Package 'bnns'

June 7, 2026

Title Bayesian Neural Network with 'Stan'

Version 1.0.0

Description Offers a flexible formula-based interface for building and training Bayesian Neural Networks powered by 'Stan'. The package supports modeling complex relationships while providing rigorous uncertainty quantification via posterior distributions. With features like user chosen priors, clear predictions, and support for regression, binary, and multi-class classification, it is well-suited for applications in clinical trials, finance, and other fields requiring robust Bayesian inference and decision-making. References: Neal(1996) <doi:10.1007/978-1-4612-0745-0>.

License MIT + file LICENSE

Depends R (>= 4.2.0)

Encoding UTF-8

Suggests bayesplot, covr, dials, dplyr, ggplot2, knitr, mlbench, OpenCL, parsnip (>= 1.0.0), ranger, recipes, rlang, rmarkdown, rsample, testthat (>= 3.0.0), tidymodels, tune, workflows

Config/testthat/edition 3

Imports BH, digest, loo, pROC, RcppEigen, rstan, stats, tibble

URL https://github.com/swarnendu-stat/bnns, https://swarnendu-stat.github.io/bnns/

BugReports https://github.com/swarnendu-stat/bnns/issues

VignetteBuilder knitr

Config/Needs/website rmarkdown

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Swarnendu Chatterjee [aut, cre, cph]

Maintainer Swarnendu Chatterjee <swarnendu.stat@gmail.com>

Repository CRAN

Date/Publication 2026-06-07 18:30:02 UTC

Contents

---

bnns *Generic Function for Fitting Bayesian Neural Network Models*

Description

This is a generic function for fitting Bayesian Neural Network (BNN) models. It dispatches to methods based on the class of the input data.

Usage

```
bnns(
  formula,
  data,
  L = 1,
  nodes = rep(2, L),
  act_fn = rep(2, L),
  out_act_fn = 1,
  algorithm = c("NUTS", "HMC"),
  iter = 1000,
  warmup = 200,
  thin = 1,
  chains = 2,
  cores = 2,
  seed = 123,
```

```
  prior_weights = NULL,
  prior_bias = NULL,
  prior_sigma = NULL,
  verbose = FALSE,
  refresh = max(iter/10, 1),
  normalize = TRUE,
  backend = c("rstan", "cmdstanr"),
  use_gpu = FALSE,
  opencl_ids = c(0, 0),
  ...
)
```

Arguments

formula	A symbolic description of the model to be fitted. The formula should specify the response variable and predictors (e.g., y ~ x1 + x2). y must be continuous for regression (out_act_fn = 1), numeric 0/1 for binary classification (out_act_fn = 2), and factor with at least 3 levels for multi-classification (out_act_fn = 3).
data	A data frame containing the variables in the model.
L	An integer specifying the number of hidden layers in the neural network. Default is 1.
nodes	An integer or vector specifying the number of nodes in each hidden layer. If a single value is provided, it is applied to all layers. Default is 16.
act_fn	An integer or vector specifying the activation function(s) for the hidden layers. Options are: • 1 for tanh • 2 for sigmoid (default) • 3 for softplus • 4 for ReLU • 5 for linear
out_act_fn	An integer or character string specifying the activation function for the output layer. Options are: • 1 or "linear" for linear (default) • 2 or "sigmoid" for sigmoid • 3 or "softmax" for softmax
algorithm	A character string specifying the MCMC algorithm. Options are "NUTS" (default) or "HMC".
iter	An integer specifying the total number of iterations for the Stan sampler. Default is 1e3.
warmup	An integer specifying the number of warmup iterations for the Stan sampler. Default is 2e2.
thin	An integer specifying the thinning interval for Stan samples. Default is 1.
chains	An integer specifying the number of Markov chains. Default is 2.

`cores` An integer specifying the number of CPU cores to use for parallel sampling. Default is 2.

`seed` An integer specifying the random seed for reproducibility. Default is 123.

`prior_weights` A list specifying the prior distribution for the weights in the neural network. The list must include two components:

- `dist`: A character string specifying the distribution type. Supported values are `"normal"`, `"uniform"`, `"cauchy"`, and `"horseshoe"`.
- `params`: A named list specifying the parameters for the chosen distribution:
  - For `"normal"`: Provide `mean` (mean of the distribution) and `sd` (standard deviation).
  - For `"uniform"`: Provide `alpha` (lower bound) and `beta` (upper bound).
  - For `"cauchy"`: Provide `mu` (location parameter) and `sigma` (scale parameter).

For the `"horseshoe"` prior, `params` is not needed as it uses a standard half-Cauchy setup. If `prior_weights` is `NULL`, the default prior is a `normal(0, 1)` distribution. For example:

- `list(dist = "normal", params = list(mean = 0, sd = 1))`
- `list(dist = "uniform", params = list(alpha = -1, beta = 1))`
- `list(dist = "cauchy", params = list(mu = 0, sigma = 2.5))`

`prior_bias` A list specifying the prior distribution for the biases in the neural network. The list must include two components:

- `dist`: A character string specifying the distribution type. Supported values are `"normal"`, `"uniform"`, `"cauchy"`, and `"horseshoe"`.
- `params`: A named list specifying the parameters for the chosen distribution:
  - For `"normal"`: Provide `mean` (mean of the distribution) and `sd` (standard deviation).
  - For `"uniform"`: Provide `alpha` (lower bound) and `beta` (upper bound).
  - For `"cauchy"`: Provide `mu` (location parameter) and `sigma` (scale parameter).

If `prior_bias` is `NULL`, the default prior is a `normal(0, 1)` distribution. For example:

- `list(dist = "normal", params = list(mean = 0, sd = 1))`
- `list(dist = "uniform", params = list(alpha = -1, beta = 1))`
- `list(dist = "cauchy", params = list(mu = 0, sigma = 2.5))`

`prior_sigma` A list specifying the prior distribution for the `sigma` parameter in regression models (`out_act_fn = 1`). This allows for setting priors on the standard deviation of the residuals. The list must include two components:

- `dist`: A character string specifying the distribution type. Supported values are `"half-normal"` and `"inverse-gamma"`.
- `params`: A named list specifying the parameters for the chosen distribution:
  - For `"half-normal"`: Provide `sd` (standard deviation of the half-normal distribution).

  - For "inverse-gamma": Provide shape (shape parameter) and scale (scale parameter).

If prior_sigma is NULL, the default prior is a half-normal(0, 1) distribution. For example:

- list(dist = "half_normal", params = list(mean = 0, sd = 1))
- list(dist = "inv_gamma", params = list(alpha = 1, beta = 1))

verbose — TRUE or FALSE: flag indicating whether to print intermediate output from Stan on the console, which might be helpful for model debugging.

refresh — refresh (integer) can be used to control how often the progress of the sampling is reported (i.e. show the progress every refresh iterations). By default, refresh = max(iter/10, 1). The progress indicator is turned off if refresh <= 0.

normalize — Logical. If TRUE (default), the input predictors are normalized to have zero mean and unit variance before training. Normalization ensures stable and efficient Bayesian sampling by standardizing the input scale, which is particularly beneficial for neural network training. If FALSE, no normalization is applied, and it is assumed that the input data is already pre-processed appropriately.

backend — A character string specifying the Stan backend to use. Options are "rstan" (default) or "cmdstanr".

use_gpu — Logical. If TRUE, enables OpenCL for GPU acceleration. Default is FALSE. (Requires the "cmdstanr" backend).

opencl_ids — A vector of two integers specifying the OpenCL platform and device IDs. Default is c(0, 0).

... — Currently not in use.

Details

The function serves as a generic interface to different methods of fitting Bayesian Neural Networks. The specific method dispatched depends on the class of the input arguments, allowing for flexibility in the types of inputs supported.

Value

The result of the method dispatched by the class of the input data. Typically, this would be an object of class "bnns" containing the fitted model and associated information.

References

1. Bishop, C.M., 1995. Neural networks for pattern recognition. Oxford university press.
2. Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M.A., Guo, J., Li, P. and Riddell, A., 2017. Stan: A probabilistic programming language. Journal of statistical software, 76.
3. Neal, R.M., 2012. Bayesian learning for neural networks (Vol. 118). Springer Science & Business Media.

See Also

bnns.default

Examples

```
# Example usage with formula interface:
data <- data.frame(x1 = runif(10), x2 = runif(10), y = rnorm(10))
model <- bnns(y ~ -1 + x1 + x2,
  data = data, L = 1, nodes = 2, act_fn = 1,
  iter = 1e1, warmup = 5, chains = 1
)

# See the documentation for bnns.default for more details on the default implementation.
```

`bnns.default` *Bayesian Neural Network Model Using Formula(default) Interface*

Description

Fits a Bayesian Neural Network (BNN) model using a formula interface. The function parses the formula and data to create the input feature matrix and target vector, then fits the model using `bnns.default`.

Usage

```
## Default S3 method:
bnns(
  formula,
  data,
  L = 1,
  nodes = rep(2, L),
  act_fn = rep(2, L),
  out_act_fn = 1,
  algorithm = c("NUTS", "HMC"),
  iter = 1000,
  warmup = 200,
  thin = 1,
  chains = 2,
  cores = 2,
  seed = 123,
  prior_weights = NULL,
  prior_bias = NULL,
  prior_sigma = NULL,
  verbose = FALSE,
  refresh = max(iter/10, 1),
  normalize = TRUE,
  backend = c("rstan", "cmdstanr"),
  use_gpu = FALSE,
  opencl_ids = c(0, 0),
  ...
)
```

Arguments

formula	A symbolic description of the model to be fitted. The formula should specify the response variable and predictors (e.g., y ~ x1 + x2).
data	A data frame containing the variables in the model.
L	An integer specifying the number of hidden layers in the neural network. Default is 1.
nodes	An integer or vector specifying the number of nodes in each hidden layer. If a single value is provided, it is applied to all layers. Default is 16.
act_fn	An integer or vector specifying the activation function(s) for the hidden layers. Options are: • 1 for tanh • 2 for sigmoid (default) • 3 for softplus • 4 for ReLU • 5 for linear
out_act_fn	An integer or character string specifying the activation function for the output layer. Options are: • 1 or "linear" for linear (default) • 2 or "sigmoid" for sigmoid • 3 or "softmax" for softmax
algorithm	A character string specifying the MCMC algorithm. Options are "NUTS" (default) or "HMC".
iter	An integer specifying the total number of iterations for the Stan sampler. Default is 1e3.
warmup	An integer specifying the number of warmup iterations for the Stan sampler. Default is 2e2.
thin	An integer specifying the thinning interval for Stan samples. Default is 1.
chains	An integer specifying the number of Markov chains. Default is 2.
cores	An integer specifying the number of CPU cores to use for parallel sampling. Default is 2.
seed	An integer specifying the random seed for reproducibility. Default is 123.
prior_weights	A list specifying the prior distribution for the weights in the neural network. The list must include two components: • dist: A character string specifying the distribution type. Supported values are "normal", "uniform", "cauchy", and "horseshoe". • params: A named list specifying the parameters for the chosen distribution: – For "normal": Provide mean (mean of the distribution) and sd (standard deviation). – For "uniform": Provide alpha (lower bound) and beta (upper bound). – For "cauchy": Provide mu (location parameter) and sigma (scale parameter).

For the "horseshoe" prior, params is not needed as it uses a standard half-Cauchy setup. If prior_weights is NULL, the default prior is a normal(0, 1) distribution. For example:

- list(dist = "normal", params = list(mean = 0, sd = 1))
- list(dist = "uniform", params = list(alpha = -1, beta = 1))
- list(dist = "cauchy", params = list(mu = 0, sigma = 2.5))

prior_bias — A list specifying the prior distribution for the biases in the neural network. The list must include two components:

- dist: A character string specifying the distribution type. Supported values are "normal", "uniform", and "cauchy".
- params: A named list specifying the parameters for the chosen distribution:
  - For "normal": Provide mean (mean of the distribution) and sd (standard deviation).
  - For "uniform": Provide alpha (lower bound) and beta (upper bound).
  - For "cauchy": Provide mu (location parameter) and sigma (scale parameter).

If prior_bias is NULL, the default prior is a normal(0, 1) distribution. For example:

- list(dist = "normal", params = list(mean = 0, sd = 1))
- list(dist = "uniform", params = list(alpha = -1, beta = 1))
- list(dist = "cauchy", params = list(mu = 0, sigma = 2.5))

prior_sigma — A list specifying the prior distribution for the sigma parameter in regression models (out_act_fn = 1). This allows for setting priors on the standard deviation of the residuals. The list must include two components:

- dist: A character string specifying the distribution type. Supported values are "half-normal" and "inverse-gamma".
- params: A named list specifying the parameters for the chosen distribution:
  - For "half-normal": Provide sd (standard deviation of the half-normal distribution).
  - For "inverse-gamma": Provide shape (shape parameter) and scale (scale parameter).

If prior_sigma is NULL, the default prior is a half-normal(0, 1) distribution. For example:

- list(dist = "half_normal", params = list(mean = 0, sd = 1))
- list(dist = "inv_gamma", params = list(alpha = 1, beta = 1))

verbose — TRUE or FALSE: flag indicating whether to print intermediate output from Stan on the console, which might be helpful for model debugging.

refresh — refresh (integer) can be used to control how often the progress of the sampling is reported (i.e. show the progress every refresh iterations). By default, refresh = max(iter/10, 1). The progress indicator is turned off if refresh <= 0.

normalize — Logical. If TRUE (default), the input predictors are normalized to have zero mean and unit variance before training. Normalization ensures stable and efficient Bayesian sampling by standardizing the input scale, which is particularly beneficial for neural network training. If FALSE, no normalization is applied, and it is assumed that the input data is already pre-processed appropriately.

backend	A character string specifying the Stan backend to use. Options are "rstan" (default) or "cmdstanr".
use_gpu	Logical. If TRUE, enables OpenCL for GPU acceleration. Default is FALSE. (Requires the "cmdstanr" backend).
opencl_ids	A vector of two integers specifying the OpenCL platform and device IDs. Default is c(0, 0).
...	Currently not in use.

Details

The function uses the provided formula and data to generate the design matrix for the predictors and the response vector. It then calls helper function bnns_train to fit the Bayesian Neural Network model.

Value

An object of class "bnns" containing the fitted model and associated information, including:

- fit: The fitted Stan model object.
- data: A list containing the processed training data.
- call: The matched function call.
- formula: The formula used for the model.

Examples

```
# Example usage:
data <- data.frame(x1 = runif(10), x2 = runif(10), y = rnorm(10))
model <- bnns(y ~ -1 + x1 + x2,
  data = data, L = 1, nodes = 2, act_fn = 3,
  iter = 1e1, warmup = 5, chains = 1
)
```

---

bnns_params *Parameter functions for Bayesian Neural Networks*

Description

These functions provide dials parameter objects for tuning the hyperparameters of Bayesian Neural Networks.

Usage

```
L(range = c(1L, 5L), trans = NULL)

warmup(range = c(100L, 1000L), trans = NULL)

chains(range = c(1L, 4L), trans = NULL)

iter(range = c(500L, 2000L), trans = NULL)

nodes(range = c(1L, 64L), trans = NULL)

act_fn(values = c("tanh", "sigmoid", "softplus", "relu", "linear"))
```

Arguments

range	A two-element vector holding the defaults for the smallest and largest possible values.
trans	A trans object from the scales package, could be NULL.
values	A character vector of possible values.

Value

A quant_param or qual_param object from the dials package.

---

load_bnns *Load a fitted bnns model from disk*

Description

Load a fitted bnns model from disk

Usage

```
load_bnns(file)
```

Arguments

file	A character string specifying the path to the saved model.

Value

A fitted bnns object.

loo.bnns *Leave-One-Out Cross-Validation (LOO) for bnns models*

Description

Leave-One-Out Cross-Validation (LOO) for bnns models

Usage

```
## S3 method for class 'bnns'
loo(x, ...)
```

Arguments

x	A fitted bnns model object.
...	Additional arguments passed to loo::loo().

Value

A loo object containing model comparison metrics.

measure_bin *Measure Performance for Binary Classification Models*

Description

Evaluates the performance of a binary classification model using a confusion matrix and accuracy.

Usage

```
measure_bin(obs, pred, cut = 0.5)
```

Arguments

obs	A numeric or integer vector of observed binary class labels (0 or 1).
pred	A numeric vector of predicted probabilities for the positive class.
cut	A numeric threshold (between 0 and 1) to classify predictions into binary labels.

Value

A list containing:

conf_mat A confusion matrix comparing observed and predicted class labels.

accuracy The proportion of correct predictions.

ROC ROC generated using pROC::roc

AUC Area under the ROC curve.

Examples

```
obs <- c(1, 0, 1, 1, 0)
pred <- c(0.9, 0.4, 0.8, 0.7, 0.3)
cut <- 0.5
measure_bin(obs, pred, cut)
# Returns: list(conf_mat = <confusion matrix>, accuracy = 1, ROC = <ROC>, AUC = 1)
```

---

measure_cat *Measure Performance for Multi-Class Classification Models*

Description

Evaluates the performance of a multi-class classification model using log loss and multiclass AUC.

Usage

```
measure_cat(obs, pred)
```

Arguments

obs	A factor vector of observed class labels. Each level represents a unique class.
pred	A numeric matrix of predicted probabilities, where each row corresponds to an observation, and each column corresponds to a class. The number of columns must match the number of levels in obs.

Details

The log loss is calculated as:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(p_{ic})$$

where y_{ic} is 1 if observation i belongs to class c, and p_{ic} is the predicted probability for that class.

The AUC is computed using the pROC::multiclass.roc function, which provides an overall measure of model performance for multiclass classification.

Value

A list containing:

log_loss The negative log-likelihood averaged across observations.

ROC ROC generated using pROC::roc

AUC The multiclass Area Under the Curve (AUC) as computed by pROC::multiclass.roc.

Examples

```
library(pROC)
obs <- factor(c("A", "B", "C"), levels = LETTERS[1:3])
pred <- matrix(
  c(
    0.8, 0.1, 0.1,
    0.2, 0.6, 0.2,
    0.7, 0.2, 0.1
  ),
  nrow = 3, byrow = TRUE
)
measure_cat(obs, pred)
# Returns: list(log_loss = 1.012185, ROC = <ROC>, AUC = 0.75)
```

`measure_cont` *Measure Performance for Continuous Response Models*

Description

Evaluates the performance of a continuous response model using RMSE and MAE.

Usage

```
measure_cont(obs, pred)
```

Arguments

`obs`	A numeric vector of observed (true) values.
`pred`	A numeric vector of predicted values.

Value

A list containing:

`rmse` Root Mean Squared Error.

`mae` Mean Absolute Error.

Examples

```
obs <- c(3.2, 4.1, 5.6)
pred <- c(3.0, 4.3, 5.5)
measure_cont(obs, pred)
# Returns: list(rmse = 0.1732051, mae = 0.1666667)
```

opencl_diagnostics *OpenCL Diagnostic Information*

Description

This helper function lists the available OpenCL platforms and devices on your system. It is useful for determining the correct `opencl_ids` to pass to `bnns()` when using GPU acceleration.

Usage

```
opencl_diagnostics()
```

Details

The function first checks if the `clinfo` system command is available. If not, it falls back to looking for the `OpenCL` R package to retrieve the platforms and devices.

Value

Invoked for its side effect of printing OpenCL diagnostic information.

plot.bnns *Plot diagnostics for a fitted Bayesian Neural Network*

Description

Generates Markov Chain Monte Carlo (MCMC) trace plots, posterior density plots, Posterior Predictive Checks (PPC), or predicted probability distributions for the fitted model.

Usage

```
## S3 method for class 'bnns'
plot(
  x,
  type = c("trace", "density", "posterior_predictive", "pred_prob"),
  pars = NULL,
  ...
)
```

Arguments

x	A fitted bnns model object.
type	Character string indicating the type of plot. Options are "trace" for MCMC trace plots, "density" for posterior density plots, "posterior_predictive" for Posterior Predictive Checks, and "pred_prob" for visualizing the predicted class probability distributions (classification only).
pars	A character vector of parameter names to include in the plot. By default, this focuses on the output layer ("w_out", "b_out", and "sigma") to avoid cluttering the plot device with hundreds of hidden layer weights.
...	Additional arguments passed to stan_trace, stan_dens, or ppc_dens_overlay.

Value

A ggplot object containing the requested diagnostic plots.

---

predict.bnns *Predictions from a fitted Bayesian Neural Network*

Description

Predictions from a fitted Bayesian Neural Network

Usage

```
## S3 method for class 'bnns'
predict(
  object,
  newdata = NULL,
  type = c("samples", "mean", "median", "quantile", "prob", "class"),
  quantiles = c(0.025, 0.975),
  ...
)
```

Arguments

object	A fitted bnns model object.
newdata	A data frame containing new data for prediction. If not provided, the predictions will be generated using the training data.
type	Character string indicating the type of prediction. Options are "samples" (default), "mean", "median", "quantile", "prob" (class probabilities), or "class" (predicted class labels).
quantiles	Numeric vector of probabilities used when type = "quantile". Default is c(0.025, 0.975).
...	Additional arguments passed to internal prediction methods.

Value

- For type = "samples": A matrix (regression/binary) or 3D array (multiclass) of posterior predictions.
- For type = "mean" or "median": A vector or matrix of aggregated predictions. For classification tasks, type = "mean" returns the posterior mean class probabilities.
- For type = "quantile": A matrix or array of quantiles.
- For type = "prob": A matrix of class probabilities (for classification models).
- For type = "class": A vector of predicted class labels (for classification models).

---

print.bnns *Print Method for "bnns" Objects*

Description

Displays a summary of a fitted Bayesian Neural Network (BNN) model, including the function call and the Stan fit details.

Usage

```
## S3 method for class 'bnns'
print(x, ...)
```

Arguments

x	An object of class "bnns", typically the result of a call to bnns.default.
...	Additional arguments (currently not used).

Value

The function is called for its side effects and does not return a value. It prints the following:

- The function call used to generate the "bnns" object.
- A summary of the Stan fit object stored in x$fit.

See Also

bnns, summary.bnns

Examples

```
# Example usage:
data <- data.frame(x1 = runif(10), x2 = runif(10), y = rnorm(10))
model <- bnns(y ~ -1 + x1 + x2,
  data = data, L = 1, nodes = 2, act_fn = 2,
  iter = 1e1, warmup = 5, chains = 1
)
print(model)
```

relu *relu transformation*

Description

relu transformation

Usage

```
relu(x)
```

Arguments

x	A numeric vector or matrix on which relu transformation is going to be applied.

Value

A numeric vector or matrix after relu transformation.

Examples

```
relu(matrix(1:4, , nrow = 2))
```

save_bnns *Save a fitted bnns model to disk*

Description

Save a fitted bnns model to disk

Usage

```
save_bnns(object, file)
```

Arguments

object	A fitted bnns object.
file	A character string specifying the path where the model should be saved (usually ending in .rds).

sigmoid *sigmoid transformation*

Description

sigmoid transformation

Usage

```
sigmoid(x)
```

Arguments

x	A numeric vector or matrix on which sigmoid transformation is going to be applied.

Value

A numeric vector or matrix after sigmoid transformation.

Examples

```
sigmoid(matrix(1:4, nrow = 2))
```

softmax_3d *Apply Softmax Function to a 3D Array*

Description

This function applies the softmax transformation along the third dimension of a 3D array. The softmax function converts raw scores into probabilities such that they sum to 1 for each slice along the third dimension.

Usage

```
softmax_3d(x)
```

Arguments

x	A 3D array. The input array on which the softmax function will be applied.

Details

The softmax transformation is computed as:

$$\text{softmax}(x_{ijk}) = \frac{\exp(x_{ijk})}{\sum_l \exp(x_{ijl})}$$

This is applied for each pair of indices `(i, j)` across the third dimension `(k)`.

The function processes the input array slice-by-slice for the first two dimensions `(i, j)`, normalizing the values along the third dimension `(k)` for each slice.

Value

A 3D array of the same dimensions as `x`, where the values along the third dimension are transformed using the softmax function.

Examples

```
# Example: Apply softmax to a 3D array
x <- array(runif(24), dim = c(2, 3, 4)) # Random 3D array (2x3x4)
softmax_result <- softmax_3d(x)
```

---

softplus *softplus transformation*

Description

softplus transformation

Usage

```
softplus(x)
```

Arguments

`x`	A numeric vector or matrix on which softplus transformation is going to be applied.

Value

A numeric vector or matrix after softplus transformation.

Examples

```
softplus(matrix(1:4, nrow = 2))
```

summary.bnns *Summary of a Bayesian Neural Network (BNN) Model*

Description

Provides a comprehensive summary of a fitted Bayesian Neural Network (BNN) model, including details about the model call, data, network architecture, posterior distributions, and model fitting information.

Usage

```
## S3 method for class 'bnns'
summary(object, ...)
```

Arguments

object	An object of class bnns, representing a fitted Bayesian Neural Network model.
...	Additional arguments (currently unused).

Details

The function prints the following information:

- **Call:** The original function call used to fit the model.
- **Data Summary:** Number of observations and features in the training data.
- **Network Architecture:** Structure of the BNN including the number of hidden layers, nodes per layer, and activation functions.
- **Posterior Summary:** Summarized posterior distributions of key parameters (e.g., weights, biases, and noise parameter).
- **Model Fit Information:** Bayesian sampling details, including the number of iterations, warmup period, thinning, and chains.
- **Notes:** Remarks and warnings, such as checks for convergence diagnostics.

Value

A list (returned invisibly) containing the following elements:

- "Number of observations": The number of observations in the training data.
- "Number of features": The number of features in the training data.
- "Number of hidden layers": The number of hidden layers in the neural network.
- "Nodes per layer": A comma-separated string representing the number of nodes in each hidden layer.
- "Activation functions": A comma-separated string representing the activation functions used in each hidden layer.
- "Output activation function": The activation function used in the output layer.

- "Stanfit Summary": A summary of the Stan model, including key parameter posterior distributions.
- "Iterations": The total number of iterations used for sampling in the Bayesian model.
- "Warmup": The number of iterations used as warmup in the Bayesian model.
- "Thinning": The thinning interval used in the Bayesian model.
- "Chains": The number of Markov chains used in the Bayesian model.
- "Performance": Predictive performance metrics, which vary based on the output activation function.

The function also prints the summary to the console.

See Also

bnns, print.bnns

Examples

```
# Fit a Bayesian Neural Network
data <- data.frame(x1 = runif(10), x2 = runif(10), y = rnorm(10))
model <- bnns(y ~ -1 + x1 + x2,
  data = data, L = 1, nodes = 2, act_fn = 2,
  iter = 1e1, warmup = 5, chains = 1
)

# Get a summary of the model
summary(model)
```

---

waic.bnns *Watanabe-Akaike Information Criterion (WAIC) for bnns models*

Description

Watanabe-Akaike Information Criterion (WAIC) for bnns models

Usage

```
## S3 method for class 'bnns'
waic(x, ...)
```

Arguments

x	A fitted bnns model object.
...	Additional arguments passed to loo::waic().

Value

A waic object containing model comparison metrics.

Index