Package 'nnfor'

November 13, 2026

Type Package

Title Time Series Forecasting with Neural Networks

Version 0.10.0

Description Automatic time series modelling with neural networks.
Allows fully automatic, semi-manual or fully manual specification of networks. For details of the specification methodol-
ogy see: (i) Crone and Kourentzes (2010) <doi:10.1016/j.neucom.2010.01.017>;
and (ii) Kourentzes et al. (2014) <doi:10.1016/j.eswa.2013.12.011>.

License GPL-3

Encoding UTF-8

Depends generics

Imports forecast, glmnet, neuralnet, plotrix, MASS, tsutils (>= 0.9.5), uroot, parallel, datasets

Suggests thief, torch, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://kourentzes.com/forecasting/2019/01/16/tutorial-for-the-nnfor-r-package/

BugReports https://github.com/trnnick/nnfor/issues

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Nikolaos Kourentzes [aut, cre]

Maintainer Nikolaos Kourentzes <nikolaos@kourentzes.com>

Repository CRAN

Date/Publication 2026-11-13 16:40:07 UTC

Contents

nnfor-package 2
compare_backends 4

elm . . . 6
elm.fast . . . 11
elm.thief . . . 14
forecast.elm . . . 15
forecast.mlp . . . 16
linscale . . . 18
mlp . . . 19
mlp.thief . . . 26
nnfor_map . . . 27
plot.elm . . . 28
plot.elm.fast . . . 29
plot.forecast.net . . . 30
plot.mlp . . . 31
plot_fit . . . 33
plot_network . . . 34
predict.elm.fast . . . 35
print.nnfor_map . . . 36

Index **39**

`nnfor-package` *nnfor: Time Series Forecasting with Neural Networks*

Description

The **nnfor** package provides automatic time series modelling with neural networks. It facilitates fully automatic, semi-manual or fully manual specification of networks, using multilayer perceptrons (`mlp`) and extreme learning machines (`elm`). Specification covers the input lags, differencing, deterministic seasonality, exogenous regressors and the number of hidden nodes, and forecasts are combined across an ensemble of networks.

Training backends

Multilayer perceptrons are trained with `neuralnet` by default, which is installed with **nnfor**. As an option, they can be trained with torch by setting `backend = "torch"` in `mlp`. torch adds modern activation functions and optimizers, custom architectures (`hd` given as a function), and trains all networks of an ensemble simultaneously. It is optional: install it with `install.packages("torch")` followed by `torch::install_torch()`. Nothing else in **nnfor** needs it.

By default the torch backend reproduces neuralnet's training setup and accepts neuralnet's argument names; the two backends still produce different networks. The torch settings, suggested settings and the differences between the backends are described in the Details of `mlp`, and `compare_backends` compares them on a series. Extreme learning machines (`elm`, `elm.fast`) estimate output weights by regression and do not use a training backend.

Functions

Modelling:

`mlp` Multilayer perceptrons, trained with neuralnet or torch.

`elm` Extreme learning machines.

`elm.fast` Fast extreme learning machine for general regression problems (inputs and target given directly); the engine of `elm(barebone = TRUE)`.

Fitted networks can be printed, and reused on new data with the `model` argument of `mlp` and `elm`.

Forecasting:

`forecast(fit)` Forecasts from `mlp` and `elm` fits (`forecast.mlp`, `forecast.elm`).

`predict(fit, newx)` Predictions from `elm.fast` fits for new inputs.

`mlp.thief`, `elm.thief` Wrappers to forecast with temporal hierarchies using the **thief** package.

Visualisation:

`plot(fit)` For `mlp` and `elm` fits: the network architecture or the in-sample fit (`plot.mlp`, `plot.elm`).

`plot_network`, `plot_fit` The same plots, with all their options for customisation.

`plot(forecast(fit))` Forecasts from `mlp` and `elm` fits.

`plot(fit)` For `elm.fast` fits: fitted against actual values.

All plots respect graphical parameters set by the user and accept further graphical arguments.

Transparency and comparison:

`nnfor_map` Map of how information flows through the package when fitting or forecasting, optionally annotated with a fitted network; see `plot.nnfor_map`.

`compare_backends` Fits and forecasts a series with several backend configurations, reporting run times, in-sample, and out-of-sample errors.

Utilities:

`linscale` Linear scaling of a vector to a given range, and its reverse.

Parallel training

Ensemble members can be trained in parallel with `options(nnfor.cores = 4)` (forked workers on Linux and macOS) or `options(nnfor.cores = cl)` with a cluster from `parallel::makeCluster()`, which is recommended on Windows. All random numbers are drawn before members are dispatched, so results are identical for any number of cores and reproducible with `set.seed()`.

Note

You can find a tutorial how to use the package here.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

References

- For an introduction to neural networks see: Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting 2e. *Wessex Press Publishing Co.*, Chapter 10.
- For ensemble combination operators see: Kourentzes N., Barrow B.K., Crone S.F. (2014) Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, **41**(**9**), 4235-4244.
- For variable selection see: Crone S.F., Kourentzes N. (2010) Feature selection for time series prediction – A combined filter and wrapper approach for neural networks. *Neurocomputing*, **73**(**10**), 1923-1936.

See Also

Useful links:

- https://kourentzes.com/forecasting/2019/01/16/tutorial-for-the-nnfor-r-package/
- Report bugs at https://github.com/trnnick/nnfor/issues

`compare_backends` *Compare network backends*

Description

Fits `mlp` with each backend configuration on a time series, holding out the last `h` observations, and reports run times with in-sample and out-of-sample mean squared errors. It is meant to give a feel for the backends on a given series, not as a benchmark.

Usage

```
compare_backends(
  y = datasets::AirPassengers,
  h = NULL,
  reps = 10,
  configs = NULL,
  seed = 1,
  ...
)

## S3 method for class 'nnfor_comparison'
print(x, digits = 3, ...)

## S3 method for class 'nnfor_comparison'
plot(x, col = NULL, lty = NULL, legend = TRUE, ...)
```

Arguments

y	Time series to use.
h	Number of observations held out and forecast. Defaults to the longest seasonal period of y, or 10 for non-seasonal series.
reps	Number of networks in each ensemble.
configs	Named list of configurations, each a list of arguments for mlp, e.g. list(backend = "torch", act.fct = "tanh"). If NULL, compares neuralnet, torch with neuralnet's settings, and torch with the settings suggested in mlp.
seed	Random seed used for every configuration.
...	For compare_backends, further arguments passed to mlp for every configuration. For plot, arguments passed to plot (e.g. xlim, main); by default the plot shows the hold-out and the 3h observations before it. Unused by print.
x	Object of class nnfor_comparison.
digits	Number of significant digits to print.
col	Colours of the forecasts. Recycled if there are more configurations than colours, with a new line type each round so that every configuration stays distinguishable.
lty	Line types of the forecasts. By default solid for the first round of colours, dashed for the second, and so on.
legend	Logical, add a legend.

Details

Every configuration is fitted from the same random seed and with the same arguments in ..., so the specification of the network (lags, differencing, seasonal dummies, hidden nodes) is the same and only the training differs. Configurations that need torch are skipped with a message if torch is not installed.

Value

An object of class nnfor_comparison: a list with results (a data frame with one row per configuration: fit and forecast time in seconds, in-sample and out-of-sample MSE), the forecasts, the fits, and the series split into train and test. Use plot to see the forecasts.

See Also

mlp, nnfor_map.

Examples

```
# Runs with neuralnet only, so it does not need torch
cmp <- compare_backends(AirPassengers, reps = 2, configs = list(
  "neuralnet: logistic" = list(backend = "neuralnet"),
  "neuralnet: tanh"     = list(backend = "neuralnet", act.fct = "tanh")))
print(cmp)
plot(cmp)
```

```
## Not run:
 cmp <- compare_backends(AirPassengers)
 print(cmp)
 plot(cmp)
 # Own configurations
 compare_backends(AirPassengers, configs = list(
   "torch, tanh" = list(backend = "torch", act.fct = "tanh"),
   "torch, relu" = list(backend = "torch", act.fct = "relu", algorithm = "adam",
                        learningrate = 0.01, stepmax = 2000)))

## End(Not run)

## Not run:
 # Stress test: many backend configurations with large ensembles. This takes a
 # while; neuralnet's time grows with reps (training it in parallel helps),
 # while torch trains all members of an ensemble in one batch.
 options(nnfor.cores = max(1, parallel::detectCores() - 1))  # neuralnet only
 adam <- list(algorithm = "adam", learningrate = 0.01, stepmax = 3000)
 configs <- list(
   "neuralnet: logistic, rprop+" = list(backend = "neuralnet"),
   "neuralnet: tanh, rprop+"     = list(backend = "neuralnet", act.fct = "tanh"),
   "torch: logistic, rprop"      = list(backend = "torch"),
   "torch: tanh, rprop"          = list(backend = "torch", act.fct = "tanh"),
   "torch: gelu, adam"           = c(list(backend = "torch", act.fct = "gelu"), adam),
   "torch: silu, adam"           = c(list(backend = "torch", act.fct = "silu"), adam),
   "torch: relu, adam"           = c(list(backend = "torch", act.fct = "relu"), adam),
  "torch: leaky_relu, adam"    = c(list(backend = "torch", act.fct = "leaky_relu"), adam),
  "torch: mish, adamw"       = list(backend = "torch", act.fct = "mish", algorithm = "adamw",
                                    learningrate = 0.01, stepmax = 3000),
  "torch: elu, rmsprop"      = list(backend = "torch", act.fct = "elu", algorithm = "rmsprop",
                                    learningrate = 0.001, stepmax = 3000),
   "torch: softplus, adagrad"    = list(backend = "torch", act.fct = "softplus",
                                    algorithm = "adagrad", learningrate = 0.05,
                                    stepmax = 3000),
  "torch: tanh, sgd"         = list(backend = "torch", act.fct = "tanh", algorithm = "sgd",
                                    learningrate = 0.001, stepmax = 5000))
 cmp <- compare_backends(AirPassengers, reps = 100, configs = configs)
 print(cmp)
 plot(cmp)
 # Configurations ranked by out-of-sample MSE
 cmp$results[order(cmp$results$MSE.out), c("configuration", "fit.time", "MSE.in", "MSE.out")]

## End(Not run)
```

---

elm *Extreme learning machines for time series forecasting*

Description

This function fits ELM neural networks for time series forecasting.

Usage

```
elm(
  y,
  m = NULL,
  hd = NULL,
  type = c("lasso", "ridge", "step", "lm"),
  reps = 20,
  comb = c("median", "mean", "mode"),
  lags = NULL,
  keep = NULL,
  difforder = NULL,
  outplot = FALSE,
  sel.lag = TRUE,
  direct = FALSE,
  allow.det.season = TRUE,
  det.type = c("auto", "bin", "trg"),
  xreg = NULL,
  xreg.lags = NULL,
  xreg.keep = NULL,
  barebone = FALSE,
  model = NULL,
  retrain = FALSE
)
```

Arguments

y	Input time series. Can be ts or msts object.
m	Seasonal period(s) of the series. Use a vector for multiple seasonalities, e.g. c(7, 365.25). If NULL (default) they are taken from y: its frequency for a ts object, or all seasonal periods for an msts object. When given, m overrides the periods of y for lag specification, seasonality and differencing tests and seasonal dummies.
hd	Number of hidden nodes. This can be a vector, where each number represents the number of hidden nodes of a different hidden layer. Use NULL to automatically specify.
type	Estimation type for output layer weights. Can be "lasso" (lasso with CV), "ridge" (ridge regression with CV), "step" (stepwise regression with AIC) or "lm" (linear regression; "ls" is accepted as a synonym).
reps	Number of networks to train, the result is the ensemble forecast.
comb	Combination operator for forecasts when reps > 1. Can be "median", "mode" (based on KDE estimation) and "mean".
lags	Lags of y to use as inputs. If none provided then 1:frequency(y) is used. Use 0 for no univariate lags.
keep	Logical vector to force lags to stay in the model if sel.lag == TRUE. If NULL then it keep = rep(FALSE,length(lags)).

`difforder` Vector including the differencing lags. For example c(1,12) will apply first and seasonal (12) differences. For no differencing use 0. For automatic selection use NULL.

`outplot` Provide plot of model fit. Can be TRUE or FALSE. For a customised plot use `plot(fit, which = "fit", ...)` after fitting.

`sel.lag` Automatically select lags. Can be TRUE or FALSE.

`direct` Use direct input-output connections to model strictly linear effects. Can be TRUE or FALSE.

`allow.det.season` Permit modelling seasonality with deterministic dummies.

`det.type` Type of deterministic seasonality dummies to use. This can be "bin" for binary or "trg" for a sine-cosine pair. With "auto" if ony a single seasonality is used and periodicity is up to 12 then "bin" is used, otherwise "trg".

`xreg` Exogenous regressors. Each column is a different regressor and the sample size must be at least as long as the target in-sample set, but can be longer.

`xreg.lags` This is a list containing the lags for each exogenous variable. Each list is a numeric vector containing lags. If xreg has 3 columns then the xreg.lags list must contain three elements. If NULL then it is automatically specified.

`xreg.keep` List of logical vectors to force lags of xreg to stay in the model if sel.lag == TRUE. If NULL then all exogenous lags can be removed.

`barebone` Use an alternative elm implementation (written in R) that is faster when the number of inputs is very high. Typically not needed. Supports a single hidden layer only.

`model` A previously trained elm object. If this is provided then the same model is fitted to y, without re-estimating any model parameters.

`retrain` If a previous model is provided, retrain the network or not. If the network is retrained the size of the hidden layer is reset.

Value

Return object of class `elm`. If barebone == TRUE then the object inherits a second class "`elm.fast`". The function `plot` produces a plot the network architecture or of the fit (see `plot.elm`). `elm` contains:

- `net`: ELM networks. If it is of class "`elm.fast`" then this is NULL.
- `hd`: Number of hidden nodes. If it is of class "`elm.fast`" this is a vector with a different number for each repetition.
- `W.in`: NULL unless it is of class "`elm.fast`". Contains the input weights.
- `W`: Output layer weights for each repetition.
- `b`: Contains the output node bias for each training repetition.
- `W.dct`: Contains the direct connection weights if argument direct == TRUE. Otherwise is NULL.
- `lags`: Input lags used.
- `xreg.lags`: `xreg` lags used.

- `difforder`: Differencing used.
- `sdummy`: Use of deterministic seasonality.
- `ff`: Seasonal periods modelled (`m`, or those of `y`).
- `ff.det`: Seasonal frequencies coded using deterministic dummies.
- `det.type`: Type of determistic seasonality.
- `y`: Input time series.
- `minmax`: Scaling structure.
- `xreg.minmax`: Scaling structure for xreg variables.
- `comb`: Combination operator used.
- `type`: Estimation used for output layer weights.
- `direct`: Presence of direct input-output connections.
- `fitted`: Fitted values.
- `MSE`: In-sample Mean Squared Error.
- `fitted.all`: Fitted values of each ensemble member.

Parallel training

Ensemble members can be trained in parallel with `options(nnfor.cores = 4)` (forked workers on Linux and macOS) or `options(nnfor.cores = cl)` with a cluster from `parallel::makeCluster()`, which is recommended on Windows. All random numbers are drawn before members are dispatched, so results are identical for any number of cores and reproducible with `set.seed()`.

Note

To use elm with Temporal Hierarchies (thief package) see `elm.thief`. The elm function by default calls the `neuralnet` function. If barebone == TRUE then it uses an alternative implementation (`elm.fast`) which is more appropriate when the number of inputs is several hundreds.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

References

- For an introduction to neural networks see: Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting 2e. *Wessex Press Publishing Co.*, Chapter 10.
- For combination operators see: Kourentzes N., Barrow B.K., Crone S.F. (2014) Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, **41**(**9**), 4235-4244.
- For variable selection see: Crone S.F., Kourentzes N. (2010) Feature selection for time series prediction – A combined filter and wrapper approach for neural networks. *Neurocomputing*, **73**(**10**), 1923-1936.
- For ELMs see: Huang G.B., Zhou H., Ding X. (2006) Extreme learning machine: theory and applications. *Neurocomputing*, **70**(**1**), 489-501.

See Also

forecast.elm, elm.thief, mlp.

Examples

```
## Not run:
 fit <- elm(AirPassengers)
 print(fit)
 plot(fit)
 frc <- forecast(fit,h=36)
 plot(frc)

## End(Not run)
## Not run:
# Data for the examples with exogenous regressors (xreg): a promotion
# indicator and a slowly moving price. xreg must cover the sample and the
# forecast horizon.
set.seed(1)
h <- 12
n <- length(AirPassengers)
z <- cbind(promo = rbinom(n + h, 1, 0.2),
           price = 50 + cumsum(rnorm(n + h)))
y <- AirPassengers + 30 * z[1:n, "promo"] - 2 * z[1:n, "price"]

# Regressors with automatic lags: each starts from the same candidate lags
# as y (here 1 to 12, so not lag 0, the current value of the regressor) and
# lag selection keeps the useful ones
fit <- elm(y, xreg = z)
print(fit)
# Forecasting needs the future values of the regressors
frc <- forecast(fit, h = h, xreg = z)

# Lags for each regressor, one vector per column of xreg, and lags kept
# whatever the lag selection says: promo at lag 0, price at lag 0
fit <- elm(y, xreg = z, xreg.lags = list(0, 0:2),
           xreg.keep = list(TRUE, c(TRUE, FALSE, FALSE)))

# Use exactly the lags given, without lag selection
fit <- elm(y, xreg = z, xreg.lags = list(0, 1:3), sel.lag = FALSE)

# Regressors as the only inputs: no lags of y (lags = 0)
fit <- elm(y, xreg = z, lags = 0, xreg.lags = list(0, 0:2), sel.lag = FALSE)

# Number of hidden nodes (hd); by default it depends on the sample size
fit <- elm(AirPassengers, hd = 50)          # one hidden layer of 50 nodes
fit <- elm(AirPassengers, hd = c(30, 10))   # two random hidden layers
plot(fit)  # hidden nodes that the output layer does not use are dashed

# Estimation of the output weights: "lasso" (default), "ridge", "step"
# (stepwise regression) or "lm"
fit <- elm(AirPassengers, type = "ridge")
fit <- elm(AirPassengers, type = "step")
```

```
# Direct connections from the inputs to the output, besides the hidden layer
fit <- elm(AirPassengers, direct = TRUE)

# The fast implementation, for many inputs; also returns an elm object
fit <- elm(AirPassengers, barebone = TRUE)

# Save a fitted network and use it later
fit <- elm(AirPassengers)
file <- file.path(tempdir(), "elm_fit.rds")
saveRDS(fit, file)
fit <- readRDS(file)
frc <- forecast(fit, h = 12)

# Apply a network to new data without re-estimating it: a network fitted up
# to 1958 is used on the full series
fit.old <- elm(window(AirPassengers, end = c(1958, 12)))
fit.new <- elm(AirPassengers, model = fit.old)
# ...or keep its specification and re-estimate on the new data (the number
# of hidden nodes is chosen again)
fit.new <- elm(AirPassengers, model = fit.old, retrain = TRUE)

# A reused network with regressors needs the regressors of the new data
fit.old <- elm(window(y, end = c(1958, 12)), xreg = z, xreg.lags = list(0, 0:2))
fit.new <- elm(y, model = fit.old, xreg = z)
frc <- forecast(fit.new, h = h, xreg = z)

## End(Not run)
```

`elm.fast` *ELM (fast) neural network*

Description

Fit ELM (fast) neural network. This is an ELM implementation that does not rely on neuralnets package.

Usage

```
elm.fast(
  y,
  x,
  hd = NULL,
  type = c("lasso", "ridge", "step", "ls"),
  reps = 20,
  comb = c("median", "mean", "mode"),
  direct = FALSE,
  linscale = TRUE,
```

```
  output = c("linear", "logistic"),
  core = FALSE,
  ortho = FALSE
)
```

Arguments

y	Target variable.
x	Explanatory variables. Each column is a variable.
hd	Starting number of hidden nodes (scalar). Use NULL to automatically specify.
type	Estimation type for output layer weights. Can be "lasso" (lasso with CV), "ridge" (ridge regression with CV), "step" (stepwise regression with AIC) or "ls" (linear regression; "lm" is accepted as a synonym).
reps	Number of networks to train.
comb	Combination operator for forecasts when reps > 1. Can be "median", "mode" (based on KDE estimation) and "mean".
direct	Use direct input-output connections to model strictly linear effects. Can be TRUE or FALSE.
linscale	Scale inputs linearly between -0.8 to 0.8. If output == "logistic" then scaling is between 0 and 1.
output	Type of output layer. It can be "linear" or "logistic". If "logistic" then type must be set to "lasso". Logistic outputs are probabilities from the inverse logit of the output layer and, if `linscale = TRUE`, back to the range of `y`.
core	If TRUE skips calculation of final fitted values and MSE. Called internally by `"elm"` function.
ortho	If TRUE then the initial weights between the input and hidden layers are orthogonal (only when number of input variable <= sample size).

Value

An object of class `"elm.fast"`. The function `plot` produces a plot the network fit (see `plot.elm.fast`). An object of class `"elm.fast"` is a list containing the following elements:

- `hd`: Number of hidden nodes. This is a vector with a different number for each training repetition.
- `W.in`: Input weights for each training repetition.
- `W`: Output layer weights for each repetition.
- `b`: Output node bias for each training repetition.
- `W.dct`: Direct connection weights argument if direct == TRUE for each training repetition. Otherwise NULL.
- `fitted.all`: Fitted values for each training repetition.
- `fitted`: Ensemble fitted values.
- `y`: Target variable.
- `type`: Estimation used for output layer weights.

- `comb`: Combination operator used.
- `direct`: Presence of direct input-output connections.
- `minmax`: If scaling is used this contains the scaling information for the target variable.
- `minmax.x`: If scaling is used this contains the scaling information for the input variables.
- `MSE`: In-sample Mean Squared Error.

Note

This implementation of ELM is more appropriate when the number of inputs is several hundreds. For time series modelling use `elm` instead.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

References

- For combination operators see: Kourentzes N., Barrow B.K., Crone S.F. (2014) Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, **41**(**9**), 4235-4244.
- For ELMs see: Huang G.B., Zhou H., Ding X. (2006) Extreme learning machine: theory and applications. *Neurocomputing*, **70**(**1**), 489-501.

See Also

`elm`.

Examples

```
## Not run:
 p <- 2000
 n <- 150
 X <- matrix(rnorm(p*n),nrow=n)
 b <- cbind(rnorm(p))
 Y <- X %*% b
 fit <- elm.fast(Y,X)
 print(fit)

## End(Not run)
```

`elm.thief` *ELM network for THieF*

Description

Function for ELM forecasting with Temporal Hierarchies.

Usage

```
elm.thief(y, h = NULL, ...)
```

Arguments

`y`	Input time series. Can be ts or msts object.
`h`	Forecast horizon. If NULL then h is set to match frequency of time series.
`...`	Additional arguments passed to `elm`.

Value

An object of classes `"forecast.net"` and `"forecast"`. The function `plot` produces a plot of the forecasts. An object of class `"forecast.net"` is a list containing the following elements:

- `method`: The name of the forecasting method as a character string
- `mean`: Point forecasts as a time series
- `all.mean`: An array h x reps of all ensemble members forecasts, where reps are the number of ensemble members.
- `x`: The original time series (either `fit` used to create the network.
- `fitted`: Fitted values. Any values not fitted for the initial period of the time series are imputted with NA.
- `residuals`: Residuals from the fitted network.

Note

This function is created to work with Temporal Hierarchies (thief package). For conventional ELM networks use `elm`.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

References

- For forecasting with temporal hierarchies see: Athanasopoulos G., Hyndman R.J., Kourentzes N., Petropoulos F. (2017) Forecasting with Temporal Hierarchies. *European Journal of Operational research*, **262**(**1**), 60-74.
- For combination operators see: Kourentzes N., Barrow B.K., Crone S.F. (2014) Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, **41**(**9**), 4235-4244.

See Also

elm, mlp.thief.

Examples

```
## Not run:
  library(thief)
  frc <- thief(AirPassengers,forecastfunction=elm.thief)
  plot(frc)

## End(Not run)
```

---

forecast.elm *Forecast using ELM neural network*

Description

Create forecasts using ELM neural networks.

Usage

```
## S3 method for class 'elm'
forecast(object, h = NULL, y = NULL, xreg = NULL, ...)
```

Arguments

object	ELM network object, produced using elm.
h	Forecast horizon. If NULL then h is set to match frequency of time series.
y	Optionally forecast using different data than what the network was trained on. Expected to create havoc and do really bad things!
xreg	Exogenous regressors. Each column is a different regressor and the sample size must be at least as long as the target in-sample set plus the forecast horizon, but can be longer. Set it to NULL if no xreg inputs are used.
...	Unused argument.

Value

An object of classes "forecast.net" and "forecast". The function plot produces a plot of the forecasts. An object of class "forecast.net" is a list containing the following elements:

- method: The name of the forecasting method as a character string
- mean: Point forecasts as a time series
- all.mean: An array h x reps of all ensemble members forecasts, where reps are the number of ensemble members.
- x: The original time series used to create the network.
- fitted: Fitted values.
- residuals: Residuals from the fitted network.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

See Also

elm, elm.thief, mlp.

Examples

```
## Not run:
 fit <- elm(AirPassengers)
 plot(fit)
 frc <- forecast(fit,h=36)
 plot(frc)

## End(Not run)
## Not run:
# Networks with regressors (xreg) need their values over the forecast horizon
set.seed(1)
z <- cbind(promo = rbinom(156, 1, 0.2))
y <- AirPassengers + 30 * z[1:144, "promo"]
fit <- elm(y, xreg = z, xreg.lags = list(0))
frc <- forecast(fit, h = 12, xreg = z)
plot(frc)

## End(Not run)
```

---

forecast.mlp *Forecast using MLP neural network*

Description

Create forecasts using MLP neural networks.

Usage

```
## S3 method for class 'mlp'
forecast(object, h = NULL, y = NULL, xreg = NULL, ...)
```

Arguments

object	MLP network object, produced using mlp.
h	Forecast horizon. If NULL then h is set to match frequency of time series.
y	Optionally forecast using different data than what the network was trained on. Expected to create havoc and do really bad things!

xreg	Exogenous regressors. Each column is a different regressor and the sample size must be at least as long as the target in-sample set plus the forecast horizon, but can be longer. Set it to NULL if no xreg inputs are used.
...	Unused argument.

Value

An object of classes "forecast.net" and "forecast". The function plot produces a plot of the forecasts. An object of class "forecast.net" is a list containing the following elements:

- method: The name of the forecasting method as a character string
- mean: Point forecasts as a time series
- all.mean: An array h x reps of all ensemble members forecasts, where reps are the number of ensemble members.
- x: The original time series used to create the network.
- fitted: Fitted values.
- residuals: Residuals from the fitted network.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

See Also

mlp, mlp.thief, elm.

Examples

```
## Not run:
 fit <- mlp(AirPassengers)
 plot(fit)
 frc <- forecast(fit,h=36)
 plot(frc)

## End(Not run)
## Not run:
# Networks with regressors (xreg) need their values over the forecast horizon
set.seed(1)
z <- cbind(promo = rbinom(156, 1, 0.2))
y <- AirPassengers + 30 * z[1:144, "promo"]
fit <- mlp(y, xreg = z, xreg.lags = list(0))
frc <- forecast(fit, h = 12, xreg = z)
plot(frc)

## End(Not run)
```

`linscale` *Apply minmax linear scaling to a vector*

Description

Apply minmax linear scaling to a vector.

Usage

```
linscale(x, minmax = NULL, rev = FALSE)
```

Arguments

`x`	Input vector.
`minmax`	minmax must be a list with elements "mn", "mx", "mn.orig" and "mx.orig", where "mn" and "mx" refer to the target min and max, and the remaining two refer to the current vector min and max. By default mn=-1 and mx=1. mn.orig and mx.orig can be missing, unless the scaling is reversed.
`rev`	Reverse scaling back to original: TRUE or FALSE.

Value

Outputs a list with elements:

- `x` - Scaled vector.
- `minmax` - List with resulting mn, mx, mn.orig and mx.orig. Can be used as input to reverse scaling.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

Examples

```
y <- rnorm(20)*100
sc <- linscale(y)
x <- sc$x
print(c(min(y),max(y)))
print(c(min(x),max(x)))
sc.rev <- linscale(x,minmax=sc$minmax,rev=TRUE)
print(c(min(sc.rev$x),max(sc.rev$x)))
```

mlp *Multilayer Perceptron for time series forecasting*

Description

This function fits MLP neural networks for time series forecasting.

Usage

```
mlp(
  y,
  m = NULL,
  hd = NULL,
  reps = 20,
  comb = c("median", "mean", "mode"),
  lags = NULL,
  keep = NULL,
  difforder = NULL,
  outplot = FALSE,
  sel.lag = TRUE,
  allow.det.season = TRUE,
  det.type = c("auto", "bin", "trg"),
  xreg = NULL,
  xreg.lags = NULL,
  xreg.keep = NULL,
  hd.auto.type = c("set", "valid", "cv", "elm"),
  hd.max = NULL,
  backend = c("neuralnet", "torch"),
  model = NULL,
  retrain = FALSE,
  ...
)
```

Arguments

y	Input time series. Can be ts or msts object.
m	Seasonal period(s) of the series. Use a vector for multiple seasonalities, e.g. `c(7, 365.25)`. If NULL (default) they are taken from `y`: its frequency for a ts object, or all seasonal periods for an msts object. When given, `m` overrides the periods of `y` for lag specification, seasonality and differencing tests and seasonal dummies.
hd	Number of hidden nodes. This can be a vector, where each number represents the number of hidden nodes of a different hidden layer. With `backend = "torch"` it can also be a function defining a custom architecture, see Details.
reps	Number of networks to train, the result is the ensemble forecast.

`comb` Combination operator for forecasts when reps > 1. Can be "median", "mode" (based on KDE estimation) and "mean".

`lags` Lags of y to use as inputs. If none provided then 1:frequency(y) is used. Use 0 for no univariate lags.

`keep` Logical vector to force lags to stay in the model if sel.lag == TRUE. If NULL then it keep = rep(FALSE,length(lags)).

`difforder` Vector including the differencing lags. For example c(1,12) will apply first and seasonal (12) differences. For no differencing use 0. For automatic selection use NULL.

`outplot` Provide plot of model fit. Can be TRUE or FALSE. For a customised plot use `plot(fit, which = "fit", ...)` after fitting.

`sel.lag` Automatically select lags. Can be TRUE or FALSE.

`allow.det.season` Permit modelling seasonality with deterministic dummies.

`det.type` Type of deterministic seasonality dummies to use. This can be "bin" for binary or "trg" for a sine-cosine pair. With "auto" if ony a single seasonality is used and periodicity is up to 12 then "bin" is used, otherwise "trg".

`xreg` Exogenous regressors. Each column is a different regressor and the sample size must be at least as long as the target in-sample set, but can be longer.

`xreg.lags` This is a list containing the lags for each exogenous variable. Each list is a numeric vector containing lags. If xreg has 3 columns then the xreg.lags list must contain three elements. If NULL then it is automatically specified.

`xreg.keep` List of logical vectors to force lags of xreg to stay in the model if sel.lag == TRUE. If NULL then all exogenous lags can be removed. The syntax for multiple xreg is the same as for xreg.lags.

`hd.auto.type` Used only if hd==NULL. "set" fixes hd=5. "valid" uses a 20% validation set (randomly) sampled to find the best number of hidden nodes. "cv" uses 5-fold cross-validation. "elm" uses ELM to estimate the number of hidden nodes (experimental).

`hd.max` When hd.auto.type is set to either "valid" or "cv" then this argument can be used to set the maximum number of hidden nodes to evaluate, otherwise the maximum is set automatically.

`backend` Library used to train the networks: "neuralnet" (default) or "torch", which needs the torch package. See Details.

`model` A previously trained mlp object. If this is provided then the same model is fitted to y, without re-estimating any model parameters.

`retrain` If a previous model is provided, retrain the network or not.

`...` Settings for training, passed to the backend: arguments of `neuralnet` for `backend = "neuralnet"`, or the torch settings listed in Details. They are also used when searching for the number of hidden nodes.

Details

Training backends: Networks are trained with `neuralnet` by default. With `backend = "torch"` they are trained with torch, which supports modern activation functions and optimizers and custom architectures. torch is optional: install it with `install.packages("torch")` followed by `torch::install_torch()`. Use `compare_backends` to compare them on a series, and `nnfor_map` to see where training sits in the modelling steps.

torch settings: The torch backend accepts neuralnet's argument names, and by default trains as neuralnet does: logistic hidden layers, linear output, full-batch Rprop on the sum of squared errors, stopping when the largest absolute gradient falls below `threshold` or after `stepmax` steps. Settings, passed in `...`:

`act.fct` Hidden layer activation: `"logistic"` (default, also `"sigmoid"`), `"tanh"`, `"relu"`, `"leaky_relu"`, `"elu"`, `"selu"`, `"gelu"`, `"silu"` (also `"swish"`), `"mish"`, `"softplus"`, or a function of a torch tensor.

`algorithm` Optimizer: `"rprop"` (default; neuralnet's `"rprop+"` and `"rprop-"` are accepted), `"adam"`, `"adamw"`, `"sgd"` (also `"backprop"`), `"rmsprop"` or `"adagrad"`. L-BFGS is not offered: it couples the ensemble members, which are trained together.

`learningrate` Learning rate. Defaults: 0.1 as the initial step of Rprop (as neuralnet), 0.001 for Adam and AdamW, 0.01 otherwise.

`threshold`, `stepmax` Stopping rule, as in neuralnet (defaults 0.01 and 1e5).

`learningrate.limit`, `learningrate.factor` Rprop step limits and factors, as in neuralnet (defaults `list(min = 1e-10, max = 0.1)` and `list(minus = 0.5, plus = 1.2)`).

Popular choices for shallow networks: `"tanh"` is a smooth, zero-centred replacement for the logistic function; `"silu"` and `"gelu"` are smooth alternatives to `"relu"`, which with few hidden nodes can leave nodes inactive; Adam with a learning rate between 0.001 and 0.01 is the usual default optimizer, and Rprop remains a good choice for full-batch training of small networks.

Suggested torch settings: The defaults reproduce neuralnet's setup, and the function defaults are kept for both backends. For torch we suggest `act.fct = "gelu"`, `algorithm = "adam"`, `learningrate = 0.01`, `stepmax = 3000`. On eight standard series (AirPassengers, co2, nottem, USAccDeaths, UKgas, ldeaths, Nile, lynx; last season or 10 years held out, 10 networks per ensemble), this was the most consistent of the settings tried: its out-of-sample MSE was on average 6% below neuralnet's, lower on five series and at most 45% higher on the others, for a similar run time. `"relu"` and `"silu"` were less reliable, failing badly on some series, and Rprop with `"tanh"` was slow to converge. Results vary from series to series: `compare_backends` runs this kind of comparison on your data. The default torch setup (neuralnet's settings) was the fastest overall.

Custom architectures: With `backend = "torch"`, `hd` can be a function that receives the number of inputs and returns the hidden layers as a torch module. nnfor builds the inputs, adds a linear output layer, trains the ensemble and produces forecasts. The number of hidden nodes is then not searched, so `hd.auto.type` is ignored, and `act.fct` is not used: activations are part of the module. For example, two hidden layers with dropout:

```
arch <- function(n_inputs) {
  torch::nn_sequential(
    torch::nn_linear(n_inputs, 16), torch::nn_silu(), torch::nn_dropout(0.1),
    torch::nn_linear(16, 8), torch::nn_silu())
}
```

```
fit <- mlp(AirPassengers, hd = arch, backend = "torch", algorithm = "adam",
           learningrate = 0.01, stepmax = 3000)
```

or a module of your own, here with a skip connection:

```
skip_block <- torch::nn_module(
  initialize = function(n_inputs, width = 8) {
    self$hidden <- torch::nn_linear(n_inputs, width)
    self$skip <- torch::nn_linear(n_inputs, width)
  },
  forward = function(x) torch::torch_tanh(self$hidden(x)) + self$skip(x))
fit <- mlp(AirPassengers, hd = function(n) skip_block(n), backend = "torch",
           algorithm = "adam", learningrate = 0.01, stepmax = 2000)
```

Ensemble members of custom architectures share one optimizer and one backward pass, but run separate forward passes, so they are slower to train than standard networks.

Differences between the backends:

- The two backends do not produce the same networks: torch computes in single precision (float32), and its Rprop does not use neuralnet's weight backtracking.
- neuralnet discards ensemble members that do not converge within `stepmax`. torch keeps them, since with optimizers such as Adam the gradient may never fall below `threshold`; the number that converged is shown by `print`.
- Ensemble members of standard networks are trained simultaneously as one batch of tensors, which gives the same result as training them one by one. Training time therefore grows little with `reps`, while neuralnet's grows in proportion; `options(nnfor.cores)` is not used by torch.
- Results are reproducible with `set.seed()`: starting weights are drawn from R's random number generator, as neuralnet draws them, and torch's own generator is seeded from it.
- Networks trained with torch need torch to forecast, also when loaded from a saved file.
- Training runs on the CPU. GPUs offer little for networks of this size; support may be added in a later version.

Value

Return object of class `mlp`. The function `plot` produces a plot the network architecture or of the fit (see `plot.mlp`). `mlp` contains:

- `net`: MLP networks.
- `hd`: Number of hidden nodes.
- `lags`: Input lags used.
- `xreg.lags`: `xreg` lags used.
- `difforder`: Differencing used.
- `sdummy`: Use of deterministic seasonality.
- `ff`: Seasonal periods modelled (`m`, or those of `y`).
- `ff.det`: Seasonal frequencies coded using deterministic dummies.
- `det.type`: Type of determistic seasonality.

- y: Input time series.
- minmax: Scaling structure.
- xreg.minmax: Scaling structure for xreg variables.
- comb: Combination operator used.
- fitted: Fitted values.
- MSE: In-sample Mean Squared Error.
- MSEH: If hd.auto.type is set to either "valid" or "cv" an array of the MSE error for each network size is provided. Otherwise this is NULL.
- fitted.all: Fitted values of each ensemble member.
- backend: Name of the library used to train the networks: "neuralnet" or "torch".

Parallel training

Ensemble members can be trained in parallel with `options(nnfor.cores = 4)` (forked workers on Linux and macOS) or `options(nnfor.cores = cl)` with a cluster from `parallel::makeCluster()`, which is recommended on Windows. All random numbers are drawn before members are dispatched, so results are identical for any number of cores and reproducible with `set.seed()`.

Note

To use mlp with Temporal Hierarchies (thief package) see `mlp.thief`.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

References

- For an introduction to neural networks see: Ord K., Fildes R., Kourentzes N. (2017) Principles of Business Forecasting 2e. *Wessex Press Publishing Co.*, Chapter 10.
- For combination operators see: Kourentzes N., Barrow B.K., Crone S.F. (2014) Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, **41**(**9**), 4235-4244.
- For variable selection see: Crone S.F., Kourentzes N. (2010) Feature selection for time series prediction – A combined filter and wrapper approach for neural networks. *Neurocomputing*, **73**(**10**), 1923-1936.

See Also

`forecast.mlp`, `mlp.thief`, `elm`.

Examples

```
## Not run:
 fit <- mlp(AirPassengers)
 print(fit)
 plot(fit)
 plot(fit, which = "fit")
 frc <- forecast(fit,h=36)
 plot(frc)

## End(Not run)
## Not run:
# Data for the examples with exogenous regressors (xreg): a promotion
# indicator and a slowly moving price. xreg must cover the sample and the
# forecast horizon.
set.seed(1)
h <- 12
n <- length(AirPassengers)
z <- cbind(promo = rbinom(n + h, 1, 0.2),
           price = 50 + cumsum(rnorm(n + h)))
y <- AirPassengers + 30 * z[1:n, "promo"] - 2 * z[1:n, "price"]

# Regressors with automatic lags: each starts from the same candidate lags
# as y (here 1 to 12, so not lag 0, the current value of the regressor) and
# lag selection keeps the useful ones
fit <- mlp(y, xreg = z)
print(fit)
# Forecasting needs the future values of the regressors
frc <- forecast(fit, h = h, xreg = z)

# Lags for each regressor, one vector per column of xreg: promo at time t
# only (lag 0), price at lags 0 to 2
fit <- mlp(y, xreg = z, xreg.lags = list(0, 0:2))

# Keep some regressor lags in the network whatever the lag selection says:
# promo at lag 0, price at lag 0 (lags 1 and 2 may still be removed)
fit <- mlp(y, xreg = z, xreg.lags = list(0, 0:2),
           xreg.keep = list(TRUE, c(TRUE, FALSE, FALSE)))

# Use exactly the lags given, without lag selection
fit <- mlp(y, xreg = z, xreg.lags = list(0, 0:2), sel.lag = FALSE)

# Regressors as the only inputs: no lags of y (lags = 0)
fit <- mlp(y, xreg = z, lags = 0, xreg.lags = list(0, 0:2), sel.lag = FALSE)

# A single regressor: drop = FALSE keeps xreg a matrix
fit <- mlp(y, xreg = z[, "promo", drop = FALSE], xreg.lags = list(0))

# Lags of y: lags 1 to 12, with lags 1 and 12 always kept
fit <- mlp(AirPassengers, lags = 1:12, keep = c(TRUE, rep(FALSE, 10), TRUE))

# Number of hidden nodes (hd)
fit <- mlp(AirPassengers, hd = 10)          # one hidden layer of 10 nodes
```

```
fit <- mlp(AirPassengers, hd = c(8, 4))     # two hidden layers, of 8 and 4 nodes
fit <- mlp(AirPassengers, hd.auto.type = "valid",
           hd.max = 6)  # chosen on a 20% validation sample
fit$MSEH                                    # validation error for each number of nodes
fit <- mlp(AirPassengers, hd.auto.type = "cv",
           hd.max = 4, reps = 5)  # 5-fold cross-validation (slow)
fit <- mlp(AirPassengers, hd.auto.type = "elm")  # estimated with an ELM (experimental)
# With backend = "torch", hd can also define a custom architecture: see the
# torch examples below and Details.

# Save a fitted network and use it later
fit <- mlp(AirPassengers)
file <- file.path(tempdir(), "mlp_fit.rds")
saveRDS(fit, file)
fit <- readRDS(file)
frc <- forecast(fit, h = 12)

# Apply a network to new data without re-estimating it: a network fitted up
# to 1958 is used on the full series
fit.old <- mlp(window(AirPassengers, end = c(1958, 12)))
fit.new <- mlp(AirPassengers, model = fit.old)
# ...or keep its specification (lags, differencing, dummies, hidden nodes)
# and retrain the weights on the new data
fit.new <- mlp(AirPassengers, model = fit.old, retrain = TRUE)
frc <- forecast(fit.new, h = 12)

# A reused network with regressors needs the regressors of the new data
fit.old <- mlp(window(y, end = c(1958, 12)), xreg = z, xreg.lags = list(0, 0:2))
fit.new <- mlp(y, model = fit.old, xreg = z)
frc <- forecast(fit.new, h = h, xreg = z)

## End(Not run)

if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  # torch, with neuralnet's settings
  fit <- mlp(AirPassengers, reps = 5, backend = "torch")
  print(fit)
  # torch, with the suggested settings
  fit <- mlp(AirPassengers, reps = 5, backend = "torch", act.fct = "gelu",
             algorithm = "adam", learningrate = 0.01, stepmax = 3000)
  plot(forecast(fit, h = 24))
  # A custom architecture
  arch <- function(n_inputs) {
    torch::nn_sequential(torch::nn_linear(n_inputs, 16), torch::nn_silu(),
                         torch::nn_linear(16, 8), torch::nn_silu())
  }
  fit <- mlp(AirPassengers, reps = 5, hd = arch, backend = "torch",
             algorithm = "adam", learningrate = 0.01, stepmax = 1000)
  plot(fit)
}
```

`mlp.thief` *MLP network for THieF*

Description

Function for MLP forecasting with Temporal Hierarchies.

Usage

```
mlp.thief(y, h = NULL, ...)
```

Arguments

`y`	Input time series. Can be ts or msts object.
`h`	Forecast horizon. If NULL then h is set to match frequency of time series.
`...`	Additional arguments passed to `mlp`.

Value

An object of classes `"forecast.net"` and `"forecast"`. The function `plot` produces a plot of the forecasts. An object of class `"forecast.net"` is a list containing the following elements:

- `method`: The name of the forecasting method as a character string
- `mean`: Point forecasts as a time series
- `all.mean`: An array h x reps of all ensemble members forecasts, where reps are the number of ensemble members.
- `x`: The original time series (either `fit` used to create the network.
- `fitted`: Fitted values. Any values not fitted for the initial period of the time series are imputted with NA.
- `residuals`: Residuals from the fitted network.

Note

This function is created to work with Temporal Hierarchies (thief package). For conventional MLP networks use `mlp`.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

References

- For forecasting with temporal hierarchies see: Athanasopoulos G., Hyndman R.J., Kourentzes N., Petropoulos F. (2017) Forecasting with Temporal Hierarchies. *European Journal of Operational research*, **262**(**1**), 60-74.
- For combination operators see: Kourentzes N., Barrow B.K., Crone S.F. (2014) Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, **41**(**9**), 4235-4244.

See Also

mlp, elm.thief.

Examples

```
## Not run:
  library(thief)
  frc <- thief(AirPassengers,forecastfunction=mlp.thief)
  plot(frc)

## End(Not run)
```

---

nnfor_map *Map of how information flows through nnfor*

Description

Builds a map of the steps that mlp and elm go through when fitting a network, or that forecast goes through when producing forecasts. Each box is an internal or exported function of the package, showing what it does and what it passes on. Plot the map with plot, or print it for a text outline.

Usage

```
nnfor_map(x = c("mlp", "elm"), stage = c("fit", "forecast"))
```

Arguments

x	Either "mlp" or "elm", or a network fitted with mlp or elm.
stage	"fit" for model fitting or "forecast" for forecasting.

Details

When x is a fitted network, the map is annotated with the values used by that fit (seasonal periods, differencing, lags, dummies, hidden nodes, ...) and steps that the fit did not use are greyed out.

Value

An object of class nnfor_map: a list with nodes (one row per step: function, description, output, stage and the condition under which it runs) and edges (flows between steps), plus information used for plotting.

See Also

plot.nnfor_map, mlp, elm.

Examples

```
plot(nnfor_map("mlp"))
print(nnfor_map("elm", stage = "forecast"))
## Not run:
 fit <- mlp(AirPassengers)
 plot(nnfor_map(fit))
 plot(nnfor_map(fit, stage = "forecast"))

## End(Not run)
```

`plot.elm` *Plot ELM network*

Description

Produces a plot of the ELM network architecture, or of the in-sample fit.

Usage

```
## S3 method for class 'elm'
plot(x, r = 1, which = c("network", "fit"), ...)
```

Arguments

`x`	ELM network object, produced using `elm`.
`r`	Ensemble member to plot (architecture plot only).
`which`	What to plot: `"network"` for the architecture of member `r`, or `"fit"` for the in-sample fit.
`...`	Graphical arguments passed on. For `which = "network"` see `plot_network`; for `which = "fit"` see `plot_fit`.

Value

Invisibly returns `x`. Function produces a plot.

Note

Neurons are coloured with `"lightgrey"`. Seasonal dummies are coloured with `"lightpink"` and xreg with `"lightblue"`. Output connections of unused hidden nodes are dashed. All colours can be changed, see `plot_network`.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

See Also

`elm`, `mlp`, `plot_network`, `plot_fit`.

Examples

```
## Not run:
fit <- elm(AirPassengers, reps = 5, direct = TRUE)
plot(fit)              # unused hidden nodes are dashed, direct connections blue
plot(fit, r = 2)       # architecture of the second network
plot(fit, which = "fit")  # in-sample fit

## End(Not run)
```

`plot.elm.fast` *Plot ELM (fast) fit*

Description

Scatter plot of actual against fitted values of an `elm.fast` network.

Usage

```
## S3 method for class 'elm.fast'
plot(
  x,
  members = TRUE,
  members.args = list(),
  fitted.args = list(),
  line.args = list(),
  ...
)
```

Arguments

`x`	Object of class `elm.fast`.
`members`	Logical, draw the fitted values of each ensemble member.
`members.args`	List of graphical parameters for member points (default `list(col = "gray30", pch = ".")`).
`fitted.args`	List of graphical parameters for the ensemble points (default `list(pch = 20)`).
`line.args`	List of graphical parameters for the diagonal reference line (default `list(col = "grey")`); `NULL` to omit it.
`...`	Arguments passed to `plot` when creating the frame, overriding the defaults.

Value

Invisibly returns `x`.

See Also

`elm.fast`.

Examples

```
## Not run:
x <- matrix(rnorm(300), 100, 3)
y <- x %*% c(1, -1, 0.5) + rnorm(100, sd = 0.3)
fit <- elm.fast(y, x, reps = 5)
plot(fit)                                    # fitted against actual values
plot(fit, members = FALSE, line.args = NULL)  # ensemble only, no reference line

## End(Not run)
```

`plot.forecast.net` *Plot forecasts of a network*

Description

Plots the series, the ensemble forecast and, optionally, the forecasts of each ensemble member and the in-sample fit.

Usage

```
## S3 method for class 'forecast.net'
plot(
  x,
  members = TRUE,
  fitted = FALSE,
  add = FALSE,
  members.args = list(),
  mean.args = list(),
  fitted.args = list(),
  restrict = TRUE,
  ...
)
```

Arguments

`x`	Object of class `forecast.net`, produced by `forecast` on an `mlp` or `elm` fit.
`members`	Logical, draw the forecast of each ensemble member.
`fitted`	Logical, draw the in-sample fit.
`add`	Logical, add to the current plot instead of starting a new one. When `TRUE` the series itself is not redrawn.
`members.args`	List of graphical parameters for member forecasts (default `list(col = "grey")`).
`mean.args`	List of graphical parameters for the ensemble forecast (default `list(col = "blue", lwd = 2)`).
`fitted.args`	List of graphical parameters for the in-sample fit (default `list(col = "blue", lty = 2)`).

`restrict`	Logical. If `TRUE` (default), the vertical axis is limited to at most twice the range of the actuals and the ensemble forecast (that range, extended by half of it on each side), so that a poorly trained member, which the ensemble combination mostly ignores, cannot stretch the plot. Plots that already fit within that limit are unchanged. Use `FALSE` to show every member in full. Ignored when `add = TRUE` or when `ylim` is given.
`...`	Arguments passed to `plot` when drawing the series, overriding the defaults (e.g. `main`, `xlim`, `ylim`, `col`).

Value

Invisibly returns `x`.

See Also

`forecast.mlp`, `forecast.elm`.

Examples

```
## Not run:
fit <- mlp(AirPassengers, reps = 5)
frc <- forecast(fit, h = 24)
plot(frc)                                # forecasts of the ensemble and of each network
plot(frc, restrict = FALSE)           # no limit on the axis, to see every network in full
plot(frc, members = FALSE, fitted = TRUE)  # ensemble only, with the in-sample fit
plot(frc, xlim = c(1958, 1963), mean.args = list(col = "red"))  # zoom, custom style
# Add the forecasts to a plot of your own
plot(AirPassengers, xlim = c(1949, 1963), ylim = c(100, 750))
plot(frc, add = TRUE, members = FALSE)

## End(Not run)
```

---

plot.mlp *Plot MLP network*

Description

Produces a plot of the MLP network architecture, or of the in-sample fit.

Usage

```
## S3 method for class 'mlp'
plot(x, r = 1, which = c("network", "fit"), ...)
```

Arguments

x	MLP network object, produced using mlp.
r	Ensemble member to plot (architecture plot only).
which	What to plot: "network" for the architecture of member r, or "fit" for the in-sample fit.
...	Graphical arguments passed on. For which = "network" see plot_network; for which = "fit" see plot_fit.

Value

Invisibly returns x. Function produces a plot.

Note

Neurons are coloured with "lightgrey". Seasonal dummies are coloured with "lightpink" and xreg with "lightblue". All colours can be changed, see plot_network.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

See Also

elm, mlp, plot_network, plot_fit.

Examples

```
## Not run:
fit <- mlp(AirPassengers, reps = 5)
plot(fit)                                  # architecture of the first network
plot(fit, r = 3)                           # architecture of the third network
plot(fit, which = "fit")                   # in-sample fit: ensemble and each network
plot(fit, which = "fit", members = FALSE)  # ensemble fit only
# Further arguments customise the plot: see plot_network() and plot_fit()
plot(fit, main = "My network", col.neuron = "gold")
plot(fit, which = "fit", fitted.args = list(col = "red", lwd = 2))

## End(Not run)
```

plot_fit *Plot the in-sample fit of a network*

Description

Plots the series together with the ensemble fit and, optionally, the fit of each ensemble member. This is the plot produced by `outplot = TRUE` in `mlp` and `elm`, and by `plot(fit, which = "fit")`.

Usage

```
plot_fit(
  object,
  members = TRUE,
  add = FALSE,
  members.args = list(),
  fitted.args = list(),
  ...
)
```

Arguments

`object`	Network object, produced using `mlp` or `elm`.
`members`	Logical, draw the fit of each ensemble member.
`add`	Logical, add to the current plot instead of starting a new one. When `TRUE` the series itself is not redrawn.
`members.args`	List of graphical parameters for the member fits (default `list(col = "grey")`).
`fitted.args`	List of graphical parameters for the ensemble fit (default `list(col = "blue")`).
`...`	Arguments passed to `plot` when drawing the series, overriding the defaults (e.g. `main`, `ylim`, `col`, `lwd`).

Value

Invisibly returns `object`.

Note

Members are only available for models fitted with nnfor 0.9.9 or later.

See Also

`plot.mlp`, `plot.elm`.

Examples

```
## Not run:
fit <- mlp(AirPassengers, reps = 5)
# Style of the network fits (members) and of the ensemble fit
plot_fit(fit, members.args = list(col = "pink"), fitted.args = list(col = "red", lwd = 2))
# Zoom in with arguments for plot(), e.g. xlim
plot_fit(fit, xlim = c(1958, 1961))
# Add the fit to a plot of your own
plot(AirPassengers, col = "grey40", main = "Air passengers")
plot_fit(fit, add = TRUE, members = FALSE)

## End(Not run)
```

---

plot_network *Plot the architecture of a network*

Description

Draws the inputs, hidden layers and output of one ensemble member of an `mlp` or `elm` fit. Called by `plot(fit)`. The plot is drawn in the current plotting region, respecting any layout or graphical parameters already set by the user.

Usage

```
plot_network(
  x,
  r = 1,
  col.neuron = "lightgrey",
  col.xreg = "lightblue",
  col.season = "lightpink",
  col.connection = par("fg"),
  col.border = par("fg"),
  col.inactive = "grey",
  col.direct = "blue",
  arrow.length = 0.1,
  axis.args = list(),
  ...
)
```

Arguments

`x`	Network object, produced using `mlp` or `elm`.
`r`	Ensemble member to plot.
`col.neuron`	Fill colour of hidden and output neurons, and of lag inputs.
`col.xreg`	Fill colour of exogenous inputs.
`col.season`	Fill colour of seasonal dummy inputs.

col.connection	Colour of connections and arrows. Defaults to par("fg").
col.border	Border colour of neurons. Defaults to par("fg").
col.inactive	Colour of ELM output connections from unused hidden neurons.
col.direct	Colour of ELM direct input-output connections.
arrow.length	Length of the arrow heads, as in arrows.
axis.args	List of arguments passed to axis for the layer labels, e.g. list(cex.axis = 0.8). Use axis.args = NULL to omit the labels.
...	Arguments passed to plot when creating the frame, overriding the defaults (e.g. main, sub, cex.main).

Value

Invisibly returns x.

See Also

plot.mlp, plot.elm.

Examples

```
## Not run:
fit <- mlp(AirPassengers, reps = 1)
# Colours of the inputs: lags of y, regressors (xreg) and seasonal dummies
plot_network(fit, col.neuron = "white", col.xreg = "skyblue", col.season = "orange")
# Without the layer labels, and with a title of your own
plot_network(fit, axis.args = NULL, main = "")
# Graphical parameters set with par() are respected, e.g. a dark theme
op <- par(bg = "grey15", fg = "white", col.main = "white", col.axis = "white")
plot_network(fit)
par(op)

## End(Not run)
```

---

predict.elm.fast *Predictions for ELM (fast) network*

Description

Calculate predictions for ELM (fast) network.

Usage

```
## S3 method for class 'elm.fast'
predict(object, newx, na.rm = FALSE, ...)
```

Arguments

object	ELM network object, produced using elm.fast.
newx	Explanatory variables. Each column is a variable.
na.rm	If TRUE remove columns and object produces an ensemble forecast, then remove any members that give NA in their forecasts.
...	Unused argument.

Value

Returns a list with:

- Y.hat: Ensemble prediction.
- Y.all: Predictions of each training repetition.

Author(s)

Nikolaos Kourentzes, <nikolaos@kourentzes.com>

See Also

elm.fast.

Examples

```
## Not run:
 p <- 2000
 n <- 150
 X <- matrix(rnorm(p*n),nrow=n)
 b <- cbind(rnorm(p))
 Y <- X %*% b
 fit <- elm.fast(Y,X)
 predict(fit,X)

## End(Not run)
```

---

print.nnfor_map *Plot or print a map of nnfor*

Description

Draws the map produced by nnfor_map. Columns are stages, boxes are functions, with what they produce underneath. Solid arrows pass data, dotted arrows pass stored settings (such as the differencing needed to reverse a transformation) and dashed arrows are repeated steps. Dashed boxes are conditional steps; grey boxes were not used by the annotated fit.

Usage

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

## S3 method for class 'nnfor_map'
plot(
  x,
 col.stages = c("#0072B2", "#009E73", "#E69F00", "#D55E00", "#CC79A7", "#56B4E9",
    "#882255"),
  col.text = par("fg"),
  col.value = "#0072B2",
  col.inactive = "grey60",
  col.arrows = par("fg"),
  legend = TRUE,
  cex = NULL,
  cex.max = 1.2,
  ...
)
```

Arguments

x	Object of class nnfor_map.
...	Arguments passed to plot when creating the frame, overriding the defaults (e.g. main, cex.main).
col.stages	Colours for the stages (recycled).
col.text	Colour of the text. Defaults to par("fg").
col.value	Colour of values from an annotated fit.
col.inactive	Colour of steps not used by an annotated fit.
col.arrows	Colour of the arrows. Defaults to par("fg").
legend	Logical, add a legend.
cex	Size of the text in the boxes. NULL (default) chooses the largest size at which the text of every box fits, up to cex.max.
cex.max	Largest text size used when cex = NULL.

Details

The map is drawn in the current plotting region and respects graphical parameters set by the user. Text is sized to the plotting area: one size for all boxes, the largest at which every box fits, with descriptions re-wrapped to the box width. Set cex to fix the size instead. The map is best viewed on a wide device; in RStudio the plot pane can be small, so saving the map to a file, as in the examples, gives a better view.

Value

Invisibly returns x.

See Also

nnfor_map.

Examples

```
plot(nnfor_map("mlp"), main = "")
## Not run:
 # Save full-screen maps as images, outside the RStudio plot pane
 png("nnfor_map_mlp.png", width = 1920, height = 1080, res = 120)
 plot(nnfor_map("mlp"))
 dev.off()

 # A map annotated with a fitted network
 fit <- mlp(AirPassengers)
 png("nnfor_map_fit.png", width = 1920, height = 1080, res = 120)
 plot(nnfor_map(fit))
 dev.off()

 # Or as a PDF, which can be zoomed without losing quality
 pdf("nnfor_map_forecast.pdf", width = 16, height = 9)
 plot(nnfor_map(fit, stage = "forecast"))
 dev.off()

## End(Not run)
```

Index

∗ **elm**
elm.thief, 14
forecast.elm, 15
plot.elm, 28
predict.elm.fast, 35
∗ **mlp**
elm, 6
elm.fast, 11
forecast.mlp, 16
mlp, 19
mlp.thief, 26
plot.mlp, 31
∗ **package**
nnfor-package, 2
∗ **thief**
elm, 6
elm.fast, 11
elm.thief, 14
forecast.elm, 15
forecast.mlp, 16
mlp, 19
mlp.thief, 26
plot.elm, 28
plot.mlp, 31
∗ **ts**
elm, 6
elm.fast, 11
elm.thief, 14
forecast.elm, 15
forecast.mlp, 16
mlp, 19
mlp.thief, 26
nnfor-package, 2
plot.elm, 28
plot.mlp, 31

arrows, *35*
axis, *35*

compare_backends, *2, 3*, 4, *21*
elm, *2, 3*, 6, *13–17*, *23*, *27, 28*, *30*, *32–34*
elm.fast, *2, 3*, *9*, 11, *29*, *36*
elm.thief, *3*, *9, 10*, 14, *16*, *27*

forecast, *3*
forecast.elm, *3*, *10*, 15, *31*
forecast.mlp, *3*, 16, *23*, *31*

linscale, *3*, 18

mlp, *2–5*, *10*, *16, 17*, 19, *26–28*, *30*, *32–34*
mlp.thief, *3*, *15*, *17*, *23*, 26

neuralnet, *2*, *20, 21*
nnfor *(*nnfor-package*)*, 2
nnfor-package, 2
nnfor_map, *3*, *5*, *21*, 27, *36*, *38*

plot, *3*, *5*, *29*, *31*, *33*, *35*, *37*
plot.elm, *3*, *8*, 28, *33*, *35*
plot.elm.fast, *12*, 29
plot.forecast.net, 30
plot.mlp, *3*, *22*, 31, *33*, *35*
plot.nnfor_comparison
*(*compare_backends*)*, 4
plot.nnfor_map, *3*, *27*
plot.nnfor_map *(*print.nnfor_map*)*, 36
plot_fit, *3*, *28*, *32*, 33
plot_network, *3*, *28*, *32*, 34
predict, *3*
predict.elm.fast, 35
print.nnfor_comparison
*(*compare_backends*)*, 4
print.nnfor_map, 36