Multi-layer Perceptron: nagdmc_mlp

Purpose

nagdmc_mlp computes a feed-forward multi-layer perceptron with a single hidden layer. The values of free parameters are optimised by using conjugate gradients.

Declaration

```
#include <nagdmc.h>
void nagdmc_mlp(long rec1, long nvar, long nrec, long dblk, double data[],
                long nxvar, long xvar[], long yvar, long nhid, int phi,
                int psi, double gain, long *nit, long nit2, long ninit,
                long iseed, double model[], long val[], double vdata[],
                double res[], double *objf, int *info);
```

Parameters

1: **rec1** – `long` *Input*

On entry: the index in the data of the first data record used in the analysis.

Constraint: **rec1** ≥ 0.

2: **nvar** – `long` *Input*

On entry: the number of variables in the data.

Constraint: **nvar** > 1.

3: **nrec** – `long` *Input*

On entry: the number of consecutive records, beginning at **rec1**, used in the analysis.

Constraint: **nrec** > 1.

4: **dblk** – `long` *Input*

On entry: the total number of records in the data block.

Constraint: **dblk** $\geq$ **rec1** + **nrec**.

5: **data**[**dblk** $*$ **nvar**] – `double` *Input*

On entry: the data values for the jth variable (for $j = 0, 1, \ldots,$ **nvar**-1) are stored in **data**[$i*$**nvar**$+j$], for $i = 0, 1, \ldots,$ **dblk** $- 1$.

6: **nxvar** – `long` *Input*

On entry: the number of independent variables. If **nxvar** $= 0$ then all variables in the data, excluding **yvar**, are treated as independent variables.

Constraint: $0 \leq$ **nxvar** $<$ **nvar**.

7: **xvar**[**nxvar**] – `long` *Input*

On entry: the indices indicating the position in **data** in which values of the independent variables are stored. If **nxvar** $= 0$ then **xvar** must be 0, and the indices of independent variables are given by $j = 0, 1, \ldots,$ **nvar** $- 1$; $j \neq$ **yvar**.

Constraints: if **nxvar** > 0, $0 \leq$ **xvar**[i] $<$ **nvar**, for $i = 0, 1, \ldots,$ **nxvar** $- 1$; otherwise **xvar** must be 0.

8: **yvar** – `long` *Input*

On entry: the index in **data** in which values of the dependent variable are stored.

Constraints: $0 \leq$ **yvar** $<$ **nvar**; if **nxvar** > 0, **yvar** $\neq$ **xvar**[i], for $i = 0, 1, \ldots,$ **nxvar** $- 1$.

9: **nhid** – `long` *Input*

On entry: the number of functions in the hidden layer.

Constraint: **nhid** > 0.

10: **phi** – `int` *Input*

On entry: an integer that signifies the hidden layer activation function.

Constraint: **phi** must be one of either 1 (logistic) or 2 (hyperbolic tangent).

11: **psi** – `int` *Input*

On entry: an integer that signifies the output layer activation function.

Constraint: **psi** must be one of either 0 (linear), 1 (logistic) or 2 (hyperbolic tangent).

12: **gain** – `double` *Input*

On entry: the value of the multiplicative constant in the argument to logistic or hyperbolic tangent functions.

Constraint: **gain** > 0.0.

13: **nit** – `long *` *Input/Output*

On entry: the maximum number of iterations of the conjugate gradients algorithm.
On exit: the number of completed iterations of the conjugate gradients algorithm.

Constraint: on entry the value pointed to by **nit** must be > 1.

14: **nit2** – `long` *Input*

On entry: the number of iterations used to find a good starting point for the optimisation.

Constraint: **nit2** ≥ 1.

15: **ninit** – `long` *Input*

On entry: the number of times to search for a good initial starting point for the optimisation.

Constraint: **ninit** ≥ 0.

16: **iseed** – `long` *Input*

On entry: the value of the random seed used to compute initial values of free parameters. If **iseed** is less than zero, the random seed is taken from the system clock; otherwise the value of **iseed** is used as the seed value.

17: **model**[$9 + p$+**nhid**$*(p+1)$] – `double` *Output*

On exit: the optimised free parameter values for a model with p independent variables.

18: **val**[**4**] – `long` *Input*

On entry: the information used to validate the MLP model and halt the optimisation; if **val** is 0, early stopping is not used; otherwise:

val[0] contains the number of data records used to validate the model;

val[1] contains the number of iterations of the optimising function to complete before beginning validation;

val[2] contains the number of iterations between consecutive validations;

val[3] contains number of iterations past the current minimum of the objective function to continue validating, thus trying to eliminate the possibility of the solution getting stuck in local minima.

Constraints: if referenced, **val**[0] ≥ 1; $1 <$ **val**[1] $< t$, where t is the value pointed to by **nit** on entry; **val**[2] ≥ 0; and **val**[3] ≥ 0.

19: **vdata**[**nvar**$*q$] – `double` *Input*

On entry: if **val** is not 0, **vdata**$[i *$ **nvar** $+ j]$ for $j = 0, 1, \ldots,$ **nvar** $- 1$ is the ith data record used to validate the model, for $i = 0, 1, \ldots, q - 1$, where $q =$ **val**[0]; otherwise **vpara** is not referenced.

20: **res**[**nrec**] – `double` *Output*

On exit: **res**[i] contains the MLP approximation to the value of the dependent variable in the data for the ith training data record, for $i = 0, 1, \ldots,$ **nrec** $- 1$.

21: **objf** – `double *` *Output*

On exit: the value of the sum of squares objective function at the end of the optimisation.

22: **info** – `int *` *Output*

On exit: **info** gives information on the success of the function call:

0: the function successfully completed its task.

i; $i = 1, 2, 3, 4, 6, 7, \ldots, 15, 18$: the specification of the ith formal parameter was incorrect.

99: the function failed to allocate enough memory.

Notation

nrec	the number of data records, n.
data	stores the data records x_i, for $i = 1, 2, \ldots, n$.
nxvar	determines the number of independent variables (equals the number of input units), d.
yvar	the index in **data** giving the values for the dependent variable y_i, for $i = 1, 2, \ldots, n$.
nhid	the number of hidden units in the model, m.
phi	signifies the function choice for $\phi(\cdot)$.
psi	signifies the function choice for $\psi(\cdot)$..
gain	if required, the value of γ.
res	the fitted values $\hat{y}_i$, for $i = 1, 2, \ldots, n$.

Description

A fully connected, feed-forward multi-layer perceptron (MLP) has d input units, m units in its hidden layer and a single output unit. Given the ith training data record x_i with scalar elements x_{ij}, for $i = 1, 2, \ldots, d$, the output $\hat{y}_i$ for the value y_i of the dependent variable in X is given by,

$$\hat{y}_i = \psi\left(\sum_{k=1}^{m} w_k \phi\left(\sum_{j=1}^{d} w_{jk} x_{ij} - \theta\right) - \eta\right), \quad \begin{cases} w_{jk}, w_k, \theta_k, \eta \in \mathbb{R}, \\ \phi(\cdot), \psi(\cdot) : \ \mathbb{R} \rightarrow \mathbb{R}, \end{cases},$$

where:

(a) w_{jk} denotes the weight value that connects the ith unit in the input layer to the kth unit in the hidden layer, for $j = 1, 2, \ldots, d$; for $k = 1, 2, \ldots, m$.

(b) θ is the threshold value subtracted at the hidden layer.

(c) w_k denotes the weight value that connects the kth unit in the hidden layer to the single unit in the output layer, for $k = 1, 2, \ldots, m$.

(d) η is the threshold value subtracted at the single unit in the output layer.

(e) $\phi(\cdot)$ is the activation function applied at the hidden layer and may be any one of the following functions:

$$\begin{array}{lllll} \text{logistic:} & \phi(z) & = & \dfrac{1}{1+\mathrm{e}^{-\gamma z}}, & \gamma, z \in \mathbb{R}, \\ \text{hyperbolic tangent:} & \phi(z) & = & \tanh(\gamma z), & \gamma, z \in \mathbb{R}, \end{array}$$

where the value of γ is supplied by the user.

(f) $\psi(\cdot)$ is the activation function applied at the output layer and may be any one of the following functions:

$$\begin{array}{lllll} \text{linear:} & \psi(z) & = & z, & z \in \mathbb{R}, \\ \text{logistic:} & \psi(z) & = & \dfrac{1}{1+\mathrm{e}^{-\gamma z}}, & \gamma, z \in \mathbb{R}, \\ \text{hyperbolic tangent:} & \psi(z) & = & \tanh(\gamma z), & \gamma, z \in \mathbb{R}. \end{array}$$

where, again, the value of γ for the logistic function is supplied by the user.

Values for the free parameters in the multi-layer perceptron model are optimised by using a pre-conditioned, limited memory quasi-Newton conjugate gradients method to minimise the objective (sum of squares) function:

$$\frac{1}{2n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2.$$

In order to improve the accuracy of MLP approximations to new data records, usually it is desriable to halt the optimisation before the value of the sum of squares error function, as measured on the training data records, reaches a global minimum. This method of improving the accuracy of MLPs on new data is known as early stopping, and can be performed by using a validation set of data records. In particular, the optimisation is halted when the sum of squares error function is minimised over a validation set of data records which are not (directly) used to determine values for the free parameters in the model.

References and Further Reading

Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press London.

Haykin S (1994) *Neural Networks* MacMillan.

See Also

nagdmc_predict_mlp	computes a predictions given a trained MLP model.
mlp_ex.c	the example calling program.