Package 'gnn'

July 22, 2025

Version 0.0-4

Title Generative Neural Networks

Description Tools to set up, train, store, load, investigate and analyze generative neural networks. In particular, functionality for generative moment matching networks is provided.

Maintainer Marius Hofert <mhofert@hku.hk>

Depends R (>= 3.5.0)

Imports keras, tensorflow, methods, R6, qrng, tools, copula

Suggests nvmix, qrmdata, RColorBrewer

License GPL (>= 3)

SystemRequirements TensorFlow (https://www.tensorflow.org/)

NeedsCompilation no

Repository CRAN

Encoding UTF-8

Date/Publication 2024-02-28 08:30:11 UTC

Author Marius Hofert [aut, cre] (ORCID: <https://orcid.org/0000-0001-8009-4665>), Avinash Prasad [aut]

Contents

`catch` *Catching Results, Warnings and Errors Simultaneously*

Description

Catches results, warnings and errors.

Usage

```
catch(expr)
```

Arguments

`expr`	expression to be evaluated, typically a function call.

Details

This function is particularly useful for large(r) simulation studies to not fail until finished.

Value

`list` with components:

`value`	value of `expr` or `NULL` in case of an error.
`warning`	warning message (see `simpleWarning` or `warning()`) or `NULL` in case of no warning.
`error`	error message (see `simpleError` or `stop()`) or `NULL` in case of no error.

Author(s)

Marius Hofert (based on `doCallWE()` and `tryCatch.W.E()` in the R package **simsalapar**).

Examples

```
library(gnn) # for being standalone

catch(log(2))
catch(log(-1))
catch(log("a"))
```

ffGNN *Feedforward for Generative Neural Networks*

Description

Feedforward method for objects of S3 class "gnn_GNN".

Usage

```
## S3 method for class 'gnn_GNN'
ffGNN(x, data)
```

Arguments

x	object of S3 class "gnn_GNN".
data	matrix to be fed forward through x.

Value

The output matrix of x when fed with data.

Author(s)

Marius Hofert

Examples

```
if(TensorFlow_available()) { # rather restrictive (due to R-Forge, winbuilder)
library(gnn) # for being standalone

## Define dummy model
d <- 2 # bivariate case
GMMN <- FNN(c(d, 300, d)) # Feedforward NN with MMD loss (a GMMN; random weights)

## Feedforward
n <- 3
set.seed(271)
X <- ffGNN(GMMN, data = matrix(runif(n * d), ncol = d))
stopifnot(dim(X) == c(n, d))

}
```

find_box *Box Numbers (Multivariate) Points Fall Into*

Description

Finding the numbers of boxes that given (multivariate) points fall into (the default is similar to `findInterval()` but other methods are provided, too).

Usage

```
find_box(x, endpoints = NULL,
         method = c("per.dim", "lexicographic", "nested", "diagonal"),
         rightmost.closed = TRUE, left.open = TRUE, ...)
```

Arguments

`x`	(n, d)-matrix containing n d-dimensional data points, $n \geq 1$, $d \geq 1$.
`endpoints`	d-`list` containing numeric vectors of endpoints of the intervals in each dimension (each of the d elements is an argument `vec` as required by `findInterval()`).
`method`	`character` string indicating the method to be used. Available are:
	`"per.dim"` the default. Each row `x[i,]` of `x` produces d numbers, where the jth indicates in which interval `x[i,j]` falls. This is essentially `findInterval()` applied to the d coordinate samples (the columns of `x`).
	`"lexicographic"` Each row `x[i,]` produces one number, indicating in which box `x[i,]` falls if all *nonempty* boxes are numbered consecutively in lexicographic order.
	`"nested"` Each row `x[i,]` produces one number, indicating in which box `x[i,]` falls along the d-dimensional diagonal in a nested way, with a non-nested middle part if the number of interval endpoints per dimension is even (note that this method requires all elements of `endpoints` to have the same length, so that the diagonal is well-defined).
	`"diagonal"` Each row `x[i,]` produces one number, indicating in which box `x[i,]` falls along the d-dimensional diagonal (note that this method requires all elements of `endpoints` to have the same length, so that the diagonal is well-defined).
`rightmost.closed`	see `findInterval()` (note the different default here).
`left.open`	see `findInterval()` (note the different default here).
`...`	additional arguments passed to the underlying `findInterval()`.

Details

The box numbers can be used, for example, to color points; see the examples below.

Value

`"per.dim"` (n, d)-matrix of box numbers per dimension.

`"lexicographic"`, `"nested"`, `"diagonal"` n-vector with box numbers.

Note that, as `findInterval()`, 0 means 'in no box'.

Author(s)

Marius Hofert

Examples

```
## Example data
n <- 1000
d <- 2
set.seed(271)
U <- matrix(runif(n * d), ncol = d) # (n, d)-matrix of data (here: in [0,1]^d)


### 1 Basic example calls ##################################################

## Define endpoints and evaluate for different methods
epts <- seq(0, 1, by = 1/5) # 5 boxes per dimension
find_box(U, endpoints = epts)[1:10,] # default "per.dim" (first 10 points only)
boxes.lexi <- find_box(U, endpoints = epts, method = "lexicographic")
boxes.nest <- find_box(U, endpoints = epts, method = "nested")
boxes.diag <- find_box(U, endpoints = epts, method = "diagonal")

## Special cases
## First row of U (n = 1)
U[1,] # ~= (0.25, 0.14)
stopifnot(find_box(U[1, 1:2], endpoints = epts) == c(2, 1))
stopifnot(find_box(U[1, 1:2], endpoints = epts, method = "lexicographic") == 1)
## Note concerning the last line: It's 1 because all other boxes are empty
stopifnot(find_box(U[1, 1:2], endpoints = epts, method = "nested") == 2)
stopifnot(find_box(U[1, 1:2], endpoints = epts, method = "diagonal") == 0)
## Single number U[1,1] (d = 1)
U[1,1] # ~= 0.25
stopifnot(find_box(U[1,1], endpoints = epts) == 2)
stopifnot(find_box(U[1,1], endpoints = epts, method = "lexicographic") == 1)
stopifnot(find_box(U[1,1], endpoints = epts, method = "nested") == 2)
stopifnot(find_box(U[1,1], endpoints = epts, method = "diagonal") == 2)


### 2 Coloring points in lexicographic ordering ############################

## Define color palette
library(RColorBrewer)
basecols <- c("#000000", brewer.pal(8, name = "Dark2")[c(8,7,3,1,5,4,2,6)])
mypal <- function(n) rep_len(basecols, length.out = n)

## Colors
```

```
ncols <- diff(range(boxes.lexi)) + 1 # maximal number of colors needed
palette(mypal(ncols)) # set palette according to maximum number of colors needed


## Boxes of equal size
boxes.lexi <- find_box(U, endpoints = epts, method = "lexicographic")
cols <- if(min(boxes.lexi) == 0) boxes.lexi + 1 else boxes.lexi
plot(U, pch = 20, xlab = expression(U[1]), ylab = expression(U[2]), col = cols)
abline(v = epts, h = epts, col = "gray50") # guides

## Boxes of different sizes and numbers
epts. <- list(seq(0.2, 1, by = 1/5), seq(1/3, 1, by = 1/3))
boxes.lexi <- find_box(U, endpoints = epts., method = "lexicographic")
cols <- if(min(boxes.lexi) == 0) boxes.lexi + 1 else boxes.lexi
plot(U, pch = 20, xlab = expression(U[1]), ylab = expression(U[2]), col = cols)
abline(v = epts.[[1]], h = epts.[[2]], col = "gray50")


### 3 Coloring points along the diagonal in a nested way ########################

## Boxes of equal size (with 'middle' part)
boxes.nest <- find_box(U, endpoints = epts, method = "nested")
cols <- if(min(boxes.nest) == 0) boxes.nest + 1 else boxes.nest # color numbers
plot(U, pch = 20, xlab = expression(U[1]), ylab = expression(U[2]), col = cols)
abline(v = epts, h = epts, col = "gray50") # guides

## Boxes of different sizes (without 'middle' part; have to be the same number of
## boxes per dimension, otherwise there is no obvious 'diagonal')
epts. <- lapply(1:d, function(j) c(0, 0.1, 0.3, 0.6, 1)) # 4 boxes per dimension
boxes.nest <- find_box(U, endpoints = epts., method = "nested")
cols <- if(min(boxes.nest) == 0) boxes.nest + 1 else boxes.nest # color numbers
plot(U, pch = 20, xlab = expression(U[1]), ylab = expression(U[2]), col = cols)
abline(v = epts.[[1]], h = epts.[[2]], col = "gray50") # guides


### 4 Coloring points along the diagonal ########################################

## Boxes of equal size
boxes.diag <- find_box(U, endpoints = epts, method = "diagonal")
cols <- if(min(boxes.diag) == 0) boxes.diag + 1 else boxes.diag # color numbers
plot(U, pch = 20, xlab = expression(U[1]), ylab = expression(U[2]), col = cols)
abline(v = epts, h = epts, col = "gray50") # guides

## Boxes of different sizes (have to be the same number of
## boxes per dimension, otherwise there is no obvious 'diagonal')
epts. <- lapply(1:d, function(j) c(0, 0.05, 0.1, 0.3, 0.6, 1))
boxes.diag <- find_box(U, endpoints = epts., method = "diagonal")
cols <- if(min(boxes.diag) == 0) boxes.diag + 1 else boxes.diag # color numbers
plot(U, pch = 20, xlab = expression(U[1]), ylab = expression(U[2]), col = cols)
abline(v = epts.[[1]], h = epts.[[2]], col = "gray50") # guides
```

fitGNN *Functions and Methods for Training of Generative Neural Networks*

Description

Functions and methods for training generative neural networks.

Usage

```
## S3 method for class 'gnn_GNN'
fitGNN(x, data, batch.size = nrow(data), n.epoch = 100,
    prior = NULL, max.n.prior = 5000, verbose = 2, ...)
## S3 method for class 'gnn_GNN'
fitGNNonce(x, data, batch.size = nrow(data), n.epoch = 100,
    prior = NULL, verbose = 2, file = NULL, name = NULL, ...)
## S3 method for class 'gnn_GNN'
is.trained(x)
## S3 method for class 'list'
is.trained(x)
```

Arguments

x	**fitGNN(), fitGNNonce(), is.trained.gnn_GNN()** object of class "gnn_GNN" to be trained. **is.trained.gnn_GNN()** object of class "gnn_GNN" to be trained or a list of such.
data	(n, d)-matrix containing the n d-dimensional observations of the training data.
batch.size	number of samples used per stochastic gradient step.
n.epoch	number of epochs (one epoch equals one pass through the complete training dataset while updating the GNN's parameters through stochastic gradient steps).
prior	(n, d)-matrix of prior samples; see also rPrior(). If prior = NULL a sample of independent N(0,1) random variates is generated.
max.n.prior	maximum number of prior samples stored in x after training.
verbose	integer verbose level. Choices are: **0** silent (no output). **1** progress bar (via txtProgressBar()). **2** output after each block of epochs (block size is ceiling(n.epoch/10) if n.epoch <= 100 and ceiling(sqrt(n.epoch)) if n.epoch > 100). **3** output after each expoch.
file	NULL or a character string specifying the file in which the trained GNN(s) is (are) saved. If file is provided and the specified file exists, it is loaded and returned via load_gnn().
name	character string giving the name under which the fitted x is saved (if NULL the fitted x is saved under the name "x").
...	additional arguments passed to the underlying fit() (which is keras:::fit.keras.engine.training.

Value

fitGNN() the trained x.

fitGNNonce() object of class as x with the trained GNN.

is.trained.gnn_GNN() logical indicating whether x is trained.

is.trained.list() logical of length length(x) indicating, for each component, whether it is trained.

Author(s)

Marius Hofert

See Also

FNN(), save_gnn(), load_gnn().

FNN *Generative Moment Matching Network*

Description

Constructor for a generative feedforward neural network (FNN) model, an object of S3 class "gnn_FNN".

Usage

```
FNN(dim = c(2, 2), activation = c(rep("relu", length(dim) - 2), "sigmoid"),
    batch.norm = FALSE, dropout.rate = 0, loss.fun = "MMD", n.GPU = 0, ...)
```

Arguments

dim	integer vector of length at least two, giving the dimensions of the input layer, the hidden layer(s) (if any) and the output layer (in this order).
activation	character vector of length length(dim) - 1 specifying the activation functions for all hidden layers and the output layer (in this order); note that the input layer does not have an activation function.
loss.fun	loss function specified as character or function.
batch.norm	logical indicating whether batch normalization layers are to be added after each hidden layer.
dropout.rate	numeric value in [0,1] specifying the fraction of input to be dropped; see the rate parameter of layer_dropout(). Note that only if positive, dropout layers are added after each hidden layer.
n.GPU	non-negative integer specifying the number of GPUs available if the GPU version of TensorFlow is installed. If positive, a (special) multiple GPU model for data parallelism is instantiated. Note that for multi-layer perceptrons on a few GPUs, this model does not yet yield any scale-up computational factor (in fact, currently very slightly negative scale-ups are likely due to overhead costs).
...	additional arguments passed to loss().

Details

The S3 class "gnn_FNN" is a subclass of the S3 class "gnn_GNN" which in turn is a subclass of "gnn_Model".

Value

FNN() returns an object of S3 class "gnn_FNN" with components

model FNN model (a **keras** object inheriting from the R6 classes "keras.engine.training.Model", "keras.engine.network.Network", "keras.engine.base_layer.Layer" and "python.builtin.object", or a raw object).

type character string indicating the type of model.

dim see above.

activation see above.

batch.norm see above.

dropout.rate see above.

n.param number of trainable, non-trainable and total number of parameters.

loss.type type of loss function (character).

n.train number of training samples (NA_integer_ unless trained).

batch.size batch size (NA_integer_ unless trained).

n.epoch number of epochs (NA_integer_ unless trained).

loss numeric(n.epoch) containing the loss function values per epoch.

time object of S3 class "proc_time" containing the training time (if trained).

prior matrix containing a (sub-)sample of the prior (if trained).

Author(s)

Marius Hofert and Avinash Prasad

References

Li, Y., Swersky, K. and Zemel, R. (2015). Generative moment matching networks. *Proceedings of Machine Learning Research*, **37** (International Conference on Maching Learning), 1718–1727. See http://proceedings.mlr.press/v37/li15.pdf (2019-08-24)

Dziugaite, G. K., Roy, D. M. and Ghahramani, Z. (2015). Training generative neural networks via maximum mean discrepancy optimization. *AUAI Press*, 258–267. See http://www.auai.org/uai2015/proceedings/papers/230.p (2019-08-24)

Hofert, M., Prasad, A. and Zhu, M. (2020). Quasi-random sampling for multivariate distributions via generative neural networks. *Journal of Computational and Graphical Statistics*, doi:10.1080/10618600.2020.1868302.

Hofert, M., Prasad, A. and Zhu, M. (2020). Multivariate time-series modeling with generative neural networks. See https://arxiv.org/abs/2002.10645.

Hofert, M. Prasad, A. and Zhu, M. (2020). Applications of multivariate quasi-random sampling with neural networks. See https://arxiv.org/abs/2012.08036.

Examples

```
if(TensorFlow_available()) { # rather restrictive (due to R-Forge, winbuilder)
library(gnn) # for being standalone

## Training data
d <- 2 # bivariate case
P <- matrix(0.9, nrow = d, ncol = d); diag(P) <- 1 # correlation matrix
ntrn <- 60000 # training data sample size
set.seed(271)
library(nvmix)
X <- abs(rNorm(ntrn, scale = P)) # componentwise absolute values of N(0,P) sample

## Plot a subsample
m <- 2000 # subsample size for plots
opar <- par(pty = "s")
plot(X[1:m,], xlab = expression(X[1]), ylab = expression(X[2])) # plot |X|
U <- apply(X, 2, rank) / (ntrn + 1) # pseudo-observations of |X|
plot(U[1:m,], xlab = expression(U[1]), ylab = expression(U[2])) # visual check

## Model 1: A basic feedforward neural network (FNN) with MSE loss function
fnn <- FNN(c(d, 300, d), loss.fun = "MSE") # define the FNN
fnn <- fitGNN(fnn, data = U, n.epoch = 40) # train with batch optimization
plot(fnn, kind = "loss") # plot the loss after each epoch

## Model 2: A GMMN (FNN with MMD loss function)
gmmn <- FNN(c(d, 300, d)) # define the GMMN (initialized with random weights)
## For training we need to use a mini-batch optimization (batch size < nrow(U)).
## For a fair comparison (same number of gradient steps) to NN, we use 500
## samples (25% = 4 gradient steps/epoch) for 10 epochs for GMMN.
library(keras) # for callback_early_stopping()
## We monitor the loss function and stop earlier if the loss function
## over the last patience-many epochs has changed by less than min_delta
## in absolute value. Then we keep the weights that led to the smallest
## loss seen throughout training.
gmmn <- fitGNN(gmmn, data = U, batch.size = 500, n.epoch = 10,
               callbacks = callback_early_stopping(monitor = "loss",
                                                   min_delta = 1e-3, patience = 3,
                                                   restore_best_weights = TRUE))
plot(gmmn, kind = "loss") # plot the loss after each epoch
## Note:
## - Obviously, in a real-world application, batch.size and n.epoch
##   should be (much) larger (e.g., batch.size = 5000, n.epoch = 300).
## - Training is not reproducible (due to keras).

## Model 3: A FNN with CvM loss function
fnnCvM <- FNN(c(d, 300, d), loss.fun = "CvM")
fnnCvM <- fitGNN(fnnCvM, data = U, batch.size = 500, n.epoch = 10,
                 callbacks = callback_early_stopping(monitor = "loss",
                                                     min_delta = 1e-3, patience = 3,
                                                     restore_best_weights = TRUE))
plot(fnnCvM, kind = "loss") # plot the loss after each epoch
```

```
## Sample from the different models
set.seed(271)
V.fnn <- rGNN(fnn, size = m)
set.seed(271)
V.gmmn <- rGNN(gmmn, size = m)
set.seed(271)
V.fnnCvM <- rGNN(fnnCvM, size = m)

## Joint plot of training subsample with GMMN PRNs. Clearly, the MSE
## cannot be used to learn the distribution correctly.
layout(matrix(1:4, ncol = 2, byrow = TRUE))
plot(U[1:m,], xlab = expression(U[1]), ylab = expression(U[2]), cex = 0.2)
mtext("Training subsample", side = 4, line = 0.4, adj = 0)
plot(V.fnn,    xlab = expression(V[1]), ylab = expression(V[2]), cex = 0.2)
mtext("Trained NN with MSE loss", side = 4, line = 0.4, adj = 0)
plot(V.gmmn,  xlab = expression(V[1]), ylab = expression(V[2]), cex = 0.2)
mtext("Trained NN with MMD loss", side = 4, line = 0.4, adj = 0)
plot(V.fnnCvM,    xlab = expression(V[1]), ylab = expression(V[2]), cex = 0.2)
mtext("Trained NN with CvM loss", side = 4, line = 0.4, adj = 0)

## Joint plot of training subsample with GMMN QRNs
library(qrng) # for sobol()
V.fnn.    <- rGNN(fnn,    size = m, method = "sobol", randomize = "Owen")
V.gmmn.   <- rGNN(gmmn,   size = m, method = "sobol", randomize = "Owen")
V.fnnCvM. <- rGNN(fnnCvM, size = m, method = "sobol", randomize = "Owen")
plot(U[1:m,], xlab = expression(U[1]), ylab = expression(U[2]), cex = 0.2)
mtext("Training subsample", side = 4, line = 0.4, adj = 0)
plot(V.fnn., xlab = expression(V[1]), ylab = expression(V[2]), cex = 0.2)
mtext("Trained NN with MSE loss", side = 4, line = 0.4, adj = 0)
plot(V.gmmn., xlab = expression(V[1]), ylab = expression(V[2]), cex = 0.2)
mtext("Trained NN with MMD loss", side = 4, line = 0.4, adj = 0)
plot(V.fnnCvM., xlab = expression(V[1]), ylab = expression(V[2]), cex = 0.2)
mtext("Trained NN with CvM loss", side = 4, line = 0.4, adj = 0)
layout(1)
par(opar)
}
```

GNN_basics *Basic Functions and Methods*

Description

Basic functions and methods for objects of S3 class "gnn_GNN".

Usage

```
## S3 method for class 'gnn_GNN'
print(x, ...)
## S3 method for class 'gnn_GNN'
str(object, ...)
```

```
## S3 method for class 'gnn_GNN'
summary(object, ...)
## S3 method for class 'gnn_GNN'
dim(x)
## S3 method for class 'gnn_GNN'
is.GNN(x)
## S3 method for class 'list'
is.GNN(x)
```

Arguments

x	**print(), dim()** object of S3 class "gnn_GNN".
	is.GNN() object of S3 class "gnn_GNN" or a list of such.
object	object of S3 class "gnn_GNN".
...	currently not used.

Value

print() return value of the print() method for objects of class "list".

str() nothing, as str() returns nothing when applied to objects of class "list".

summary() return value of the summary() method for objects of class "list".

dim() slot dim of x, so a vector of dimensions of input, hidden and output layers.

is.GNN() logical of length equal to the length of x indicating, for each component, whether it is an object of class "gnn_GNN".

Author(s)

Marius Hofert

Examples

```
if(TensorFlow_available()) { # rather restrictive (due to R-Forge, winbuilder)
library(gnn) # for being standalone

d <- 2
dim <- c(d, 300, d) # dimensions of the input, hidden and output layers
GMMN <- FNN(dim) # define the GMMN model
stopifnot(is.GNN(GMMN)) # check for being a GNN
GMMN # print() method
str(GMMN) # str() method
summary(GMMN) # summary() method
stopifnot(dim(GMMN) == c(d, 300, d)) # dim() method

}
```

loss *Loss Function*

Description

Implementation of various loss functions to measure statistical discrepancy between two datasets.

Usage

```
loss(x, y, type = c("MMD", "CvM", "MSE", "BCE"), ...)

MMD(x, y, ...)
CvM(x, y)
```

Arguments

x	2d-tensor or (n, d)-matrix (during training, n is the batch size and d is the dimension of the input dataset).
y	2d-tensor or (m, d)-matrix (during training, m is the batch size (and typically equal to n) and d is the dimension of the input dataset).
type	character string indicating the type of loss used. Currently available are the (kernel) maximum mean discrepancy ("MMD", calling MMD()), the Cramer-von Mises statistc ("CvM", calling CvM()) of Rémillard and Scaillet (2009), the mean squared error ("MSE") and the binary cross entropy ("BCE").
...	additional arguments passed to the underlying functions, most notably bandwidth (a number or numeric vector of bandwidths for the radial basis function kernels) in case type = "MMD".

Value

loss() returns a 0d tensor containing the loss.

MMD() and CvM() return a 0d tensor (if x and y are tensors) or numeric(1) (if x or y are R matrices).

Author(s)

Marius Hofert and Avinash Prasad

References

Kingma, D. P. and Welling, M. (2014). Stochastic gradient VB and the variational auto-encoder. *Second International Conference on Learning Representations (ICLR)*. See https://keras.rstudio.com/articles/examples/variat

Rémillard, B. and Scaillet, O. (2009). Testing for equality between two copulas. *Journal of Multivariate Analysis* **100**, 377–386.

See Also

FNN() where loss() is used.

plot *Functions for Plotting*

Description

Functions for plotting.

Usage

```
## S3 method for class 'gnn_GNN'
plot(x, kind = c("scatter", "loss"), max.n.samples = NULL,
    type = NULL, xlab = NULL, ylab = NULL,
    y2lab = NULL, labels = "X", pair = NULL, ...)
```

Arguments

x	trained object of class "gnn_GNN" whose loss function (loss per epoch of training) is to be plotted.
kind	character() indicating the type of plot.
max.n.samples	maximal number of samples to be plotted.
type	line type; see plot().
xlab	x-axis label; see plot().
ylab	y-axis label; see plot().
y2lab	secondary y-axis label.
labels	character() vector indicating the labels to be used; if of length 1, then the base label to be used.
pair	numeric(2) containing the indices of the pair to be plotted.
...	additional arguments passed to the underlying plot().

Value

Plot by side-effect.

Author(s)

Marius Hofert

See Also

fitGNN().

raw_keras *Convert GNN model Slots to raw or keras Objects*

Description

Keras objects cannot be saved like other R objects. The methods `as.raw()` and `as.keras()` can be used to convert the `model` slots of objects of S3 class `"gnn_GNN"` to `"raw"` objects (which can be saved) or `"keras.engine.training.Model"` objects (which can be trained).

Usage

```
## S3 method for class 'gnn_GNN'
as.raw(x)
## S3 method for class 'gnn_GNN'
as.keras(x)
```

Arguments

x	object of S3 class `"gnn_GNN"`.

Value

object of S3 class `"gnn_GNN"` with slot `method` converted by the respective method if necessary.

Author(s)

Marius Hofert

rGNN *Sampling from a Generative Neural Network*

Description

Sampling method for objects of S3 class `"gnn_GNN"`.

Usage

```
## S3 method for class 'gnn_GNN'
rGNN(x, size, prior = NULL, pobs = FALSE, ...)
```

Arguments

x	object of S3 class "gnn_GNN".
size	sample size, a positive integer. Ignored if prior is a matrix.
prior	one of
	NULL the default, generates independent N(0,1) realizations as prior sample.
	matrix passes the given sample through the GNN x. Such a matrix is internally (if prior = NULL) and typically obtained via rPrior().
pobs	logical indicating whether the pseudo-observations of the generated samples should be returned.
...	additional arguments passed to the underlying rPrior() if prior = NULL.

Value

(size, dim(x)[1])-matrix of samples.

Author(s)

Marius Hofert

Examples

```
if(TensorFlow_available()) { # rather restrictive (due to R-Forge, winbuilder)
library(gnn) # for being standalone

## Define dummy model
d <- 2 # bivariate case
GMMN <- FNN(c(d, 300, d)) # Feedforward NN with MMD loss (a GMMN; random weights)

## Sampling
n <- 3
(X1 <- rGNN(GMMN, size = n)) # default (independent N(0,1) samples as prior)
(X2 <- rGNN(GMMN, size = n, # passing additional arguments to rPrior()
            qmargins = qexp, method = "sobol", seed = 271))
(X3 <- rGNN(GMMN, prior = matrix(rexp(n * d), ncol = d))) # providing 'prior'
stopifnot(dim(X1) == c(n, d), dim(X2) == c(n, d), dim(X3) == c(n, d))

}
```

---

rm_ext *Remove a File Extension*

Description

Fixes the removal of file extensions of file_path_sans_ext() in the case where file names contain digits after the last dot (which is often used to incorporate numeric numbers into file names).

Usage

```
rm_ext(x)
```

Arguments

x	file name(s) with extension(s) to be stripped off.

Value

The file name without its extension (if the file name had an extension).

Author(s)

Marius Hofert

Examples

```
library(gnn) # for being standalone

myfilepath1 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75.rda"
myfilepath2 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75"
myfilepath3 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75."
myfilepath4 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75._"
myfilepath5 <- "/myusername/my_filename_with_dots_0.25_0.50_0.75._*.rda"
library(tools)
file_path_sans_ext(myfilepath2) # fails (only case)

stopifnot(rm_ext(myfilepath1) == file_path_sans_ext(myfilepath1))
stopifnot(rm_ext(myfilepath2) == myfilepath2)
stopifnot(rm_ext(myfilepath3) == file_path_sans_ext(myfilepath3))
stopifnot(rm_ext(myfilepath4) == file_path_sans_ext(myfilepath4))
stopifnot(rm_ext(myfilepath5) == file_path_sans_ext(myfilepath5))
```

---

rPrior *Sampling from a Prior Distribution*

Description

Sampling from a prior distribution.

Usage

```
rPrior(n, copula, qmargins = qnorm, method = c("pseudo", "sobol"), ...)
```

Arguments

n	sample size, a positive integer.
copula	object of S4 class "Copula" for which the method rCopula() is available; see the R package **copula**.
qmargins	marginal quantile function or a list of length dim(x)[1] of such.
method	character string indicating the sampling method. If "sobol", then randomization "digital.shift" is used (pass seed via ... for reproducibilty; see the R package **qrng**).
...	additional arguments passed to method.

Value

(n, dim(copula))-matrix of samples.

Author(s)

Marius Hofert

Examples

```
library(gnn) # for being standalone

n <- 5
d <- 3
library(copula)
cop <- claytonCopula(2, dim = d)
X1 <- rPrior(n, copula = cop) # Clayton copula and N(0,1) margins
X2 <- rPrior(n, copula = cop, qmargins = qexp) # Exp(1) margins
X3 <- rPrior(n, copula = cop, qmargins = qexp, method = "sobol", seed = 271)
stopifnot(dim(X1) == c(n, d), dim(X2) == c(n, d), dim(X3) == c(n, d))
```

---

save_load_rda *Save and Load .rda Files with Conversion to Raw and Keras Models*

Description

Save and load .rda files with conversion to objects of class raw (for save_gnn()) or "keras.engine.training.Model" (for load_gnn()).

Usage

```
save_gnn(..., file, name = NULL)
load_gnn(file)
```

Arguments

...	objects to be saved in file (under the provided names if name was provided). Those objects which are of class "gnn_GNN" are converted with as.raw() before they are saved.
file	file name; see the underlying save() and load().
name	character (vector) of name(s) under which the objects in ... are to be saved in file. If NULL, the names of the objects provided by ... are taken as name.

Value

save_gnn() nothing (generates an .rda by side-effect).

load_gnn() the loaded object(s). Those of class "gnn_GNN" are converted with as.keras() before they are returned; this also applies to a component of a loaded object of class list.

Author(s)

Marius Hofert

See Also

See the underlying functions load() and save() (among others).

Examples

```
if(TensorFlow_available()) { # rather restrictive (due to R-Forge, winbuilder)
library(gnn) # for being standalone

file <- tempfile("foo", fileext = ".rda")
GMMN1 <- FNN()
save_gnn(GMMN1, file = file) # converts GMMN via as.raw()
GMMN2 <- load_gnn(file) # converts loaded object via as.keras()
stopifnot(is.GNN(GMMN2), inherits(GMMN2[["model"]], "keras.engine.training.Model"))
rm(GMMN1, GMMN2) # clean-up
stopifnot(file.remove(file))

}
```

TensorFlow_available *A Simple Check whether TensorFlow is Available*

Description

A simple (and restrictive) check whether TensorFlow is available.

Usage

```
TensorFlow_available()
```

Details

Essentially calls "pip list | grep tensorflow" via system(). Only available on non-Windows operating systems; returns FALSE on Windows.

Value

logical indicating whether TensorFlow was found.

Author(s)

Marius Hofert

Examples

```
library(gnn) # for being standalone

TensorFlow_available()
```

---

time *Human-Readable Time Measurement*

Description

Functions and methods for extracting and printing timings in human-readable format.

Usage

```
as.human(x, fmt = "%.2f")
human.time(expr, print = TRUE, ...)
## S3 method for class 'gnn_GNN'
time(x, human = FALSE, ...)
## S3 method for class 'gnn_proc_time'
print(x, ...)
```

Arguments

x	**as.human()** object of class "proc_time" as returned by system.time(). **time.gnn_GNN()** object of class "gnn_GNN". **print.gnn_proc_time()** object of class "gnn_proc_time" as returned by time().
fmt	format string as required by sprintf().
expr	see system.time().
print	logical indicating whether to print the result; either way, it is returned (invisibly if print = TRUE).
human	logical indicating whether the result is to be returned in human-readable format.
...	for human.time() and time.gnn_GNN() additional arguments passed to the underlying as.human(); unused for print.gnn_proc_time().

Value

as.human(), human.time() named `character(3)` providing user, system and elapsed time in human-readable format.

time.gnn_GNN() object of class `"gnn_proc_time"`.

print.gnn_proc_time() `x` (invisibly).

Author(s)

Marius Hofert

Examples

```
if(TensorFlow_available()) { # rather restrictive (due to R-Forge, winbuilder)
library(gnn) # for being standalone

human.time(Sys.sleep(0.1)) # print human-readable time
(proc.obj <- human.time(Sys.sleep(0.1), print = FALSE)) # save the timing (character(3))
fnn <- FNN()
time(fnn) # default print method for objects of class "gnn_proc_time"
time(fnn, human = TRUE) # human-readable print method for such objects
}
```

`trafos_dimreduction` *Dimension-Reduction Transformations for Training or Sampling*

Description

Dimension-reduction transformations applied to an input data matrix. Currently on the principal component transformation and its inverse.

Usage

```
PCA_trafo(x, mu, Gamma, inverse = FALSE, ...)
```

Arguments

`x`	(n, d)-matrix of data (typically before training or after sampling). If `inverse = FALSE`, then, conceptually, an (n, d)-matrix with $1 \leq k \leq d$, where d is the dimension of the original data whose dimension was reduced to k.
`mu`	if `inverse = TRUE`, a d-vector of centers, where d is the dimension to transform `x` to.
`Gamma`	if `inverse = TRUE`, a (d, k)-matrix with k at least as large as `ncol(x)` containing the k orthonormal eigenvectors of a covariance matrix sorted in decreasing order of their eigenvalues; in other words, the columns of `Gamma` contain principal axes or loadings. If a matrix with k greater than `ncol(x)` is provided, only the first k-many are considered.

`inverse`	`logical` indicating whether the inverse transformation of the principal component transformation is applied.
`...`	additional arguments passed to the underlying `prcomp()`.

Details

Conceptually, the principal component transformation transforms a vector $\boldsymbol{X}$ to a vector $\boldsymbol{Y}$ where $\boldsymbol{Y} = \Gamma^T(\boldsymbol{X} - \boldsymbol{\mu})$, where $\boldsymbol{\mu}$ is the mean vector of $\boldsymbol{X}$ and Γ is the (d, d)-matrix whose columns contains the orthonormal eigenvectors of `cov(X)`.

The corresponding (conceptual) inverse transformation is $\boldsymbol{X} = \boldsymbol{\mu} + \Gamma\boldsymbol{Y}$.

See McNeil et al. (2015, Section 6.4.5).

Value

If `inverse = TRUE`, the transformed data whose rows contain $\boldsymbol{X} = \boldsymbol{\mu} + \Gamma\boldsymbol{Y}$, where Y is one row of `x`. See the details below for the notation.

If `inverse = FALSE`, a `list` containing:

`PCs`: (n, d)-matrix of principal components.

`cumvar`: cumulative variances; the jth entry provides the fraction of the explained variance of the first j principal components.

`sd`: sample standard deviations of the transformed data.

`lambda`: eigenvalues of `cov(x)`.

`mu`: d-vector of centers of `x` (see also above) typically provided to `PCA_trafo(, inverse = TRUE)`.

`Gamma`: (d, d)-matrix of principal axes (see also above) typically provided to `PCA_trafo(, inverse = TRUE)`.

Author(s)

Marius Hofert

References

McNeil, A. J., Frey, R., and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Examples

```
library(gnn) # for being standalone

## Generate data
library(copula)
set.seed(271)
X <- qt(rCopula(1000, gumbelCopula(2, dim = 10)), df = 3.5)
pairs(X, gap = 0, pch = ".")

## Principal component transformation
PCA <- PCA_trafo(X)
```

```
Y <- PCA$PCs
PCA$cumvar[3] # fraction of variance explained by the first 3 principal components
which.max(PCA$cumvar > 0.9) # number of principal components it takes to explain 90%

## Biplot (plot of the first two principal components = data transformed with
## the first two principal axes)
plot(Y[,1:2])

## Transform back and compare
X. <- PCA_trafo(Y, mu = PCA$mu, Gamma = PCA$Gamma, inverse = TRUE)
stopifnot(all.equal(X., X))

## Note: One typically transforms back with only some of the principal axes
X. <- PCA_trafo(Y[,1:3], mu = PCA$mu, # mu determines the dimension to transform to
                Gamma = PCA$Gamma, # must be of dim. (length(mu), k) for k >= ncol(x)
                inverse = TRUE)
stopifnot(dim(X.) == c(1000, 10))
## Note: We (typically) transform back to the original dimension.
pairs(X., gap = 0, pch = ".") # pairs of back-transformed first three PCs
```

`trafos_margins` *Data Transformations for Training or Sampling*

Description

Transformations applied to each marginal component sample to map given data to a different range.

Usage

```
range_trafo(x, lower, upper, inverse = FALSE)
logis_trafo(x, mean = 0, sd = 1, slope = 1, intercept = 0, inverse = FALSE)
```

Arguments

`x`	(n, d)-matrix of data (typically before training or after sampling).
`lower`	value or d-vector typically containing the smallest value of each column of `x`.
`upper`	value or d-vector typically containing the largest value of each column of `x`.
`mean`	value or d-vector.
`sd`	value or d-vector.
`slope`	value or d-vector of slopes of the linear transformations applied after applying `plogis()` (before applying `qlogis()` if `inverse = TRUE`).
`intercept`	value or d-vector of intercepts of the linear transformations applied after applying `plogis()` (before applying `qlogis()` if `inverse = TRUE`).
`inverse`	`logical` indicating whether the inverses of the respective transformations are to be computed (typically used after generating data from a neural network trained on data transformed with the respective transformation and `inverse = FALSE`).

Value

An object as x containing the componentwise transformed data.

Author(s)

Marius Hofert

Examples

```
library(gnn) # for being standalone

## Generate data
n <- 100
set.seed(271)
x <- cbind(rnorm(n), (1-runif(n))^(-1/2)-1) # normal and Pareto(2) margins
plot(x)

## Range transformation
ran <- apply(x, 2, range) # column j = range of the jth column of x
x.ran <- range_trafo(x, lower = ran[1,], upper = ran[2,]) # marginally transform to [0,1]
plot(x.ran) # => now range [0,1] but points a bit clustered around small y-values
x. <- range_trafo(x.ran, lower = ran[1,], upper = ran[2,], inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Logistic transformation
x.logis <- logis_trafo(x) # marginally transform to [0,1] via plogis()
plot(x.logis) # => y-range is [1/2, 1] which can be harder to train
x. <- logis_trafo(x.logis, inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Logistic transformation with scaling to all of [0,1] in the second coordinate
x.logis.scale <- logis_trafo(x, slope = 2, intercept = -1)
plot(x.logis.scale) # => now y-range is scaled to [0,1]
x. <- logis_trafo(x.logis.scale, slope = 2, intercept = -1, inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Logistic transformation with sample mean and standard deviation and then
## transforming the range to [0,1] with a range transformation (note that
## slope = 2, intercept = -1 would not help here as the y-range is not [1/2, 1])
mu <- colMeans(x)
sig <- apply(x, 2, sd)
x.logis.fit <- logis_trafo(x, mean = mu, sd = sig) # marginally plogis(, location, scale)
plot(x.logis.fit) # => y-range is not [1/2, 1] => use range transformation
ran <- apply(x.logis.fit, 2, range)
x.logis.fit.ran <- range_trafo(x.logis.fit, lower = ran[1,], upper = ran[2,])
plot(x.logis.fit.ran) # => now y-range is [1/2, 1]
x. <- logis_trafo(range_trafo(x.logis.fit.ran, lower = ran[1,], upper = ran[2,],
                              inverse = TRUE),
                  mean = mu, sd = sig, inverse = TRUE) # transform back
stopifnot(all.equal(x., x)) # check

## Note that for heavy-tailed data, plogis() can fail to stay inside (0,1)
```

```
## even with adapting to sample mean and standard deviation. We now present
## a case where we see that using a fitted logistic distribution function
## is *just* good enough to numerically keep the data inside (0,1).
set.seed(271)
x <- cbind(rnorm(n), (1-runif(n))^(-2)-1) # normal and Pareto(1/2) margins
plot(x) # => heavy-tailed in y-coordinate
## Transforming with standard logistic distribution function
x.logis <- logis_trafo(x)
stopifnot(any(x.logis[,2] == 1))
## => There is value numerically indistinguishable from 1 to which applying
##    the inverse transform will lead to Inf
stopifnot(any(is.infinite(logis_trafo(x.logis, inverse = TRUE))))
## Now adapt the logistic distribution to share the mean and standard deviation
## with the data
mu <- colMeans(x)
sig <- apply(x, 2, sd)
x.logis.scale <- logis_trafo(x, mean = mu, sd = sig)
stopifnot(all(x.logis.scale[,2] != 1)) # => no values equal to 1 anymore

## Alternatively, log() the data first, thus working with a log-logistic
## distribution as transformation
lx <- cbind(x[,1], log(x[,2])) # 2nd coordinate only
lmu <- c(mu[1], mean(lx[,2]))
lsig <- c(sig[1], sd(lx[,2]))
x.llogis <- logis_trafo(lx, mean = lmu, sd = lsig)
x. <- logis_trafo(x.llogis, mean = lmu, sd = lsig, inverse = TRUE)
x.. <- cbind(x.[,1], exp(x.[,2])) # undo log()
stopifnot(all.equal(x.., x))
```

Index