Package: ONAM (via r-universe)

November 3, 2026

Type Package

Title Fitting Interpretable Neural Additive Models Using Orthogonalization

Version 1.1.0

Description An algorithm for fitting interpretable additive neural networks for identifiable and visualizable feature effects using post hoc orthogonalization. Fit custom neural networks intuitively using established 'R' 'formula' notation, including interaction effects of arbitrary order while preserving identifiability to enable a functional decomposition of the prediction function. For more details see Koehler et al. (2025) <doi:10.1038/s44387-025-00033-7>.

License MIT + file LICENSE

BugReports https://github.com/Koehlibert/ONAM_R/issues

Depends keras3, reticulate

Imports dplyr, scales, rlang, ggplot2, pROC

Suggests akima, RColorBrewer, testthat (>= 3.0.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

NeedsCompilation no

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libpng-dev python3

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

Date/Publication 2026-11-03 15:57:06 UTC

RemoteUrl https://github.com/koehlibert/onam_r

RemoteRef HEAD

RemoteSha 1aab3f41f0be348e00b174efd22573f97b125b80

Contents

build_dnn . . . 2
gen_sobol . . . 3
install_conda_env . . . 4
load_onam . . . 5
onam . . . 6
plot.gen_sobol . . . 8
plot.var_decomp . . . 8
plot_inter_effect . . . 9
plot_main_effect . . . 10
predict.onam . . . 11
save_onam . . . 12
summary.onam . . . 12
var_decomp . . . 13

Index **15**

`build_dnn` *Build a stacked dense (DNN) model constructor*

Description

Returns a *function* `function(inputs) {...}` that builds a feed-forward stack of `layer_dense()` calls on top of the `inputs` tensor it is later called with, and returns the resulting `keras3::keras_model()`.

Usage

```
build_dnn(units = NULL, layer_df = NULL, default_activation = NULL)
```

Arguments

`units`	Optional integer vector of layer sizes. Ignored if `layer_df` is supplied. Every layer uses `default_activation` as activation function, with `activation = "gelu"` as the default. `use_bias = TRUE`.
`layer_df`	Optional data frame / tibble with one row per layer. Must contain a `units` column. May also contain any of: • `activation` (character) • `use_bias` (logical) • `kernel_initializer`, `bias_initializer` (character, or a list-column of initializer objects, e.g. `keras3::initializer_he_normal()`) • `kernel_regularizer`, `bias_regularizer`, `activity_regularizer` (list-column; each element is either `NULL` or an object like `keras3::regularizer_l2(0.01)`) • `name` (character, layer name)

Any column not supplied falls back to `layer_dense()`'s own default. Columns holding actual keras objects (regularizers, and initializers if you don't just want a string) must be *list-columns* so each row can carry a different object or NULL, e.g.:

```
tibble::tibble(
  units = c(64, 32, 1),
  activation = c("gelu", "gelu", "linear"),
 kernel_regularizer = list(keras3::regularizer_l2(0.001), NULL, NULL)
)
```

`default_activation`
: Activation function to be used if not otherwise specified in `layer_df`.

Details

Layers can be specified either as a simple vector of unit counts, or as a data frame / tibble giving full per-layer control over activation, use_bias, initializers, regularizers, layer name, etc.

Value

A function of a single argument `inputs` (a keras input tensor) that returns a `keras3::keras_model()`.

Examples

```
mod_units <- build_dnn(units = c(16, 8, 1))
mod_df <- build_dnn(layer_df = data.frame(
  units = c(32, 16, 8, 1),
  activation = c("gelu", "gelu", "gelu", "linear"),
  use_bias = c(FALSE, FALSE, FALSE, TRUE)
))
```

---

gen_sobol *Compute Generalized Sobol Indices for dependent features*

Description

Compute Generalized Sobol Indices for dependent features

Usage

```
gen_sobol(object, data = NULL)
```

Arguments

`object`	Either model of class `onam` as returned from onam or evaluation outcome as returned from predict.onam.
`data`	Data for which the model is to be evaluated. If `NULL` (DEFAULT), the data from model fitting is used.

Details

For details on generalized sobol indices, see Chastaing et al. (2012) doi:10.1214/12EJS749.

Value

Returns a named vector of percentage of variance explained by each interaction order.

Examples

```
# Basic example for a simple ONAM-model
# Create training data
n <- 1000
x1 <- runif(n, -2, 2)
x2 <- runif(n, -2, 2)
y <- sin(x1) + ifelse(x2 > 0, pweibull(x2, shape = 3),
  pweibull(-x2, shape = 0.5)) +
  x1 * x2
data_train <- cbind(x1, x2, y)
# Define model
model_formula <- y ~ mod1(x1) + mod1(x2) +
  mod1(x1, x2)
mod1 <-  build_dnn(units = c(16, 8, 1))
list_of_deep_models <- list(mod1 = mod1)
# Fit model
mod <- onam(model_formula, list_of_deep_models,
            data_train, n_ensemble = 1, epochs = 10)
gen_sobol(mod)
```

---

install_conda_env *Set up conda environment for keras functionality*

Description

Helper function to install Keras and packages necessary for package functionality into a conda environment. Use this function if `keras3::install_keras()` does not work, esp. on windows machines.

Usage

```
install_conda_env(
  envname = "r-keras",
  python_version = "python=3.10",
  overwrite = FALSE
)
```

Arguments

envname	Name for the conda environment to be created.
python_version	Python version to be installed in the conda environment.
overwrite	Should an existing conda environment of name envname be overwritten if present?

Value

No return value, called for side effects

See Also

keras3::install_keras()

---

load_onam *Load a fitted onam model from disk*

Description

Load a fitted onam model from disk

Usage

```
load_onam(dir)
```

Arguments

dir	Directory as created by save_onam.

Value

Returns the restored object of class onam.

See Also

save_onam

onam *Fit orthogonal neural additive model*

Description

Fits an interpretable neural additive model with post hoc orthogonalization for a given network architecture and user-specified feature sets.

Usage

```
onam(
  formula,
  list_of_deep_models,
  data,
  model = NULL,
  prediction_function = NULL,
  model_data = NULL,
  categorical_features = NULL,
  target = "continuous",
  n_ensemble = 10,
  epochs = 500,
  learning_rate = 0.001,
  callback = NULL,
  seed = NULL,
  progresstext = FALSE,
  verbose = 0
)
```

Arguments

`formula`	Formula for model fitting. Specify deep parts with the same name as `list_of_deep_models`.
`list_of_deep_models`	List of named models used in `model_formula`. Simple fully connected model architectures can be created using `build_dnn`. When specifying a custom architecture, architectures must accept both single inputs and input lists (see example).
`data`	Data to be fitted
`model`	Prediction model that is to be explained. Output of the model as returned from `prediction_function(model)` will be used as model output. If `NULL`(default), the outcome has to be present in `data`.
`prediction_function`	Prediction function to be used to generate the outcome. Only used if `model` is specified. If `NULL`(default), S3-method based on the `model` argument is used.
`model_data`	Data used for generating predictions of `model`. Necessary for some models that require specific data formats, i.e. xgboost. If `NULL`(default), `data` is used. Only used if `model` is specified.

categorical_features	Vector of feature names of categorical features.
target	Target of prediction task. Can be either "continuous" or "binary". For "continuous"(default), an additive model for the prediction of a continuous outcome is fitted. For "binary", a binary classification with sigmoid activation in the last layer is fitted.
n_ensemble	Number of orthogonal neural additive model ensembles
epochs	Number of epochs to train the model. See fit for details.
learning_rate	Learning rate for model fitting. See compile for details.
callback	Callback to be called during training. See fit for details.
seed	Random seed used by R, python, numpy, and backend framework. See set_random_seed for details.
progresstext	Show model fitting progress. If TRUE, shows current number of ensemble being fitted
verbose	Verbose argument for internal model fitting. Used for debugging. See fit for details.

Details

For more details see Koehler et al. (2025) doi:10.1038/s44387-025-00033-7.

Value

Returns a model object of class onam, containing all ensemble members, ensemble weights, and main and interaction effect outputs.

Examples

```
# Basic example for a simple ONAM-model
# Create training data
n <- 1000
x1 <- runif(n, -2, 2)
x2 <- runif(n, -2, 2)
y <- sin(x1) + ifelse(x2 > 0, pweibull(x2, shape = 3),
  pweibull(-x2, shape = 0.5)) +
  x1 * x2
data_train <- cbind(x1, x2, y)
# Define model
model_formula <- y ~ mod1(x1) + mod1(x2) +
  mod1(x1, x2)
mod1 <- function(inputs) {
  # `inputs` is a list of shared feature inputs for interaction terms
  combined <- if (is.list(inputs)) layer_concatenate(inputs) else inputs
  outputs <- combined %>%
    layer_dense(units = 16, activation = "gelu") %>%
 layer_dense(units = 8, activation = "linear",
             use_bias = TRUE) %>%
 layer_dense(units = 1, activation = "linear",
             use_bias = TRUE)
```

```
 keras_model(inputs, outputs)
}
list_of_deep_models <- list(mod1 = mod1)
# Fit model
mod <- onam(model_formula, list_of_deep_models,
            data_train, n_ensemble = 1, epochs = 10)
summary(mod)
```

`plot.gen_sobol` *Plot generalized Sobol indices*

Description

Plot generalized Sobol indices

Usage

```
## S3 method for class 'gen_sobol'
plot(x, ...)
```

Arguments

`x`	Object of class `gen_sobol` as returned by gen_sobol
`...`	further arguments, currently unused

Details

For details on generalized sobol indices, see Chastaing et al. (2012) doi:10.1214/12EJS749.

Value

Returns a 'ggplot2' object showing, for each fitted effect, the generalized Sobol index split into the effect's own contribution (`gen_sobol_index_1`) and its contribution through interactions with other effects of the same order (`gen_sobol_index_2`)

`plot.var_decomp` *Plot variance decomposition*

Description

Plot variance decomposition

Usage

```
## S3 method for class 'var_decomp'
plot(x, ...)
```

Arguments

x	Object of class var_decomp as returned by var_decomp
...	further arguments, currently unused

Value

Returns a 'ggplot2' object showing the fraction of total variance explained by each interaction order

---

plot_inter_effect *Plot Interaction Effect*

Description

Plot Interaction Effect

Usage

```
plot_inter_effect(
  object,
  feature1,
  feature2,
  interpolate = FALSE,
  labs = NULL,
  custom_colors = "spectral",
  n_interpolate = 200,
  include_main = FALSE
)
```

Arguments

object	Either model of class onam as returned from onam or model evaluation outcome as returned from predict.onam
feature1, feature2	Effects to be plotted.
interpolate	If TRUE, values will be interpolated for a smooth plot. If FALSE (default), only observations in the data will be plotted.
labs	An optional named vector that can contain axis labels and the name of the effect. Expected vector names are 'xlab', 'ylab' and 'effect'.
custom_colors	color palette object for the interaction plot. Default is "spectral", returning a color palette based on the spectral theme.
n_interpolate	number of values per coordinate axis to interpolate. Ignored if 'interpolate = FALSE'.
include_main	If TRUE, main effects for features feature1 and feature2 will be added to the interaction term to give combined effect of main and interaction effects. Default is FALSE.

Value

Returns a 'ggplot2' object of the specified effect interaction

Examples

```
# Basic example for a simple ONAM-model
# Create training data
n <- 1000
x1 <- runif(n, -2, 2)
x2 <- runif(n, -2, 2)
y <- sin(x1) + ifelse(x2 > 0, pweibull(x2, shape = 3),
  pweibull(-x2, shape = 0.5)) +
  x1 * x2
data_train <- cbind(x1, x2, y)
# Define model
model_formula <- y ~ mod1(x1) + mod1(x2) +
  mod1(x1, x2)
mod1 <-  build_dnn(units = c(16, 8, 1))
list_of_deep_models <- list(mod1 = mod1)
# Fit model
mod <- onam(model_formula, list_of_deep_models,
            data_train, n_ensemble = 1, epochs = 10)
plot_inter_effect(mod, "x1", "x2")
```

---

plot_main_effect *Plot Main Effect*

Description

Plot Main Effect

Usage

```
plot_main_effect(object, feature, reference_level = NULL, labs = NULL)
```

Arguments

object	Either model of class onam as returned from onam or model evaluation outcome as returned from predict.onam
feature	Feature for which the effect is to be plotted, must be present in the model formula. For interaction terms, use plotInteractionEffect
reference_level	Reference level to be used when plotting categorical effects for better interpretability. Effect of the reference level will be set to zero and subtracted from all over factor levels.
labs	An optional named vector that can contain axis labels. Expected vector names are 'xlab', 'ylab' and 'effect'. If both 'ylab' and 'effect' are specified, 'effect' will be used as y-axis label.

Value

Returns a ggplot2 object of the specified effect

Examples

```
# Basic example for a simple ONAM-model
# Create training data
n <- 1000
x1 <- runif(n, -2, 2)
x2 <- runif(n, -2, 2)
y <- sin(x1) + ifelse(x2 > 0, pweibull(x2, shape = 3),
  pweibull(-x2, shape = 0.5)) +
  x1 * x2
data_train <- cbind(x1, x2, y)
# Define model
model_formula <- y ~ mod1(x1) + mod1(x2) +
  mod1(x1, x2)
mod1 <-  build_dnn(units = c(16, 8, 1))
list_of_deep_models <- list(mod1 = mod1)
# Fit model
mod <- onam(model_formula, list_of_deep_models,
            data_train, n_ensemble = 1, epochs = 10)
plot_main_effect(mod, "x1")
```

`predict.onam` *Evaluate orthogonal neural additive model*

Description

Evaluate orthogonal neural additive model

Usage

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

Arguments

`object`	model of class `onam` as returned from onam to be evaluated
`newdata`	Data for which the model is to be evaluated. If NULL (default), data with which `model` was fitted is used.
`...`	some methods for this generic require additional arguments. None are used in this method.

Value

Returns a list containing data, model output for each observation in `newdata` and main and interaction effects obtained by the model

save_onam *Save a fitted onam model to disk*

Description

An onam object holds one or more fitted 'keras' submodels per ensemble member (in `object$ensemble[[i]]$model_list`), which cannot be serialized with `saveRDS()` alone. This function saves each submodel to its own `.keras` file and the remaining orthogonalization metadata (weights, `model_info`, data, predictions, ...) to a single `.rds` file, all inside `dir`. Use load_onam to restore the object.

Usage

```
save_onam(object, dir, overwrite = FALSE)
```

Arguments

`object`	Object of class onam as returned by onam.
`dir`	Directory the model is saved to. Created if it does not exist.
`overwrite`	Should an existing directory of the same name be overwritten?

Value

No return value, called for side effects.

See Also

load_onam

summary.onam *Get summary of an onam object*

Description

generates a summary of a fitted onam object including information on ensembling strategy and performance metrics such as correlation and degree of interpretability

Usage

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

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

Arguments

object	onam object of class onam as returned from onam to be summarized
...	further arguments passed to or from other methods.
x	object of class summary.onam.

Details

For examples see example(onam)

Value

Gives summary of the onam object, including model inputs, number of ensembles, correlation of model output and original outcome variable, and interpretability metrics i_1 and i_2

---

var_decomp *Get variance decomposition of orthogonal neural additive model*

Description

Get variance decomposition of orthogonal neural additive model

Usage

```
var_decomp(object, data = NULL)
```

Arguments

object	Either model of class onam as returned from onam or model evaluation outcome as returned from predict.onam
data	Data for which the model is to be evaluated. If NULL (DEFAULT), the data from model fitting is used.

Value

Returns a named vector of percentage of variance explained by each interaction order.

Examples

```
# Basic example for a simple ONAM-model
# Create training data
n <- 1000
x1 <- runif(n, -2, 2)
x2 <- runif(n, -2, 2)
y <- sin(x1) + ifelse(x2 > 0, pweibull(x2, shape = 3),
  pweibull(-x2, shape = 0.5)) +
  x1 * x2
```

```
data_train <- cbind(x1, x2, y)
# Define model
model_formula <- y ~ mod1(x1) + mod1(x2) +
  mod1(x1, x2)
mod1 <-  build_dnn(units = c(16, 8, 1))
list_of_deep_models <- list(mod1 = mod1)
# Fit model
mod <- onam(model_formula, list_of_deep_models,
            data_train, n_ensemble = 1, epochs = 10)
var_decomp(mod)
```

Index

build_dnn, 2, *6*

compile, *7*

example(onam), *13*

fit, *7*

gen_sobol, 3, *8*

install_conda_env, 4

keras3::install_keras(), *4, 5*

load_onam, 5, *12*

onam, *3*, 6, *9–13*

plot.gen_sobol, 8
plot.var_decomp, 8
plot_inter_effect, 9
plot_main_effect, 10
predict *(*predict.onam*)*, 11
predict.onam, *3*, *9, 10*, 11, *13*
print.summary.onam *(*summary.onam*)*, 12

save_onam, *5*, 12
set_random_seed, *7*
summary.onam, 12, *13*

var_decomp, *9*, 13