **It must exactly correspond to the number of input columns in the dataset input file.**

- **Number of first hidden layer neurons**

**this parameter is a field required!**

It is the number of neurons of the first hidden layer of the network. **In the run phase the number must correspond to the exact number used in the training/test step. Otherwise the experiment will crash..**

- **Number of second hidden layer neurons**

It is the number of neurons of the second hidden layer of the network. As suggestion this should be selected smaller than the previous layer.

For most experiments, this layer is considered redundant, except for particular complex cases. So use it only if really needed. Moreover some theoretical issues have demonstrated that one hidden layer only is sufficient to solve usual non linear classification problems.

**If used, in the run phase the number must correspond to the exact number used in the training/test step. Otherwise the experiment will crash.**

By default the second hidden layer is empty (not used)

- **Number of output neurons**

**this parameter is a field required!**

It is the number of neurons in the output layer of the network. **In the run phase the number must correspond to the exact number used in the training/test step. Otherwise the experiment will crash.**



# DAta Mining & Exploration Program

## 4 Examples

This section is dedicated to show some practical examples of the correct use of the web application. Not all aspects and available options are reported, but a significant sample of features useful for beginners of DAME suite and with a poor experience about data mining methodologies with machine learning algorithms. In order to do so, very simple and trivial problems will be described. Further complex examples will be integrated here in the next releases of the documentation.

### 4.1 Regression XOR problem

The problem can be stated as follows: we want to train a model to learn the logical XOR function between two binary variables. As known, the XOR problem is not a linearly separable problem, so we require to obtain a neural network able to learn to identify the right output value of the XOR function, having a BoK made by possible combinations of two input variable and related correct output. This is a very trivial problem and in principle it should not be needed any machine learning method. But as remarked, the scope is not to obtain a scientific benefit, but to make practice with the web application. Let say, it is an example comparable with the classical “print <Hello World> on standard output” implementation problem for beginners in C language.

As first case, we will use the MLPQNA model associated to the regression functionality.

The starting point is to create a new workspace, named **mlpqnaExp** and to populate it by uploading two files:

- **xor.csv**: CSV dataset file for training and test use cases;
- **xor\_run.csv**: CSV dataset file for run use case;

Their content description is already described in section 3 of this document.

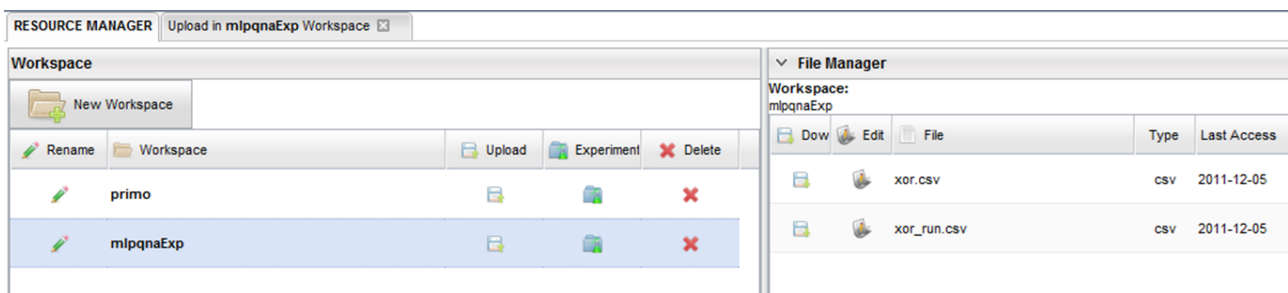


Fig. 11 – The starting point, with a Workspace (mlpqnaExp) created and two data files uploaded

#### 4.1.1 Regression MLPQNA – Train use case

Let suppose we create an experiment named **XorTrain** and we want to configure it. After creation, the new configuration tab is open. Here we select **Regression\_MLPQNA** as couple functionality-model of the current experiment and we select also **Train** as use case.



# DAta Mining & Exploration Program

RESOURCE MANAGER Upload in mlpqnaExp Workspace Experiment Setup

Workspace: mlpqnaExp Experiment: XorTrain Select a Running Mode: Train

Select a Functionality: Regression\_MLPQNA \*- Field is Required

Input dataset\*: /xor.csv

Trained network weights file:

Number of input neurons\*: 2

Number of first hidden layer neurons\*: 5

Number of second hidden layer neurons:

Max number of iterations:

Restarts:

Minimum weight:

Decay:

Apply k-fold cross validation

Cross validation k value:

Statistical train

Submit

Fig. 12 – The xorTrain experiment configuration tab

Now we have to configure parameters for the experiment. In particular, we will leave empty the not required fields (labels without asterisk).

The meaning of the parameters for this use case are described in section 3.1.1 of this document. As alternative, you can click on the Help button to obtain detailed parameter description and their default values directly from the webapp.

We give xor.csv as training dataset, specifying:

- **Number of input nodes:** 2, because 2 are the input columns in the file;
- **Number of hidden nodes (first level):** 2, as minimal number of hidden nodes (no particularly complex network brain is required to solve the XOR problem). Anyway, we suggest to try with different numbers of such nodes, by gradually incrementing them, to see what happens in terms of training error and convergence speed;
- **Number of output nodes:** 1, because the third column in the input file is the target (correct output for input patterns);

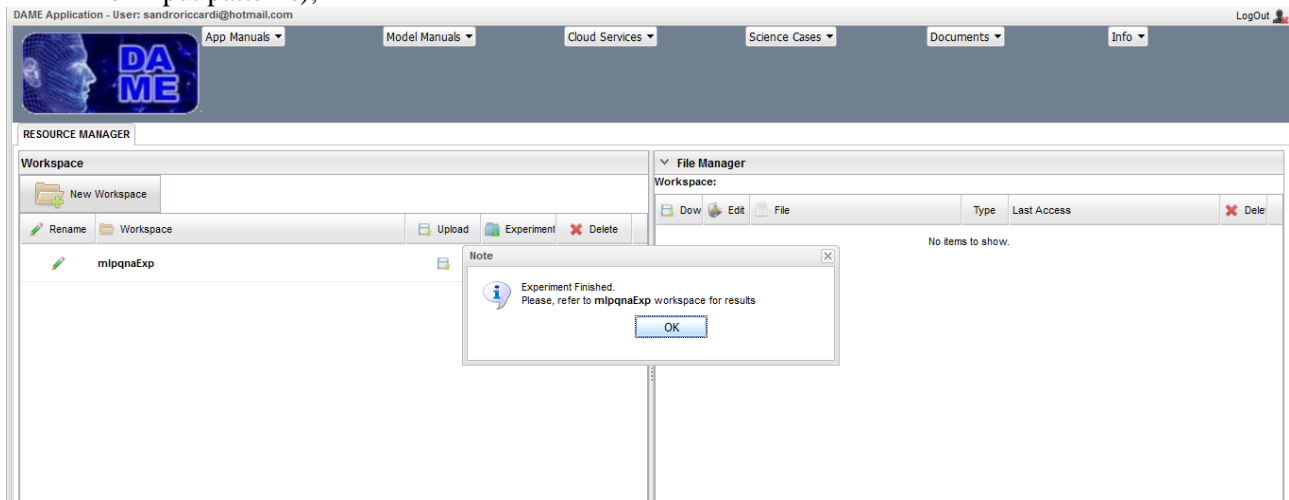


Fig. 13 – The xorTrain experiment status after submission



# DAta Mining & Exploration Program

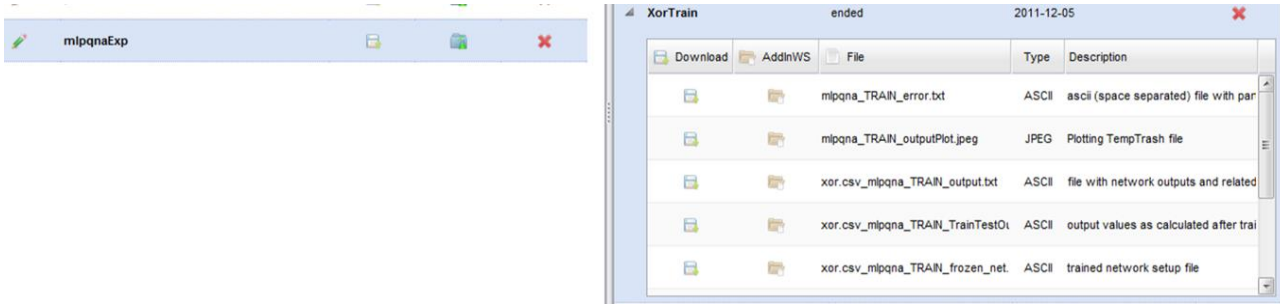


Fig. 14 – The xorTrain experiment output files

The content of output files, obtained at the end of the experiment (available when the status is “ended”) is shown in the following (note that in the training phase the file **train\_out** is not much relevant). The file **error** reports the training error after a set of iterations indicated in the first column (the error is the MSE of the difference between network output and the target values).

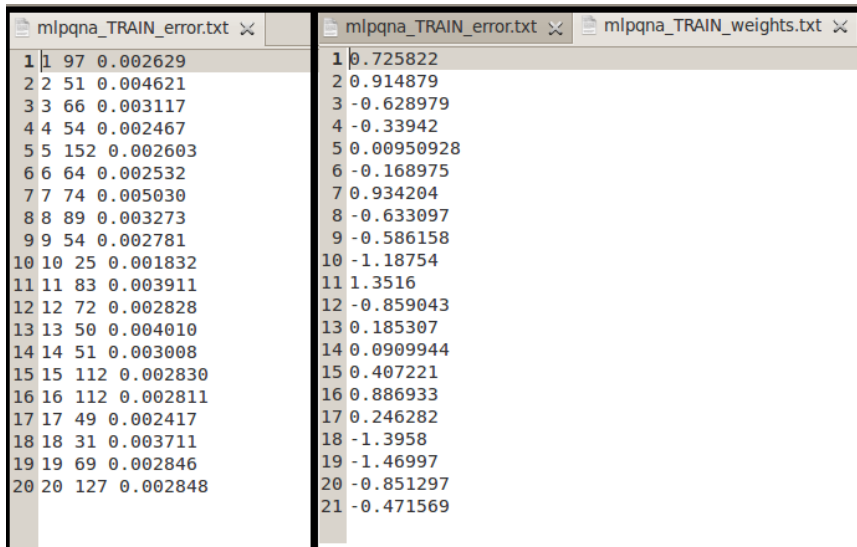


Fig. 15 – The files error (left) and weights (right) output of the xorTrain experiment

## 4.1.2 Regression MLPQNA – Test use case

The file **weights** and **frozen\_train\_net** can be copied into the input file area (**File Manager**) of the workspace, in order to be re-used in future experiments (for example in this case the test use case).



# DAta Mining & Exploration Program

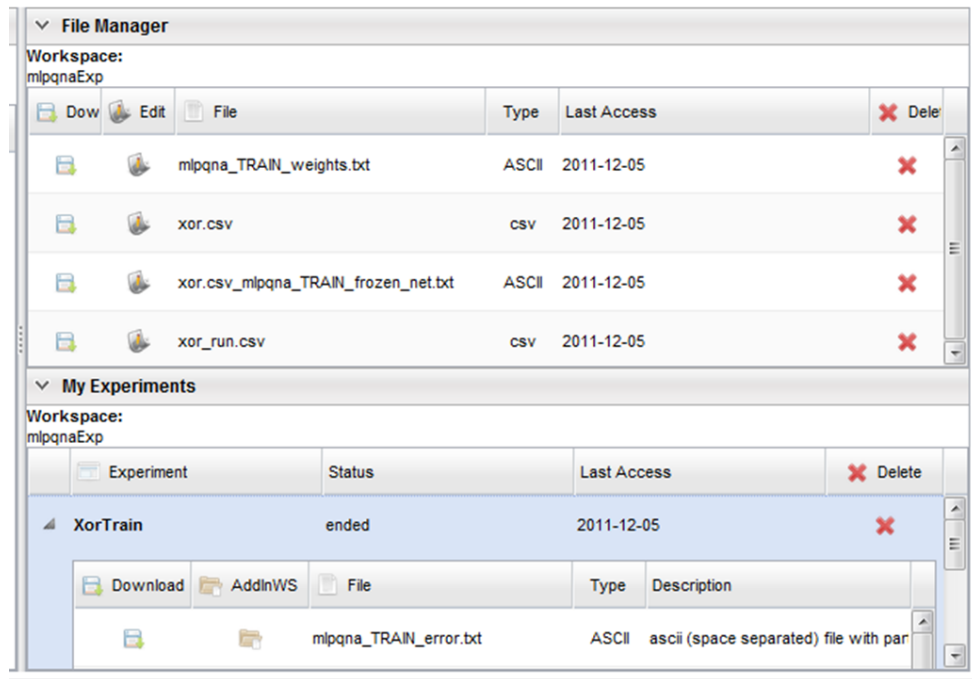


Fig. 16 – The file “weights” and “frozen\_train\_net” copied in the WS input file area for next purposes

So far, we proceed to create a new experiment, named **xorTest**, to verify the training of the network. For simplicity we will re-use the same input dataset (file xor.csv) but in general, the user could use another dataset, uploaded from scratch or extracted from the original training dataset, through file editing options.

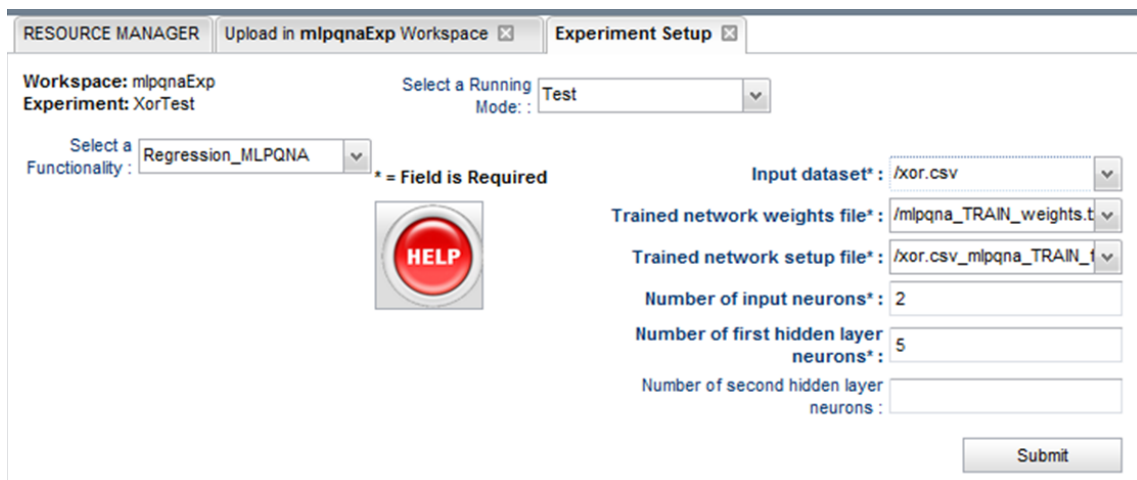


Fig. 17 – The xorTest experiment configuration tab (note “weights” file and frozen\_train\_net file inserted)

After execution, the experiment **xorTest** will show the output files available.



# DAta Mining & Exploration Program

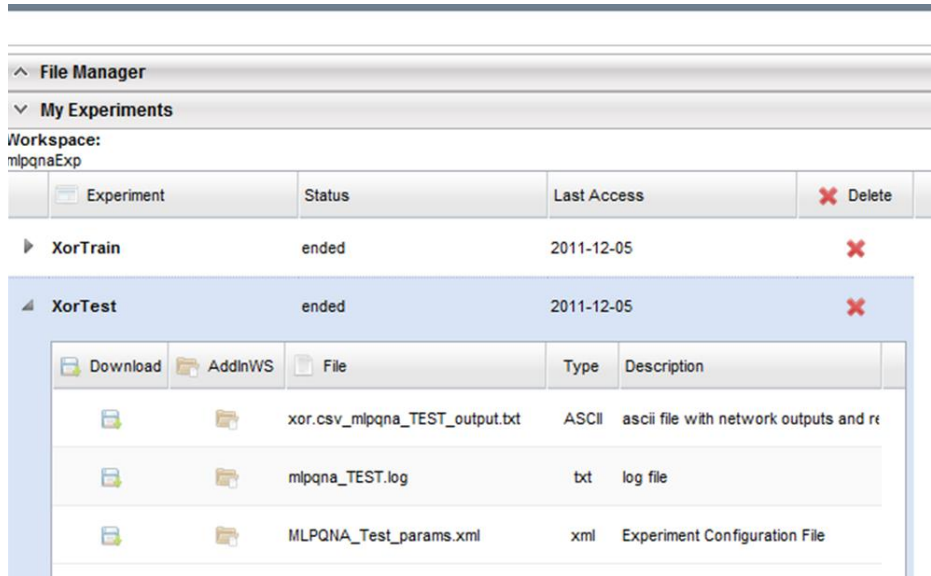


Fig. 18 – The xorTest experiment output files



# DAta Mining & Exploration Program

## 5 Appendix – References and Acronyms

### Abbreviations & Acronyms

A & A	Meaning	A & A	Meaning
AI	Artificial Intelligence	KDD	Knowledge Discovery in Databases
ANN	Artificial Neural Network	IEEE	Institute of Electrical and Electronic Engineers
ARFF	Attribute Relation File Format	INAF	Istituto Nazionale di Astrofisica
ASCII	American Standard Code for Information Interchange	JPEG	Joint Photographic Experts Group
BoK	Base of Knowledge	LAR	Layered Application Architecture
BP	Back Propagation	MDS	Massive Data Sets
BLL	Business Logic Layer	MLC	Multi Layer Clustering
CE	Cross Entropy	MLP	Multi Layer Perceptron
CSOM	Clustering SOM	MSE	Mean Square Error
CSV	Comma Separated Values	NN	Neural Network
DAL	Data Access Layer	OAC	Osservatorio Astronomico di Capodimonte
DAME	DAta Mining & Exploration	PC	Personal Computer
DAMEWARE	DAME Web Application REsource	PI	Principal Investigator
DAPL	Data Access & Process Layer	REDB	Registry & Database
DL	Data Layer	RIA	Rich Internet Application
DM	Data Mining	SDSS	Sloan Digital Sky Survey
DMM	Data Mining Model	SL	Service Layer
DMS	Data Mining Suite	SOFM	Self Organizing Feature Map
FITS	Flexible Image Transport System	SOM	Self Organizing Map
FL	Frontend Layer	SW	Software
FW	FrameWork	UI	User Interface
GRID	Global Resource Information Database	URI	Uniform Resource Indicator
GSOM	Gated SOM	VO	Virtual Observatory
GUI	Graphical User Interface	XML	eXtensible Markup Language
HW	Hardware		

**Tab. 3 – Abbreviations and acronyms**



# DAta Mining & Exploration Program

## Reference & Applicable Documents

ID	Title / Code	Author	Date
R1	“The Use of Multiple Measurements in Taxonomic Problems”, in Annals of Eugenics, 7, p. 179--188	Ronald Fisher	1936
R2	<i>Neural Networks for Pattern Recognition</i> . Oxford University Press, GB	Bishop, C. M.	1995
R3	<i>Neural Computation</i>	Bishop, C. M., Svensen, M. & Williams, C. K. I.	1998
R4	Data Mining Introductory and Advanced Topics, Prentice-Hall	Dunham, M.	2002
R5	<i>The Fourth Paradigm</i> . Microsoft research, Redmond Washington, USA	Hey, T. et al.	2009
R6	Artificial Intelligence, A modern Approach. Second ed. (Prentice Hall)	Russell, S., Norvig, P.	2003
R7	Neural Networks - A comprehensive Foundation, Second Edition, Prentice Hall	Haykin, S.,	1999
R8	<i>A practical application of simulated annealing to clustering</i> . Pattern Recognition 25(4): 401-412	Donald E. Brown D.E., Huntley, C. L.:	1991
R9	Mathematical Background on MLPQNA model <a href="http://dame.dsf.unina.it/machine_learning.html#mlpqna">http://dame.dsf.unina.it/machine_learning.html#mlpqna</a>	M. Brescia	2012
R10	The detection of globular clusters in galaxies as a data mining problem. MNRAS, Volume 421, Issue 2, pp. 1155-1165 <a href="http://adsabs.harvard.edu/abs/2012MNRAS.421.1155B">http://adsabs.harvard.edu/abs/2012MNRAS.421.1155B</a>	Brescia et al.	2012
R11	Photometric redshifts with the quasi Newton algorithm (MLPQNA) Results in the PHAT1 contest. Astronomy & Astrophysics, Volume 546, id.A13, 8 pp <a href="http://adsabs.harvard.edu/abs/2012A%26A...546A..13C">http://adsabs.harvard.edu/abs/2012A%26A...546A..13C</a>	Cavuoti et al.	2012
R12	Photometric redshifts for Quasars in multi band Surveys. ApJ (in press). <a href="http://adsabs.harvard.edu/abs/2013arXiv1305.5641B">http://adsabs.harvard.edu/abs/2013arXiv1305.5641B</a>	Brescia et al.	2013

**Tab. 4 – Reference Documents**





# DAta Mining & Exploration Program

ID	Title / Code	Author	Date
A1	SuiteDesign_VONEURAL-PDD-NA-0001-Rel2.0	DAME Working Group	15/10/2008
A2	project_plan_VONEURAL-PLA-NA-0001-Rel2.0	Brescia	19/02/2008
A3	statement_of_work_VONEURAL-SOW-NA-0001-Rel1.0	Brescia	30/05/2007
A4	mlpGP_DAME-MAN-NA-0008-Rel2.0	Brescia	04/04/2011
A5	pipeline_test_VONEURAL-PRO-NA-0001-Rel.1.0	D'Abrusco	17/07/2007
A6	scientific_example_VONEURAL-PRO-NA-0002-Rel.1.1	D'Abrusco/Cavuoti	06/10/2007
A7	frontend_VONEURAL-SDD-NA-0004-Rel1.4	Manna	18/03/2009
A8	FW_VONEURAL-SDD-NA-0005-Rel2.0	Fiore	14/04/2010
A9	REDB_VONEURAL-SDD-NA-0006-Rel1.5	Nocella	29/03/2010
A10	driver_VONEURAL-SDD-NA-0007-Rel0.6	d'Angelo	03/06/2009
A11	dm-model_VONEURAL-SDD-NA-0008-Rel2.0	Cavuoti/Di Guido	22/03/2010
A12	ConfusionMatrixLib_VONEURAL-SPE-NA-0001-Rel1.0	Cavuoti	07/07/2007
A13	softmax_entropy_VONEURAL-SPE-NA-0004-Rel1.0	Skordovski	02/10/2007
A14	MLPQNA_DAME-SRS-NA-0009-Rel_1.0	Riccardi, Brescia	09/02/2011
A15	dm_model_VONEURAL-SRS-NA-0005-Rel0.4	Cavuoti	05/01/2009
A16	MLPQNA_DAME-SDD-NA-0015-Rel_1.0	Riccardi, Brescia	02/06/2011
A17	DMPlugins_DAME-TRE-NA-0016-Rel0.3	Di Guido, Brescia	14/04/2010
A18	BetaRelease_ReferenceGuide_DAME-MAN-NA-0009-Rel1.0	Brescia	28/10/2010
A19	BetaRelease_GUI_UserManual_DAME-MAN-NA-0010-Rel1.0	Brescia	03/12/2010

**Tab. 5 – Applicable Documents**



# DAta Mining & Exploration Program

\_\_oOo\_\_



# DAta Mining & Exploration Program



*DAME Program*  
*“we make science discovery happen”*

