

Package ‘NeuralEstimators’

September 9, 2026

Title Likelihood-Free Parameter Estimation using Neural Networks

Version 0.2.2

Description An 'R' interface to the 'Julia' package 'NeuralEstimators.jl'. The package facilitates the user-friendly development of neural Bayes estimators, which are neural networks that map data to a point summary of the posterior distribution (Sainsbury-Dale et al., 2024, <[doi:10.1080/00031305.2023.2249522](https://doi.org/10.1080/00031305.2023.2249522)>). These estimators are likelihood-free and amortised, in the sense that, once the neural networks are trained on simulated data, inference from observed data can be made in a fraction of the time required by conventional approaches. The package also supports amortised Bayesian or frequentist inference using neural networks that approximate the posterior or likelihood-to-evidence ratio (Zammit-Mangion et al., 2025, Sec. 3.2, 5.2, <[doi:10.48550/arXiv.2404.12484](https://doi.org/10.48550/arXiv.2404.12484)>). The package accommodates any model for which simulation is feasible by allowing users to define models implicitly through simulated data.

Maintainer Matthew Sainsbury-Dale <msainsburydale@gmail.com>

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.2

Imports JuliaConnectoR, magrittr

Suggests dplyr, ggplot2, ggplotify, ggpubr, gridExtra, knitr, rmarkdown, markdown, R.rsp, testthat (>= 3.0.0)

Config/testthat/edition 3

SystemRequirements Julia (>= 1.11)

VignetteBuilder R.rsp

URL <https://github.com/msainsburydale/NeuralEstimators>,
<https://msainsburydale.github.io/NeuralEstimators.jl/dev/>

NeedsCompilation no

Author Matthew Sainsbury-Dale [aut, cre]

Repository CRAN

Date/Publication 2026-09-09 06:30:03 UTC

Contents

assess	2
bias	3
bootstrap	4
cpu_device	4
encodedata	5
estimate	6
gpu_device	6
infer	7
loadstate	7
logratio	8
plotdistribution	9
plotestimates	11
PointEstimator	12
PosteriorEstimator	13
RatioEstimator	14
reactant_device	14
risk	15
rmse	16
sampleposterior	16
savestate	17
spatialgraph	17
tanhloss	19
train	20
Index	25

assess	<i>assess a neural estimator</i>
--------	----------------------------------

Description

assess a neural estimator

Usage

assess(estimator, parameters, Z, ...)

Arguments

estimator	a neural estimator (or a list of neural estimators)
parameters	true parameters, stored as a $d \times K$ matrix, where d is the dimension of the parameter vector and K is the number of sampled parameter vectors
Z	data simulated conditionally on the parameters. If $\text{length}(Z) > K$, the parameter matrix will be recycled by horizontal concatenation as $\text{parameters} = \text{parameters}[, \text{rep}(1:K, J)]$, where $J = \text{length}(Z) / K$
...	additional keyword arguments passed to the Julia version of <code>assess()</code>

Value

a list with the following elements:

- `estimates`: a long-form data frame with columns
 - `"parameter"`; the name of the parameter
 - `"truth"`; the true value of the parameter
 - `"estimate"`; the estimated value of the parameter
 - `"k"`; the index of the parameter vector in the test set
 - `"j"`; the index of the data set

For an `IntervalEstimator`, `estimate` is replaced by columns `lower` and `upper`. For a `QuantileEstimator`, there is an additional column `prob` giving the probability level of each quantile estimate.

- `runtimes`: a data frame giving the total time taken for each estimator
- `samples` (posterior and ratio estimators only): posterior samples stored as a long-form Julia `DataFrame` with columns `parameter`, `truth`, `k`, and `j` as above, plus `draw` (the index of the draw) and `value` (the sampled value). This object is not converted to an R data frame, which is slow for large sample collections; extract or summarise it in Julia if needed.

See Also

[risk\(\)](#), [rmse\(\)](#), [bias\(\)](#), [plotestimates\(\)](#), and [plotdistribution\(\)](#) for computing various empirical diagnostics and visualisations from an object returned by [assess\(\)](#)

 bias

computes a Monte Carlo approximation of an estimator's bias

Description

computes a Monte Carlo approximation of an estimator's bias

Usage

```
bias(assessment, ...)
```

Arguments

<code>assessment</code>	an object returned by assess()
<code>...</code>	optional arguments inherited from risk() (excluding the argument <code>loss</code>)

Value

a dataframe giving the estimated bias

See Also

[assess\(\)](#), [risk\(\)](#), [rmse\(\)](#)

bootstrap

*bootstrap***Description**

Compute bootstrap estimates from a neural point estimator

Usage

```
bootstrap(estimator, Z, B = 400, blocks = NULL, use_gpu = TRUE)
```

Arguments

estimator	a neural point estimator
Z	either a list of data sets simulated conditionally on the fitted parameters (parametric bootstrap); or a single observed data set containing independent replicates, which will be sampled with replacement B times (non-parametric bootstrap)
B	number of non-parametric bootstrap samples
blocks	integer vector specifying the blocks in non-parameteric bootstrap. For example, with 5 replicates, the first two corresponding to block 1 and the remaining three corresponding to block 2, blocks should be <code>c(1, 1, 2, 2, 2)</code>
use_gpu	boolean indicating whether to use the GPU if it is available

Value

$d \times B$ matrix, where d is the dimension of the parameter vector

cpu_device

*CPU device***Description**

A CPU compute device from [MLDataDevices.jl](#) for the device argument of [train\(\)](#).

Usage

```
cpu_device(...)
```

Arguments

... arguments passed to the Julia function.

Value

a Julia device object

See Also

`gpu_device()`, `reactant_device()`, `train()`

encodedata

encodedata

Description

For data Z with missing (NA) entries, computes an augmented data set (U, W) where W encodes the missingness pattern as an indicator vector and U is the original data Z with missing entries replaced by a fixed constant c .

Usage

```
encodedata(Z, c = 0)
```

Arguments

Z	data containing NA entries
c	fixed constant with which to replace NA entries

Value

Augmented data set (U, W) . If Z is provided as a list, the return type will be a `JuliaProxy` object; these objects can be indexed in the usual manner using `[]`, or converted to an R object using `juliaGet()` (note however that `juliaGet()` can be slow for large data sets).

See Also

the Julia version of `encodedata()`

Examples

```
## Not run:
library("NeuralEstimators")
Z <- matrix(c(1, 2, NA, NA, 5, 6, 7, NA, 9), nrow = 3)
encodedata(Z)
encodedata(list(Z, Z))
## End(Not run)
```

estimate	<i>estimate</i>
----------	-----------------

Description

Apply a neural Bayes estimator to data

Usage

```
estimate(estimator, Z, batchsize = 32, ...)
```

Arguments

estimator	a neural estimator that can be applied to data in a call of the form <code>estimator(Z)</code>
Z	data in a format amenable to the neural-network architecture of estimator
batchsize	the batch size for applying estimator to Z
...	additional keyword arguments passed to the Julia version of <code>estimate()</code>

Value

a matrix of outputs resulting from applying estimator to Z

See Also

[infer\(\)](#) for a unified inference interface, and [sampleposterior\(\)](#) for making inference with neural posterior or likelihood-to-evidence-ratio estimators

gpu_device	<i>GPU device</i>
------------	-------------------

Description

A GPU compute device from [MLDataDevices.jl](#) for the device argument of [train\(\)](#).

Usage

```
gpu_device(...)
```

Arguments

...	arguments passed to the Julia function (e.g. <code>force</code>).
-----	--

Value

a Julia device object

See Also

`cpu_device()`, `reactant_device()`, `train()`

infer	<i>infer</i>
-------	--------------

Description

Unified inference interface that dispatches to `estimate()` for neural Bayes estimators (e.g., `PointEstimator`) and `sampleposterior()` for posterior or ratio estimators.

Usage

```
infer(estimator, Z, ...)
```

Arguments

<code>estimator</code>	a neural estimator
<code>Z</code>	data in a format amenable to the neural-network architecture of estimator
<code>...</code>	additional keyword arguments passed to the Julia version of <code>infer()</code> (and hence to <code>estimate()</code> or <code>sampleposterior()</code>)

Value

For neural Bayes estimators, a matrix of point estimates (see `estimate()`). For posterior or ratio estimators, a $d \times N \times K$ array of posterior samples (see `sampleposterior()`).

See Also

`estimate()`, `sampleposterior()`

loadstate	<i>Load a saved state of a neural estimator</i>
-----------	---

Description

Load a saved state of a neural estimator (e.g., optimised neural-network parameters). Useful for amortised inference, whereby a neural network is trained once and then used repeatedly to make inference with new data sets.

Usage

```
loadstate(estimator, filename)
```

Arguments

estimator	the neural estimator that we wish to load the state into
filename	file name (including path) of the neural-network state stored in a bson file

Value

estimator updated with the saved state

logratio	<i>logratio</i>
----------	-----------------

Description

Apply a neural ratio estimator to data

Usage

```
logratio(estimator, Z, grid, batchsize = 32, ...)
```

Arguments

estimator	a neural ratio estimator
Z	data in a format amenable to the neural-network architecture of estimator
grid	matrix of parameter values, where each column is a parameter configuration
batchsize	the batch size
...	additional keyword arguments passed to the Julia version of <code>logratio()</code>

Value

A matrix of log ratios with one row per data set and one column per grid point

See Also

`sampleposterior()` and `infer()` for making posterior inferences

plotdistribution	<i>Plot the empirical sampling distribution of an estimator.</i>
------------------	--

Description

Plot the empirical sampling distribution of an estimator.

Usage

```
plotdistribution(
  df,
  type = c("box", "density", "scatter"),
  parameter_labels = NULL,
  estimator_labels = ggplot2::waiver(),
  truth_colour = "black",
  truth_size = 8,
  truth_line_size = NULL,
  pairs = FALSE,
  upper_triangle_plots = NULL,
  legend = TRUE,
  return_list = FALSE,
  flip = FALSE
)
```

Arguments

<code>df</code>	a long form data frame containing fields <code>estimator</code> , <code>parameter</code> , <code>estimate</code> , <code>truth</code> , and a column (e.g., <code>replicate</code>) to uniquely identify each observation.
<code>type</code>	string indicating whether to plot kernel density estimates for each individual parameter (<code>type = "density"</code>) or scatter plots for all parameter pairs (<code>type = "scatter"</code>).
<code>parameter_labels</code>	a named vector containing parameter labels.
<code>estimator_labels</code>	a named vector containing estimator labels.
<code>truth_colour</code>	the colour used to denote the true parameter value.
<code>truth_size</code>	the size of the point used to denote the true parameter value (applicable only for <code>type = "scatter"</code>).
<code>truth_line_size</code>	the size of the cross-hairs used to denote the true parameter value. If <code>NULL</code> (default), the cross-hairs are not plotted. (applicable only for <code>type = "scatter"</code>).
<code>pairs</code>	logical; should we combine the scatter plots into a single pairs plot (applicable only for <code>type = "scatter"</code>)?
<code>upper_triangle_plots</code>	an optional list of plots to include in the uppertriangle of the pairs plot.

legend	Flag; should we include the legend (only applies when constructing a pairs plot)
return_list	Flag; should the parameters be split into a list?
flip	Flag; should the boxplots be "flipped" using coord_flip() (default FALSE)?

Value

a list of 'ggplot' objects or, if pairs = TRUE, a single 'ggplot'.

Examples

```
## Not run:
# In the following, we have two estimators and, for each parameter, 50 estimates
# from each estimator.

estimators <- c("Estimator 1", "Estimator 2")
estimator_labels <- c("Estimator 1" = expression(hat(theta)[1](".")),
                      "Estimator 2" = expression(hat(theta)[2](".")))

# Single parameter:
df <- data.frame(
  estimator = estimators, truth = 0, parameter = "mu",
  estimate = rnorm(2*50),
  replicate = rep(1:50, each = 2)
)
parameter_labels <- c("mu" = expression(mu))
plotdistribution(df)
plotdistribution(df, type = "density")
plotdistribution(df, parameter_labels = parameter_labels, estimator_labels = estimator_labels)

# Two parameters:
df <- rbind(df, data.frame(
  estimator = estimators, truth = 1, parameter = "sigma",
  estimate = rgamma(2*50, shape = 1, rate = 1),
  replicate = rep(1:50, each = 2)
))
parameter_labels <- c(parameter_labels, "sigma" = expression(sigma))
plotdistribution(df, parameter_labels = parameter_labels)
plotdistribution(df, parameter_labels = parameter_labels, type = "density")
plotdistribution(df, parameter_labels = parameter_labels, type = "scatter")

# Three parameters:
df <- rbind(df, data.frame(
  estimator = estimators, truth = 0.25, parameter = "alpha",
  estimate = 0.5 * runif(2*50),
  replicate = rep(1:50, each = 2)
))
parameter_labels <- c(parameter_labels, "alpha" = expression(alpha))
plotdistribution(df, parameter_labels = parameter_labels)
plotdistribution(df, parameter_labels = parameter_labels, type = "density")
plotdistribution(df, parameter_labels = parameter_labels, type = "scatter")
plotdistribution(df, parameter_labels = parameter_labels, type = "scatter", pairs = TRUE)
```

```

# Pairs plot with user-specified plots in the upper triangle:
upper_triangle_plots <- lapply(1:3, function(i) {
  x = rnorm(10)
  y = rnorm(10)
  shape = sample(c("Class 1", "Class 2"), 10, replace = TRUE)
  ggplot() +
    geom_point(aes(x = x, y = y, shape = shape)) +
    labs(shape = "") +
    theme_bw()
})
plotdistribution(
  df,
  parameter_labels = parameter_labels, estimator_labels = estimator_labels,
  type = "scatter", pairs = TRUE, upper_triangle_plots = upper_triangle_plots
)
## End(Not run)

```

plotestimates

Plot estimates vs. true values.

Description

Plot estimates vs. true values.

Usage

```

plotestimates(
  df,
  estimator_labels = ggplot2::waiver(),
  parameter_labels = NULL
)

```

Arguments

df a long form data frame containing fields estimator, parameter, estimate, and truth.

estimator_labels a named vector containing estimator labels.

parameter_labels a named vector containing parameter labels.

Value

a 'ggplot' of the estimates for each parameter against the true value.

Examples

```
## Not run:
K <- 50
df <- data.frame(
  estimator = c("Estimator 1", "Estimator 2"),
  parameter = rep(c("mu", "sigma"), each = K),
  truth = 1:(2*K),
  estimate = 1:(2*K) + rnorm(4*K)
)
estimator_labels <- c("Estimator 1" = expression(hat(theta)[1](".")),
  "Estimator 2" = expression(hat(theta)[2](".")))
parameter_labels <- c("mu" = expression(mu), "sigma" = expression(sigma))

plotestimates(df, parameter_labels = parameter_labels, estimator_labels)
## End(Not run)
```

PointEstimator

PointEstimator

Description

Construct a neural Bayes point estimator. The neural-network architecture is still defined in Julia (e.g. via `juliaEval()`); this function wraps that architecture in a `PointEstimator`.

Typical constructors, matching the Julia methods:

- `PointEstimator(network)`: a single network mapping data to the parameter space.
- `PointEstimator(num_parameters, summary_network, num_summaries = ...)`: a summary network with an MLP inference network built internally.

Usage

```
PointEstimator(...)
```

Arguments

... arguments passed to the Julia version of `PointEstimator()`

Value

a `PointEstimator`

See Also

[PosteriorEstimator\(\)](#), [RatioEstimator\(\)](#), [train\(\)](#), [infer\(\)](#)

Examples

```
## Not run:
library("NeuralEstimators")
library("JuliaConnectoR")
juliaEval("using NeuralEstimators, Flux")
network <- juliaEval("
  n = 1; d = 2; w = 32
  psi = Chain(Dense(n, w, relu), Dense(w, w, relu))
  phi = Chain(Dense(w, w, relu), Dense(w, d))
  DeepSet(psi, phi)
")
estimator <- PointEstimator(network)

## End(Not run)
```

PosteriorEstimator	<i>PosteriorEstimator</i>
--------------------	---------------------------

Description

Construct a neural posterior estimator. The neural-network architecture is still defined in Julia (e.g. via `juliaEval()`); this function wraps that architecture in a `PosteriorEstimator`.

Typical constructors, matching the Julia methods:

- `PosteriorEstimator(num_parameters, summary_network, num_summaries = ...)`: builds the approximate distribution internally (default `q = "NormalisingFlow"`).
- `PosteriorEstimator(summary_network, q)`: an explicit approximate distribution `q`.

The argument `q` may be a Julia type or a string naming one (e.g. `"Gaussian"`, `"GaussianMixture"`, `"NormalisingFlow"`).

Usage

```
PosteriorEstimator(...)
```

Arguments

... arguments passed to the Julia version of `PosteriorEstimator()`

Value

a `PosteriorEstimator`

See Also

[PointEstimator\(\)](#), [RatioEstimator\(\)](#), [train\(\)](#), [infer\(\)](#)

RatioEstimator

RatioEstimator

Description

Construct a neural likelihood-to-evidence-ratio estimator. The neural-network architecture is still defined in Julia (e.g. via `juliaEval()`); this function wraps that architecture in a `RatioEstimator`.

Typical constructor, matching the Julia method:

- `RatioEstimator(num_parameters, summary_network, num_summaries = num_summaries)`

Usage

`RatioEstimator(...)`

Arguments

... arguments passed to the Julia version of `RatioEstimator()`

Value

a `RatioEstimator`

See Also

`PointEstimator()`, `PosteriorEstimator()`, `train()`, `infer()`

reactant_device

Reactant/XLA device

Description

A Reactant/XLA compute device from `MLDataDevices.jl` for the device argument of `train()`. Requires Lux.

Usage

`reactant_device(...)`

Arguments

... arguments passed to the Julia function (e.g. `force`).

Value

a Julia device object

See Also

`cpu_device()`, `gpu_device()`, `train()`

risk	<i>computes a Monte Carlo approximation of an estimator's Bayes risk</i>
------	--

Description

computes a Monte Carlo approximation of an estimator's Bayes risk

Usage

```
risk(
  assessment,
  loss = function(x, y) abs(x - y),
  average_over_parameters = FALSE,
  average_over_sample_sizes = TRUE
)
```

Arguments

assessment an object returned by `assess()`

loss a binary operator defining the loss function (default absolute-error loss)

average_over_parameters if TRUE, the loss is averaged over all parameters; otherwise (default), the loss is averaged over each parameter separately

average_over_sample_sizes if TRUE (default), the loss is averaged over all sample sizes (the column `m` in `df`); otherwise, the loss is averaged over each sample size separately

Value

a dataframe giving an estimate of the Bayes risk

See Also

`assess()`, `bias()`, `rmse()`

rmse	<i>computes a Monte Carlo approximation of an estimator's root-mean-square error (RMSE)</i>
------	---

Description

computes a Monte Carlo approximation of an estimator's root-mean-square error (RMSE)

Usage

```
rmse(assessment, ...)
```

Arguments

assessment	an object returned by <code>assess()</code>
...	optional arguments inherited from <code>risk()</code> (excluding the argument <code>loss</code>)

Value

a dataframe giving the estimated RMSE

See Also

`assess()`, `bias()`, `risk()`

sampleposterior	<i>sampleposterior</i>
-----------------	------------------------

Description

Samples from the approximate posterior distribution given data Z .

Usage

```
sampleposterior(estimator, Z, N = 1000, ...)
```

Arguments

estimator	a neural posterior or likelihood-to-evidence-ratio estimator
Z	data in a format amenable to the neural-network architecture of estimator
N	number of approximate posterior samples to draw
...	additional keyword arguments passed to the Julia version of <code>sampleposterior()</code>

Value

a $d \times N \times K$ array of posterior samples, where d is the dimension of the parameter vector, N is the number of draws, and K is the number of independent data sets in Z (so a single data set yields a $d \times N \times 1$ array)

See Also

[infer\(\)](#) for a unified inference interface, and [estimate\(\)](#) for making inference with neural Bayes estimators

savestate	<i>save the state of a neural estimator</i>
-----------	---

Description

Save the state of a neural estimator (e.g., optimised neural-network parameters). Supports both Flux and Lux estimators.

Usage

```
savestate(estimator, filename)
```

Arguments

- | | |
|-----------|---|
| estimator | the neural estimator that we wish to save |
| filename | file in which to save the neural-network state as a bson file |

Value

No return value, called for side effects

spatialgraph	<i>spatialgraph</i>
--------------	---------------------

Description

Constructs a graph object for use in a graph neural network (GNN).

Usage

```
spatialgraph(S, Z, isotropic = TRUE, stationary = TRUE, ...)
```

Arguments

S	Spatial locations, provided as: • An $n \times 2$ matrix when locations are fixed across replicates, where n is the number of spatial locations. • A list of $n_i \times 2$ matrices when locations vary across replicates. • A list of the above elements (i.e., a list of matrices or a list of lists of matrices) when constructing graphs from multiple data sets.
Z	Spatial data, provided as: • An $n \times m$ matrix when locations are fixed, where m is the number of replicates. • A list of n_i-vectors when locations vary across replicates. • A list of the above elements (i.e., a list of matrices or a list of lists of vectors) when constructing graphs from multiple data sets.
isotropic	Logical. If TRUE, edge features store the spatial distance (magnitude) between nodes. If FALSE, the spatial displacement or spatial location is stored, depending on the value of stationary .
stationary	Logical. If TRUE, edge features store the spatial displacement (vector difference) between nodes, capturing both magnitude and direction. If FALSE, edge features include the full spatial locations of both nodes.
...	Additional keyword arguments from the Julia function <code>adjacencymatrix()</code> that define the neighborhood of each node, with the default being a randomly selected set of $k=30$ neighbors within a radius of $r=0.15$ spatial distance units.

Value

A GNNGraph (JuliaProxy object) or, if multiple data sets are provided, a vector of GNNGraph objects which can be indexed in the usual manner using `[[` or converted to an R list using a combination of indexing and `lapply`.

Examples

```
## Not run:
library("NeuralEstimators")

# Number of replicates
m <- 5

# Spatial locations fixed for all replicates
n <- 100
S <- matrix(runif(n * 2), n, 2)
Z <- matrix(runif(n * m), n, m)
g <- spatialgraph(S, Z)

# Spatial locations varying between replicates
n <- sample(50:100, m, replace = TRUE)
S <- lapply(n, function(ni) matrix(runif(ni * 2), ni, 2))
Z <- lapply(n, function(ni) runif(ni))
```

```

g <- spatialgraph(S, Z)

# Multiple data sets: Spatial locations fixed for all replicates within a given data set
K <- 15 # number of data sets
n <- sample(50:100, K, replace = TRUE) # number of spatial locations can vary between data sets
S <- lapply(1:K, function(k) matrix(runif(n[k] * 2), n[k], 2))
Z <- lapply(1:K, function(k) matrix(runif(n[k] * m), n[k], m))
g <- spatialgraph(S, Z)

# Multiple data sets: Spatial locations varying between replicates within a given data set
S <- lapply(1:K, function(k) {
  lapply(1:m, function(i) {
    ni <- sample(50:100, 1) # randomly generate the number of locations for each replicate
    matrix(runif(ni * 2), ni, 2) # generate the spatial locations
  })
})
Z <- lapply(1:K, function(k) {
  lapply(1:m, function(i) {
    n <- nrow(S[[k]][[i]])
    runif(n)
  })
})
g <- spatialgraph(S, Z)

## End(Not run)

```

tanhloss

tanhloss

Description

For $k > 0$, defines Julia code that defines the loss function,

$$L(\hat{\theta}, \theta) = \tanh \left(\frac{|\hat{\theta} - \theta|}{k} \right),$$

which approximates the 0-1 loss as k tends to zero.

The resulting string is intended to be used in the function `train`, but can also be converted to a callable function using `juliaEval`.

Usage

```
tanhloss(k)
```

Arguments

k Positive numeric value that controls the smoothness of the approximation.

Value

String defining the tanh loss function in Julia code.

train	<i>Train a neural estimator</i>
-------	---------------------------------

Description

The function caters for different variants of "on-the-fly" simulation. Specifically, a sampler can be provided to continuously sample new parameter vectors from the prior, and a simulator can be provided to continuously simulate new data conditional on the parameters. If provided with specific sets of parameters (`theta_train` and `theta_val`) and/or data (`Z_train` and `Z_val`), they will be held fixed during training.

Note that using R functions to perform "on-the-fly" simulation requires the user to have installed the Julia package `RCall`.

Usage

```
train(
  estimator,
  sampler = NULL,
  simulator = NULL,
  theta_train = NULL,
  theta_val = NULL,
  Z_train = NULL,
  Z_val = NULL,
  K = 10000,
  K_val = NULL,
  m = NULL,
  sampler_args = NULL,
  sampler_kwargs = NULL,
  simulator_args = NULL,
  simulator_kwargs = NULL,
  loss = "absolute-error",
  learning_rate = 5e-04,
  epochs = 100,
  batchsize = 32,
  savepath = NULL,
  stopping_epochs = 5,
  epochs_per_refresh = 1,
  epochs_per_Z_refresh = NULL,
  epochs_per_theta_refresh = NULL,
  simulate_just_in_time = FALSE,
  device = NULL,
  use_gpu = TRUE,
  shuffle = TRUE,
  partial = NULL,
  freeze_summary_network = FALSE,
  verbose = TRUE,
  ...
)
```

)

Arguments

estimator	a neural estimator
sampler	a function that takes an integer K, samples K parameter vectors from the prior, and returns them as a pxK matrix
simulator	a function that takes a pxK matrix of parameters and an integer m, and returns K simulated data sets each containing m independent replicates
theta_train	a set of parameters used for updating the estimator using stochastic gradient descent
theta_val	a set of parameters used for monitoring the performance of the estimator during training
Z_train	a simulated data set used for updating the estimator using stochastic gradient descent
Z_val	a simulated data set used for monitoring the performance of the estimator during training
K	the number of parameter vectors sampled in the training set at each epoch
K_val	the number of parameter vectors in the validation set; if NULL (default), the Julia default is used
m	deprecated; use simulator_args
sampler_args	a list of positional arguments passed to the parameter sampler; if provided, sampler is called as sampler(K, sampler_args...).
sampler_kwargs	a list of positional arguments passed to the parameter sampler; if provided, sampler is called as sampler(K; sampler_kwargs...).
simulator_args	a list of positional arguments passed to the data simulator; if provided, simulator is called as simulator(theta, simulator_args...).
simulator_kwargs	a list of positional arguments passed to the data simulator; if provided, simulator is called as simulator(theta; simulator_kwargs...).
loss	the loss function: a string ('absolute-error' for mean-absolute-error loss or 'squared-error' for mean-squared-error loss), or a string of Julia code defining the loss function. For some classes of estimators (e.g., PosteriorEstimator, QuantileEstimator, RatioEstimator), the loss function does not need to be specified.
learning_rate	the initial learning rate for the optimiser ADAM (default 5e-4)
epochs	the number of epochs to train the neural network. An epoch is one complete pass through the entire training data set when doing stochastic gradient descent.
batchsize	the batchsize to use when performing stochastic gradient descent, that is, the number of training samples processed between each update of the neural-network parameters.
savepath	path to save the trained estimator and other information; if null (default), nothing is saved. Otherwise, the neural-network parameters (i.e., the weights and biases) will be saved during training as bson files; the risk function evaluated

over the training and validation sets will also be saved, in the first and second columns of `loss_per_epoch.csv`, respectively; the best parameters (as measured by validation risk) will be saved as `best_network.bson`.

<code>stopping_epochs</code>	cease training if the risk doesn't improve in this number of epochs (default 5).
<code>epochs_per_refresh</code>	integer indicating how often to refresh the training data
<code>epochs_per_Z_refresh</code>	deprecated; use <code>epochs_per_refresh</code>
<code>epochs_per_theta_refresh</code>	integer indicating how often to refresh the training parameters; must be a multiple of <code>epochs_per_refresh</code> . If NULL (default), it is set equal to <code>epochs_per_refresh</code>
<code>simulate_just_in_time</code>	flag indicating whether we should simulate "just-in-time", in the sense that only a batchsize number of parameter vectors and corresponding data are in memory at a given time
<code>device</code>	the compute device, as a Julia object from <code>MLDataDevices.jl</code> , e.g. <code>cpu_device()</code> , <code>gpu_device()</code> , or <code>reactant_device()</code> (the latter requires Lux). If NULL (default), the device is inferred from <code>use_gpu</code> . Takes priority over <code>use_gpu</code> .
<code>use_gpu</code>	a boolean indicating whether to use the GPU if one is available (ignored if device is provided)
<code>shuffle</code>	whether to shuffle the training set at each epoch
<code>partial</code>	whether to include the final incomplete batch; if NULL (default), the Julia default is used
<code>freeze_summary_network</code>	if TRUE and the estimator has a summary network, freeze those parameters during training
<code>verbose</code>	a boolean indicating whether information, including empirical risk values and timings, should be printed to the console during training.
<code>...</code>	additional keyword arguments passed to the Julia version of <code>train()</code>

Value

a trained neural estimator or, if `m` is a vector, a list of trained neural estimators

See Also

the Julia version of `train()`, `assess()` for assessing an estimator post training, and `infer()/estimate()/sampleposterior` for making inference with observed data

Examples

```
## Not run:
# Construct a neural Bayes estimator for replicated univariate Gaussian
# data with unknown mean and standard deviation.

# Load R and Julia packages
```

```

library("NeuralEstimators")
library("JuliaConnector")
juliaEval("using NeuralEstimators, Flux")

# Define the neural-network architecture
estimator <- juliaEval('
n = 1    # dimension of each replicate
d = 2    # number of parameters in the model
w = 32   # width of each hidden layer
psi = Chain(Dense(n, w, relu), Dense(w, w, relu))
phi = Chain(Dense(w, w, relu), Dense(w, d))
deepset = DeepSet(psi, phi)
estimator = PointEstimator(deepset)
')

# Sampler from the prior
sampler <- function(K) {
  mu    <- rnorm(K)      # Gaussian prior for the mean
  sigma <- rgamma(K, 1)  # Gamma prior for the standard deviation
  theta <- matrix(c(mu, sigma), byrow = TRUE, ncol = K)
  return(theta)
}

# Data simulator
simulator <- function(theta_set, m) {
  apply(theta_set, 2, function(theta) {
    t(rnorm(m, theta[1], theta[2]))
  }, simplify = FALSE)
}

# Train using fixed parameter and data sets
theta_train <- sampler(10000)
theta_val   <- sampler(2000)
m <- 30 # number of iid replicates
Z_train <- simulator(theta_train, m)
Z_val   <- simulator(theta_val, m)
estimator <- train(estimator,
  theta_train = theta_train,
  theta_val = theta_val,
  Z_train = Z_train,
  Z_val = Z_val)

##### Simulation on-the-fly using R functions #####

juliaEval("using RCall") # requires the Julia package RCall
estimator <- train(estimator, sampler = sampler, simulator = simulator, m = m)

##### Simulation on-the-fly using Julia functions #####

# Defining the sampler and simulator in Julia can improve computational
# efficiency by avoiding the overhead of communicating between R and Julia.

juliaEval("using Distributions")

```

```
# Parameter sampler
sampler <- juliaEval("
    function sampler(K)
        mu = rand(Normal(0, 1), K)
        sigma = rand(Gamma(1), K)
        theta = hcat(mu, sigma)'
        return theta
    end")

# Data simulator
simulator <- juliaEval("
    function simulator(theta_matrix, m)
        Z = [rand(Normal(theta[1], theta[2]), 1, m) for theta in eachcol(theta_matrix)]
        return Z
    end")

# Train
estimator <- train(estimator, sampler = sampler, simulator = simulator, m = m)

## End(Not run)
```

Index

assess, [2](#)
assess(), [3](#), [15](#), [16](#), [22](#)

bias, [3](#)
bias(), [3](#), [15](#), [16](#)
bootstrap, [4](#)

cpu_device, [4](#)
cpu_device(), [7](#), [15](#), [22](#)

encodedata, [5](#)
estimate, [6](#)
estimate(), [7](#), [17](#), [22](#)

gpu_device, [6](#)
gpu_device(), [5](#), [15](#), [22](#)

infer, [7](#)
infer(), [6](#), [8](#), [12–14](#), [17](#), [22](#)

loadstate, [7](#)
logratio, [8](#)

plotdistribution, [9](#)
plotdistribution(), [3](#)
plotestimates, [11](#)
plotestimates(), [3](#)
PointEstimator, [12](#)
PointEstimator(), [13](#), [14](#)
PosteriorEstimator, [13](#)
PosteriorEstimator(), [12](#), [14](#)

RatioEstimator, [14](#)
RatioEstimator(), [12](#), [13](#)
reactant_device, [14](#)
reactant_device(), [5](#), [7](#), [22](#)
risk, [15](#)
risk(), [3](#), [16](#)
rmse, [16](#)
rmse(), [3](#), [15](#)

sampleposterior, [16](#)
sampleposterior(), [6–8](#), [22](#)
savestate, [17](#)
spatialgraph, [17](#)

tanhloss, [19](#)
train, [19](#), [20](#)
train(), [4–7](#), [12–15](#)