Package ‘LBBNN’

December 10, 2025

Title Latent Binary Bayesian Neural Networks Using 'torch'

Version 0.1.2

Maintainer Lars Skaaret-Lund <lars.skaaret-lund@nmbu.no>

Description Latent binary Bayesian neural networks (LBBNNs) are implemented using 'torch', an R interface to the LibTorch backend. Supports mean-field variational inference as well as flexible variational posteriors using normalizing flows. The standard LBBNN implementation follows Hubin and Storvik (2024) <doi:10.3390/math12060788>, using the local reparametrization trick as in Skaaret-Lund et al. (2024) <https://openreview.net/pdf?id=d6kqUKzG3V>. Input-skip connections are also supported, as described in Høyheim et al. (2025) <doi:10.48550/arXiv.2503.10496>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Language en-US

Suggests testthat (>= 3.0.0), knitr, rmarkdown, torchvision

Config/testthat/edition 3

Depends R (>= 3.5)

LazyData true

VignetteBuilder knitr

Imports ggplot2, torch, igraph, coro, svglite

NeedsCompilation no

Author Lars Skaaret-Lund [aut, cre],
Aliaksandr Hubin [aut],
Eirik Høyheim [aut]

Repository CRAN

Date/Publication 2025-12-10 09:50:02 UTC

Contents

`coef.LBBNN_Net` *Get model coefficients (local explanations) of an* `LBBNN_Net` *object*

Description

Given an input sample x_1,... x_j (with j the number of variables), the local explanation is found by considering active paths. If relu activation functions are assumed, each path is a piecewise linear function, so the contribution for x_j is just the sum of the weights associated with the paths connecting x_j to the output. The contributions are found by taking the gradient wrt x.

Usage

```
## S3 method for class 'LBBNN_Net'
coef(
  object,
  dataset,
  inds = NULL,
  output_neuron = 1,
  num_data = 1,
  num_samples = 10,
```

```
  ...
)
```

Arguments

`object`	an object of class `LBBNN_Net`.
`dataset`	Either a `torch::dataloader` object, or a `torch::torch_tensor` object. The former is assumed to be the same `torch::dataloader` used for training or testing. The latter can be any user-defined data.
`inds`	Optional integer vector of row indices in the dataset to compute explanations for.
`output_neuron`	integer, which output neuron to explain (default = 1).
`num_data`	integer, if no indices are chosen, the first `num_data` of `dataset` are automatically used for explanations.
`num_samples`	integer, how many samples to use for model averaging when sampling the weights in the active paths.
`...`	further arguments passed to or from other methods.

Details

- If `num_data` = 1, confidence intervals are computed using model averaging over `num_samples` weight samples.
- If `num_data` > 1, confidence intervals are computed across the mean explanations for each sample.
- The output is a data frame with row names as input variables (`x0`, `x1`, `x2`, ...) and columns giving mean and 95% confidence intervals for each variable.

Value

A data frame with rows corresponding to input variables and the following columns:

- `lower`: lower bound of the 95% confidence interval
- `mean`: mean contribution of the variable
- `upper`: upper bound of the 95% confidence interval

Examples

```
x<-torch::torch_randn(3,2)
b <- torch::torch_rand(2)
y <- torch::torch_matmul(x,b)
train_data <- torch::tensor_dataset(x,y)
train_loader <- torch::dataloader(train_data,batch_size = 3,shuffle=FALSE)
problem<-'regression'
sizes <- c(2,1,1)
inclusion_priors <-c(0.9,0.2)
inclusion_inits <- matrix(rep(c(-10,10),2),nrow = 2,ncol = 2)
stds <- c(1.0,1.0)
model <- LBBNN_Net(problem,sizes,inclusion_priors,stds,inclusion_inits,flow = FALSE,
```

```
input_skip = TRUE)
train_LBBNN(epochs = 1,LBBNN = model, lr = 0.01,train_dl = train_loader)
coef(model,dataset = x, num_data = 1)
```

Custom_activation *Generate a custom activation function.*

Description

The first 3 entries are customized in order to see if we can learn that structure. The rest will be relu as usual

Usage

```
Custom_activation()
```

Value

Returns a ‘torch::nn_module that can be used in an LBBNN_Net

FLOW *Class to generate a normalizing flow*

Description

Used inLBBNN_Net when the argument flow = TRUE. Contains a torch::nn_module where the initial vector gets transformed through all the layers in the module. Also computes the log-determinant of the Jacobian for the entire transformation, which is just the sum of log-determinant of the independent layers.

Usage

```
FLOW(input_dim, transform_type, num_transforms)
```

Arguments

input_dim	numeric vector, the dimensionality of each layer. The first item is the input vector size.
transform_type	Type of transformation. Currently only RNVP is implemented.
num_transforms	integer, how many layers of transformations to include in the flow.

Value

A torch::nn_module object representing the normalizing flow. The module provides:

forward(z) Applies all flow transformation layers to the input tensor z. Returns a named list containing:

- z A torch_tensor containing the transformed version of the input, with the same shape as z.
- logdet A scalar torch_tensor equal to the sum of the log-determinants of all transformation layers.

Examples

```
flow <- FLOW(c(200,100,100),transform_type = 'RNVP',num_transforms = 3)
flow$to(device = 'cpu')
x <- torch::torch_rand(200,device = 'cpu')
output <- flow(x)
z_out <- output$z
print(dim(z_out))
log_det <- output$logdet
print(log_det)
```

Gallstone_Dataset *Gallstone Dataset*

Description

Taken from the UCI machine learning repository. The task is to classify whether the patient had gallstones or not. It contains a mix of demographic data and bioimpedance data.

Usage

```
Gallstone_Dataset
```

Format

This dataset has 319 rows and 38 columns.

Source

https://pmc.ncbi.nlm.nih.gov/articles/PMC11309733/#T2

get_dataloaders — *Wrapper around* `torch::dataloader`

Description

Avoids users having to manually define their own dataloaders.

Usage

```
get_dataloaders(
  dataset,
  train_proportion,
  train_batch_size,
  test_batch_size,
  standardize = TRUE,
  shuffle_train = TRUE,
  shuffle_test = FALSE,
  seed = 1
)
```

Arguments

`dataset`	A `data.frame`. The last column is assumed to be the dependent variable.
`train_proportion`	numeric, between 0 and 1. Proportion of data to be used for training.
`train_batch_size`	integer, number of samples per batch in the training dataloader.
`test_batch_size`	integer, number of sampels per batch in the testing dataloader.
`standardize`	logical, whether to standardize input-features, default is TRUE.
`shuffle_train`	logical, whether to shuffle the training data each epoch. default is TRUE
`shuffle_test`	logical, shuffle test data, default is FALSE. Usually not needed.
`seed`	integer. Used for reproducibility purposes in the train/test split.

Value

A list containing:

train_loader A `torch::dataloader` for the training data.

test_loader A `torch::dataloader` for the test data.

get_local_explanations_gradient

Function to get gradient based local explanations for input-skip LBBNNs.

Description

Works by computing the gradient wrt to input, given we have relu activation functions.

Usage

```
get_local_explanations_gradient(
  model,
  input_data,
  num_samples = 1,
  magnitude = TRUE,
  include_potential_contribution = FALSE,
  device = "cpu"
)
```

Arguments

`model`	A `LBBNN_Net` with input-skip
`input_data`	The data to be explained (one sample).
`num_samples`	integer, how many samples to use to produce credible intervals.
`magnitude`	If TRUE, only return explanations. If FALSE, multiply by input values.
`include_potential_contribution`	IF TRUE, If covariate=0, we assume that the contribution is negative (good/bad that it is not included) if FALSE, just removes zero covariates.
`device`	character, the device to be trained on. Default is 'cpu', can be 'mps' or 'gpu'.

Value

A list with the following elements:

explanations A `torch::tensor` of shape (num_samples, p, num_classes).

p integer, the number of input features.

predictions A `torch::tensor` of shape (num_samples, num_classes).

LBBNN_Conv2d *Class to generate an LBBNN convolutional layer.*

Description

It supports:

- Prior inclusion probabilities for weights and biases in each layer.
- Standard deviation priors for weights and biases in each layer.
- Optional normalizing flows (RNVP) for a more flexible variational posterior.
- Forward pass using either the full model or the Median Probability Model (MPM).
- Computation of the KL-divergence.

Usage

```
LBBNN_Conv2d(
  in_channels,
  out_channels,
  kernel_size,
  prior_inclusion,
  standard_prior,
  density_init,
  flow = FALSE,
  num_transforms = 2,
  hidden_dims = c(200, 200),
  device = "cpu"
)
```

Arguments

`in_channels`	integer, number of input channels.
`out_channels`	integer, number of output channels.
`kernel_size`	size of the convolving kernel.
`prior_inclusion`	numeric scalar, prior inclusion probability for each weight and bias in the layer.
`standard_prior`	numeric scalar, prior standard deviation for weights and biases in each layer.
`density_init`	A numeric of size 2, used to initialize the inclusion parameters, one for each layer.
`flow`	logical, whether to use normalizing flows
`num_transforms`	integer, number of transformations for `flow`. Default is 2.
`hidden_dims`	numeric vector, dimension of the hidden layer(s) in the neural networks of the RNVP transform.
`device`	The device to be used. Default is CPU.

Value

A `torch::nn_module` object representing a convolutional LBBNN layer. The module has the following methods:

- `forward(input, MPM = FALSE)`: Computes activation (using the LRT at training time) of a batch of inputs.
- `kl_div()`: Computes the KL-divergence.
- `sample_z()`: Samples from the flow if `flow = TRUE`, in addition returns the log-determinant of the Jacobian of the transformation.

Examples

```
layer <- LBBNN_Conv2d(in_channels = 1,out_channels = 32,kernel_size = c(3,3),
prior_inclusion = 0.2,standard_prior = 1,density_init = c(-10,10),device = 'cpu')
x <-torch::torch_randn(100,1,28,28)
out <-layer(x)
print(dim(out))
```

`LBBNN_Linear` *Class to generate an LBBNN feed forward layer*

Description

This module implements a fully connected LBBNN layer. It supports:

- Prior inclusion probabilities for weights and biases in each layer.
- Standard deviation priors for weights and biases in each layer.
- Optional normalizing flows (RNVP) for a more flexible variational posterior.
- Forward pass using either the full model or the Median Probability Model (MPM).
- Computation of the KL-divergence.

Usage

```
LBBNN_Linear(
  in_features,
  out_features,
  prior_inclusion,
  standard_prior,
  density_init,
  flow = FALSE,
  num_transforms = 2,
  hidden_dims = c(200, 200),
  device = "cpu",
  bias_inclusion_prob = FALSE,
  conv_net = FALSE
)
```

Arguments

- `in_features` integer, number of input neurons.
- `out_features` integer, number of output neurons.
- `prior_inclusion` numeric scalar, prior inclusion probability for each weight and bias in the layer.
- `standard_prior` numeric scalar, prior standard deviation for weights and biases in each layer.
- `density_init` A numeric of size 2, used to initialize the inclusion parameters, one for each layer.
- `flow` logical, whether to use normalizing flows
- `num_transforms` integer, number of transformations for `flow`. Default is 2.
- `hidden_dims` numeric vector, dimension of the hidden layer(s) in the neural networks of the RNVP transform.
- `device` The device to be used. Default is CPU.
- `bias_inclusion_prob` logical, determines whether the bias should be as associated with inclusion probabilities.
- `conv_net` logical, whether the layer is to be used in a convolutional net.

Value

A `torch::nn_module` object representing a fully connected LBBNN layer. The module has the following methods:

- `forward(input, MPM = FALSE)`: Computes activation (using the LRT at training time) of a batch of inputs.
- `kl_div()`: Computes the KL-divergence.
- `sample_z()`: Samples from the flow if `flow = TRUE`, in addition returns the log-determinant of the Jacobian of the transformation.

Examples

```
l1 <- LBBNN_Linear(in_features = 10,out_features = 5,prior_inclusion = 0.25,
standard_prior = 1,density_init = c(0,1),flow = FALSE)
x <- torch::torch_rand(20,10,requires_grad = FALSE)
output <- l1(x,MPM = FALSE) #the forward pass, output has shape (20,5)
print(l1$kl_div()$item()) #compute KL-divergence after the forward pass
```

LBBNN_Net *Feed-forward Latent Binary Bayesian Neural Network (LBBNN)*

Description

Each layer is defined by LBBNN_Linear. For example, sizes = c(20, 200, 200, 5) generates a network with:

- 20 input features,
- two hidden layers of 200 neurons each,
- an output layer with 5 neurons.

Usage

```
LBBNN_Net(
  problem_type,
  sizes,
  prior,
  std,
  inclusion_inits,
  input_skip = FALSE,
  flow = FALSE,
  num_transforms = 2,
  dims = c(200, 200),
  device = "cpu",
  raw_output = FALSE,
  custom_act = NULL,
  link = NULL,
  nll = NULL,
  bias_inclusion_prob = FALSE
)
```

Arguments

problem_type	character, one of: 'binary classification', 'multiclass classification', 'regression', or 'custom'.
sizes	Integer vector specifying the layer sizes of the network. The first element is the input size, the last is the output size, and the intermediate integers represent hidden layers.
prior	numeric vector of prior inclusion probabilities for each weight matrix. length must be length(sizes) - 1.
std	numeric vector of prior standard deviation for each weight matrix. length must be length(sizes) - 1.
inclusion_inits	numeric matrix of shape (2, number of weight matrices) specifying the lower and upper bounds for initializations of the inclusion parameters.

input_skip	logical, whether to include input_skip.
flow	logical, whether to use normalizing flows.
num_transforms	integer, how many transformations to use in the flow.
dims	numeric vector, hidden dimension for the neural network in the RNVP transform.
device	the device to be trained on. Can be 'cpu', 'gpu' or 'mps'. Default is cpu.
raw_output	logical, whether the network skips the last sigmoid/softmax layer to compute local explanations.
custom_act	Allows the user to submit their own customized activation function.
link	User can define their own link function (not implemented yet).
nll	User can define their own likelihood function (not implemented yet).
bias_inclusion_prob	logical, determines whether the bias should be as associated with inclusion probabilities.

Value

A `torch::nn_module` object representing the LBBNN. It includes the following methods:

- `forward(x, MPM = FALSE)`: Performs a forward pass through the whole network.
- `kl_div()`: Returns the KL divergence of the network.
- `density()`: Returns the density of the whole network, i.e. the proportion of weights with inclusion probabilities greater than 0.5.
- `compute_paths()`: Computes active paths through the network without input-skip.
- `compute_paths_input_skip()`: Computes active paths with input-skip enabled.
- `density_active_path()`: Returns network density after removing inactive paths.

Examples

```
layers <- c(10,2,5)
alpha <- c(0.3,0.9)
stds <- c(1.0,1.0)
inclusion_inits <- matrix(rep(c(-10,10),2),nrow = 2,ncol = 2)
prob <- 'multiclass classification'
net <- LBBNN_Net(problem_type = prob, sizes = layers, prior = alpha,std = stds
,inclusion_inits = inclusion_inits,input_skip = FALSE,flow = FALSE,device = 'cpu')
x <- torch::torch_rand(20,10,requires_grad = FALSE)
output <- net(x)
net$kl_div()$item()
net$density()
```

LBBNN_plot — *Function to plot an input skip structure after removing weights in non-active paths.*

Description

Uses igraph to plot.

Usage

```
LBBNN_plot(
  model,
  layer_spacing = 1,
  neuron_spacing = 1,
  vertex_size = 10,
  label_size = 0.5,
  edge_width = 0.5,
  save_svg = NULL
)
```

Arguments

model	A trained LBBNN_Net model with input_skip enabled.
layer_spacing	numeric, spacing in between layers.
neuron_spacing	numeric, spacing between neurons within a layer.
vertex_size	numeric, size of the neurons.
label_size	numeric, size of the text within neurons.
edge_width	numeric, width of the edges connecting neurons.
save_svg	the path where the plot will be saved if save_svg is not null.

Value

This function produces plots as a side effect and does not return a value.

Examples

```
sizes <- c(2,3,3,2)
problem <- 'multiclass classification'
inclusion_priors <- c(0.1,0.1,0.1)
std_priors <- c(1.0,1.0,1.0)
inclusion_inits <- matrix(rep(c(-10,10),3), nrow = 2, ncol = 3)
device <- 'cpu'
torch::torch_manual_seed(0)
model <- LBBNN_Net(problem_type = problem, sizes = sizes,
                   prior = inclusion_priors, inclusion_inits = inclusion_inits,
                   input_skip = TRUE, std = std_priors, flow = FALSE,
```

```
                     num_transforms = 2, dims = c(200,200), device = device)
model$compute_paths_input_skip()
LBBNN_plot(model, 1, 1, 14, 1)
```

mgp_dataset *Auto MPG daataset*

Description

Taken from the UCI machine learning repository.

Usage

```
mgp_dataset
```

Format

This dataset has 398 rows and 24 columns.

Source

https://archive.ics.uci.edu/dataset/9/auto+mpg

Mice_Dataset *Mice Dataset*

Description

Only used for internal testing for now.

Usage

```
Mice_Dataset
```

Format

This dataset has 1080 rows and 78 columns.

Source

https://pubmed.ncbi.nlm.nih.gov/26111164/

plot.LBBNN_Net — *Plot* LBBNN_Net *objects*

Description

Given a trained LBBNN_Net model, this function produces either:

- **Global plot**: a visualization of the network structure, showing only the active paths.
- **Local explanation**: a plot of the local explanation for a single input sample, including error bars obtained from Monte Carlo sampling of the network weights.

Usage

```
## S3 method for class 'LBBNN_Net'
plot(x, type = c("global", "local"), data = NULL, num_samples = 100, ...)
```

Arguments

x	An instance of LBBNN_Net.
type	Either "global" or "local".
data	If local is chosen, one sample must be provided to obtain the explanation. Must be a torch::torch_tensor of shape (1,p).
num_samples	integer, how many samples to use for model averaging over the weights in case of local explanations.
...	further arguments passed to or from other methods.

Value

No return value. Called for its side effects of producing a plot.

plot_local_explanations_gradient — *Plot the gradient based local explanations for one sample with input-skip LBBNNs.*

Description

Plots the contribution of each covariate, and the prediction, with error bars.

Usage

```
plot_local_explanations_gradient(
  model,
  input_data,
  num_samples,
  device = "cpu",
  save_svg = NULL
)
```

Arguments

`model`	An instance of `LBBNN_Net` with input-skip enabled.
`input_data`	The data to be explained (one sample).
`num_samples`	integer, how many sample to use to produce credible intervals.
`device`	character, the device to be trained on. Default is cpu. Can be 'mps' or 'gpu'.
`save_svg`	the path where the plot will be saved as svg, if save_svg is not NULL.

Value

This function produces plots as a side effect and does not return a value.

`predict.LBBNN_Net` *Obtain predictions from the variational posterior of an* `LBBNN model`

Description

Draw from the (variational) posterior distribution of a trained `LBBNN_Net` object.

Usage

```
## S3 method for class 'LBBNN_Net'
predict(
  object,
  newdata,
  mpm = FALSE,
  draws = 10,
  device = "cpu",
  link = NULL,
  ...
)
```

Arguments

`object`	A trained `LBBNN_Net` object
`newdata`	A `torch::dataloader` object containing the data with which to predict.
`mpm`	logical, whether to use the median probability model.
`draws`	integer, the number of samples to draw from the posterior.
`device`	character, device for computation (default = `"cpu"`).
`link`	Optional link function to apply to the network output. Currently not implemented.
`...`	further arguments passed to or from other methods.

Value

A `torch::torch_tensor` of shape `(draws,N,C)` where `N` is the number of samples in `newdata`, and `C` the number of outputs.

Examples

```
x<-torch::torch_randn(3,2)
b <- torch::torch_rand(2)
y <- torch::torch_matmul(x,b)
train_data <- torch::tensor_dataset(x,y)
train_loader <- torch::dataloader(train_data,batch_size = 3,shuffle=FALSE)
problem<-'regression'
sizes <- c(2,1,1)
inclusion_priors <-c(0.9,0.2)
inclusion_inits <- matrix(rep(c(-10,10),2),nrow = 2,ncol = 2)
stds <- c(1.0,1.0)
model <- LBBNN_Net(problem,sizes,inclusion_priors,stds,inclusion_inits,flow = FALSE,
input_skip = TRUE)
train_LBBNN(epochs = 1,LBBNN = model, lr = 0.01,train_dl = train_loader)
predict(model,mpm = FALSE,newdata = train_loader,draws = 1)
```

`print.LBBNN_Net` *Print summary of an* `LBBNN_Net` *object*

Description

Provides a summary of a trained `LBBNN_Net` object. Includes the model type (input-skip or not), whether normalizing flows are used, module and sub-module structure, number of trainable parameters, and prior variance and inclusion probabilities for the weights.

Usage

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

Arguments

x	An object of class LBBNN_Net.
...	Further arguments passed to or from other methods.

Value

Invisibly returns the input x.

Examples

```
x<-torch::torch_randn(3,2)
b <- torch::torch_rand(2)
y <- torch::torch_matmul(x,b)
train_data <- torch::tensor_dataset(x,y)
train_loader <- torch::dataloader(train_data,batch_size = 3,shuffle=FALSE)
problem<-'regression'
sizes <- c(2,1,1)
inclusion_priors <-c(0.9,0.2)
inclusion_inits <- matrix(rep(c(-10,10),2),nrow = 2,ncol = 2)
stds <- c(1.0,1.0)
model <- LBBNN_Net(problem,sizes,inclusion_priors,stds,inclusion_inits,flow = FALSE,
input_skip = TRUE)
print(model)
```

---

quants — *Function to obtain empirical 95% confidence interval, including the mean*

Description

Using the built in quantile function to return 95% confidence interval

Usage

```
quants(x)
```

Arguments

x	numeric vector whose sample quantiles is desired.

Value

The quantiles in addition to the mean.

`Raisin_Dataset` *Raisins Dataset*

Description

Ilkay Cinar, Murat Kokl and Sakir Tasdemi(2020) provide a dataset consisting of 2 varieties of Turkish raisins, with 450 samples of each type. The dataset contains 7 morphological features, extracted from images taken of the Raisins. The goal is to classify to one of the two types of Raisins.

Usage

```
Raisin_Dataset
```

Format

this data frame has 900 rows and the following 8 columns:

Area Number of pixels within the boundary

MajorAxisLength Length of the main axis

MinorAxisLength Length of the small axis

Eccentricity Measure of the eccentricity of the ellipse

ConvexArea The number of pixels of the smallest convex shell of the region formed by the raisin grain

Extent Ratio of the region formed by the raisin grain to the total pixels in the bounding box

Perimeter distance between the boundaries of the raisin grain and the pixels around it

Class Kecimen or Besni raisin.

Source

https://archive.ics.uci.edu/dataset/850/raisin

`residuals.LBBNN_Net` *Residuals from LBBNN fit*

Description

Residuals from an object of the `LBBNN_Net` class.

Usage

```
## S3 method for class 'LBBNN_Net'
residuals(object, type = c("response"), ...)
```

Arguments

`object`	An object of class `LBBNN_Net`.
`type`	Currently only 'response' is implemented i.e. y_true - y_predicted.
`...`	further arguments passed to or from other methods.

Value

A numeric vector of residuals (`y_true - y_predicted`)

Examples

```
x<-torch::torch_randn(3,2)
b <- torch::torch_rand(2)
y <- torch::torch_matmul(x,b)
train_data <- torch::tensor_dataset(x,y)
train_loader <- torch::dataloader(train_data,batch_size = 3,shuffle=FALSE)
problem<-'regression'
sizes <- c(2,1,1)
inclusion_priors <-c(0.9,0.2)
inclusion_inits <- matrix(rep(c(-10,10),2),nrow = 2,ncol = 2)
stds <- c(1.0,1.0)
model <- LBBNN_Net(problem,sizes,inclusion_priors,stds,inclusion_inits,flow = FALSE,
input_skip = TRUE)
train_LBBNN(epochs = 1,LBBNN = model, lr = 0.01,train_dl = train_loader)
residuals(model)
```

`RNVP_layer` *Single RNVP transform layer.*

Description

Affine half flow aka Real Non-Volume Preserving (x = z * exp(s) + t), where a randomly selected half z1 of the dimensions in z are transformed as an Affine function of the other half z2, i.e. scaled by s(z2) and shifted by t(z2). From "Density estimation using Real NVP", Dinh et al. (May 2016) https://arxiv.org/abs/1605.08803 This implementation uses the numerically stable updates introduced by IAF: https://arxiv.org/abs/1606.04934

Usage

```
RNVP_layer(hidden_sizes, device = "cpu")
```

Arguments

`hidden_sizes`	A vector of integers. The first is the dimensionality of the vector, to be transformed by RNVP. The subsequent are hidden dimensions in the MLP.
`device`	The device to be used. Default is CPU.

Value

A `torch::nn_module` object representing a single RNVP layer. The module has the following methods:

`forward(z)` Applies the RNVP transformation. Returns a `torch::torch_tensor` with the same shape as z.

`log_det()` A scalar `torch::torch_tensor` giving the log-determinant of the Jacobian of the transformation.

Examples

```
z <- torch::torch_rand(200)
layer <- RNVP_layer(c(200,50,100))
out <- layer(z)
print(dim(out))
print(layer$log_det())
```

`summary.LBBNN_Net` *Summary of LBBNN fit*

Description

Summary method for objects of the `LBBNN_Net` class. Only applies to objects trained with `input_skip = TRUE`.

Usage

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

Arguments

`object`	An object of class `LBBNN_Net`.
`...`	further arguments passed to or from other methods.

Details

The returned table combines two types of information:

- Number of times each input variable is included in the active paths from each layer (obtained from `get_input_inclusions()`).
- Average inclusion probabilities for each input variable from each layer, including a final column showing the average across all layers.

Value

A `data.frame` containing the above information. The function prints a formatted summary to the console. The returned `data.frame` is invisible.

Examples

```
x<-torch::torch_randn(3,2)
b <- torch::torch_rand(2)
y <- torch::torch_matmul(x,b)
train_data <- torch::tensor_dataset(x,y)
train_loader <- torch::dataloader(train_data,batch_size = 3,shuffle=FALSE)
problem<-'regression'
sizes <- c(2,1,1)
inclusion_priors <-c(0.9,0.2)
inclusion_inits <- matrix(rep(c(-10,10),2),nrow = 2,ncol = 2)
stds <- c(1.0,1.0)
model <- LBBNN_Net(problem,sizes,inclusion_priors,stds,inclusion_inits,flow = FALSE,
input_skip = TRUE)
train_LBBNN(epochs = 1,LBBNN = model, lr = 0.01,train_dl = train_loader)
summary(model)
```

---

train_LBBNN *Train an instance of* `LBBNN_Net`.

Description

Function that for each epoch iterates through each mini-batch, computing the loss and using back-propagation to update the network parameters.

Usage

```
train_LBBNN(
  epochs,
  LBBNN,
  lr,
  train_dl,
  device = "cpu",
  scheduler = NULL,
  sch_step_size = NULL
)
```

Arguments

`epochs`	integer, total number of epochs to train for, where one epoch is a pass through the entire training dataset (all mini batches).
`LBBNN`	An instance of `LBBNN_Net`, to be trained.
`lr`	numeric, the learning rate to be used in the Adam optimizer.
`train_dl`	An instance of `torch::dataloader` consisting of a tensor dataset with features and targets.
`device`	the device to be trained on. Default is 'cpu', also accepts 'gpu' or 'mps'.

scheduler	A torch learning rate scheduler object. Can be used to decay learning rate for better convergence, currently only supports 'step'.
sch_step_size	Where to decay if using torch::lr_step. E.g. 1000 means learning rate is decayed every 1000 epochs.

Value

a list containing the losses and accuracy (if classification) and density for each epoch during training. For comparisons sake we show the density with and without active paths.

A list with elements (returned invisibly):

accs Vector of accuracy per epoch (classification only).

loss Vector of average loss per epoch.

density Vector of network densities per epoch.

Examples

```
x<-torch::torch_randn(3,2)
b <- torch::torch_rand(2)
y <- torch::torch_matmul(x,b)
train_data <- torch::tensor_dataset(x,y)
train_loader <- torch::dataloader(train_data,batch_size = 3,shuffle=FALSE)
problem<-'regression'
sizes <- c(2,1,1)
inclusion_priors <-c(0.9,0.2)
inclusion_inits <- matrix(rep(c(-10,10),2),nrow = 2,ncol = 2)
stds <- c(1.0,1.0)
model <- LBBNN_Net(problem,sizes,inclusion_priors,stds,inclusion_inits,flow = FALSE)
output <- train_LBBNN(epochs = 1,LBBNN = model, lr = 0.01,train_dl = train_loader)
```

---

validate_LBBNN *Validate a trained LBBNN model.*

Description

Computes metrics on a validation dataset without computing gradients. Supports model averaging (recommended) by sampling from the variational posterior (num_samples > 1) to improve predictions. Returns metrics for both the full model and the sparse model.

Usage

```
validate_LBBNN(LBBNN, num_samples, test_dl, device = "cpu")
```

Arguments

`LBBNN`	An instance of a trained `LBBNN_Net` to be validated.
`num_samples`	integer, the number of samples from the variational posterior to be used for model averaging.
`test_dl`	An instance of `torch::dataloader`, containing the validation data.
`device`	The device to perform validation on. Default is 'cpu'; other options include 'gpu' and 'mps'.

Value

A list containing the following elements:

accuracy_full_model Classification accuracy of the full (dense) model (if classification).

accuracy_sparse Classification accuracy using only weights in active paths (if classification).

validation_error Root mean squared error for the full model (if regression).

validation_error_sparse Root mean squared error using only weights in active paths (if regression).

density Proportion of weights with posterior inclusion probability > 0.5 in the whole network.

density_active_path Proportion of weights with inclusion probability > 0.5 after removing weights not in active paths.

Index