Package ‘brnn’

October 12, 2022

Version 0.9.2
Date 2022-05-16
Title Bayesian Regularization for Feed-Forward Neural Networks
Author Paulino Perez Rodriguez, Daniel Gianola
Maintainer Paulino Perez Rodriguez <perpdgo@colpos.mx>
Depends R (>= 3.5.0), Formula, truncnorm
Description Bayesian regularization for feed-forward neural networks.
LazyLoad true
License GPL-2
NeedsCompilation yes
Repository CRAN
Date/Publication 2022-05-16 21:00:02 UTC

R topics documented:

  brnn ................................................................. 2
  brnn_extended ................................................... 7
  brnn_ordinal .................................................... 10
  D ................................................................. 13
  estimate.trace ................................................... 14
  G ................................................................. 15
  GOord ........................................................... 16
  initnw ............................................................ 16
  jacobian .......................................................... 18
  normalize ......................................................... 18
  partitions ......................................................... 18
  pheno ............................................................. 19
  phenoOrd ........................................................ 19
  predict.brnn ..................................................... 20
  predict.brnn_extended .......................................... 20
  predict.brnn_ordinal ............................................ 21
  twoinput ........................................................ 22
  un_normalize ..................................................... 22
Description

The `brnn` function fits a two layer neural network as described in MacKay (1992) and Foresee and Hagan (1997). It uses the Nguyen and Widrow algorithm (1990) to assign initial weights and the Gauss-Newton algorithm to perform the optimization. This function implements the functionality of the function `trainbr` in Matlab 2010b.

Usage

```r
brnn(x, ...)  
## S3 method for class 'formula'
brnn(formula, data, contrasts=NULL,...)
## Default S3 method:
brnn(x,y,neurons=2,normalize=TRUE,epochs=1000,mu=0.005,mu_dec=0.1,
    mu_inc=10,mu_max=1e10,min_grad=1e-10,change = 0.001,cores=1,
    verbose=FALSE,Monte_Carlo = FALSE,tol = 1e-06, samples = 40,...)
```

Arguments

- **formula**: A formula of the form `y ~ x1 + x2 + ...`
- **data**: Data frame from which variables specified in `formula` are preferentially to be taken.
- **x**: (numeric, `n × p`) incidence matrix.
- **y**: (numeric, `n`) the response data-vector (NAs not allowed).
- **neurons**: positive integer that indicates the number of neurons.
- **normalize**: logical, if TRUE will normalize inputs and output, the default value is TRUE.
- **epochs**: positive integer, maximum number of epochs(iterations) to train, default 1000.
- **mu**: positive number that controls the behaviour of the Gauss-Newton optimization algorithm, default value 0.005.
- **mu_dec**: positive number, is the mu decrease ratio, default value 0.1.
- **mu_inc**: positive number, is the mu increase ratio, default value 10.
- **mu_max**: maximum mu before training is stopped, strict positive number, default value `1 × 10^11`.
- **min_grad**: minimum gradient.
- **change**: The program will stop if the maximum (in absolute value) of the differences of the F function in 3 consecutive iterations is less than this quantity.
**cores** Number of cpu cores to use for calculations (only available in UNIX-like operating systems). The function detectCores in the R package parallel can be used to attempt to detect the number of CPUs in the machine that R is running, but not necessarily all the cores are available for the current user, because for example in multi-user systems it will depend on system policies. Further details can be found in the documentation for the parallel package.

**verbose** logical, if TRUE will print iteration history.

**Monte_Carlo** If TRUE it will estimate the trace of the inverse of the hessian using Monte Carlo procedures, see Bai et al. (1996) for more details. This routine calls the function estimate.trace() to perform the computations.

**tol** numeric tolerance, a tiny number useful for checking convergence in the Bai’s algorithm.

**samples** positive integer, number of Monte Carlo replicates to estimate the trace of the inverse, see Bai et al. (1996) for more details.

**contrasts** an optional list of contrasts to be used for some or all of the factors appearing as variables in the model formula.

... arguments passed to or from other methods.

**Details**

The software fits a two layer network as described in MacKay (1992) and Foresee and Hagan (1997). The model is given by:

\[ y_i = g(x_i) + e_i = \sum_{k=1}^{s} w_k g_k(b_k + \sum_{j=1}^{p} x_{ij} \beta_j^{[k]} ) + e_i, i = 1, ..., n \]

where:

- \( e_i \sim N(0, \sigma_e^2) \).
- \( s \) is the number of neurons.
- \( w_k \) is the weight of the \( k \)-th neuron, \( k = 1, ..., s \).
- \( b_k \) is a bias for the \( k \)-th neuron, \( k = 1, ..., s \).
- \( \beta_j^{[k]} \) is the weight of the \( j \)-th input to the net, \( j = 1, ..., p \).
- \( g_k(\cdot) \) is the activation function, in this implementation \( g_k(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \).

The software will minimize

\[ F = \beta E_D + \alpha E_W \]

where

- \( E_D = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \), i.e. the error sum of squares.
- \( E_W \) is the sum of squares of network parameters (weights and biases).
- \( \beta = \frac{1}{2\sigma_e^2} \).
- \( \alpha = \frac{1}{2\sigma^2} \), \( \sigma^2 \) is a dispersion parameter for weights and biases.
Value

object of class "brnn" or "brnn.formula". Mostly internal structure, but it is a list containing:

$theta  A list containing weights and biases. The first $s$ components of the list contains vectors with the estimated parameters for the $k$-th neuron, i.e. $(w_k, b_k, \beta_1^k, \ldots, \beta_p^k)'$.

$message  String that indicates the stopping criteria for the training process.

$alpha  $alpha$ parameter.

$beta  $beta$ parameter.

$gamma  effective number of parameters.

$Ew  The sum of the squares of the bias and weights.

$Ed  The sum of the squares between observed and predicted values.

References


See Also

predict.brnn

Examples

## Not run:

#Load the library
library(brnn)

#Example 1
#Noise triangle wave function, similar to example 1 in Foresee and Hagan (1997)

#Generating the data
x1=seq(0,0.23,length.out=25)
y1=4*x1+rnorm(25,sd=0.1)
x2=seq(0.25, 0.75, length.out=50)
y2=2-4*x2+rnorm(50, sd=0.1)
x3=seq(0.77, 1, length.out=25)
y3=4*x3-4+rnorm(25, sd=0.1)
x=c(x1, x2, x3)
y=c(y1, y2, y3)

# With the formula interface
out=brnn(y~x, neurons=2)

# With the default S3 method the call is
out=brnn(y=y, x=as.matrix(x), neurons=2)

plot(x, y, ylim=c(-1.5, 1.5),
     main="Bayesian Regularization for ANN 1-2-1")
lines(x, predict(out), col="blue", lty=2)
legend("topright", legend="Fitted model", col="blue", lty=2, bty="n")

###############################################################
# Example 2
# Sine wave function, example in the Matlab 2010b demo.

x = seq(-1, 0.5, length.out=100)
y = sin(2*pi*x)+rnorm(length(x), sd=0.1)

# With the formula interface
out=brnn(y~x, neurons=3)

# With the default method the call is
out=brnn(y=y, x=as.matrix(x), neurons=3)

plot(x, y)
lines(x, predict(out), col="blue", lty=2)
legend("bottomright", legend="Fitted model", col="blue", lty=2, bty="n")

###############################################################
# Example 3
# 2 Inputs and 1 output
# the data used in Paciorek and
# Schervish (2004). The data is from a two input one output function with Gaussian noise
# with mean zero and standard deviation 0.25

data(twoinput)

data=brnn(y~x1+x2, data=twoinput, neurons=10)

# With the default S3 method
out=brnn(y=as.vector(twoinput$y), x=as.matrix(cbind(twoinput$x1, twoinput$x2)), neurons=10)
f=function(x1, x2) predict(out, cbind(x1, x2))
x1 = seq(min(twoinput$x1), max(twoinput$x1), length.out=50)
x2 = seq(min(twoinput$x2), max(twoinput$x2), length.out=50)
z = outer(x1, x2, f) # calculating the density values

transformation_matrix = persp(x1, x2, z,
   main = "Fitted model",
   sub = expression(y == italic(g) ~(bold(x)) + e),
   col = "lightgreen", theta = 30, phi = 20, r = 50,
   d = 0.1, expand = 0.5, ltheta = 90, lphi = 180,
   shade = 0.75, ticktype = "detailed", nticks = 5)
points(trans3d(twoinput$x1, twoinput$x2, f(twoinput$x1, twoinput$x2),
   transformation_matrix), col = "red")

# Example 4
# Gianola et al. (2011).
# Warning, it will take a while

# Load the Jersey dataset
data(Jersey)

# Fit the model with the FULL DATA
# Formula interface
out = brnn(pheno$yield_devMilk ~ G, neurons = 2, verbose = TRUE)

# Obtain predictions and plot them against fitted values
plot(pheno$yield_devMilk, predict(out))

# Predictive power of the model using the SECOND set for 10 fold CROSS-VALIDATION
data = pheno
data$X = G
data$partitions = partitions

# Fit the model for the TESTING DATA
out = brnn(yield_devMilk ~ X,
   data = subset(data, partitions != 2), neurons = 2, verbose = TRUE)

# Plot the results
# Predicted vs observed values for the training set
par(mfrow = c(2, 1))
plot(out$y, predict(out), xlab = expression(hat(y)), ylab = "y")
cor(out$y, predict(out))

# Predicted vs observed values for the testing set
yhat_R_testing = predict(out, newdata = subset(data, partitions == 2))
ytesting = pheno$yield_devMilk[partitions == 2]
plot(ytesting, yhat_R_testing, xlab = expression(hat(y)), ylab = "y")
cor(ytesting, yhat_R_testing)

## End(Not run)
Description

The `brnn_extended` function fits a two layer neural network as described in MacKay (1992) and Foresee and Hagan (1997). It uses the Nguyen and Widrow algorithm (1990) to assign initial weights and the Gauss-Newton algorithm to perform the optimization. The hidden layer contains two groups of neurons that allow us to assign different prior distributions for two groups of input variables.

Usage

```r
brnn_extended(x, ...)
```

## S3 method for class 'formula'

```r
brnn_extended(formula, data, contrastsx=NULL, contrastsz=NULL, ...)
```

## Default S3 method:

```r
brnn_extended(x, y, z, neurons1, neurons2, normalize=TRUE, epochs=1000, mu=0.005, mu_dec=0.1, mu_inc=10, mu_max=1e10, min_grad=1e-10, change = 0.001, cores=1, verbose = FALSE, ...)
```

Arguments

- `formula` A formula of the form `y ~ x1 + x2 ... | z1 + z2 ...`, the `|` is used to separate the two groups of input variables.
- `data` Data frame from which variables specified in `formula` are preferentially to be taken.
- `y` (numeric, `n`) the response data-vector (NAs not allowed).
- `x` (numeric, `n × p`) incidence matrix for variables in group 1.
- `z` (numeric, `n × q`) incidence matrix for variables in group 2.
- `neurons1` positive integer that indicates the number of neurons for variables in group 1.
- `neurons2` positive integer that indicates the number of neurons for variables in group 2.
- `normalize` logical, if TRUE will normalize inputs and output, the default value is TRUE.
- `epochs` positive integer, maximum number of epochs to train, default 1000.
- `mu` positive number that controls the behaviour of the Gauss-Newton optimization algorithm, default value 0.005.
- `mu_dec` positive number, is the mu decrease ratio, default value 0.1.
- `mu_inc` positive number, is the mu increase ratio, default value 10.
- `mu_max` maximum mu before training is stopped, strict positive number, default value `1 × 10^{10}`.
- `min_grad` minimum gradient.
The program will stop if the maximum (in absolute value) of the differences of the F function in 3 consecutive iterations is less than this quantity.

Number of cpu cores to use for calculations (only available in UNIX-like operating systems). The function detectCores in the R package parallel can be used to attempt to detect the number of CPUs in the machine that R is running, but not necessarily all the cores are available for the current user, because for example in multi-user systems it will depend on system policies. Further details can be found in the documentation for the parallel package.

logical, if TRUE will print iteration history.

an optional list of contrasts to be used for some or all of the factors appearing as variables in the first group of input variables in the model formula.

an optional list of contrasts to be used for some or all of the factors appearing as variables in the second group of input variables in the model formula.

... arguments passed to or from other methods.

The software fits a two layer network as described in MacKay (1992) and Foresee and Hagan (1997). The model is given by:

\[ y_i = \sum_{k=1}^{s_1} w_1^k g_k (b_1^k + \sum_{j=1}^{p} x_{ij} \beta_1^{[k]}) + \sum_{k=1}^{s_2} w_2^k g_k (b_2^k + \sum_{j=1}^{q} z_{ij} \beta_2^{[k]}) + e_i, \quad i = 1, \ldots, n \]

- \( e_i \sim N(0, \sigma_e^2) \).
- \( g_k (\cdot) \) is the activation function, in this implementation \( g_k (x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \).

The software will minimize

\[ F = \beta E_D + \alpha \theta_1^T \theta_1 + \delta \theta_2^T \theta_2 \]

where

- \( E_D = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \), i.e. the sum of squared errors.
- \( \beta = \frac{1}{2\sigma_e^2} \).
- \( \alpha = \frac{1}{2\sigma_{\theta_1}^2}, \sigma_{\theta_1}^2 \) is a dispersion parameter for weights and biases for the associated to the first group of neurons.
- \( \delta = \frac{1}{2\sigma_{\theta_2}^2}, \sigma_{\theta_2}^2 \) is a dispersion parameter for weights and biases for the associated to the second group of neurons.

object of class "brnn_extended" or "brnn_extended.formula". Mostly internal structure, but it is a list containing:

- \$theta1 A list containing weights and biases. The first \( s_1 \) components of the list contain vectors with the estimated parameters for the \( k \)-th neuron, i.e. \((w_1^k, b_1^k, \beta_1^{[k]}, \ldots, \beta_p^{[k]})'\). \( s_1 \) corresponds to neurons1 in the argument list.
$\theta_2$ A list containing weights and biases. The first $s_2$ components of the list contains vectors with the estimated parameters for the $k$-th neuron, i.e. $(w_2^k, b_2^k, \beta_2^2[k], \ldots, \beta_2^q[k])'$. $s_2$ corresponds to neurons2 in the argument list.

$message$ String that indicates the stopping criteria for the training process.

References


See Also

predict.brnn_extended

Examples

## Not run:

# Example 5
# Warning, it will take a while

# Load the Jersey dataset
data(Jersey)

# Predictive power of the model using the SECOND set for 10 fold CROSS-VALIDATION
data=pheno
data$G=G
data$D=D
data$partitions=partitions

# Fit the model for the TESTING DATA for Additive + Dominant
out=brnn_extended(yield_devMilk ~ G | D,
data=subset(data,partitions!=2),
neurons1=2,neurons2=2,epochs=100,verbose=TRUE)

# Plot the results
# Predicted vs observed values for the training set
par(mfrow=c(2,1))
yhat_R_training=predict(out)
plot(out$y,yhat_R_training,xlab=expression(hat(y)),ylab="y")
cor(out$y,yhat_R_training)

# Predicted vs observed values for the testing set
newdata=subset(data,partitions==2,select=c(D,G))
ytesting=pheno$yield_devMilk[partitions==2]
yhat_R_testing=predict(out,newdata=newdata)
plot(ytesting,yhat_R_testing,xlab=expression(hat(y)),ylab="y")
cor(ytesting,yhat_R_testing)

## End(Not run)

### Description

The brnn_ordinal function fits a Bayesian Regularized Neural Network for Ordinal data.

#### Usage

```r
brnn_ordinal(x, ...)
```

```r
## S3 method for class 'formula'
brnn_ordinal(formula, data, contrasts=NULL, ...)
```

```r
## Default S3 method:
brnn_ordinal(x, y, 
neurons=2, 
normalize=TRUE, 
ePOCHS=1000, 
mu=0.005, 
mu_dec=0.1, 
mu_inc=10, 
mu_max=1e10, 
min_grad=1e-10, 
change_f=0.01, 
change_par=0.01, 
iter_EM=1000, 
verbose=FALSE, 
...)
```

#### Arguments

- `formula`: A formula of the form `y ~ x1 + x2 + ...`
- `data`: Data frame from which variables specified in `formula` are preferentially to be taken.
- `x`: (numeric, `n x p`) incidence matrix.
- `y`: (numeric, `n`) the response data-vector (NAs not allowed).
- `neurons`: positive integer that indicates the number of neurons.
normalize logical, if TRUE will normalize inputs and output, the default value is TRUE.
epochs positive integer, maximum number of epochs (iterations) to train, default 1000.
u positive number that controls the behaviour of the Gauss-Newton optimization algorithm, default value 0.005.
u_dec positive number, is the mu decrease ratio, default value 0.1.
u_inc positive number, is the mu increase ratio, default value 10.
u_max maximum mu before training is stopped, strict positive number, default value $1 \times 10^{10}$.
min_grad minimum gradient.
change_F the program will stop if the maximum (in absolute value) of the differences of the F function in 3 consecutive iterations is less than this quantity.
change_par the program will stop iterations of the EM algorithm when the maximum of absolute values of differences between parameters in two consecutive iterations is less than this quantity.
iter_EM positive integer, maximum number of iteration for the EM algorithm.
verbose logical, if TRUE will print iteration history.
contrasts an optional list of contrasts to be used for some or all of the factors appearing as variables in the model formula.
... arguments passed to or from other methods.

Details

The software fits a Bayesian Regularized Neural Network for Ordinal data. The model is an extension of the two layer network as described in MacKay (1992); Foresee and Hagan (1997), and Gianola et al. (2011). We use the latent variable approach described in Albert and Chib (1993) to model ordinal data, the Expectation maximization (EM) and Levenberg-Marquardt algorithm (Levenberg, 1944; Marquardt, 1963) to fit the model.

Following Albert and Chib (1993), suppose that $Y_1, ..., Y_n$ are observed and $Y_i$ can take values on $L$ ordered values. We are interested in modelling the probability $p_{ij} = P(Y_i = j)$ using the covariates $x_{i1}, ..., x_{ip}$. Let

$$g(x_i) = \sum_{k=1}^{s} w_k g_k(b_k + \sum_{j=1}^{p} x_{ij} \beta_j^{[k]}),$$

where:

- $s$ is the number of neurons.
- $w_k$ is the weight of the $k$-th neuron, $k = 1, ..., s$.
- $b_k$ is a bias for the $k$-th neuron, $k = 1, ..., s$.
- $\beta_j^{[k]}$ is the weight of the $j$-th input to the net, $j = 1, ..., p$.
- $g_k(\cdot)$ is the activation function, in this implementation $g_k(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1}$.

Let

$$Z_i = g(x_i) + e_i,$$

where:
• \( e_i \sim N(0, 1) \).
• \( Z_i \) is an unobserved (latent variable).

The output from the model for latent variable is related to observed data using the approach employed in the probit and logit ordered models, that is \( Y_i = j \) if \( \lambda_{j-1} < Z_i < \lambda_j \), where \( \lambda_j \) are a set of unknown thresholds. We assign prior distributions to all unknown quantities (see Albert and Chib, 1993; Gianola et al., 2011) for further details. The Expectation maximization (EM) and Levenberg-Marquardt algorithm (Levenberg, 1944; Marquardt, 1963) to fit the model.

Value

Object of class "brnn_ordinal". Mostly internal structure, but it is a list containing:

- \$theta: A list containing weights and biases. The first \( s \) components of the list contains vectors with the estimated parameters for the \( k \)-th neuron, i.e. \( (w_k, b_k, \beta_1^{[k]}, ..., \beta_p^{[k]})' \).
- \$threshold: A vector with estimates of thresholds.
- \$alpha: \( \alpha \) parameter.
- \$gamma: Effective number of parameters.

References


See Also

- `predict.brnn_ordinal`

Examples

```r
## Not run:
# Load the library
library(brnn)

# Load the dataset
data(GLS)
```
#Subset of data for location Harare
HarareOrd=subset(phenOOrd,Loc=="Harare")

#Eigen value decomposition for GOOrd keep those
#eigen vectors whose corresponding eigen-vectors are bigger than 1e-10
#and then compute principal components

evd=eigen(GOOrd)
evd$vectors=evd$vectors[,evd$value>1e-10]
evd$values=evd$values[evd$values>1e-10]
PC=evd$vectors
rownames(PC)=rownames(GOOrd)

#Response variable
y=phenOOrd$rating
gid=as.character(phenOOrd$Stock)

Z=model.matrix(~gid-1)
colnames(Z)=gsub("gid","",colnames(Z))

if(any(colnames(Z)!=rownames(PC))) stop("Ordering problem\n")

#Matrix of predictors for Neural net
X=Z%*%PC

#Cross-validation
set.seed(1)
testing=sample(1:length(y),size=as.integer(0.10*length(y)),replace=FALSE)
isNa=(1:length(y)%in%testing)
yTrain=y[!isNa]
XTrain=X[!isNa,]
nTest=sum(isNa)

neurons=2

fmOrd=brnn_ordinal(XTrain,yTrain,neurons=neurons,verbose=FALSE)

#Predictions for testing set
XTest=X[isNa,]
predictions=predict(fmOrd,XTest)
predictions

## End(Not run)
Description

This matrix was calculated by using the dominance incidence matrix derived from 33,267 Single Nucleotide Polymorphisms (SNPs) information on 297 individually cows,

\[ D = \frac{X_d X_d'}{2 \sum_{j=1}^{p} (p_j^2 + q_j^2) p_j q_j}, \]

where

- \( X_d \) is the design matrix for allele substitution effects for dominance.
- \( p_j \) is the frequency of the second allele at locus \( j \) and \( q_j = 1 - p_j \).

Source

University of Wisconsin at Madison, USA.

estimate.trace

Description

The estimate.trace function estimates the trace of the inverse of a positive definite and symmetric matrix using the algorithm developed by Bai et al. (1996). It is specially useful when the matrix is huge.

Usage

```
estimate.trace(A, tol=1E-6, samples=40, cores=1)
```

Arguments

- \( A \) (numeric), positive definite and symmetric matrix.
- \( \text{tol} \) numeric tolerance, a very small number useful for checking convergence in the Bai’s algorithm.
- \( \text{samples} \) integer, number of Monte Carlo replicates to estimate the trace of the inverse.
- \( \text{cores} \) Number of cpu cores to use for calculations (only available in UNIX-like operating systems).

References

Examples

```r
## Not run:
library(brnn)
data(Jersey)

#Estimate the trace of the inverse of G matrix
estimate.trace(G)

#The TRUE value
sum(diag(solve(G)))

## End(Not run)
```

---

**G**

*Genomic additive relationship matrix for the Jersey dataset.*

---

**Description**

A matrix, similar to this was used in Gianola et al. (2011) for predicting milk, fat and protein production in Jersey cows. In this software version we do not center the incidence matrix for the additive effects.

\[
G = \frac{X_a X_a'}{2 \sum_{j=1}^{p_j(1-p_j)}},
\]

where

- \(X_a\) is the design matrix for allele substitution effects for additivity.
- \(p_j\) is the frequency of the second allele at locus \(j\) and \(q_j = 1 - p_j\).

**Source**

University of Wisconsin at Madison, USA.

**References**

**G0rd**

*Genomic additive relationship matrix for the GLS dataset.*

**Description**

Genomic relationship matrix for the GLS dataset. The matrix was derived from 46347 markers for the 278 individuals. The matrix was calculated as follows $G = MM'/p$, where $M$ is the matrix of markers centered and standarized and $p$ is the number of markers.

**Source**

International Maize and Wheat Improvement Center (CIMMYT), Mexico.

**References**


---

**initnw**

*Initialize networks weights and biases*

**Description**

Function to initialize the weights and biases in a neural network. It uses the Nguyen-Widrow (1990) algorithm.

**Usage**

`initnw(neurons, p, n, npar)`

**Arguments**

- `neurons` Number of neurons.
- `p` Number of predictors.
- `n` Number of cases.
- `npar` Number of parameters to be estimate including only weights and biases, and should be equal to $neurons \times (1 + 1 + p) + 1$. 
Details

The algorithm is described in Nguyen-Widrow (1990) and in other books, see for example Sivanandam and Sumathi (2005). The algorithm is briefly described below.

• 1.- Compute the scaling factor $\theta = 0.7p^{1/n}$.

• 2.- Initialize the weight and biases for each neuron at random, for example generating random numbers from $U(-0.5, 0.5)$.

• 3.- For each neuron:
  – compute $\eta_k = \sqrt{\sum_{j=1}^{p} (\beta_{j}^{(k)})^2}$,
  – update $(\beta_{1}^{(k)}, ..., \beta_{p}^{(k)})'$,
    \[
    \beta_{j}^{(k)} = \frac{\theta \beta_{j}^{(k)}}{\eta_k}, j = 1, ..., p,
    \]
  – Update the bias ($b_k$) generating a random number from $U(-\theta, \theta)$.

Value

A list containing initial values for weights and biases. The first $s$ components of the list contains vectors with the initial values for the weights and biases of the $k$-th neuron, i.e. $(\omega_k, b_k, \beta_{1}^{(k)}, ..., \beta_{p}^{(k)})'$.

References


Examples

```r
# Not run:
# Load the library
library(brnn)

# Set parameters
neurons=3
p=4
n=10
npar=neurons*(1+1+p)+1
initnw(neurons=neurons,p=p,n=n,npar=npar)
```

# End(Not run)
**jacobian**

*Jacobian*

**Description**

Internal function for the calculation of the Jacobian.

**normalize**

*normalize*

**Description**

Internal function for normalizing the data. This function makes a linear transformation of the inputs such that the values lie between -1 and 1.

**Usage**

```r
normalize(x, base, spread)
```

**Arguments**

- **x**: a vector or matrix that needs to be normalized.
- **base**: If `x` is a vector, `base` is the minimum of `x`. If `x` is a matrix, `base` is a vector with the minimum for each of the columns of the matrix `x`.
- **spread**: if `x` is a vector, `spread` = `base` - `max(x)`. If `x` is a matrix, `spread` is a vector calculated for each of the columns of `x`.

**Details**

\[ z = 2 \times \frac{x - base}{spread} - 1 \]

**Value**

A vector or matrix with the resulting normalized values.

**partitions**

*Partitions for cross validation (CV)*

**Description**

Is a vector \((297 \times 1)\) that assigns observations to 10 disjoint sets; the assignment was generated at random. This is used later to conduct a 10-fold CV.

**Source**

University of Wisconsin at Madison, USA.
Phenotypic information for Jersey

**Description**

The format of the phenotype dataframe is animal ID, herd, number of lactations, average milk yield, average fat yield, average protein yield, yield deviation for milk, yield deviation for fat, and yield deviation for protein. Averages are adjusted for age, days in milk, and milking frequency. Yield deviations are adjusted further, for herd-year-season effect. You may wish to use yield deviations, because there are relatively few cows per herd (farmers don’t pay to genotype all of their cows, just the good ones).

**Source**

University of Wisconsin at Madison, USA.

**References**


Phenotypic information for GLS (ordinal trait)

**Description**

The format of the phenotype dataframe is Location (Loc), Replicate (Rep), Genotype ID (Stock) and rating (ordinal score). Gray Leaf Spot (GLS) is a disease caused by the fungus *Cercospora zeae-maydis*. This dataset consists of genotypic and phenotypic information for 278 maize lines from the Drought Tolerance Maize (DTMA) project of CIMMYT’s Global Maize Program. The dataset includes information on disease severity measured on an ordinal scale with 5 points: 1= no disease, 2= low infection, 3=moderate infection, 4=high infection and 5= totally infected.

**Source**

International Maize and Wheat Improvement Center (CIMMYT), Mexico.

**References**

predict.brnn

Description
The function produces the predictions for a two-layer feed-forward neural network.

Usage
```r
## S3 method for class 'brnn'
predict(object, newdata, ...)
```

Arguments
- `object`: an object of the class `brnn` as returned by `brnn`
- `newdata`: matrix or data frame of test examples. A vector is considered to be a row vector comprising a single case.
- `...`: arguments passed to or from other methods.

Details
This function is a method for the generic function `predict()` for class "brnn". It can be invoked by calling `predict(x)` for an object `x` of the appropriate class, or directly by calling `predict.brnn(x)` regardless of the class of the object.

Value
A vector containing the predictions

predict.brnn_extended

Description
The function produces the predictions for a two-layer feed-forward neural network.

Usage
```r
## S3 method for class 'brnn_extended'
predict(object, newdata, ...)
```

Arguments
- `object`: an object of the class `brnn_extended` as returned by `brnn_extended`
- `newdata`: matrix or data frame of test examples. A vector is considered to be a row vector comprising a single case.
- `...`: arguments passed to or from other methods.
Details

This function is a method for the generic function \texttt{predict()} for class "brnn\_extended". It can be invoked by calling \texttt{predict(x)} for an object \texttt{x} of the appropriate class, or directly by calling \texttt{predict.brnn(x)} regardless of the class of the object.

Value

A vector containing the predictions

Description

The function produces the predictions for a two-layer feed-forward neural network for ordinal data.

Usage

\begin{verbatim}
## S3 method for class 'brnn\_ordinal'
predict(object, newdata, ...)
\end{verbatim}

Arguments

- \texttt{object}: an object of the class \texttt{brnn\_ordinal} as returned by \texttt{brnn\_ordinal}
- \texttt{newdata}: matrix or data frame of test examples. A vector is considered to be a row vector comprising a single case.
- \texttt{...}: arguments passed to or from other methods.

Details

This function is a method for the generic function \texttt{predict()} for class "brnn\_ordinal". It can be invoked by calling \texttt{predict(x)} for an object \texttt{x} of the appropriate class, or directly by calling \texttt{predict.brnn\_ordinal(x)} regardless of the class of the object.

Value

A list with components:

- \texttt{class}: Predicted class (an integer).
- \texttt{probability}: Posterior probability of belonging to a class given the covariates.
twoinput

2 Inputs and 1 output.

Description
The data used in Paciorek and Schervish (2004). This is a data.frame with 3 columns, columns 1 and 2 corresponds to the predictors and column 3 corresponds to the target.

References

un_normalize

Description
Internal function for going back to the original scale.

Usage
un_normalize(z, base, spread)

Arguments
z
a vector or matrix with values normalized between -1 and 1, this vector was obtained when normalizing a vector or matrix x.
base
If z is a vector, base is the minimum of x. If x is a matrix, base is a vector with the minimum for each of the columns of the matrix x.
spread
if z is a vector, spread=base-max(x). If x is a matrix, spread is a vector calculated for each of the columns of x.

Details
x=base+0.5*spread*(z+1)

Value
A vector or matrix with the resulting un normalized values.
Index

* datasets
  D, 13
  G, 15
  GOrd, 16
  partitions, 18
  pheno, 19
  phenoOrd, 19
  twoinput, 22

* matrix
  estimate.trace, 14

* models
  brnn, 2
  brnn_extended, 7
  brnn_ordinal, 10
  initnw, 16
  jacobian, 18
  normalize, 18
  predict.brnn, 20
  predict.brnn_extended, 20
  predict.brnn_ordinal, 21
  un_normalize, 22

brnn, 2
brnn_extended, 7
brnn_ordinal, 10

coeff.brnn (brnn), 2
coeff.brnn_extended (brnn_extended), 7
D, 13
estimate.trace, 14
G, 15
GOrd, 16

initnw, 16
jacobian, 18
normalize, 18

partitions, 18
pheno, 19
phenoOrd, 19
predict.brnn, 4, 20
predict.brnn_extended, 9, 20
predict.brnn_ordinal, 12, 21
print.brnn (brnn), 2
print.brnn_extended (brnn_extended), 7
print.brnn_ordinal (brnn_ordinal), 10

summary.brnn (brnn), 2
summary.brnn_extended (brnn_extended), 7
summary.brnn_ordinal (brnn_ordinal), 10
twoinput, 22
un_normalize, 22

23