Package: LBBNN (via r-universe)

October 29, 2026

Title Latent Binary Bayesian Neural Networks Using 'torch'

Version 0.1.6

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

Roxygen list(markdown = TRUE)

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

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev libfribidi-dev libglpk-dev libharfbuzz-dev libicu-dev libpng-dev libxml2-dev

Repository https://larselund.r-universe.dev

Date/Publication 2026-07-01 09:39:48 UTC

RemoteUrl https://github.com/larselund/lbbnn

RemoteRef HEAD

RemoteSha 1176f4995076e8aad7042b0d153e7c5ff318cc13

Contents

coef.lbbnn_net . . . 2
custom_activation . . . 4
gallstone_dataset . . . 4
get_dataloaders . . . 5
get_local_explanations_gradient . . . 6
lbbnn_conv2d . . . 7
lbbnn_linear . . . 8
lbbnn_net . . . 10
normalizing_flow . . . 12
plot.lbbnn_net . . . 13
plot_active_paths . . . 14
plot_local_explanations_gradient . . . 15
predict.lbbnn_net . . . 16
print.lbbnn_net . . . 17
quants . . . 18
raisin_dataset . . . 18
residuals.lbbnn_net . . . 19
resolve_device . . . 20
rnvp_layer . . . 20
summary.lbbnn_net . . . 21
torch_available . . . 22
train_lbbnn . . . 22
validate_lbbnn . . . 24

Index **26**

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

Description

Given an input sample x_1,... x_j (with j the number of inputs, 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

```
if (torch_available()) {
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

`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, samples per batch in the train dataloader.
`test_batch_size`	integer, samples per batch in the test dataloader.
`standardize`	logical, standardize input-features, default is TRUE.
`shuffle_train`	logical, shuffle training data each epoch. default TRUE
`shuffle_test`	logical, shuffle test data, default is FALSE.
`seed`	integer. Used for reproducibility 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` *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, 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 used. One of `'cpu'` (default), `'gpu'` or `'cuda'` (both select a CUDA GPU), or `'mps'`. Requesting an accelerator that is not available raises an error.

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 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",
  weight_init = "uniform"
)
```

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	Controls initialization of the inclusion parameters. A string keyword or a numeric vector c(lower, upper). String options: "polarized", "polarized_mild", "polarized_sparse", "polarized_dense", "dense", "sparse", "balanced".
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. One of 'cpu' (default), 'gpu' or 'cuda' (both select a CUDA GPU), or 'mps'. Requesting an accelerator that is not available raises an error.
weight_init	Controls how weight_mean is initialized. A string keyword or a numeric vector c(lower, upper) for custom uniform bounds. String options: "uniform" (default, small uniform), "he" / "he_relu" (Kaiming uniform for ReLU), "he_tanh" (Kaiming uniform for tanh), "glorot" / "xavier" (Xavier uniform).

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

```
if (torch_available()) {
layer <- lbbnn_conv2d(in_channels = 1,out_channels = 32,kernel_size = c(3,3),
prior_inclusion = 0.2,standard_prior = 1,density_init = c(0,1),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 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,
  weight_init = "uniform"
)
```

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	Controls initialization of the inclusion parameters. A string keyword or a numeric vector c(lower, upper). String options: "polarized", "polarized_mild", "polarized_sparse", "polarized_dense", "dense", "sparse", "balanced".
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. One of 'cpu' (default), 'gpu' or 'cuda' (both select a CUDA GPU), or 'mps'. Requesting an accelerator that is not available raises an error.
bias_inclusion_prob	logical, determines whether the bias should be as associated with inclusion probabilities.
conv_net	logical, whether the layer is used in a convolutional net.
weight_init	Controls how weight_mean is initialized. A string keyword or a numeric vector c(lower, upper) for custom uniform bounds. String options: "uniform" (default, small uniform), "he" / "he_relu" (Kaiming uniform for ReLU), "he_tanh" (Kaiming uniform for tanh), "glorot" / "xavier" (Xavier uniform).

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

```
if (torch_available()) {
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,
  weight_init = "uniform"
)
```

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. NOTE: Can also be given as a keyword with the following options: 'balanced', 'dense', 'polarized', 'polarized_dense', 'polarized_mild', 'polarized_sparse', 'sparse'.
input_skip	logical, whether to include input_skip.
flow	logical, 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 used. One of 'cpu' (default), 'gpu' or 'cuda' (both select a CUDA GPU), or 'mps'. Requesting an accelerator that is not available raises an error. 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).
nll	User can define their own likelihood function (not implemented).
bias_inclusion_prob	logical, determines whether the bias should be as associated with inclusion probabilities.
weight_init	Controls how weight_mean is initialized in each layer. A string keyword or a numeric vector c(lower, upper) for custom uniform bounds. String options: "uniform" (default), "he" / "he_relu" (Kaiming uniform for ReLU), "he_tanh" (Kaiming uniform for tanh), "glorot" / "xavier" (Xavier uniform).

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

```
if (torch_available()) {
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()}
```

---

normalizing_flow *Class to generate a normalizing flow*

Description

Used in`LBBNN_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, the sum of the log-determinants of the independent layers.

Usage

```
normalizing_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`	Transformation type. Currently RNVP is implemented.
`num_transforms`	integer, how many layers of transformations to include.

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

```
if (torch_available()) {
flow <- normalizing_flow(c(2,5,5), transform_type='RNVP', num_transforms = 3)
flow$to(device = 'cpu')
x <- torch::torch_rand(2, device = 'cpu')
output <- flow(x)
z_out <- output$z
print(dim(z_out))
log_det <- output$logdet
print(log_det)}
```

---

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

Description

Uses igraph to plot.

Usage

```
plot_active_paths(
  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.

Value

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

Examples

```
if (torch_available()) {
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_active_paths(model, 1, 1, 14, 1)}
```

`plot_local_explanations_gradient` *Plot the gradient based local explanations for one sample.*

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, samples to use to produce credible intervals.
`device`	character, the device to be used. One of `'cpu'` (default), `'gpu'` or `'cuda'` (both select a CUDA GPU), or `'mps'`. Requesting an accelerator that is not available raises an error.
`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 posterior of an* LBBNN model

Description

Draw from the posterior 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. One of 'cpu' (default), 'gpu' or 'cuda' (both select a CUDA GPU), or 'mps'. Requesting an accelerator that is not available raises an error.
link	Optional link function to apply to the network output.
...	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

```
if (torch_available()) {
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

```
if (torch_available()) {
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`	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

```
if (torch_available()) {
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)}
```

resolve_device *Resolve a device string to a valid torch device.*

Description

Maps the device aliases used throughout the package to the device names understood by `torch`. `'gpu'` is treated as an alias for `'cuda'`. If an accelerator is requested but unavailable, an error is raised.

Usage

```
resolve_device(device = "cpu")
```

Arguments

`device`	A device specification. Accepts `'cpu'`, `'gpu'`, `'cuda'` and `'mps'` (case-insensitive). A `torch_device` object is returned unchanged.

Value

A valid torch device string (`'cpu'`, `'cuda'` or `'mps'`).

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

```
if (torch_available()) {
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

```
if (torch_available()) {
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)}
```

---

torch_available *Check if torch/libtorch is available for examples*

Description

Check if torch/libtorch is available for examples

Usage

```
torch_available()
```

---

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 network parameters.

Usage

```
train_lbbnn(
  epochs,
  LBBNN,
  lr,
  train_dl,
  device = "cpu",
  scheduler = NULL,
  sch_step_size = NULL,
  min_density = NULL,
  verbose = TRUE
)
```

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 used. One of `'cpu'` (default), `'gpu'` or `'cuda'` (both select a CUDA GPU), or `'mps'`. Requesting an accelerator that is not available raises an error.
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.
min_density	Optional argument to stop training if the density reaches this value. Currently, this is the overall density, not density in active paths, as the latter is expensive to compute for each epoch.
verbose	Control whether metrics are printed to console.

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

```
if (torch_available()) {
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. One of `'cpu'` (default), `'gpu'` or `'cuda'` (both select a CUDA GPU), or `'mps'`. Requesting an accelerator that is not available raises an error.

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

∗ **datasets**
gallstone_dataset, 4
raisin_dataset, 18

coef.lbbnn_net, 2
custom_activation, 4

gallstone_dataset, 4
get_dataloaders, 5
get_local_explanations_gradient, 6

lbbnn_conv2d, 7
lbbnn_linear, 8
lbbnn_net, 10

normalizing_flow, 12

plot.lbbnn_net, 13
plot_active_paths, 14
plot_local_explanations_gradient, 15
predict.lbbnn_net, 16
print.lbbnn_net, 17

quants, 18

raisin_dataset, 18
residuals.lbbnn_net, 19
resolve_device, 20
rnvp_layer, 20

summary.lbbnn_net, 21

torch_available, 22
train_lbbnn, 22

validate_lbbnn, 24