Package: pinnr (via r-universe)

November 1, 2026

Type Package

Title Physics-Informed Neural Networks in Native R with 'torch'

Version 0.1.1

Description Fits physics-informed neural networks (PINNs) that solve ordinary and partial differential equations and estimate unknown model parameters from data. Networks are trained by penalising the equation residual, boundary and initial conditions, and misfit to observations, following Raissi, Perdikaris and Karniadakis (2019) <doi:10.1016/j.jcp.2018.10.045>. Everything runs in native R on top of 'torch', with no 'Python' dependency, and ordinary differential equation models written as 'deSolve'-style derivative functions can be used directly.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

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

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

VignetteBuilder knitr

Config/testthat/edition 3

URL https://obasanmii.github.io/pinnr/, https://github.com/Obasanmii/pinnr

BugReports https://github.com/Obasanmii/pinnr/issues

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

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

Date/Publication 2026-11-01 22:09:58 UTC

RemoteUrl https://github.com/Obasanmii/pinnr

RemoteRef HEAD

RemoteSha 1d900c81efe9b4c9316db3f09f04fe1329f677b1

Contents

coef.pinn_fit . . . 2
conditions . . . 3
pinn_domain . . . 5
pinn_fit . . . 5
pinn_mlp . . . 7
pinn_ode . . . 8
pinn_problem . . . 9
pinn_sample . . . 11
pinn_save . . . 12
plot.pinn_fit . . . 13
predict.pinn_fit . . . 13
residuals.pinn_fit . . . 14

Index **15**

`coef.pinn_fit` *Estimated parameters*

Description

Estimated parameters

Usage

```
## S3 method for class 'pinn_fit'
coef(object, ...)
```

Arguments

`object`	A `pinn_fit`.
`...`	Unused.

Value

Named numeric vector of estimated unknown parameters (empty if the problem had none).

conditions *Boundary, initial and custom conditions*

Description

Conditions are enforced as soft penalties on points sampled where they apply. Each condition fixes one or more domain variables (`where`) and samples the remaining variables over their full range.

Usage

```
pinn_dirichlet(
  where,
  value = 0,
  output = NULL,
  n = 100,
  weight = 1,
  name = NULL
)

pinn_neumann(
  where,
  along,
  value = 0,
  output = NULL,
  n = 100,
  weight = 1,
  name = NULL
)

pinn_initial(
  value,
  time = "t",
  output = NULL,
  n = 100,
  weight = 1,
  name = NULL
)

pinn_condition(where, fn, n = 100, weight = 1, name = NULL)
```

Arguments

`where`	Named list giving the fixed value(s) of one or more domain variables. See *Locations*.
`value`	Target value. See *Values*.
`output`	Name(s) of the output(s) the condition applies to. The default `NULL` applies it to every output.

n	Number of points used for this condition.
weight	Relative weight of this condition in the loss.
name	Optional label used in printed output.
along	Name of the domain variable to differentiate with respect to.
time	Name of the time variable for pinn_initial().
fn	For pinn_condition(), a function returning the residual (a tensor, or a list of tensors) that should be zero. Arguments are matched by name to outputs, domain variables, parameters and the derivative operator d (see pinn_problem()).

Details

- pinn_dirichlet() fixes the value of the solution, $u = g$.
- pinn_neumann() fixes a first derivative, $\partial u/\partial v = g$.
- pinn_initial() is a Dirichlet condition at the lower end of the time variable.
- pinn_condition() takes any residual function, for Robin, periodic or other conditions.

Value

An object of class pinn_condition.

Locations

Each element of where is a number or one of the strings "lower", "upper" or "both", which refer to the ends of that variable's range. where = list(x = "both") places points on both ends of x. When several variables are fixed, every combination of their values is used: where = list(x = "both", y = "both") places points on the four corners.

Values

value can be a number, a numeric vector named by output, or a function. A function's arguments are matched by name to the domain variables, outputs and parameters, so value = function(x) sin(pi * x) uses the spatial variable x. Inside these functions the arguments are torch tensors, so use arithmetic and maths functions (sin, exp, log, ...) rather than control flow such as if or ifelse.

Examples

```
# u = 0 on both ends of x
pinn_dirichlet(where = list(x = "both"), value = 0)

# u(x, 0) = sin(pi x)
pinn_initial(function(x) sin(pi * x), time = "t")

# insulated right boundary: du/dx = 0 at x = 1
pinn_neumann(where = list(x = "upper"), along = "x")

# Robin condition du/dx + 2 u = 0 at x = 1, as a custom residual
pinn_condition(where = list(x = "upper"),
               fn = function(u, x, d) d(u, x) + 2 * u)
```

pinn_domain *Define a rectangular domain*

Description

Each argument names an input variable of the problem and gives its range as `c(lower, upper)`. The order of the arguments is the order in which the variables enter the network.

Usage

```
pinn_domain(...)
```

Arguments

`...`	Named numeric vectors of length 2, for example `x = c(0, 1), t = c(0, 10)`.

Value

An object of class `pinn_domain`.

Examples

```
pinn_domain(x = c(-1, 1), t = c(0, 1))
```

pinn_fit *Train a physics-informed neural network*

Description

Fits a fully connected network to a `pinn_problem()` (or a problem from `pinn_ode()`) by minimising

$$w_r L_{residual} + w_c L_{conditions} + w_d L_{data},$$

where each term is a mean squared error. Training runs `epochs` steps of Adam and then, optionally, `lbfgs_iter` iterations of L-BFGS, which often sharpens the solution considerably.

Usage

```
pinn_fit(
  problem,
  hidden = c(32, 32, 32),
  activation = "tanh",
  epochs = 5000,
  lr = 0.001,
  lr_unknown = NULL,
  lbfgs_iter = 0,
```

```
  n_collocation = 1000,
  sampling = c("lhs", "uniform"),
  resample_every = 0,
  weights = c(residual = 1, conditions = 1, data = 1),
  precision = c("float32", "float64"),
  device = "cpu",
  seed = NULL,
  verbose = interactive(),
  record_every = 10,
  windows = 1
)
```

Arguments

problem	A pinn_problem.
hidden	Integer vector of hidden layer widths.
activation	Activation function; see pinn_mlp().
epochs	Number of Adam steps.
lr	Adam learning rate for the network weights.
lr_unknown	Adam learning rate for unknown parameters. Defaults to 10 * lr.
lbfgs_iter	Number of L-BFGS iterations after Adam (0 to skip).
n_collocation	Number of points inside the domain where the residual is evaluated.
sampling	"lhs" (Latin hypercube) or "uniform".
resample_every	Draw new collocation and condition points every this many Adam steps (0 keeps one fixed set). Resampling reduces overfitting to the training points.
weights	Named numeric vector with loss weights residual, conditions and data. Missing names default to 1.
precision	"float32" (default) or "float64". Double precision is slower but can help L-BFGS reach smaller losses.
device	A torch device, such as "cpu", "cuda" or "mps".
seed	Optional integer seed for reproducible sampling and weights.
verbose	Print progress messages.
record_every	Store the loss every this many steps.
windows	Number of time windows for ODE problems; see *Long time horizons*.

Value

An object of class pinn_fit, with methods for predict(), coef(), residuals(), plot() and summary().

Choosing settings

The defaults solve small problems on a CPU in seconds to minutes. If the loss stalls, try more epochs, a wider or deeper network, more collocation points, or lbfgs_iter = 500. For parameter estimation, a larger lr_unknown lets the parameters move faster than the network weights. Long time horizons are hard for PINNs; shorter windows, or more data, help.

Long time horizons

PINNs often converge to a wrong, flat solution when an ODE is solved over a long horizon with fast early dynamics (for example an epidemic starting from a few cases). For problems from `pinn_ode()`, `windows = k` splits the time range into `k` equal windows and trains one network per window, each starting from the end state of the previous one ("time marching"). This is available for forward problems (no `unknown` parameters).

Pressing Esc (or Ctrl+C) stops training early and returns the model as trained so far, with a warning. If the loss becomes `NaN` or infinite, training stops and the weights with the lowest loss seen so far are returned, also with a warning (and `diverged = TRUE` in the result).

Examples

```
# Exponential decay dy/dt = -k y
decay <- function(t, y, parms) list(-parms["k"] * y)
prob <- pinn_ode(decay, y0 = c(y = 1), times = seq(0, 5, by = 0.5),
                 parms = c(k = 0.5))
fit <- pinn_fit(prob, hidden = c(16, 16), epochs = 1500, seed = 1)
head(predict(fit))
```

`pinn_mlp` *Fully connected network used by pinnr*

Description

A multilayer perceptron with Xavier-normal initial weights and zero biases, the standard choice for PINNs.

Usage

```
pinn_mlp(n_in, hidden = c(32, 32, 32), n_out = 1, activation = "tanh")
```

Arguments

`n_in`	Number of inputs.
`hidden`	Integer vector of hidden layer widths.
`n_out`	Number of outputs.
`activation`	`"tanh"` (default), `"sin"`, `"silu"`, `"gelu"` or `"softplus"`. Smooth activations are needed because the loss involves derivatives of the network; avoid ReLU.

Value

A `torch::nn_module`.

Examples

```
net <- pinn_mlp(2, c(16, 16), 1)
net(torch::torch_randn(5, 2))
```

---

pinn_ode *Build a problem from a deSolve-style ODE model*

Description

Turns an ordinary differential equation written for `deSolve::ode()` into a `pinn_problem()`. The same derivative function can be used to solve the system (a forward problem) or, with `unknown` and `data`, to estimate its parameters from observations.

Usage

```
pinn_ode(
  func,
  y0,
  times,
  parms = NULL,
  unknown = NULL,
  positive = TRUE,
  data = NULL,
  time = "time",
  hard_initial = TRUE
)
```

Arguments

`func`	A derivative function `function(t, y, parms)` in `deSolve` style. See *Writing the model*.
`y0`	Named numeric vector of initial states at the first time.
`times`	Numeric vector of times. Only its range is used for training; the full vector becomes the default grid for `predict()`.
`parms`	Named numeric vector or list of known parameters.
`unknown`	Named numeric vector of initial guesses for parameters to estimate. These are passed to `func` alongside `parms`.
`positive`	Logical; see `pinn_problem()`.
`data`	Optional data frame of observations with a column named as `time` and one column per observed state. The output of `deSolve::ode()` converted with `as.data.frame()` works directly.
`time`	Name of the time variable. The default, `"time"`, matches the column name in `deSolve` output.
`hard_initial`	Logical; enforce the initial state exactly. See *Initial conditions*.

Value

A pinn_problem() object.

Writing the model

func has the usual deSolve signature function(t, y, parms) and returns a list whose first element holds the derivatives, in the order of y0. The recommended style works unchanged:

```
function(t, y, parms) {
  with(as.list(c(y, parms)), {
    dS <- -beta * S * I
    ...
    list(c(dS, dI, dR))
  })
}
```

Indexing with y["S"], y[1] or parms["beta"] also works. Because the states are torch tensors, the body must use arithmetic and maths functions; if, ifelse, max and pmin on states do not work. Forcing functions of time can be written with tensor operations, for example torch_sin(t) or sin(t).

Initial conditions

By default the initial state is enforced exactly by writing each state as $y(t) = y_0 + \frac{t-t_0}{t_1-t_0} N(t)$, where N is the network. This removes one loss term and usually speeds up training. Set hard_initial = FALSE to use a soft penalty instead.

Examples

```
# Logistic growth dN/dt = r N (1 - N / K)
logistic <- function(t, y, parms) {
  with(as.list(c(y, parms)), list(r * N * (1 - N / K)))
}
prob <- pinn_ode(logistic, y0 = c(N = 0.1), times = seq(0, 10, by = 0.5),
                 parms = c(r = 1, K = 1))
prob
```

---

pinn_problem *Define a physics-informed problem*

Description

Collects everything a PINN needs: the domain, the names of the unknown functions, the equation residual, conditions, optional observations and the equation's parameters.

Usage

```
pinn_problem(
  domain,
  outputs,
  residual,
  conditions = list(),
  data = NULL,
  parms = list(),
  unknown = NULL,
  positive = TRUE,
  transform = NULL
)
```

Arguments

domain	A pinn_domain(), or a named list of ranges.
outputs	Character vector naming the unknown functions.
residual	Function returning the equation residual. See *Writing the residual*.
conditions	A condition or list of conditions from pinn_dirichlet(), pinn_neumann(), pinn_initial() or pinn_condition().
data	Optional data frame of observations, with one column per domain variable and one column for each observed output. Missing values (NA) are ignored, so outputs can be observed at different points.
parms	Named list or numeric vector of known constants.
unknown	Named numeric vector of initial guesses for parameters to estimate.
positive	Logical, recycled or named by unknown. Positive parameters are estimated on the log scale, which keeps them positive and usually helps convergence.
transform	Optional output transform. See *Output transforms*.

Value

An object of class pinn_problem.

Writing the residual

residual is an R function whose arguments are matched **by name** to:

- the outputs (for example u),
- the domain variables (for example x, t),
- the parameters in parms and unknown, and
- d, the derivative operator.

d(f, x) differentiates f with respect to the variable x; d(f, x, x) gives the second derivative and d(f, x, t) a mixed derivative. The function returns the expression that should equal zero, or a list of such expressions for a system of equations. For the viscous Burgers equation $u_t + uu_x - \nu u_{xx} = 0$:

```
function(u, x, t, nu, d) d(u, t) + u * d(u, x) - nu * d(u, x, x)
```

All arguments are torch tensors with one row per point, so use arithmetic and maths functions (`sin`, `exp`, `log`, `sqrt`, ...) and avoid `if`, `ifelse` or functions that leave R's torch methods.

Output transforms

`transform` can enforce conditions exactly ("hard constraints"). It is called with the raw network outputs, domain variables and parameters (matched by name, as above) and returns a named list of modified outputs. For example, to make $u(0,t) = u(1,t) = 0$ hold exactly: `transform = function(u, x) list(u = x * (1 - x) * u)`.

See Also

`pinn_ode()` for ordinary differential equations written in `deSolve` style, and `pinn_fit()` to train the network.

Examples

```
# Heat equation u_t = a u_xx with u(x, 0) = sin(pi x) and u = 0 at both ends
heat <- pinn_problem(
  domain = pinn_domain(x = c(0, 1), t = c(0, 1)),
  outputs = "u",
  residual = function(u, x, t, a, d) d(u, t) - a * d(u, x, x),
  conditions = list(
    pinn_initial(function(x) sin(pi * x), time = "t"),
    pinn_dirichlet(where = list(x = "both"), value = 0)
  ),
  parms = list(a = 0.1)
)
heat
```

---

pinn_sample *Sample points in a domain*

Description

Draws training points uniformly at random or with Latin hypercube sampling. Variables listed in `fixed` are held at the given values, which is how boundary and initial points are generated.

Usage

```
pinn_sample(n, domain, method = c("lhs", "uniform"), fixed = list())
```

Arguments

n	Number of points.
domain	A pinn_domain().
method	"lhs" (Latin hypercube, the default) or "uniform".
fixed	Named list of numeric values. When a variable has several values, the points are split evenly between them. With several fixed variables, every combination of their values is used, so list(x = c(0, 1), y = c(0, 1)) gives the four corners. Each value (or combination) gets at least one point, so the result can have more than n rows when n is smaller than the number of combinations.

Value

A data frame with one column per domain variable.

Examples

```
dom <- pinn_domain(x = c(0, 1), t = c(0, 2))
head(pinn_sample(5, dom))
pinn_sample(4, dom, fixed = list(x = c(0, 1)))
```

---

pinn_save *Save and load fitted PINNs*

Description

Torch modules cannot be stored with saveRDS() directly. pinn_save() stores the network weights as plain arrays alongside the rest of the fit, and pinn_load() rebuilds the network.

Usage

```
pinn_save(fit, path)

pinn_load(path)
```

Arguments

fit	A pinn_fit.
path	File path, conventionally ending in .rds.

Value

pinn_save() returns path invisibly; pinn_load() returns a pinn_fit.

plot.pinn_fit *Plot a fitted PINN*

Description

Plot a fitted PINN

Usage

```
## S3 method for class 'pinn_fit'
plot(x, type = c("loss", "solution", "parameters"), output = NULL, ...)
```

Arguments

x	A pinn_fit.
type	"loss" for the training history, "solution" for the fitted outputs (1D and 2D domains), or "parameters" for the path of estimated parameters during training.
output	For 2D solutions, which output to draw.
...	Passed to the underlying base graphics function.

Value

x, invisibly.

predict.pinn_fit *Predict from a fitted PINN*

Description

Evaluates the trained network at new points.

Usage

```
## S3 method for class 'pinn_fit'
predict(object, newdata = NULL, ...)
```

Arguments

object	A pinn_fit.
newdata	Data frame with one column per domain variable. When omitted, the times given to pinn_ode() are used for ODE problems, and otherwise a regular grid (201 points in 1D, 101 x 101 in 2D).
...	Unused.

Value

A data frame with the domain variables followed by one column per output.

Examples

```
decay <- function(t, y, parms) list(-parms["k"] * y)
prob <- pinn_ode(decay, y0 = c(y = 1), times = c(0, 5), parms = c(k = 0.5))
fit <- pinn_fit(prob, hidden = c(16, 16), epochs = 1000, seed = 1)
predict(fit, data.frame(time = c(0, 1, 2)))
```

---

residuals.pinn_fit *Equation residuals of a fitted PINN*

Description

Evaluates the equation residual at new points. Values close to zero mean the network satisfies the equation there; large values show where the solution is unreliable.

Usage

```
## S3 method for class 'pinn_fit'
residuals(object, newdata = NULL, ...)
```

Arguments

object	A pinn_fit.
newdata	Data frame with one column per domain variable. When omitted, the times given to pinn_ode() are used for ODE problems, and otherwise a regular grid (201 points in 1D, 101 x 101 in 2D).
...	Unused.

Value

A data frame with the domain variables followed by one residual column per equation.

Index

as.data.frame(), *8*

coef(), *6*
coef.pinn_fit, 2
conditions, 3

deSolve::ode(), *8*

pinn_condition *(*conditions*)*, 3
pinn_condition(), *10*
pinn_dirichlet *(*conditions*)*, 3
pinn_dirichlet(), *10*
pinn_domain, 5
pinn_domain(), *10*, *12*
pinn_fit, 5
pinn_fit(), *11*
pinn_initial *(*conditions*)*, 3
pinn_initial(), *10*
pinn_load *(*pinn_save*)*, 12
pinn_mlp, 7
pinn_mlp(), *6*
pinn_neumann *(*conditions*)*, 3
pinn_neumann(), *10*
pinn_ode, 8
pinn_ode(), *5*, *7*, *11*, *13*, *14*
pinn_problem, 9
pinn_problem(), *4*, *5*, *8*, *9*
pinn_sample, 11
pinn_save, 12
plot(), *6*
plot.pinn_fit, 13
predict(), *6*, *8*
predict.pinn_fit, 13

residuals(), *6*
residuals.pinn_fit, 14

saveRDS(), *12*
summary(), *6*