Package 'splineKANr'

November 17, 2026

Type Package

Title B-Spline Kolmogorov-Arnold Networks in Native R 'torch'

Version 0.5.1

Description Implements Kolmogorov-Arnold Networks (KAN) whose edge functions are B-splines plus a residual base activation, written in native R on top of the 'torch' package. Provides Cox-de Boor basis evaluation with explicit extrapolation policies, multi-layer networks, Adam and L-BFGS training with validation, early stopping, input and output normalization, gradient clipping and reproducible seeds, adaptive (sample-based) grid updates, grid refinement, KAN-specific sparsity regularization, pruning, edge-function visualization and model persistence. A dependency-free base R reference implementation with analytic gradients is included as a numerical oracle and as a training backend that does not need 'libtorch'. The design follows Liu et al. (2024) <doi:10.48550/arXiv.2404.19756>.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports graphics, grDevices, splines, stats, torch (>= 0.13.0), utils

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

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Thang Le-Dinh [aut, cre, cph]

Maintainer Thang Le-Dinh <thang87sg@gmail.com>

Repository CRAN

Date/Publication 2026-11-17 18:30:02 UTC

Contents

kan_activation_check . . . 3
kan_benchmark . . . 3
kan_bspline_basis . . . 4
kan_bspline_basis_r . . . 5
kan_bspline_deriv_r . . . 6
kan_edge_function . . . 6
kan_edge_importance . . . 7
kan_extend_grid . . . 8
kan_fd_gradient_r . . . 8
kan_fit . . . 9
kan_forward_r . . . 11
kan_gradient_r . . . 12
kan_init_state . . . 13
kan_layer . . . 14
kan_make_grid . . . 15
kan_metrics . . . 15
kan_module_from_state . . . 16
kan_net . . . 17
kan_partial_dependence . . . 17
kan_plot_history . . . 18
kan_plot_network . . . 19
kan_prune . . . 19
kan_refine . . . 20
kan_regularization_r . . . 21
kan_reg_control . . . 22
kan_save . . . 22
kan_scale . . . 23
kan_scaler . . . 24
kan_spec . . . 24
kan_synthetic . . . 26
kan_update_grid . . . 26
kan_validate_all . . . 27
kan_validate_basis . . . 28
kan_validate_derivative . . . 28
kan_validate_gradient . . . 29
plot.kan_model . . . 30
predict.kan_model . . . 31

Index **32**

kan_activation_check *Fraction of layer inputs outside the spline domain*

Description

Fraction of layer inputs outside the spline domain

Usage

```
kan_activation_check(state, x)
```

Arguments

state	A kan_state or kan_model (for a model, x is on the original scale).
x	Input matrix.

Value

A data frame with one row per layer: observed input range, spline domain and the fraction of input values outside the domain.

Examples

```
st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
kan_activation_check(st, matrix(runif(40, -1.5, 1.5), ncol = 2))
```

kan_benchmark *Multi-seed convergence and accuracy benchmark*

Description

For each problem and seed: simulates training and an independent test set, fits a KAN and reports out-of-sample RMSE, MAE and R^2 (against the noisy test response) plus the RMSE against the noise-free function.

Usage

```
kan_benchmark(
  problems = c("linear", "sin", "poly", "interaction", "noisy_sin"),
  seeds = 1:5,
  n_train = 500L,
  n_test = 1000L,
  hidden = NULL,
  ...
)
```

Arguments

problems	Problem names, see kan_synthetic().
seeds	Integer seeds (one replicate each).
n_train, n_test	Sample sizes.
hidden	Hidden widths; NULL chooses them per problem, see Details.
...	Further arguments for kan_fit().

Details

With hidden = NULL, problems with one input use an additive single-layer KAN and two-input problems use one hidden layer of width 4, which can approximate x1 * x2 through (((x1 + x2)^2 - (x1 - x2)^2) / 4).

Value

A data frame with one row per problem and seed.

Examples

```
kan_benchmark("linear", seeds = 1, epochs = 50, backend = "r")
```

---

kan_bspline_basis *B-spline basis in torch (Cox-de Boor, batched)*

Description

Differentiable with respect to x; used inside kan_layer().

Usage

```
kan_bspline_basis(x, grid, spline_order = 3L, right_closed = FALSE)
```

Arguments

x	Tensor of shape (batch, n_in).
grid	Tensor of shape (n_in, n_knots) (extended knots).
spline_order	Spline degree k.
right_closed	Logical, see kan_bspline_basis_r().

Value

Tensor of shape (batch, n_in, n_knots - k - 1).

Examples

```
if (torch::torch_is_installed()) {
  g <- torch::torch_tensor(kan_make_grid(2, 5, 3))
  x <- torch::torch_rand(4, 2) * 2 - 1
  kan_bspline_basis(x, g, 3)$sum(dim = 3) # ~1 on the domain
}
```

`kan_bspline_basis_r` *B-spline basis by the Cox-de Boor recursion (base R reference)*

Description

Evaluates all B-spline basis functions of degree `spline_order` defined by the knot vector `knots` at the points `x`. This is the dependency-free numerical oracle against which the torch backend is validated.

Usage

```
kan_bspline_basis_r(x, knots, spline_order = 3L, right_closed = FALSE)
```

Arguments

`x`	Numeric vector of evaluation points.
`knots`	Strictly increasing numeric knot vector.
`spline_order`	Spline degree k (>= 0).
`right_closed`	Logical. If `TRUE`, points equal to the right end of the domain `knots[length(knots) - k]` are evaluated as left limits (see Details). Use `TRUE` to match `splines::splineDesign()` at the boundary.

Details

Intervals are half-open `[t_j, t_{j+1})`. Points outside `range(knots)` get all-zero rows. Basis functions with zero-length support denominators are treated as zero (`0/0 := 0`).

Value

A matrix with `length(x)` rows and `length(knots) - k - 1` columns.

Examples

```
g <- kan_make_grid(1, grid_size = 4, spline_order = 3)[1, ]
B <- kan_bspline_basis_r(seq(-1, 1, by = 0.25), g, 3)
rowSums(B) # partition of unity on [-1, 1]
```

kan_bspline_deriv_r *First derivative of the B-spline basis (base R reference)*

Description

Uses the analytic identity

$$\frac{d}{dx}N_{i,k}(x) = \frac{k}{t_{i+k} - t_i}N_{i,k-1}(x) - \frac{k}{t_{i+k+1} - t_{i+1}}N_{i+1,k-1}(x).$$

Usage

```
kan_bspline_deriv_r(x, knots, spline_order = 3L, right_closed = FALSE)
```

Arguments

x	Numeric vector of evaluation points.
knots	Strictly increasing numeric knot vector.
spline_order	Spline degree k (>= 0).
right_closed	Logical. If TRUE, points equal to the right end of the domain knots[length(knots) - k] are evaluated as left limits (see Details). Use TRUE to match splines::splineDesign() at the boundary.

Value

A matrix of the same shape as kan_bspline_basis_r().

Examples

```
g <- kan_make_grid(1, grid_size = 4, spline_order = 3)[1, ]
kan_bspline_deriv_r(c(-0.3, 0.2), g, 3)
```

kan_edge_function *Evaluate one edge function on a grid of inputs*

Description

Evaluate one edge function on a grid of inputs

Usage

```
kan_edge_function(model, layer, i, j, from = NULL, to = NULL, n = 200L)
```

Arguments

model	A kan_model or kan_state.
layer, i, j	Layer index and edge i -> j.
from, to	Range of the (normalized) layer input. Defaults to the spline domain of the edge.
n	Number of evaluation points.

Value

A data frame with columns x, phi (full edge function), spline and base (the weighted components).

Examples

```
st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
head(kan_edge_function(st, layer = 1, i = 1, j = 2))
```

---

kan_edge_importance *Edge importance (mean absolute edge activation)*

Description

Edge importance (mean absolute edge activation)

Usage

```
kan_edge_importance(model, x)
```

Arguments

model	A kan_model (then x is on the original scale) or a kan_state (then x is already normalized).
x	Input sample.

Value

A list with one n_in x n_out matrix per layer holding mean_n |phi_ij(x_n)|.

Examples

```
st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
kan_edge_importance(st, matrix(runif(40, -1, 1), ncol = 2))
```

kan_extend_grid *Extend a domain grid by uniform knots on both sides*

Description

Extend a domain grid by uniform knots on both sides

Usage

```
kan_extend_grid(grid, spline_order)
```

Arguments

grid	Numeric matrix of domain knots (one row per input), with at least two strictly increasing columns.
spline_order	Number of knots k to add on each side.

Value

A matrix with `2 * spline_order` more columns than `grid`. The spacing of the added knots is `(max - min) / (ncol(grid) - 1)`, as in pykan's `extend_grid()`.

Examples

```
kan_extend_grid(matrix(c(-1, 0, 1), nrow = 1), spline_order = 2)
```

kan_fd_gradient_r *Central finite-difference gradient of the MSE loss (base R)*

Description

Central finite-difference gradient of the MSE loss (base R)

Usage

```
kan_fd_gradient_r(state, x, y, layer, param, index, h = 1e-06)
```

Arguments

state	A kan_state.
x	Numeric input matrix.
y	Numeric response (vector or matrix with `width[length(width)]` columns).
layer	Layer index.
param	One of `"coef"`, `"scale_base"`, `"scale_sp"`, `"bias"`.
index	Integer vector of linear indices into the parameter array.
h	Step size.

Value

Numeric vector of finite-difference derivatives (`L(p + h) - L(p - h)) / (2h)`.

Examples

```
st <- kan_init_state(kan_spec(c(1, 2, 1)), seed = 2)
x <- matrix(seq(-0.9, 0.9, length.out = 7))
kan_fd_gradient_r(st, x, sin(3 * x), layer = 1, param = "coef", index = 1:3)
```

---

kan_fit *Fit a B-spline Kolmogorov-Arnold Network*

Description

Trains a KAN of architecture `c(ncol(x), hidden, ncol(y))` by minimizing the mean squared error (plus optional KAN regularization) on normalized data. Normalization constants are estimated on the training part only.

Usage

```
kan_fit(
  x,
  y,
  hidden = 5L,
  grid_size = 5L,
  spline_order = 3L,
  grid_range = c(-1, 1),
  extrapolate = c("linear", "none", "clamp"),
  base_fun = c("silu", "identity", "none"),
  bias = TRUE,
  epochs = 200L,
  batch_size = NULL,
  optimizer = c("adam", "lbfgs"),
  lr = NULL,
  weight_decay = 0,
  validation_split = 0.2,
  validation_data = NULL,
  patience = 30L,
  min_delta = 0,
  restore_best = TRUE,
  normalize_x = c("minmax", "standardize", "none"),
  normalize_y = TRUE,
  reg = kan_reg_control(),
  grid_update_every = 0L,
  grid_update_until = 50L,
  grid_eps = 0.02,
  grid_ridge = 1e-08,
```

```
  grad_clip = NULL,
  seed = NULL,
  backend = c("torch", "r"),
  dtype = "float32",
  init = NULL,
  init_control = list(),
  checkpoint_path = NULL,
  verbose = FALSE
)
```

Arguments

x	Numeric matrix or data frame of predictors.
y	Numeric response vector, or matrix for several outputs.
hidden	Integer vector of hidden layer widths; `integer()` gives a single-layer (additive) KAN.
grid_size, spline_order, grid_range, extrapolate, base_fun, bias	Architecture, see `kan_spec()`.
epochs	Maximum number of passes over the training data.
batch_size	Mini-batch size; `NULL` uses full batches.
optimizer	`"adam"` or `"lbfgs"` (full batch, torch backend only).
lr	Learning rate; defaults to `0.01` for Adam and `1` for L-BFGS.
weight_decay	L2 weight decay passed to Adam.
validation_split	Fraction of rows held out for validation (and early stopping) when `validation_data` is `NULL`.
validation_data	Optional `list(x = , y = )` used for validation.
patience	Stop after this many epochs without an improvement of at least `min_delta` in the monitored loss (validation loss if available, otherwise training loss). `NULL` disables early stopping.
min_delta	Minimum improvement counted by early stopping.
restore_best	Return the parameters of the best epoch.
normalize_x	`"minmax"` (map training range onto `grid_range`), `"standardize"` or `"none"`.
normalize_y	Standardize the response(s) during training.
reg	Regularization settings from `kan_reg_control()` (torch backend only when `reg$lambda > 0`).
grid_update_every	Update the grids from the training inputs every this many epochs (`0` = never), see `kan_update_grid()`.
grid_update_until	Last epoch at which grids may be updated.
grid_eps	Uniform/quantile blending for grid updates.

grid_ridge	Relative ridge penalty of the coefficient refit after a grid update, see kan_update_grid(). 0 gives plain least squares, as in pykan.
grad_clip	Maximum gradient norm (NULL = no clipping).
seed	Integer seed for splitting, initialization and batching.
backend	"torch" (autograd, the default) or "r" (the base R reference with analytic gradients; Adam only, no regularization, slower, but needs no libtorch).
dtype	"float32" or "float64" for the torch backend.
init	Optional kan_model to warm start from (its architecture and scalers are reused, e.g. after kan_refine()).
init_control	List of initialization options passed to kan_init_state().
checkpoint_path	Optional file; the best model so far is written there with kan_save() whenever the monitored loss improves.
verbose	Print progress every max(1, epochs %/% 10) epochs.

Value

An object of class kan_model with elements state, spec, scalers, history (data frame of losses per epoch), best_epoch, stopped_early, activation_check, settings and meta.

Examples

```
set.seed(1)
x <- matrix(runif(300, -1, 1))
y <- sin(pi * x[, 1]) + rnorm(300, sd = 0.05)
fit <- kan_fit(x, y, hidden = integer(), epochs = 30, backend = "r",
               seed = 1)
kan_metrics(y, predict(fit, x, backend = "r"))
```

---

kan_forward_r *Forward pass of a KAN in base R*

Description

Dependency-free reference implementation of the network defined by a kan_state (see kan_init_state()). It is used as a numerical oracle for the torch backend and for analysis (edge plots, attribution, grid updates).

Usage

```
kan_forward_r(state, x, detail = FALSE)
```

Arguments

state	A kan_state (see kan_init_state()) or a fitted kan_model.
x	Numeric matrix with width[1] columns (already normalized).
detail	Logical; if TRUE, return per-layer intermediate values.

Value

If `detail = FALSE`, the output matrix. Otherwise a list with `output` and `layers`; each layer entry has `input`, `phi` (array `n x n_in x n_out`), `spline`, `spline_deriv`, `base`, `base_deriv` and `features` (list of feature matrices per input).

Examples

```
st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
kan_forward_r(st, matrix(c(0.1, -0.4, 0.5, 0.9), ncol = 2))
```

---

kan_gradient_r *Mean squared error loss and analytic gradients in base R*

Description

Computes `L = mean((f(x) - y)^2)` and its exact gradient with respect to every trainable parameter by manual backpropagation. Used to validate torch autograd and finite differences (Phase 2 of the validation plan).

Usage

```
kan_gradient_r(state, x, y)
```

Arguments

`state`	A `kan_state`.
`x`	Numeric input matrix.
`y`	Numeric response (vector or matrix with `width[length(width)]` columns).

Value

A list with `loss`, `grads` (per layer: `coef`, `scale_base`, `scale_sp`, `bias`) and `input_grad` (gradient with respect to `x`).

Examples

```
st <- kan_init_state(kan_spec(c(1, 2, 1)), seed = 2)
x <- matrix(seq(-0.9, 0.9, length.out = 7))
g <- kan_gradient_r(st, x, sin(3 * x))
g$loss
```

kan_init_state *Initialize KAN parameters*

Description

Draws initial parameters with base R's random number generator, so the same `seed` gives the same network for every backend. The scheme follows pykan's `KANLayer`: small random spline coefficients of size `noise_scale / G`, base weights `scale_base_mu / sqrt(n_in) + scale_base_sigma * U(-1, 1) / sqrt(n_in)` and spline weights `scale_sp / sqrt(n_in)`.

Usage

```
kan_init_state(
  spec,
  seed = NULL,
  noise_scale = 0.3,
  scale_base_mu = 0,
  scale_base_sigma = 1,
  scale_sp = 1
)
```

Arguments

`spec`	A `kan_spec()`.
`seed`	Optional integer seed passed to `set.seed()`.
`noise_scale`	Size of the initial coefficient noise.
`scale_base_mu, scale_base_sigma`	Mean and spread of the base weights.
`scale_sp`	Initial spline weight.

Value

An object of class `kan_state`: a list with `spec` and `layers`. Each layer holds `coef` (array `n_in x n_out x (G + k)`), `scale_base`, `scale_sp`, `mask` (matrices `n_in x n_out`), `bias` (length `n_out`) and `grid` (matrix `n_in x (G + 2k + 1)`).

Examples

```
st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
str(st$layers[[1]], max.level = 1)
```

kan_layer *A single B-spline KAN layer (torch module)*

Description

Maps `(batch, n_in)` to `(batch, n_out)`. Every edge `i -> j` carries `phi_ij(x) = mask_ij * (scale_base_ij * b(x) + scale_sp_ij * s_ij(x))` with `s_ij` a B-spline with coefficients `coef[i, j, ]`.

Usage

```
kan_layer(
  n_in,
  n_out,
  grid_size = 5L,
  spline_order = 3L,
  extrapolate = "linear",
  base_fun = "silu",
  bias = TRUE,
  dtype = "float32"
)
```

Arguments

`n_in`, `n_out`	Input and output widths.
`grid_size`	Number of grid intervals `G`.
`spline_order`	Spline degree `k`.
`extrapolate`, `base_fun`, `bias`	See `kan_spec()`.
`dtype`	`"float32"` or `"float64"`.

Value

An `nn_module`. Parameters: `coef`, `scale_base`, `scale_sp`, `bias`; buffers: `grid`, `mask`. Its `forward(x, detail = FALSE)` returns the output, or with `detail = TRUE` a list with `output`, `phi` and `spline`.

Examples

```
if (torch::torch_is_installed()) {
  layer <- kan_layer(2, 3)
  layer(torch::torch_randn(4, 2))
}
```

kan_make_grid *Build an extended uniform knot grid*

Description

Creates the knot matrix used by every KAN layer: `grid_size` uniform intervals on `grid_range`, extended by `spline_order` knots on each side.

Usage

```
kan_make_grid(n_in, grid_size = 5L, spline_order = 3L, grid_range = c(-1, 1))
```

Arguments

`n_in`	Number of input dimensions (rows of the result).
`grid_size`	Number of intervals `G` on the domain (integer >= 1).
`spline_order`	Spline degree `k` (integer >= 1; 3 = cubic).
`grid_range`	Numeric vector `c(a, b)` with `a < b`, or a matrix with `n_in` rows and two columns giving one range per input.

Value

A numeric matrix with `n_in` rows and ‘grid_size + 2 * spline_order

- `1columns. Rowiholds the sorted knots for inputi`‘.

Examples

```
kan_make_grid(n_in = 2, grid_size = 5, spline_order = 3)
```

kan_metrics *Regression accuracy metrics*

Description

Regression accuracy metrics

Usage

```
kan_metrics(y, yhat)
```

Arguments

`y`	Observed values (vector or matrix).
`yhat`	Predicted values, same shape as `y`.

Value

A named numeric vector with `rmse`, `mae`, `r2` (computed over all elements) and `n`. `r2` is `NA` (with a warning) when `y` is constant.

Examples

```
kan_metrics(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

---

kan_module_from_state *Build a torch module from a state*

Description

Build a torch module from a state

Usage

```
kan_module_from_state(state, dtype = "float32")
```

Arguments

`state`	A `kan_state`.
`dtype`	`"float32"` or `"float64"`.

Value

A `kan_net()` holding the state's numbers.

Examples

```
if (torch::torch_is_installed()) {
  st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
  net <- kan_module_from_state(st)
}
```

kan_net *Multi-layer B-spline KAN (torch module)*

Description

Multi-layer B-spline KAN (torch module)

Usage

```
kan_net(spec, dtype = "float32")
```

Arguments

spec	A kan_spec().
dtype	"float32" or "float64".

Value

An nn_module whose forward(x, detail = FALSE) returns the output tensor, or with detail = TRUE a list with output and the per-layer inputs, phi.

Examples

```
if (torch::torch_is_installed()) {
  net <- kan_net(kan_spec(c(2, 4, 1)))
  net(torch::torch_randn(3, 2))
}
```

kan_partial_dependence *Partial dependence of the prediction on one predictor*

Description

Averages predictions over data while var is set to each value of a grid: the "effect plot" used to interpret a fitted model.

Usage

```
kan_partial_dependence(model, data, var, n = 50L, output = 1L)
```

Arguments

model	A kan_model.
data	Reference data (original scale).
var	Column name or index of the predictor.
n	Number of grid values.
output	Output column for multi-output models.

Value

A data frame of class kan_pd with columns value and yhat; it has a plot() method.

Examples

```
set.seed(1)
x <- matrix(runif(400, -1, 1), ncol = 2, dimnames = list(NULL, c("a", "b")))
fit <- kan_fit(x, x[, "a"]^2 + x[, "b"], hidden = integer(), epochs = 20,
               backend = "r", seed = 1)
plot(kan_partial_dependence(fit, x, "a"))
```

---

kan_plot_history *Plot the training history*

Description

Plot the training history

Usage

```
kan_plot_history(model, log = TRUE)
```

Arguments

model	A kan_model.
log	Logical; log-scale the loss axis.

Value

model, invisibly.

Examples

```
set.seed(1)
x <- matrix(runif(100, -1, 1))
fit <- kan_fit(x, x[, 1]^2, hidden = integer(), epochs = 10,
               backend = "r", seed = 1)
kan_plot_history(fit)
```

kan_plot_network *Plot the network structure*

Description

Nodes are drawn layer by layer; edge width and opacity are proportional to edge importance on `data` (or 1 for active edges if `data` is `NULL`).

Usage

```
kan_plot_network(model, data = NULL)
```

Arguments

`model`	A `kan_model`.
`data`	Optional input sample on the original scale.

Value

The list of importance matrices, invisibly.

Examples

```
st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
m <- structure(list(state = st, spec = st$spec,
  scalers = list(x = kan_scaler(matrix(c(-1, 1, -1, 1), 2), "none"),
                 y = kan_scaler(matrix(c(0, 1)), "none")),
  feature_names = c("x1", "x2"), response_names = "y"),
  class = "kan_model")
kan_plot_network(m)
```

kan_prune *Prune edges and hidden nodes with small activations*

Description

Edges with importance below `edge_threshold` are masked. A hidden node is removed when both its largest incoming and its largest outgoing edge importance fall below `node_threshold`, mirroring pykan's `prune_node()` / `prune_edge()` criteria. Retrain briefly afterwards (`kan_fit(..., init = pruned)`) to recover accuracy.

Usage

```
kan_prune(model, x, node_threshold = 0.01, edge_threshold = 0.03)
```

Arguments

model	A kan_model.
x	Input sample on the original scale.
node_threshold, edge_threshold	Importance thresholds.

Value

The pruned kan_model with element pruning describing the removed nodes and masked edges.

Examples

```
set.seed(1)
x <- matrix(runif(400, -1, 1), ncol = 2)
fit <- kan_fit(x, sin(pi * x[, 1]), hidden = 3, epochs = 20,
               backend = "r", seed = 1)
pruned <- kan_prune(fit, x, edge_threshold = 0.05)
pruned$pruning
```

---

kan_refine *Refine the grid of a model (grid extension)*

Description

Convenience wrapper around kan_update_grid() that increases the number of intervals while preserving the learned functions on x. Continue training with kan_fit(..., init = refined_model).

Usage

```
kan_refine(model, x, grid_size, grid_eps = 0.02)
```

Arguments

model	A fitted kan_model.
x	Input sample on the original scale.
grid_size	New (usually larger) grid size.
grid_eps	See kan_update_grid().

Value

The refined kan_model.

Examples

```
set.seed(1)
x <- matrix(runif(200, -1, 1))
fit <- kan_fit(x, sin(3 * x), hidden = integer(), epochs = 20,
               backend = "r", verbose = FALSE)
fit10 <- kan_refine(fit, x, grid_size = 10)
```

---

kan_regularization_r *Regularization penalty in base R*

Description

Regularization penalty in base R

Usage

```
kan_regularization_r(state, x, reg = kan_reg_control())
```

Arguments

state	A kan_state or kan_model.
x	Normalized input matrix.
reg	A kan_reg_control() object.

Value

The penalty *before* multiplication by reg$lambda.

Examples

```
st <- kan_init_state(kan_spec(c(2, 3, 1)), seed = 1)
kan_regularization_r(st, matrix(runif(20, -1, 1), ncol = 2),
                     kan_reg_control(lambda = 1))
```

kan_reg_control *Regularization settings*

Description

Regularization settings

Usage

```
kan_reg_control(
  lambda = 0,
  lambda_l1 = 1,
  lambda_entropy = 2,
  lambda_coef = 0,
  lambda_coefdiff = 0
)
```

Arguments

lambda	Overall strength; 0 disables regularization.
lambda_l1	Weight of the L1 norm of edge scales.
lambda_entropy	Weight of the edge-scale entropy.
lambda_coef	Weight of the L1 norm of spline coefficients.
lambda_coefdiff	Weight of the L1 norm of coefficient differences (a smoothness penalty).

Value

A list of class kan_reg.

Examples

```
kan_reg_control(lambda = 1e-3)
```

kan_save *Save and load a fitted KAN*

Description

kan_save() writes the architecture, all parameters, knots, normalization constants, training history and metadata to an .rds file. kan_load() restores a kan_model that predicts identically.

Usage

```
kan_save(model, file)

kan_load(file)
```

Arguments

model	A kan_model.
file	Path of the .rds file.

Value

kan_save() returns file invisibly; kan_load() returns the kan_model.

Examples

```
set.seed(1)
x <- matrix(runif(60, -1, 1))
fit <- kan_fit(x, x[, 1]^2, hidden = integer(), epochs = 5,
               backend = "r", seed = 1)
f <- tempfile(fileext = ".rds")
kan_save(fit, f)
fit2 <- kan_load(f)
all.equal(predict(fit, x, backend = "r"), predict(fit2, x, backend = "r"))
```

---

kan_scale *Apply or invert a scaler*

Description

Apply or invert a scaler

Usage

```
kan_scale(x, scaler)

kan_unscale(x, scaler)
```

Arguments

x	Numeric matrix.
scaler	A kan_scaler.

Value

The transformed matrix.

Examples

```
s <- kan_scaler(matrix(1:10, ncol = 2), "standardize")
z <- kan_scale(matrix(1:10, ncol = 2), s)
kan_unscale(z, s)
```

---

kan_scaler *Fit a column scaler*

Description

Fit a column scaler

Usage

```
kan_scaler(x, method = c("minmax", "standardize", "none"), target = c(-1, 1))
```

Arguments

x	Numeric matrix.
method	"minmax" maps each column of the training data onto target (the spline domain), "standardize" centers and scales to unit variance, "none" leaves data unchanged.
target	Target interval for "minmax".

Value

A list of class kan_scaler with method, center, scale and offset, so that z = (x - center) / scale + offset.

Examples

```
s <- kan_scaler(matrix(1:10, ncol = 2), "minmax")
kan_scale(matrix(1:10, ncol = 2), s)
```

---

kan_spec *Specify a B-spline KAN architecture*

Description

Specify a B-spline KAN architecture

Usage

```
kan_spec(
  width,
  grid_size = 5L,
  spline_order = 3L,
  grid_range = c(-1, 1),
  extrapolate = c("linear", "none", "clamp"),
  base_fun = c("silu", "identity", "none"),
  bias = TRUE
)
```

Arguments

`width`	Integer vector of layer widths, e.g. `c(2, 5, 1)` for two inputs, one hidden layer of five nodes and one output.
`grid_size`	Number of grid intervals `G` per edge.
`spline_order`	Spline degree `k` (3 = cubic).
`grid_range`	Initial domain `c(a, b)` of every edge spline.
`extrapolate`	Behavior of an edge spline outside its domain: • `"linear"` (default): first-order Taylor extension from the nearest bound. The edge stays continuous and differentiable and its gradient with respect to the input is not zero outside the domain. • `"none"`: raw B-spline on the extended knots, which decays to zero outside the domain. This matches pykan. • `"clamp"`: evaluate at the nearest bound. The spline part is then flat outside the domain (zero input gradient).
`base_fun`	Residual base activation `b(x)` in `phi(x) = w_b * b(x) + w_s * spline(x)`: `"silu"` (pykan default), `"identity"` or `"none"`.
`bias`	Logical; add a trainable bias to every node of each layer.

Value

An object of class `kan_spec`.

Examples

```
kan_spec(c(2, 5, 1))
```

kan_synthetic *Synthetic regression problems*

Description

Synthetic regression problems

Usage

```
kan_synthetic(
  name = c("linear", "sin", "poly", "interaction", "noisy_sin", "additive"),
  n = 500L,
  seed = NULL
)
```

Arguments

name	One of "linear", "sin", "poly", "interaction", "noisy_sin", "additive".
n	Number of observations.
seed	Optional seed.

Value

A list with x (matrix), y (vector), f (noise-free target function) and noise_sd.

Examples

```
d <- kan_synthetic("sin", n = 50, seed = 1)
str(d)
```

kan_update_grid *Update spline grids from data (adaptive grid)*

Description

Re-places the knots of every edge spline at the empirical quantiles of the current layer inputs, blended with a uniform grid: grid = grid_eps * uniform + (1 - grid_eps) * quantile, as in pykan's update_grid_from_samples(). The spline coefficients are then refitted by (ridge-stabilized) least squares so that every edge keeps the same spline values on the sample. Layers are processed in order, so each layer sees the inputs produced by the already-updated layers before it.

Usage

```
kan_update_grid(state, x, grid_size = NULL, grid_eps = 0.02, ridge = 1e-08)
```

Arguments

state	A kan_state or kan_model. For a model, x is on the original scale and is normalized internally.
x	Sample of network inputs (matrix with width[1] columns).
grid_size	New number of intervals. NULL keeps the current size; a larger value refines the grid (pykan's grid extension).
grid_eps	Weight of the uniform grid in [0, 1]; 1 gives a uniform grid over the data range, 0 a pure quantile grid.
ridge	Relative ridge penalty for the coefficient refit (0 for plain least squares, as pykan's lstsq).

Value

An object of the same class as state with updated grid and coef arrays.

Examples

```
st <- kan_init_state(kan_spec(c(1, 2, 1)), seed = 1)
x <- matrix(rnorm(200, sd = 0.5))
st2 <- kan_update_grid(st, x)
range(st2$layers[[1]]$grid)
```

---

kan_validate_all *Run all numerical validations*

Description

Run all numerical validations

Usage

```
kan_validate_all(backend = c("both", "r", "torch"))
```

Arguments

backend	"both", "r" or "torch".

Value

A list of data frames (basis, derivative, gradient) with an attribute all_pass.

Examples

```
res <- kan_validate_all(backend = "r")
attr(res, "all_pass")
```

kan_validate_basis *Validate the B-spline basis*

Description

Compares the basis with `splines::splineDesign()` on interior points and checks non-negativity and the partition of unity on the domain.

Usage

```
kan_validate_basis(
  grid_sizes = c(3L, 5L, 10L),
  spline_orders = 1:3,
  n = 1000L,
  backend = c("both", "r", "torch"),
  tol = 1e-06
)
```

Arguments

`grid_sizes, spline_orders`	Settings to test.
`n`	Number of evaluation points per setting.
`backend`	`"r"`, `"torch"` or `"both"` (torch is skipped with a message when not installed).
`tol`	Acceptance threshold for maximum absolute errors.

Value

A data frame with one row per backend, grid size and order.

Examples

```
kan_validate_basis(backend = "r")
```

kan_validate_derivative *Validate the first derivative of the B-spline basis*

Description

Compares the analytic derivative (and, for torch, autograd) with `splines::splineDesign(derivs = 1)` and with central finite differences at points away from the knots. The relative error uses `|a - b| / max(|b|, atol)` so values near zero are judged absolutely.

Usage

```
kan_validate_derivative(
  grid_sizes = c(3L, 5L, 10L),
  spline_orders = 1:3,
  n = 500L,
  backend = c("both", "r", "torch"),
  h = 1e-06,
  atol = 1e-06,
  tol = 1e-04
)
```

Arguments

grid_sizes, spline_orders	Settings to test.
n	Number of evaluation points per setting.
backend	"r", "torch" or "both" (torch is skipped with a message when not installed).
h	Finite-difference step.
atol	Absolute floor in the relative error.
tol	Acceptance threshold for the relative error.

Value

A data frame with one row per backend and setting.

Examples

```
kan_validate_derivative(backend = "r")
```

---

kan_validate_gradient *Validate parameter gradients of the network*

Description

For random networks, compares the analytic base R gradient of the MSE loss with central finite differences, and (if available) with torch autograd in double precision. Inputs avoid the knots so the loss is smooth.

Usage

```
kan_validate_gradient(
  widths = list(c(1, 1), c(2, 3, 1), c(3, 4, 2, 1)),
  extrapolate = c("linear", "none", "clamp"),
  n = 40L,
  seed = 1L,
  backend = c("both", "r", "torch"),
```

```
  h = 1e-06,
  atol = 1e-06,
  tol = 0.001
)
```

Arguments

widths	List of architectures to test.
extrapolate	Extrapolation policies to test.
n	Number of samples.
seed	Seed.
backend	"r", "torch" or "both".
h	Finite-difference step.
atol	Absolute floor in the relative error.
tol	Acceptance threshold for the relative error.

Value

A data frame with one row per architecture, policy and comparison.

Examples

```
kan_validate_gradient(widths = list(c(2, 3, 1)), backend = "r")
```

---

plot.kan_model *Plot the edge functions of a KAN layer*

Description

Draws a panel for every edge `i -> j` of `layer`, showing `phi_ij` (solid) with its spline (dashed) and base (dotted) components, on the normalized input scale of that layer. Masked edges are gray.

Usage

```
## S3 method for class 'kan_model'
plot(x, layer = 1L, data = NULL, n = 200L, ...)
```

Arguments

x	A kan_model.
layer	Layer to draw.
data	Optional input sample (original scale) used to set the plotted range to the observed layer inputs and to add a rug.
n	Points per curve.
...	Further arguments passed to graphics::plot().

Value

x, invisibly.

Examples

```
st <- kan_init_state(kan_spec(c(2, 2, 1)), seed = 1)
m <- structure(list(state = st, spec = st$spec,
  scalers = list(x = kan_scaler(matrix(c(-1, 1, -1, 1), 2), "none"),
                 y = kan_scaler(matrix(c(0, 1)), "none")),
  history = data.frame(), feature_names = c("x1", "x2"),
  response_names = "y"), class = "kan_model")
plot(m, layer = 1)
```

---

predict.kan_model *Predict from a fitted KAN*

Description

Predict from a fitted KAN

Usage

```
## S3 method for class 'kan_model'
predict(object, newdata, backend = c("auto", "torch", "r"), ...)
```

Arguments

object	A kan_model.
newdata	Numeric matrix or data frame with the training predictors. If it has column names, they are matched to the training names.
backend	"auto" (torch if installed, otherwise base R), "torch" or "r". Both backends evaluate the same model.
...	Unused.

Value

A numeric vector (single output) or matrix of predictions on the original response scale.

Examples

```
set.seed(1)
x <- matrix(runif(100, -1, 1))
fit <- kan_fit(x, x[, 1]^2, hidden = integer(), epochs = 10,
               backend = "r", seed = 1)
head(predict(fit, x, backend = "r"))
```

Index

graphics::plot(), *30*

kan_activation_check, 3
kan_benchmark, 3
kan_bspline_basis, 4
kan_bspline_basis_r, 5
kan_bspline_basis_r(), *4*, *6*
kan_bspline_deriv_r, 6
kan_edge_function, 6
kan_edge_importance, 7
kan_extend_grid, 8
kan_fd_gradient_r, 8
kan_fit, 9
kan_fit(), *4*
kan_forward_r, 11
kan_gradient_r, 12
kan_init_state, 13
kan_init_state(), *11*
kan_layer, 14
kan_layer(), *4*
kan_load *(*kan_save*)*, 22
kan_make_grid, 15
kan_metrics, 15
kan_module_from_state, 16
kan_net, 17
kan_net(), *16*
kan_partial_dependence, 17
kan_plot_history, 18
kan_plot_network, 19
kan_prune, 19
kan_refine, 20
kan_refine(), *11*
kan_reg_control, 22
kan_reg_control(), *10*, *21*
kan_regularization_r, 21
kan_save, 22
kan_save(), *11*
kan_scale, 23
kan_scaler, 24
kan_spec, 24
kan_spec(), *10*, *13*, *14*, *17*
kan_synthetic, 26
kan_synthetic(), *4*
kan_unscale *(*kan_scale*)*, 23
kan_update_grid, 26
kan_update_grid(), *10*, *11*, *20*
kan_validate_all, 27
kan_validate_basis, 28
kan_validate_derivative, 28
kan_validate_gradient, 29

plot.kan_model, 30
predict.kan_model, 31

set.seed(), *13*
splines::splineDesign(), *5*, *6*, *28*