



# 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 with Back Propagation*

### *User Manual*

DAME-MAN-NA-0011

Issue: 1.3  
Date: September 03, 2013  
Author: S. Cavuoti, M. Brescia

Doc. : MLPBP\_UserManual\_DAME-MAN-NA-0011-Rel1.3





# DAta Mining & Exploration Program

## INDEX

1	Introduction .....	4
2	MLP Model Theoretical Overview .....	5
2.1	Multi Layer Perceptron .....	5
2.1.1	The training performance evaluation criteria.....	9
2.1.1.1	The Mean Square Error .....	9
2.1.1.2	Cross Entropy for the two-class case.....	9
2.1.1.3	Cross Entropy for the multiple-class case .....	11
2.2	The Back Propagation learning rule.....	12
2.3	MLP Practical Rules .....	14
2.3.1	Selection of neuron activation function.....	14
2.3.2	Scaling input and target values .....	14
2.3.3	Number of hidden nodes.....	15
2.3.4	Number of hidden layers.....	15
3	Use of the web application model .....	16
3.1	Use Cases .....	16
3.2	Input .....	17
3.3	Output .....	18
3.4	TRAIN Use case .....	19
3.4.1	Regression with MLP – Train Parameter Specifications .....	19
3.4.2	Classification with MLP – Train Parameter Specifications.....	21
3.5	TEST Use case .....	23
3.5.1	Regression with MLP – Test Parameter Specifications.....	23
3.5.2	Classification with MLP – Test Parameter Specifications .....	24
3.6	Run Use case.....	25
3.6.1	Regression with MLP – Run Parameter Specifications .....	25
3.6.2	Classification with MLP – Run Parameter Specifications.....	25
3.7	Full Use case .....	26
3.7.1	Regression with MLP – Full Parameter Specifications .....	26
3.7.2	Classification with MLP – Full Parameter Specifications.....	28
4	Examples .....	32
4.1	Classification XOR problem.....	32
4.1.1	Classification MLP – Train use case .....	33
4.1.2	Classification MLP – Test use case .....	35
4.1.3	Classification MLP – Full use case.....	37
5	Appendix – References and Acronyms .....	39

## TABLE INDEX

<i>Tab. 1</i>	<i>– output file list in case of regression type experiments.....</i>	<i>18</i>
<i>Tab. 2</i>	<i>– output file list in case of classification type experiments .....</i>	<i>18</i>
<i>Tab. 3</i>	<i>– Abbreviations and acronyms.....</i>	<i>39</i>
<i>Tab. 4</i>	<i>– Reference Documents .....</i>	<i>40</i>
<i>Tab. 5</i>	<i>– Applicable Documents.....</i>	<i>41</i>



# DAta Mining & Exploration Program

## FIGURE INDEX

<i>Fig. 1 – the MLP artificial and biologic brains .....</i>	<i>5</i>
<i>Fig. 2 – the sigmoid function.....</i>	<i>6</i>
<i>Fig. 3 – Example of a SLP to calculate the logic AND operation.....</i>	<i>7</i>
<i>Fig. 4 – A MLP able to calculate the logic XOR operation .....</i>	<i>8</i>
<i>Fig. 5 – The typical flow structure of the Back Propagation algorithm.....</i>	<i>13</i>
<i>Fig. 6 – The sigmoid function and its first derivative.....</i>	<i>14</i>
<i>Fig. 7 – The content of the xor.csv file used as input for training/test use cases .....</i>	<i>17</i>
<i>Fig. 8 – The content of the xor_run.csv file used as input for Run use case .....</i>	<i>17</i>
<i>Fig. 9 – The starting point, with a Workspace (mlpExp) created and two data files uploaded .....</i>	<i>32</i>
<i>Fig. 10 – The xorTrain experiment configuration tab.....</i>	<i>33</i>
<i>Fig. 11 – The xorTrain experiment status after submission .....</i>	<i>34</i>
<i>Fig. 12 – The xorTrain experiment output files.....</i>	<i>34</i>
<i>Fig. 13 – The files .csv (left) and .mlp (right) output of the xorTrain experiment.....</i>	<i>35</i>
<i>Fig. 14 – The file “mlp_TRAIN_weights.mlp” copied in the WS input file area for next purposes.....</i>	<i>35</i>
<i>Fig. 15 – The xorTest experiment configuration tab (note “weights” file inserted).....</i>	<i>36</i>
<i>Fig. 16 – The xorTest experiment output files.....</i>	<i>36</i>
<i>Fig. 17 – The xorFull experiment configuration tab.....</i>	<i>37</i>
<i>Fig. 18 – The xorFull experiment output.....</i>	<i>38</i>



# DAta Mining & Exploration Program

## 1 Introduction

**T**he present document is the user guide of the data mining model MLP (Multi Layer Perceptron trained by Back Propagation), as implemented and integrated into the DAMEWARE. 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 model, 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...*!)



# Data Mining & Exploration Program

## 2 MLP Model Theoretical Overview

This paragraph is intended to furnish a theoretical overview of the MLP model, associated to single or multiple functionality domains, in order to be used to perform practical scientific experiments with such techniques. An overview of machine learning and functionality domains, as intended in DAME Project can be found in [A18].

### 2.1 Multi Layer Perceptron

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).

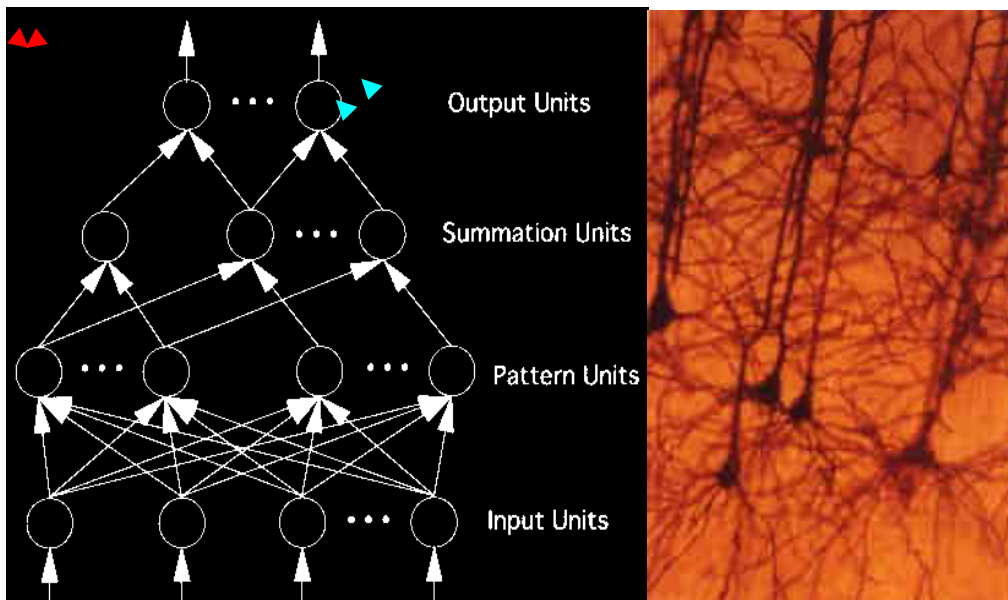
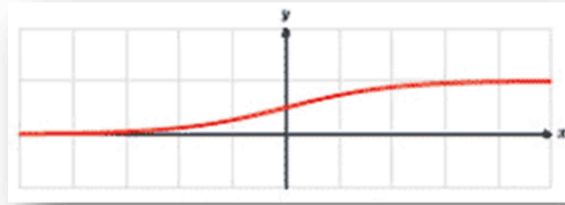


Fig. 1 – the MLP artificial and biologic brains

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*, or *softmax* (or a customized function ad hoc for the specific application).

## Sigmoid function

$$y = 1/(1+e^{(-x)})$$



**Fig. 2 – the sigmoid function**

This function is the most frequent in the MLP model. It is characterized by its smooth step between 0 and 1 with a variable threshold. But this restricted domain [0, 1] is also its limitation. It in fact can be used only in problems where expected outputs are numbers in this range.

## Softmax

In order to ensure that the outputs can be interpreted as posterior probabilities, they must be comprised between zero and one, and their sum must be equal to one. This constraint also ensures that the distribution is correctly normalized. In practice this is, for multi-class problems, achieved by using a softmax activation function in the output layer. The purpose of the softmax activation function is to enforce these constraints on the outputs. Let the network input to each output unit be  $q_i$ , with  $i = 1...c$ , where  $c$  is the number of categories. Then the softmax output is:

$$p_i = \frac{\exp(q_i)}{\sum_{j=1}^c \exp(q_j)} \quad (3)$$

Statisticians usually call softmax a "multiple logistic" function. Equation (3) is also known as normalized exponential function. It reduces to the simple logistic function when there are only two categories. Suppose you choose to set  $q_2$  to 0.

$$p_1 = \frac{\exp(q_1)}{\sum_{j=1}^c \exp(q_j)} = \frac{\exp(q_1)}{\exp(q_1) - \exp(0)} = \frac{1}{1 + \exp(-q_1)} \quad (4)$$

The term softmax is used because this activation function represents a smooth version of the winner-takes-all activation model in which the unit with the largest input has output +1 while all other units have output 0. The softmax function is also used in the hidden layer of normalized radial-basis-function networks, but in the interest of this document we would not enter into their description. To use the softmax activation function you need at least 2 columns of targets (1-N codified).

The base of the MLP is the **Perceptron**, a type of artificial neural network invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt. It can be seen as the simplest kind of feedforward neural network: a linear classifier. The Perceptron is a binary classifier which maps its input  $x$  (a real-valued vector) to an output value  $f(x)$  (a single binary value) across the matrix.

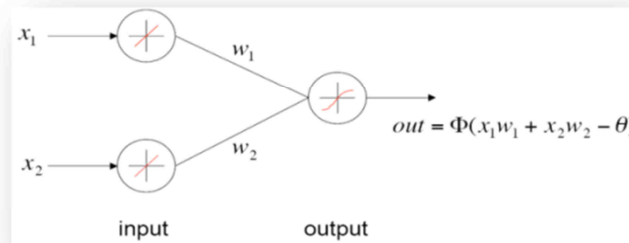


$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

where  $w$  is a vector of real-valued weights and  $w \cdot x$  is the dot product (which computes a weighted sum).  $b$  is the 'bias', a constant term that does not depend on any input value. The value of  $f(x)$  (0 or 1) is used to classify  $x$  as either a positive or a negative instance, in the case of a binary classification problem. If  $b$  is negative, then the weighted combination of inputs must produce a positive value greater than  $|b|$  in order to push the classifier neuron over the 0 threshold. Spatially, the bias alters the position (though not the orientation) of the decision boundary.

The Perceptron learning algorithm does not terminate if the learning set is not linearly separable. The Perceptron is considered the simplest kind of feed-forward neural network.

The earliest kind of neural network is a *Single Layer Perceptron* (SLP) network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1).

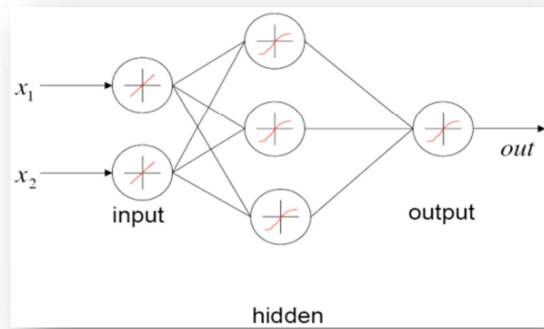


**Fig. 3 – Example of a SLP to calculate the logic AND operation**

Neurons with this kind of activation function are also called *artificial neurons* or *linear threshold units*, as described by Warren McCulloch and Walter Pitts in the 1940s.

A Perceptron can be created using any values for the activated and deactivated states as long as the threshold value lies between the two. Most perceptrons have outputs of 1 or -1 with a threshold of 0 and there is some evidence that such networks can be trained more quickly than networks created from nodes with different activation and deactivation values. SLPs are only capable of learning linearly separable patterns. In 1969 in a famous monograph entitled *Perceptrons* Marvin Minsky and Seymour Papert showed that it was impossible for a single-layer Perceptron network to learn an XOR function.

Although a single threshold unit is quite limited in its computational power, it has been shown that networks of parallel threshold units can approximate any continuous function from a compact interval of the real numbers into the interval [-1,1]. So far, it was introduced the model Multi Layer Perceptron.



**Fig. 4 – A MLP able to calculate the logic XOR operation**

This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a continuous activation function.

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.

The *universal approximation theorem* [R12] for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer Perceptron with just one hidden layer. This result holds only for restricted classes of activation functions, e.g. for the sigmoidal functions.

An extension of the universal approximation theorem states that the two layers architecture is capable of universal approximation and a considerable number of papers have appeared in the literature discussing this property. An important corollary of these results is that, in the context of a classification problem, networks with sigmoidal non-linearity and two layer of weights can approximate any decision boundary to arbitrary accuracy. Thus, such networks also provide universal non-linear discriminate functions. More generally, the capability of such networks to approximate general smooth functions allows them to model posterior probabilities of class membership. Since two layers of weights suffice to implement any arbitrary function, one would need special problem conditions, or requirements to recommend the use of more than two layers. Furthermore, it is found empirically that networks with multiple hidden layers are more prone to getting caught in undesirable local minima.

Astronomical data do not seem to require such level of complexity and therefore it is enough to use just a double weights layer, i.e. a single hidden layer.

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.

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 (usually the euclidean distance is used). 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.





# DAta Mining & Exploration Program

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. 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 be always taken 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).

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.

## 2.1.1 The training performance evaluation criteria

For the model MLP trained by Back Propagation as implemented in DAME, there is possible choice between two error evaluation criteria, respectively the MSE (Mean Square Error) between target and network output values and the Cross Entropy.

### 2.1.1.1 The Mean Square Error

Given the  $p$ -th pattern in input, a classical error function (called sum-of-squares) is:

$$E_p = \frac{1}{2} \sum_j (t_j^p - y_j^p)^2$$

Where  $t_j^p$  is the  $p$ -th desired output value and  $y_j^p$  is the output of the corresponding neuron. Due to its interpolation capabilities, the MLP is one of the most widely used neural architectures.

### 2.1.1.2 Cross Entropy for the two-class case

Learning in the neural networks is based on the definition of a suitable error function, which is then minimized with respect to the weights and biases in the network. Error functions play an important role in the use of neural networks. A variety of different error functions exist.

For regression problems the basic goal is to model the conditional distribution of the output variables, conditioned on the input variables. This motivates the use of a sum-of-squares error function. But for classification problems the sum-of-squares error function is not the most appropriate choice. In the case of a 1-of-C coding scheme, the target values sum to unity for each pattern and so the network outputs will also always sum to unity. However, there is no guarantee that they will lie in the range (0,1).

In fact, the outputs of the network trained by minimizing a sum-of-squares error function approximate the posterior probabilities of class membership, conditioned on the input vector, using the maximum likelihood principle by assuming that the target data was generated from a smooth deterministic function with added



# DAta Mining & Exploration Program

Gaussian noise. For classification problems, however, the targets are binary variables and hence far from having a Gaussian distribution, so their description cannot be given by using Gaussian noise model.

Therefore a more appropriate choice of error function is needed.

Let us now consider problems involving two classes. One approach to such problems would be to use a network with two output units, one for each class. First let's discuss an alternative approach in which we consider a network with a single output  $y$ . We would like the value of  $y$  to represent the posterior probability  $P(C_1|x)$  for class  $C_1$ . The posterior probability of class  $C_2$  will then given by  $P(C_2|x) = 1-y$ . This can be achieved if we consider a target coding scheme for which  $t = 1$  if the input vector belongs to class  $C_1$  and  $t = 0$  if it belongs to class  $C_2$ . We can combine these into a single expression, so that the probability of observing either target value is

$$p(t|x) = y^t(1-y)^{1-t} \quad (5)$$

This equation is the equation for a binomial distribution known as Bernoulli distribution. With this, interpretation of the output unit activations, the likelihood of observing the training data set, assuming the data points are drawn independently from this distribution, is then given by

$$\prod_n (y^n)^{t^n} (1-y^n)^{1-t^n} \quad (6)$$

By minimizing the negative logarithm of the likelihood we get to the cross-entropy error function (Hopfield, 1987; Baum and Wilczek, 1988; Solla et al., 1988; Hinton, 1989; Hampshire and Pearlmutter, 1990) in the form

$$E = - \sum \{ t^n \ln y^n + (1-t^n) \ln (1-y^n) \} \quad (7)$$

Let's consider some elementary properties of this error function. Differentiating this error function with respect to  $y^n$  we obtain

$$\frac{\partial E}{\partial y^n} = \frac{(y^n - t^n)}{y^n(1-y^n)} \quad (8)$$

The absolute minimum of the error function occurs when

$$y^n = t^n \quad \forall n \quad (9)$$

The considering network has one output whose value is to be interpreted as a probability, so it is appropriate to consider the logistic sigmoid activation function, equation (2), which has the property

$$g'(a) = g(a)(1-g(a)) \quad (10)$$

Combining equations (8) and (10) we see that the derivative of the error with respect to  $a$  takes a simple form



# DAta Mining & Exploration Program

$$\delta^n \equiv \frac{\partial E}{\partial a^n} = y^n - t^n \quad (11)$$

This equation gives the error quantity which is back propagated through the network in order to compute the derivatives of the error function with respect to the network weights. The same equation form can be obtained for the sum-of-squares error function and linear output units. This shows that there is a natural pairing of error function and output unit activation function.

From the equations (7) and (9) the value of the cross entropy error function at its minimum is given by

$$E_{min} = -\sum \{ t^n \ln t^n + (1-t^n) \ln(1-t^n) \} \quad (12)$$

The last equation becomes zero for 1-of-C coding scheme. However, when is a continuous variable in the range (0,1) representing the probability of the input vector belonging to class C, the error function (7) is also the correct one to use, In this case the minimum value (12) of the error does not become 0. In this case it is appropriate by subtracting this value from the original error function to get a modified error function of the form

$$E = -\sum_n \left\{ t^n \ln \frac{y^n}{t^n} + (1-t^n) \ln \frac{(1-y^n)}{(1-t^n)} \right\} \quad (13)$$

But before moving to cross-entropy for multiple classes let us describe more in detail its properties. Assume the network output for a particular pattern  $n$ , written in the form  $y^n = t^n + e^n$ . Then the cross-entropy error function (13) can be transformed to the form

$$E = -\sum_n \left\{ t^n \ln \left( 1 + \frac{\epsilon^n}{t^n} \right) + (1-t^n) \ln \left( 1 - \frac{\epsilon^n}{1-t^n} \right) \right\} \quad (14)$$

so that the error function depends on the relative errors of the network outputs. Knowing that the sum-of-squares error function depends on the squares of the absolute errors, we can make comparisons. Minimization of the cross-entropy error function will tend to result in similar relative errors on both small and large target values. By contrast, the sum-of-squares error function tends to give similar absolute errors for each pattern, and will give large relative errors for small output values. This result suggests the better functionality of the cross-entropy error function over the sum-of-squares error function at estimating small probabilities. Another advantage over the sum-of-squares error function, is that the cross-entropy error function gives much stronger weight to smaller errors.

### 2.1.1.3 Cross Entropy for the multiple-class case

Let's return to the classification problem involving mutually exclusive classes, where the number of classes is greater than two. For this problem we should seek the form which the error function should take. The network now has one output  $y_k$ , for each class, and target data which has a 1-of-c coding scheme, so that we have  $t_k^n = b_{kl}$  for a pattern  $n$  from class  $C_i$ . The probability of observing the set of target values  $t_k^n = b_{kl}$ , given an input vector  $x^n$ , is just  $P(C_i|x) = y_i$ . Therefore the conditional distribution for this pattern can be written as



# DAta Mining & Exploration Program

$$p(t^n | x^n) = \prod_{k=1}^c (y_k^n)^{t_k^n} \quad (15)$$

As before, starting from the likelihood function, by taking the negative logarithm, we obtain an error function of the form

$$E = - \sum_n \sum_{k=1}^c t_k^n \ln y_k^n \quad (16)$$

For 1-of-C coding scheme the minimum value of the error function (16) equals 0. But the error function is still valid when  $t_k^n$  is a continuous variable in the range (0,1) representing the probability that  $x^n$  belongs to  $C_k$ . To get the proper target variable the softmax activation function is used. So for the cross-entropy error function for multiple classes, equation (16), to be efficient the softmax activation function must be used. By evaluating the derivatives of the softmax error function considering all inputs to all output units, (for pattern n) it can be obtained

$$\frac{\partial E^n}{\partial a_k} = y_k - t_k \quad (17)$$

which is the same result as found for the two-class cross-entropy error (with a logistic activation function), equation (11). The same result is valid for the sum-of-squares error (with a linear activation function). This can be considered as an additional proof that there is a natural pairing of error function and activation function.

## 2.2 The Back Propagation learning rule

For better understanding, the back propagation learning algorithm can be divided into two phases: propagation and weight update.

### *Phase 1: Propagation*

Each propagation involves the following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
2. Back propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

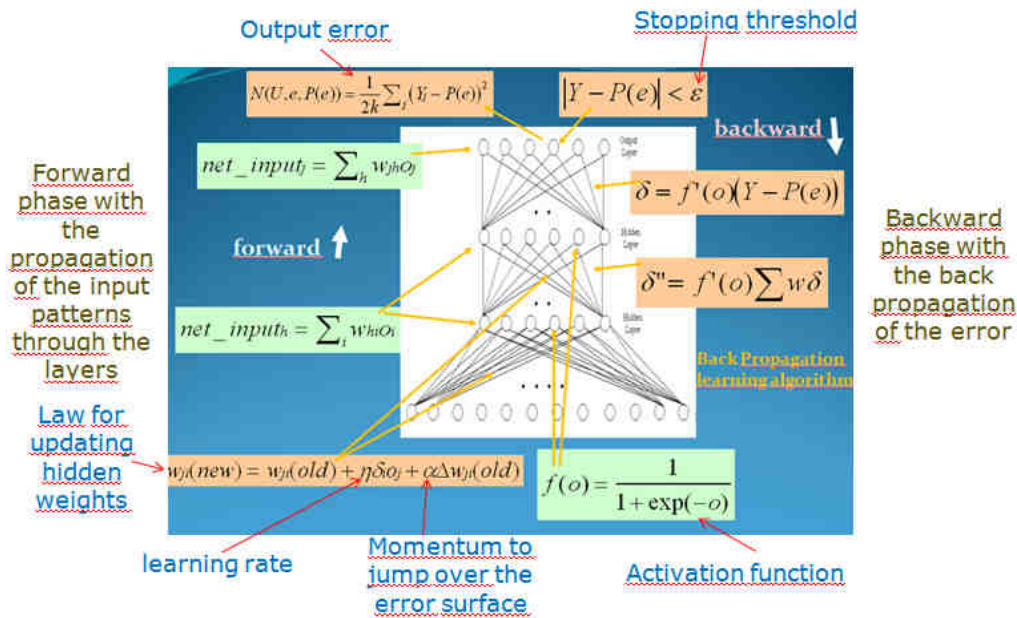
### *Phase 2: Weight update*

For each weight-synapse:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.

This ratio influences the speed and quality of learning; it is called the *learning rate*. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

Repeat the phase 1 and 2 until the performance of the network is good enough.



**Fig. 5 – The typical flow structure of the Back Propagation algorithm**

As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes. So technically speaking, back propagation is used to calculate the gradient of the error of the network with respect to the network's modifiable weights. This gradient is almost always then used in a simple stochastic gradient descent algorithm to find weights that minimize the error. Often the term "back propagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Back propagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

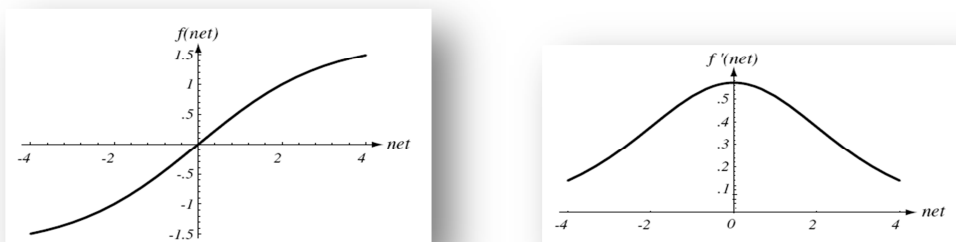
Back propagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network. Non-linear activation functions that are commonly used include the logistic function, the softmax function.

## 2.3 MLP Practical Rules

The practice and expertise in the machine learning models, such as MLP, are important factors, coming from a long training and experience within their use in scientific experiments. The speed and effectiveness of the results strongly depend on these factors. Unfortunately there are no magic ways to a priori indicate the best configuration of internal parameters, involving network topology and learning algorithm. But in some cases a set of practical rules to define best choices can be taken into account.

### 2.3.1 Selection of neuron activation function

- If there are good reasons to select a particular activation function, then do it
  - linear;
  - threshold;
  - Hyperbolic tangent;
  - sigmoid;
- General “good” properties of activation function
  - Non-linear;
  - Saturate – some max and min value;
  - Continuity and smooth;
  - Monotonicity: convenient but nonessential;
  - Linearity for a small value of net;
- Sigmoid function has all the good properties:
  - Centered at zero;
  - Anti-symmetric;
  - $f(-net) = -f(net)$ ;
  - Faster learning;
  - Overall range and slope are not important;



**Fig. 6 – The sigmoid function and its first derivative**

### 2.3.2 Scaling input and target values

- Standardize
  - Large scale difference
    - error depends mostly on large scale feature;
  - Shifted to Zero mean, unit variance
    - Need to be done once, before training;





# DAta Mining & Exploration Program

- Need full data set;
- Target value
  - Output is saturated
    - In the training, the output never reach saturated value;
      - Full training never terminated;
  - Range [-1, +1] is suggested;

## 2.3.3 Number of hidden nodes

- Number of hidden units governs the expressive power of net and the complexity of decision boundary;
- Well-separated  $\rightarrow$  fewer hidden nodes;
- From complicated density, highly interspersed  $\rightarrow$  many hidden nodes;
- Heuristics rule of thumb:
  - Use a minimum of  $2N+1$  neurons of the first hidden layer ( $N$  is the number of input nodes);
  - More training data yields better result;
  - Number of weights  $<$  number of training data;
  - Number of weights  $\approx$  (number of training data)/10;
  - Adjust number of weights in response to the training data:
    - Start with a “large” number of hidden nodes, then decay, prune weights...

## 2.3.4 Number of hidden layers

- One or two hidden layers are OK, so long as differentiable activation function;
  - But one layer is generally sufficient;
- More layers  $\rightarrow$  more chance of local minima;
- Single hidden layer vs double (multiple) hidden layer:
  - single is good for any approximation of continuous function;
  - double may be good some times;
- Problem-specific reason of more layers:
  - Each layer learns different aspects;



# DAta Mining & Exploration Program

## 3 Use of the web application model

The Multi Layer Perceptron (MLP) is one of the most common supervised neural architectures used in many application fields. It is especially related to classification and regression problems, and in DAME it is designed to be associated with such two functionality domains. The description of these two functionalities is reported in the Reference Manual [A18], available from webapp menu or from the beta intro web page.

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.

### 3.1 Use Cases

For the user the MLP with BP system offers four use cases:

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

As described in [A19] a supervised machine learning model like MLP requires different use cases, well ordered in terms of execution sequence. A typical complete experiment with this kind of models consists in 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 network weights);
2. **Test** the trained network, in order to verify training quality (it is also included the validation step, available for some models). 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.

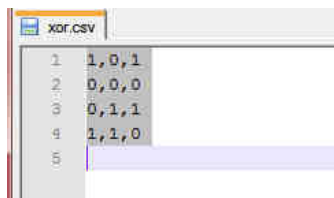
The **Full** use case includes both train and test previous cases. It can be executed as an alternative to the sequence of the two use cases. In this sense it is not to be considered as a single step of the sequence.

## 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): tabular fits files;
- VOTABLE (extension .votable): formatted files containing special fields separated by keywords coming from XML language, with more special keywords defined by VO data standards;

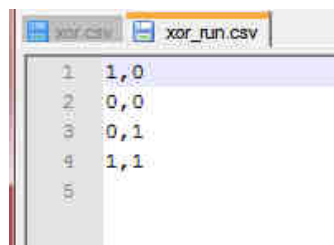
For training and test cases a correct dataset file must contain both input and target features (columns), with input type as the first group and target type as the final group.



Line	Content
1	1, 0, 1
2	0, 0, 0
3	0, 1, 1
4	1, 1, 0
5	

**Fig. 7 – The content of the xor.csv file used as input for training/test use cases**

As shown in Fig. 7, 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.



Line	Content
1	1, 0
2	0, 0
3	0, 1
4	1, 1
5	

**Fig. 8 – The content of the xor\_run.csv file used as input for Run use case**

In Fig. 8, 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	TEST	FULL	RUN
Mlp_TRAIN.log	Mlp_TEST_params.xml	Mlp_FULL_params.xml	Mlp_RUN.log
Mlp_TRAIN_params.xml	Mlp_TEST_output.csv	Mlp_FULL.log	Mlp_RUN_output.csv
Mlp_TRAIN_error.csv	Mlp_TEST_outputPlot.jpeg	Mlp_FULL_error.csv	Mlp_RUN_params.xml
Mlp_TRAIN_tmp_weights	Mlp_TEST.log	Mlp_FULL_trainOutput.csv	
Mlp_TRAIN_weights		Mlp_FULL_tmp_weights	
Mlp_TRAIN_errorPlot.jpeg		Mlp_FULL_weights	
		Mlp_FULL_output.csv	
		Mlp_FULL_errorPlot.jpeg	
		Mlp_FULL_outputPlot.jpeg	

**Tab. 1 – output file list in case of regression type experiments**

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

TRAIN	TEST	FULL	RUN
Mlp_TRAIN.log	Mlp_TEST_params.xml	Mlp_FULL_params.xml	Mlp_RUN.log
Mlp_TRAIN_params.xml	Mlp_TEST_output.csv	Mlp_FULL.log	Mlp_RUN_output.csv
Mlp_TRAIN_error.csv	Mlp_TEST_confusionMatrix	Mlp_FULL_error.csv	Mlp_RUN_params.xml
Mlp_TRAIN_tmp_weights	Mlp_TEST.log	Mlp_FULL_trainOutput.csv	
Mlp_TRAIN_weights		Mlp_FULL_tmp_weights	
Mlp_TRAIN_errorPlot.jpeg		Mlp_FULL_weights	
		Mlp_FULL_output.csv	
		Mlp_FULL_errorPlot.jpeg	
		Mlp_FULL_confusionMatrix	

**Tab. 2 – output file list in case of classification type experiments**



# DAta Mining & Exploration Program

## 3.4 TRAIN Use case

In the use case named “**Train**”, the software provides the possibility to train the MLP. The user will be able to use new or existing (already trained) MLP 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, dealing with network topology and learning algorithm. 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).

### 3.4.1 Regression with MLP – Train Parameter Specifications

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

- **Train Set**

**this parameter is a field required!**

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

- **Validation Set**

This is the dataset file to be used as input for the validation of the learning phase of the model. 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, CSV, VOTABLE).

If users leaves empty this parameter field, the validation phase of the training results is omitted.

- **Network File**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a 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 extension of such a file is usually .mlp.



# 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.

- **number of input nodes**

**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 nodes for hidden layer**

**this parameter is a field required!**

It is the number of neurons of the unique 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 output nodes**

**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).

- **number of iterations**

This the maximum number of learning iterations. It is one of the two stopping criteria for the learning algorithm. It is suggested to put an high value, in order to be sure to reach the best training results. User should use this value in combination with the error tolerance parameter.

If left empty, the default value is 1000.

- **error tolerance**

This is the threshold of the learning loop. This is one of the two stopping criteria of the algorithm. Use this parameter in combination with the number of iterations. If left empty, its default is 0.001.

- **training mode**

This is the combination of two parameters: training error evaluation criterion + the submission rule of the dataset to the model.





# DAta Mining & Exploration Program

The possible criterion for the training error evaluation is **MSE** (Mean Square Error).

The two possible submission rules are: **Batch**, where the learning error is evaluated after each entire data pattern set calculation; and **Incremental** (also known as on-line), where the error is evaluated after a single pattern submission to the network.

(See the user manual for details)

The following are the possible choices:

- **1:** MSE + Batch
- **2:** MSE + Incremental

If left empty, the default is the first one (MSE +Batch).

## 3.4.2 Classification with MLP – Train Parameter Specifications

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

- **Train Set**

**this parameter is a field required!**

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

- **Validation Set**

This is the dataset file to be used as input for the validation of the learning phase of the model. 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, CSV, VOTABLE).

If users leaves empty this parameter field, the validation phase of the training results is omitted.

- **Network File**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a training session. Usually this file should not be edited or



# DAta Mining & Exploration Program

modified by users, just to preserve its content as generated by the model itself. The extension of such a file is usually .mlp.

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.

- **number of input nodes**

**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 nodes for hidden layer**

**this parameter is a field required!**

It is the number of neurons of the unique 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 output nodes**

**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).

- **number of iterations**

This the maximum number of learning iterations. It is one of the two stopping criteria for the learning algorithm. It is suggested to put an high value, in order to be sure to reach the best training results. User should use this value in combination with the error tolerance parameter.

If left empty, the default value is 1000.

- **error tolerance**

This is the threshold of the learning loop. This is one of the two stopping criteria of the algorithm. Use this parameter in combination with the number of iterations. If left empty, its default is 0.001.

- **training mode**



# DAta Mining & Exploration Program

This is the combination of two parameters: training error evaluation criterion + the submission rule of the dataset to the model.

The two possible criteria for the training error evaluation are: **MSE** (Mean Square Error) and **CE** (Cross Entropy).

The two possible submission rules are: **Batch**, where the learning error is evaluated after each entire data pattern set calculation; and **Incremental** (also known as on-line), where the error is evaluated after a single pattern submission to the network.

(See the user manual for details)

The following are the possible choices:

- **1:** MSE + Batch
- **2:** MSE + Incremental
- **3:** CE + Batch
- **4:** CE + Incremental

If left empty, the default is the first one (MSE +Batch).

## 3.5 TEST Use case

In the use case named “**Test**”, the software provides the possibility to test the MLP. The user will be able to use already trained MLP 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 MLP – Test Parameter Specifications

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

- **Test Set**

**this parameter is a field required!**

Dataset file as input. It is a file containing input and target columns.

It must have the same number of input and target columns as for the training input file.



# DAta Mining & Exploration Program

For example, it could be the same dataset file used as the training input file.

- **Network File**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a 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 extension of such a file is usually .mlp.

### 3.5.2 Classification with MLP – Test Parameter Specifications

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

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

- **Test Set**

**this parameter is a field required!**

Dataset file as input. It is a file containing input and target columns.

It must have the same number of input and target columns as for the training input file.

For example, it could be the same dataset file used as the training input file.

- **Network File**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a 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 extension of such a file is usually .mlp.



# DAta Mining & Exploration Program

## 3.6 Run Use case

In the use case named “**Run**”, the software provides the possibility to run the MLP. The user will be able to use already trained and tested MLP 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 MLP – Run Parameter Specifications

In the case of Regression\_MLP with Run use case, the help page is at the address:

[http://dame.dsf.unina.it/mlp\\_help.html#regr\\_run](http://dame.dsf.unina.it/mlp_help.html#regr_run)

- **Run Set**

**this parameter is a field required!**

It is a file containing just input columns (NOT TARGET).

It must have the same number of input columns as for the training input file.

For example, it could be the same dataset file used as the training input file, without the last (target) columns.

- **Network File**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a 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 extension of such a file is usually .mlp.

### 3.6.2 Classification with MLP – Run Parameter Specifications

In the case of Classification\_MLP with Run use case, the help page is at the address:

[http://dame.dsf.unina.it/mlp\\_help.html#class\\_run](http://dame.dsf.unina.it/mlp_help.html#class_run)

- **Run Set**

**this parameter is a field required!**



# DAta Mining & Exploration Program

It is a file containing just input columns (NOT TARGET).

It must have the same number of input columns as for the training input file.

For example, it could be the same dataset file used as the training input file, without the last (target) columns.

- **Network File**

**this parameter is a field required!**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a 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 extension of such a file is usually .mlp.

## 3.7 Full Use case

In the use case named “**Full**”, the software provides the possibility to perform a complete sequence of train, test and run cases with the MLP.

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).

### 3.7.1 Regression with MLP – Full Parameter Specifications

In the case of Regression\_MLP with Full use case, the help page is at the address:

[http://dame.dsf.unina.it/mlp\\_help.html#regr\\_full](http://dame.dsf.unina.it/mlp_help.html#regr_full)

- **Train Set**

**this parameter is a field required!**

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

- **Validation Set**





# DAta Mining & Exploration Program

This is the dataset file to be used as input for the validation of the learning phase of the model. 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, CSV, VOTABLE).

If users leaves empty this parameter field, the validation phase of the training results is omitted.

- **Test Set**

**this parameter is a field required!**

Dataset file as input. It is a file containing input and target columns.

It must have the same number of input and target columns as for the training input file.

For example, it could be the same dataset file used as the training input file.

- **Network File**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a 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 extension of such a file is usually .mlp.

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.

- **number of input nodes**

**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 nodes for hidden layer**

**this parameter is a field required!**

It is the number of neurons of the unique 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 output nodes**



# DAta Mining & Exploration Program

**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).

- **number of iterations**

This is the maximum number of learning iterations. It is one of the two stopping criteria for the learning algorithm. It is suggested to put a high value, in order to be sure to reach the best training results. User should use this value in combination with the error tolerance parameter.

If left empty, the default value is 1000.

- **error tolerance**

This is the threshold of the learning loop. This is one of the two stopping criteria of the algorithm. Use this parameter in combination with the number of iterations. If left empty, its default is 0.001.

- **training mode**

This is the combination of two parameters: training error evaluation criterion + the submission rule of the dataset to the model.

The possible criterion for the training error evaluation is **MSE** (Mean Square Error).

The two possible submission rules are: **Batch**, where the learning error is evaluated after each entire data pattern set calculation; and **Incremental** (also known as on-line), where the error is evaluated after a single pattern submission to the network.

(See the user manual for details)

The following are the possible choices:

- **1:** MSE + Batch
- **2:** MSE + Incremental

If left empty, the default is the first one (MSE +Batch).

## 3.7.2 Classification with MLP – Full Parameter Specifications

In the case of Classification\_MLP with Full use case, the help page is at the address:  
[http://dame.dsf.unina.it/mlp\\_help.html#class\\_full](http://dame.dsf.unina.it/mlp_help.html#class_full)

- **Train Set**



# DAta Mining & Exploration Program

## **this parameter is a field required!**

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

- **Validation Set**

This is the dataset file to be used as input for the validation of the learning phase of the model. 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, CSV, VOTABLE).

If users leaves empty this parameter field, the validation phase of the training results is omitted.

- **Test Set**

## **this parameter is a field required!**

Dataset file as input. It is a file containing input and target columns.

It must have the same number of input and target columns as for the training input file.

For example, it could be the same dataset file used as the training input file.

- **Network File**

It is a file generated by the model during training phase. It contains the resulting network topology as stored at the end of a 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 extension of such a file is usually .mlp.

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.

- **number of input nodes**

## **this parameter is a field required!**



# DAta Mining & Exploration Program

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 nodes for hidden layer**

**this parameter is a field required!**

It is the number of neurons of the unique 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 output nodes**

**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).

- **number of iterations**

This the maximum number of learning iterations. It is one of the two stopping criteria for the learning algorithm. It is suggested to put an high value, in order to be sure to reach the best training results. User should use this value in combination with the error tolerance parameter.

If left empty, the default value is 1000.

- **error tolerance**

This is the threshold of the learning loop. This is one of the two stopping criteria of the algorithm. Use this parameter in combination with the number of iterations. If left empty, its default is 0.001.

- **training mode**

This is the combination of two parameters: training error evaluation criterion + the submission rule of the dataset to the model.

The two possible criteria for the training error evaluation are: **MSE** (Mean Square Error) and **CE** (Cross Entropy).

The two possible submission rules are: **Batch**, where the learning error is evaluated after each entire data pattern set calculation; and **Incremental** (also known as on-line), where the error is evaluated after a single pattern submission to the network.

(See the user manual for details)



# DAta Mining & Exploration Program

The following are the possible choices:

- **1:** MSE + Batch
- **2:** MSE + Incremental
- **3:** CE + Batch
- **4:** CE + Incremental

If left empty, the default is the first one (MSE +Batch).



# 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 Classification 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.

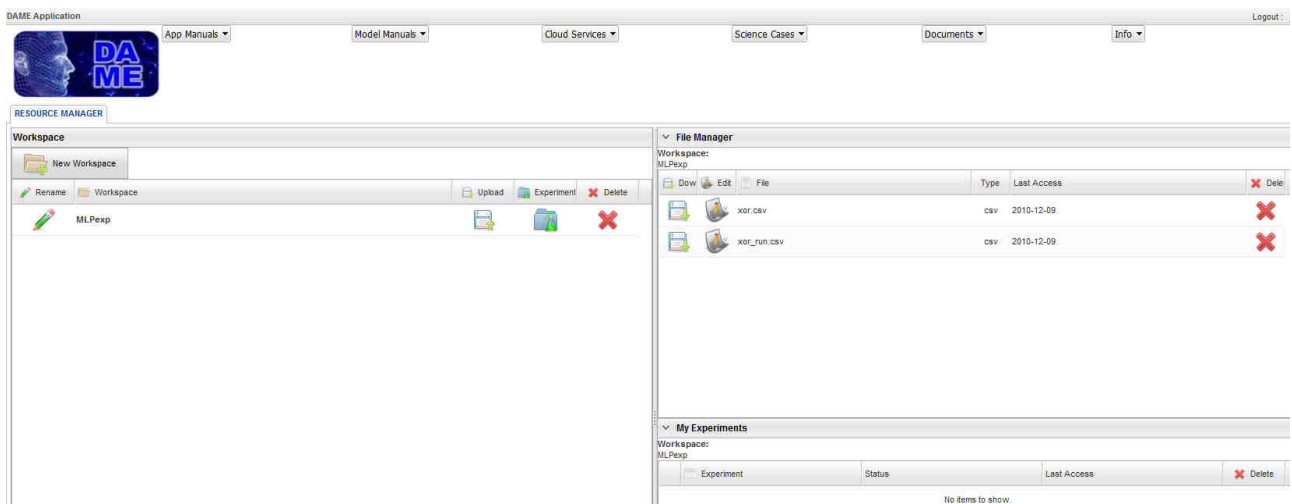
Let say, it is an example comparable with the classical “print <Hello World> on standard output” implementation problem for beginners in C language.

As first case, we will use the MLP model associated to the Classification functionality.

The starting point is to create a new workspace, named **mlpExp** 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.

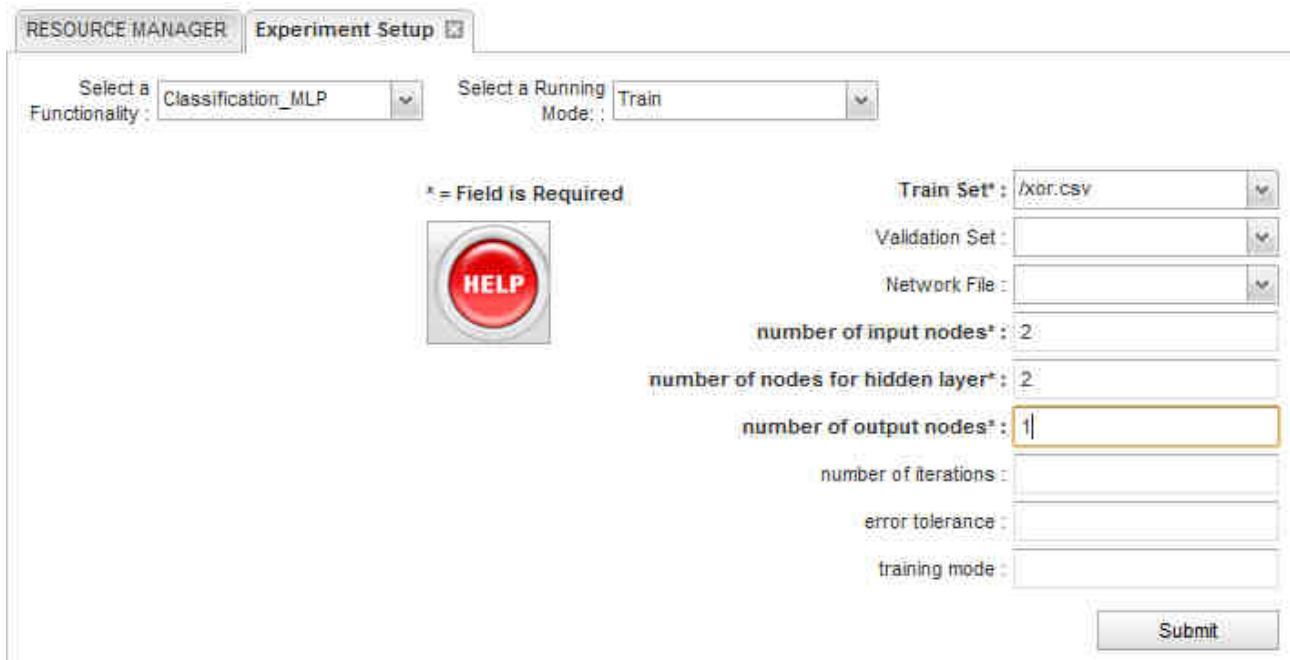


**Fig. 9 – The starting point, with a Workspace (mlpExp) created and two data files uploaded**



## 4.1.1 Classification MLP – 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 **Classification\_MLP** as couple functionality-model of the current experiment and we select also **Train** as use case.



**Fig. 10 – 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 previous sections 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);



# DAta Mining & Exploration Program

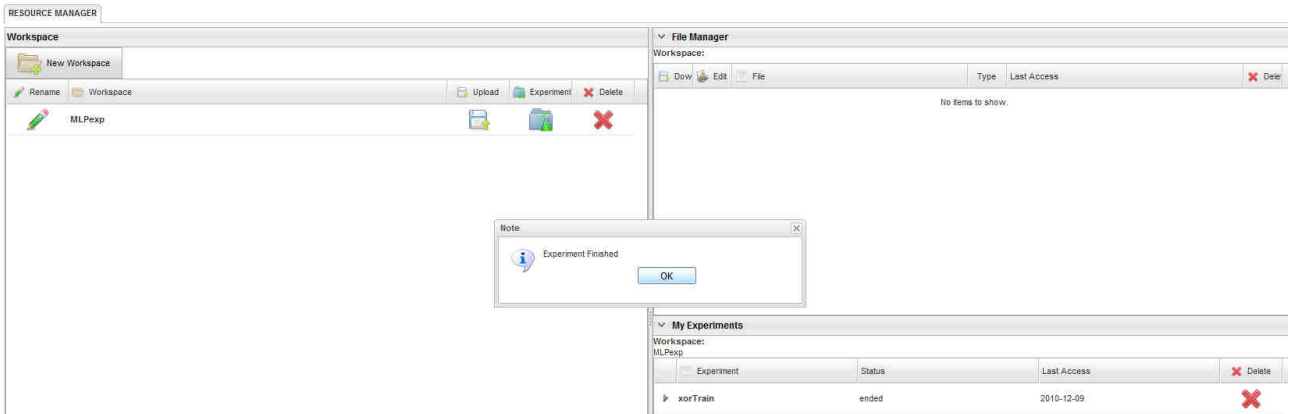


Fig. 11 – The xorTrain experiment status after submission

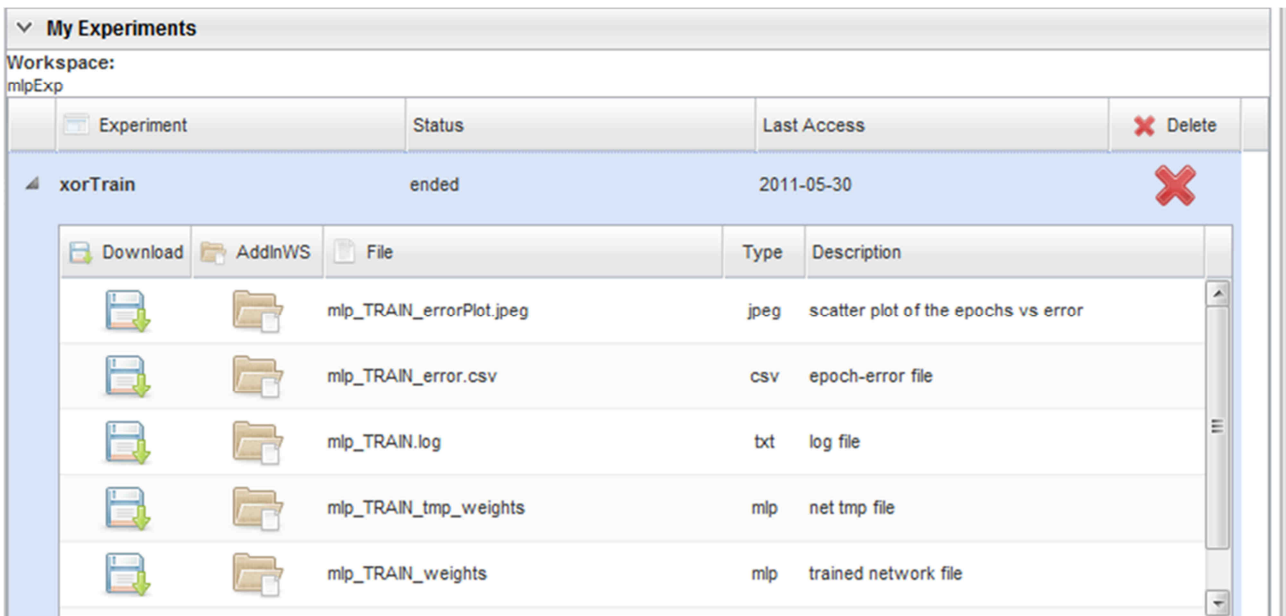


Fig. 12 – The xorTrain experiment output files

The content of output files, obtained at the end of the experiment (available when the status is “ended”) is shown in the following. The file **mlp\_TRAIN\_error.csv** reports the training error after a set of iterations indicated in the first column.



# DAta Mining & Exploration Program

```

Epochs,Current error
1, 0.2500620186
10, 0.2473851591
20, 0.1750848889
30, 0.1167778075
40, 0.0464941077
50, 0.0165067063
60, 0.0066382927
70, 0.0009171068

MAIN_FLO_2.1
num_layers=3
learning_rate=0.700000
connection_rate=1.000000
network_type=0
learning_momentum=0.600000
training_algorithm=2
train_error_function=1
train_stop_function=0
cascade_output_change_fraction=0.010000
quickprop_decay=0.000100
quickprop_mu=1.750000
rprop_increase_factor=1.200000
rprop_decrease_factor=0.500000
rprop_delta_min=0.000000
rprop_delta_max=50.000000
rprop_delta_zero=0.100000
cascade_output_stagnation_epochs=12
cascade_candidate_change_fraction=0.010000
cascade_candidate_stagnation_epochs=12
cascade_max_out_epochs=150
cascade_max_cand_epochs=150
cascade_num_candidate_groups=2
hit_fail_limit=3.49999994039535522461e-01
cascade_candidate_limit=1.00000000000000000000e+03
cascade_weight_multiplier=4.00000005960464477539e-01
cascade_activation_functions_count=10
cascade_activation_functions=5 7 8 10 11 14 15 16 17
cascade_activation_steepness_count=4
cascade_activation_steepnesses=2.50000000000000000000e-01 5.00000000000000000000e-01 7.50000000000000000000e-01 1.00000000000000000000e+00
scale_included=0
neurons (num_inputs, activation_function, activation_steepness)=(0, 0, 0.000000000000000000e+00) (0, 0, 0.000000000000000000e+00) (0, 0, 0.000000000000000000e+00) (3, 3, 5.000000000000000000e-01) (3, 3,
connections (connected_to_neuron, weight)=(0, -5.39315938949584960939e+00) (1, -5.20419979095458984375e+00) (2, 2.09722948074340820312e+00) (0, -3.5278165340423583984e+00) (1, -3.43121361732482910156e+00) (2, 4
  
```

Fig. 13 – The files .csv (left) and .mlp (right) output of the xorTrain experiment

The file **mlp\_TRAIN\_weights.mlp** contains the topology of the trained neural network and the weights of the connections between the network layers.

## 4.1.2 Classification MLP – Test use case

The file **mlp\_TRAIN\_weights.mlp** 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). This is because it represents the stored brain of the network, trained to calculate the XOR function.

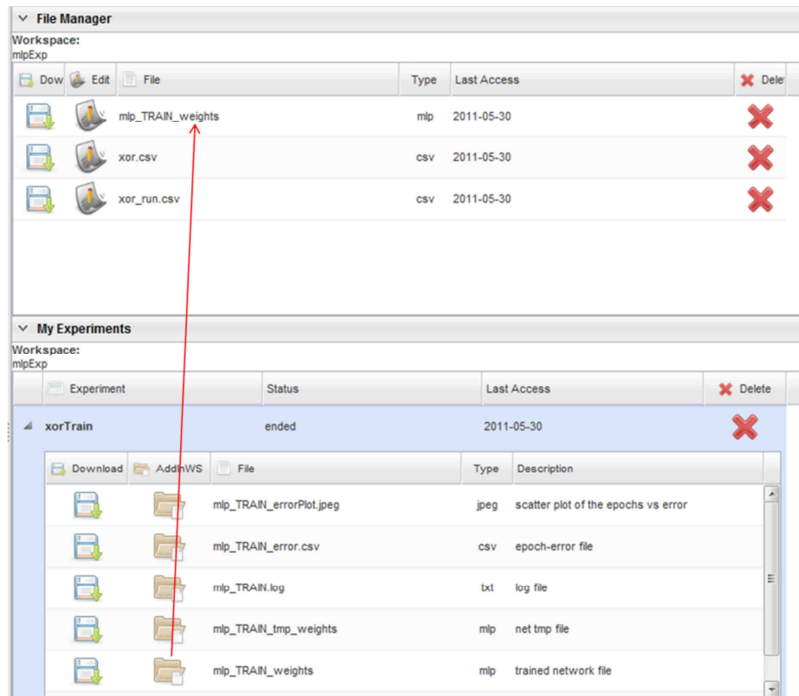
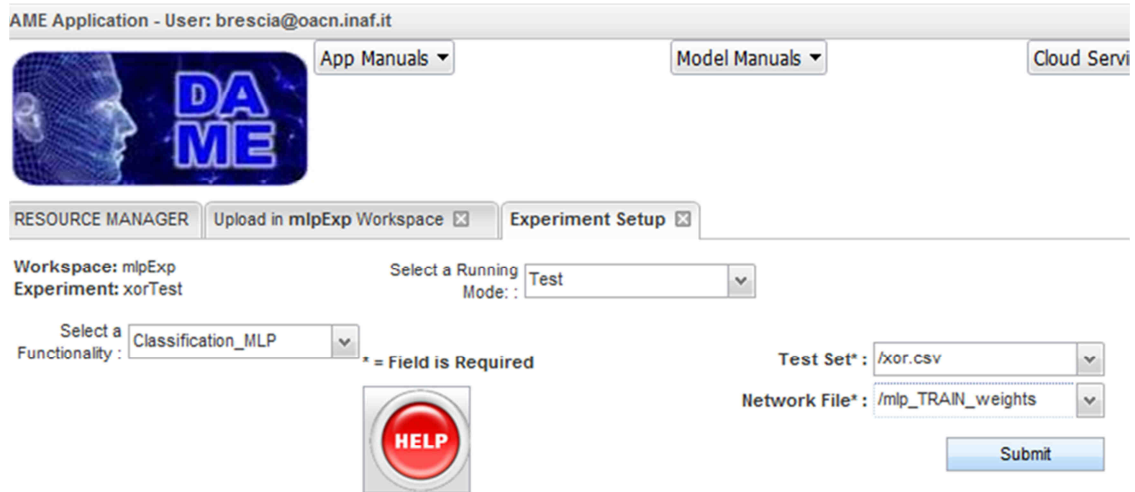


Fig. 14 – The file “mlp\_TRAIN\_weights.mlp” copied in the WS input file area for next purposes



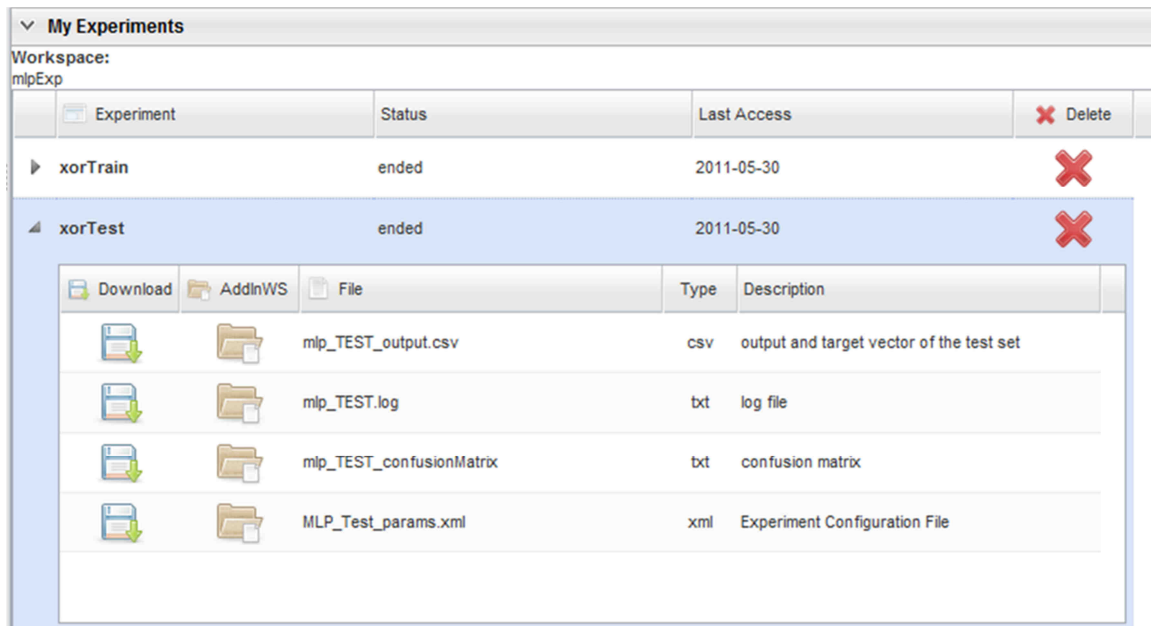
# DAta Mining & Exploration Program

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. 15 – The xorTest experiment configuration tab (note “weights” file inserted)**

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



**Fig. 16 – The xorTest experiment output files**



# DAta Mining & Exploration Program

## 4.1.3 Classification MLP – Full use case

If an automatic sequence of train and test use cases is desired, it is possible to execute an experiment by choosing Full as use case.

In this case, we create a new experiment, named **xorFull**, where we have to select parameters for both train and test use cases.

The screenshot shows the 'Experiment Setup' configuration tab in the DAME software. At the top, there are two tabs: 'RESOURCE MANAGER' and 'Experiment Setup'. Below the tabs, there are two dropdown menus: 'Select a Functionality:' set to 'Classification\_MLP' and 'Select a Running Mode:' set to 'Full'. A red 'HELP' button is visible on the left. A legend indicates '\* = Field is Required'. The configuration fields are as follows:

Train Set*	/xor.csv
Validation Set	
Test Set*	/xor.csv
Network File	
number of input nodes*	2
number of nodes for hidden layer*	2
number of output nodes*	1
number of iterations	
error tolerance	
training mode	

A 'Submit' button is located at the bottom right of the configuration area.

Fig. 17 – The xorFull experiment configuration tab

At the end, we obtain the output files of the experiment, in which both training and test outputs are present.



# DAta Mining & Exploration Program

My Experiments

Workspace:  
mlpExp

Experiment	Status	Last Access	Delete
xorTrain	ended	2011-05-30	X
xorTest	ended	2011-05-30	X
xorFull	ended	2011-05-30	X

Download AddInWS File

Type	Description
csv	output and target vector of the test set
jpeg	scatter plot of the epochs vs error
csv	epoch-error file
txt	log file
mip	net tmp file

Fig. 18 – The xorFull experiment output



# 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	MLP	Multi Layer Perceptron
CE	Cross Entropy	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
DAPL	Data Access & Process Layer	PI	Principal Investigator
DL	Data Layer	REDB	Registry & Database
DM	Data Mining	RIA	Rich Internet Application
DMM	Data Mining Model	SDSS	Sloan Digital Sky Survey
DMS	Data Mining Suite	SL	Service Layer
FITS	Flexible Image Transport System	SW	Software
FL	Frontend Layer	UI	User Interface
FW	FrameWork	URI	Uniform Resource Indicator
GRID	Global Resource Information Database	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	Mining the SDSS archive I. Photometric Redshifts in the Nearby Universe. <i>Astrophysical Journal</i> , Vol. 663, pp. 752-764	D’Abrusco, R. et al.	2007
R6	<i>The Fourth Paradigm</i> . Microsoft research, Redmond Washington, USA	Hey, T. et al.	2009
R7	Artificial Intelligence, A modern Approach. Second ed. (Prentice Hall)	Russell, S., Norvig, P.	2003
R8	Pattern Classification, A Wiley-Interscience Publication, New York: Wiley	Duda, R.O., Hart, P.E., Stork, D.G.	2001
R9	Neural Networks - A comprehensive Foundation, Second Edition, Prentice Hall	Haykin, S.,	1999
R10	<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
R11	<i>Probabilistic connectionist approaches for the design of good communication codes</i> . Proc. of the IJCNN, Japan	Babu G. P., Murty M. N.	1993
R12	<i>Approximations by superpositions of sigmoidal functions</i> . Mathematics of Control, Signals, and Systems, 2:303–314, no. 4 pp. 303-314	Cybenko, G.	1989

**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	MLP_user_manual_VONEURAL-MAN-NA-0001-Rel1.0	DAME Working Group	12/10/2007
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	VONeuralMLP2.0_VONEURAL-SPE-NA-0007-Rel1.0	Skordovski	20/02/2008
A15	dm_model_VONEURAL-SRS-NA-0005-Rel0.4	Cavuoti	05/01/2009
A16	FANN_MLP_VONEURAL-TRE-NA-0011-Rel1.0	Skordovski, Laurino	30/11/2008
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”*

