Package ‘neural’

February 20, 2015

Version 1.4.2.2

Title Neural Networks

Author Adam Nagy

Maintainer Billy Aung Myint <r@use-r.com>

Description RBF and MLP neural networks with graphical user interface

License GPL (>= 2)

Repository CRAN

NeedsCompilation no

Date/Publication 2014-09-28 18:26:46

R topics documented:

`letters_out` *Letters training sample- output*

Description

This data set gives the expected response for the `letters_train` dataset. Each letter has more than one representation.

Usage

```
data(letters_out)
```

Format

A matrix containing 40 response. Each row represents one expected response.

Source

Collecting from about 10 person and from some frequently used font type

---

`letters_recall` *Letters recall sample- input*

Description

This data set gives the 8x8 bitmatrix representation of alphabetic letters from A to E. Each letter has more than one representation. Can be useful as a recalling set for mlp network.

Usage

```
data(letters_recall)
```

Format

A matrix containing 50 representation of letters. Each row represents one letter.

Source

Collecting from about 10 person and from some frequently used font type

---

`letters_train` *Letters training sample- input*

Description

This data set gives the 8x8 bitmatrix representation of alphabetic letters from A to E. Each letter has more than one representation. Can be useful as an input training set for mlp network.

Usage

```
data(letters_train)
```

Format

A matrix containing 40 representation of letters. Each row represents one letter.

Source

Collecting from about 10 person and from some frequently used font type

mlp *MLP neural network*

Description

The recalling method of the MLP network which was trained by the mlptrain function.

Usage

```
mlp(inp,weight,dist,neurons,actfns=c(),layer=NaN, ...)
```

Arguments

`inp`	a matrix that contains one input data in each row.
`weight`	the weights of the network.
`dist`	the distortions of the network.
`neurons`	a numeric vector with length equals to the number of layers in the network, and the ith layer will contains neurons[i] neuron.
`actfns`	a list, which contains the numeric code of the activation functions, or the activation function itself. The length of the vector must be the same as the length of the neurons vector, and each element of the vector must be between 1 and 4 or a function. The possible numeric codes are the following: 1: Logistic function 2: Hyperbolic tangent function 3: Gauss function 4: Identical function.
`layer`	the number of the layer as far as the function is calculating. If the value is NaN, the function will calculate till the output layer. It can be useful if you need the output of one of the hidden layers.
`...`	currently not used.

Details

the "weight, dist, neurons, actfns" arguments can be determined by the mlptrain algorithm.

Value

a matrix that contains the response data of the network, each row contains one response.

See Also

'mlptrain' for training an MLP network, 'rbf' and 'rbftrain' for approximation.

mlptrain *MLP neural network*

Description

A simple MLP neural network that is suitable for classification tasks.

Usage

```
mlptrain(inp,neurons,out,weight=c(),dist=c(),alfa=0.2,it=200,online=TRUE,
permute=TRUE,thresh=0,dthresh=0.1,actfns=c(),diffact=c(),visual=TRUE, ...)
```

Arguments

`inp`	a matrix that contains one input data in each row.
`neurons`	a numeric vector with length equals to the number of layers in the network, and the ith layer will contains neurons[i] neuron.
`out`	a matrix that contains one output data in each row.
`weight`	the starting weights of the network.
`dist`	the starting distortions of the network.
`alfa`	the learning-rate parameter of the back-propagation algorithm.
`it`	the maximum number of training iterations.
`online`	if TRUE the algorithm will operate in sequential mode of back-propagation,if FALSE the algorithm will operate in batch mode of back-propagation.
`permute`	if TRUE the algorithm will use a random permutation of the input data in each epoch.
`thresh`	the maximal difference between the desired response and the actual response that is regarded as zero.
`dthresh`	if the difference between the desired response and the actual response is lesser than this value, the corresponding neuron is drawn in red, otherwise it is drawn in green.
`actfns`	a list, which contains the numeric code of the activation functions, or the activation function itself. The length of the vector must be the same as the length of the neurons vector, and each element of the vector must be between 1 and 4 or a function. The possible numeric codes are the following: 1: Logistic function 2: Hyperbolic tangent function 3: Gauss function 4: Identical function.
`diffact`	a list, which contains the differential of the activation functions. Only need to use if you use your own activation functions.
`visual`	a logical value, that switches on/off the graphical user interface.
`...`	currently not used.

Details

The function creates an MLP neural network on the basis of the function parameters. After the creation of the network it is trained with the back-propagation algorithm using the inp and out parameters. The inp and out parameters has to be the same number of rows, otherwise the function will stop with an error message.

If you use the weight or dist argument, than that variables won't be determined by random. This could be useful if you want to retrain your network. In that case use both of this two arguments.

From this vesion of the package there is the chance to use your own activation functions, by using the actfns argument. If you do this, don't forget to set the differential of the activation functions in the diffact argument at the same order and the same position where you are using the new activation function. (No need of using the diffact argument if you're using the preset activation functions.)

The function has a graphical user interface that can be switched on and off using the visual argument. If the graphical interface is on, the activation functions can be set in manually. If the activation functions are not set then each of them will be automatically the logistic function. The result of the function are the parameters of the trained MLP neural network. Use the mlp function for information recall.

Value

a list with 5 arguments:

`weight`	the weights of the network.
`dist`	the distortions of the network.
`neurons`	a numeric vector with length equals to the number of layers in the network, and the ith layer will contains neurons[i] neuron.
`actfns`	a list, that contains the activation functions. The length of the list is equal with the number of active layers.
`diffact`	a list, which contains the differential of the activation functions. The length of the list is equal with the number of active layers.

See Also

'mlp' for recall; 'rbftrain' and 'rbf' for training an RBF network.

Examples

```
x<-matrix(c(1,1,0,0,1,0,1,0),4,2)
y<-matrix(c(0,1,1,0),4,1)
neurons<-4
## Not run:
data<-mlptrain(x,neurons,y,it=4000);
mlp(x,data$weight,data$dist,data$neurons,data$actfns)

## End(Not run)
```

rbf *RBF neural network*

Description

The recalling method of the RBF network which was trained by the rbftrain function.

Usage

```
rbf(inp,weight,dist,neurons,sigma, ...)
```

Arguments

`inp`	a matrix that contains one input data in each row.
`weight`	the weights of the network.
`dist`	the distortion of the network.
`neurons`	a numeric vector with length equals to the number of layers in the network, and the ith layer will contains neurons[i] neuron.
`sigma`	the sigma parameters of the network.
`...`	currently not used.

Details

the last four argument can be produce by the rbftrain algorithm.

Value

a matrix that contains the response data of the network in each row.

See Also

'rbftrain' for training an RBF network, 'mlp' and 'mlptrain' for classification.

rbftrain *RBF neural network*

Description

A simple RBF neural network which suitable for approximation.

Usage

```
rbftrain(inp,neurons,out,weight=c(),dist=c(),alfa=0.2,it=40,err=0,
        sigma=NaN,online=TRUE,permute=TRUE,visual=TRUE, ...)
```

Arguments

`inp`	a matrix that contains one input data in each row.
`neurons`	the number of neurons in the hidden layer.
`out`	a matrix that contains one output data in each row.
`weight`	the starting weights of the network.
`dist`	the starting distortions of the network.
`alfa`	the learning-rate parameter of the back-propagation algorithm.
`it`	the maximum number of training iterations.
`err`	the average error at the studying points,if the average error anytime lower than this value,the algorithm will stop.
`sigma`	the width of the Gauss functions.
`online`	if TRUE the algorithm will operate in sequential mode of back-propagation,if FALSE the algorithm will operate in batch mode of back-propagation.
`permute`	if TRUE the algorithm will use a random permutation of the input data in each epoch.
`visual`	a logical value, that switches on/off the graphical user interface.
`...`	currently not used...

Details

The function creates an RBF neural network on the basis of the function parameters. After the creation of the network the function trains it using the back-propagation algorithm using the inp and out parameter. This two parameters row number must be the same, else the function will stop with an error message.

If you use the weight or dist argument, than that variables won't be determined by random. This could be useful if you want to retrain your network. In that case use both of this two arguments in the same time.

The function works with normalized Gauss-functions, which width parameter will be the sigma argument. If you want to give the values, this argument should be a matrix, with rows equal the number of neurons in the first layer, and columns equal the number of neurons in the second layer. If the sigma argument is NaN, then the width of each Gauss function will be the half of the distance between the two nearest training samples times 1,1. If the sigma argument is exactly one number, then all sigma value will be that exact number.

The function has a graphical user interface that can be switched on and off, with the visual argument. If the graphical user interface is on, then the function could show the result of the approximation in a co-ordinate system, if it's a function with one parameter.

The result of the function is the parameters of the trained RBF neural network. Use the rbf function for information recall.

Value

list with 4 argument

`weight`	the weights of the network.

`dist`	the distortion of the network.
`neurons`	a numeric vector with length equals to the number of layers in the network, and the ith layer will contains neurons[i] neuron.
`sigma`	the width of the Gauss functions.

See Also

'rbf' for recalling; 'mlp' and 'mlptrain' for classification.

Examples

```
x<-t(matrix(-5:10*24,1,16));
y<-t(matrix(sin(pi/180*(-5:10*24)),1,16));
neurons<-8;
## Not run:
data<-rbftrain(x,neurons,y,sigma=NaN)
rbf(x,data$weight,data$dist,data$neurons,data$sigma)

## End(Not run)
```

Index