

Package ‘deepregression’

September 6, 2025

Title Fitting Deep Distributional Regression

Version 2.3.2

Description Allows for the specification of semi-structured deep distributional regression models which are fitted in a neural network as proposed by Ruegamer et al. (2023) <[doi:10.18637/jss.v105.i02](https://doi.org/10.18637/jss.v105.i02)>. Predictors can be modeled using structured (penalized) linear effects, structured non-linear effects or using an unstructured deep network model.

Config/reticulate `list(packages = list(list(package = ``six", pip = TRUE), list(package = ``tensorflow", version = ``2.15", pip = TRUE), list(package = ``tensorflow_probability", version = ``0.23", pip = TRUE), list(package = ``keras", version = ``2.15", pip = TRUE)))`

Depends R (>= 4.0.0), tensorflow (>= 2.2.0), tfprobability, keras (>= 2.2.0)

Suggests testthat, knitr, covr

Imports mgcv, dplyr, R6, reticulate (>= 1.14), Matrix, magrittr, tfruns, methods, coro (>= 1.0.3), torchvision (>= 0.5.1), luz (>= 0.4.0), torch

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author David Ruegamer [aut, cre],
Christopher Marquardt [ctb],
Laetitia Frost [ctb],
Florian Pfisterer [ctb],
Philipp Baumann [ctb],
Chris Kolb [ctb],
Lucas Kook [ctb]

Maintainer David Ruegamer <david.ruegamer@gmail.com>

Repository CRAN

Date/Publication 2025-09-06 05:12:00 UTC

Contents

check_and_install	4
check_input_args_fit	5
choose_kernel_initializer_torch	5
coef.drEnsemble	6
collect_distribution_parameters	6
combine_penalties	7
create_family	7
create_family_torch	8
create_penalty	8
cv	9
deepregression	9
distfun_to_dist	12
ensemble	12
ensemble.deepregression	13
extractval	14
extractvar	15
extract_pure_gam_part	15
extract_S	16
family_to_tfd	16
family_to_trafo	17
family_to_trafo_torch	17
family_to_trochd	18
fitted.drEnsemble	18
form_control	19
from_distfun_to_dist_torch	19
from_dist_to_loss	20
from_dist_to_loss_torch	20
from_preds_to_dist	21
from_preds_to_dist_torch	22
gam_plot_data	23
get_distribution	23
get_ensemble_distribution	24
get_gamdata	24
get_gamdata_reduced_nr	25
get_gam_part	25
get_help_forward_torch	26
get_layernr_by_opname	26
get_layernr_trainable	27
get_layer_by_opname	27
get_luz_dataset	28
get_names_pfc	28
get_nodedata	29
get_node_term	29
get_partial_effect	30
get_processor_name	30
get_special	31

get_type_pfc	31
get_weight_by_name	32
get_weight_by_opname	32
handle_gam_term	33
import_packages	33
import_tf_dependings	34
import_torch_dependings	34
keras_dr	34
layer_add_identity	36
layer_dense_module	37
layer_dense_torch	37
layer_generator	38
layer_node	40
layer_sparse_batch_normalization	41
layer_sparse_conv_2d	41
layer_spline	42
layer_spline_torch	43
log_score	43
loop_through_pfc_and_call_trafo	44
makeInputs	45
makelayername	45
make_folds	46
make_generator	46
make_generator_from_matrix	47
make_tfd_dist	48
model_torch	50
multioptimizer	50
names_families	51
na_omit_list	51
nn_init_no_grad_constant_deepreg	52
orthog_control	52
orthog_P	53
orthog_post_fitting	54
orthog_structured_smooths_Z	54
penalty_control	55
plot.deepregression	56
plot_cv	59
precalc_gam	59
predict_gam_handler	60
predict_gen	60
prepare_data	61
prepare_data_torch	62
prepare_input_list_model	62
prepare_newdata	63
prepare_torch_distr_mixdistr	64
process_terms	64
quant	65
reinit_weights	66

reinit_weights.deepregression	66
re_layer	67
separate_define_relation	67
simplyconnected_layer_torch	68
stddev	69
stop_iter_cv_result	70
subnetwork_init	70
subnetwork_init_torch	71
tfd_mse	72
tfd_zinb	73
tfd_zip	73
tf_repeat	73
tf_row_tensor	74
tf_split_multiple	74
tf_stride_cols	75
tf_stride_last_dim_tensor	75
tib_layer	76
torch_dr	77
update_miniconda_deepregression	78
weight_control	78
Index	80

check_and_install	<i>Function to check python environment and install necessary packages</i>
-------------------	--

Description

If you encounter problems with installing the required python modules please make sure, that a correct python version is configured using py_discover_config and change the python version if required. Internally uses keras::install_keras.

Usage

```
check_and_install(force = FALSE, engine)
```

Arguments

- force if TRUE, forces the installations
- engine character; check if tf(= tensorflow) or torch is available

Value

Function that checks if a Python environment is available and contains TensorFlow. If not the recommended version is installed.

check_input_args_fit *Function to check if inputs are supported by corresponding fit function*

Description

Function to check if inputs are supported by corresponding fit function

Usage

```
check_input_args_fit(args, fit_fun)
```

Arguments

args	list; list of arguments used in fit process
fit_fun	used fit function (e.g. fit.keras.engine.training.Model)

Value

stop message if inputs are not supported

choose_kernel_initializer_torch
Function to choose a kernel initializer for a torch layer

Description

Function to choose a kernel initializer for a torch layer

Usage

```
choose_kernel_initializer_torch(kernel_initializer, value = NULL)
```

Arguments

kernel_initializer	string; initializer
value	numeric; value used for a constant initializer

Value

kernel initializer

coef.drEnsemble	<i>Method for extracting ensemble coefficient estimates</i>
-----------------	---

Description

Method for extracting ensemble coefficient estimates

Usage

```
## S3 method for class 'drEnsemble'
coef(object, which_param = 1, type = NULL, ...)
```

Arguments

object	object of class "drEnsemble"
which_param	integer, indicating for which distribution parameter coefficients should be returned (default is first parameter)
type	either NULL (all types of coefficients are returned), "linear" for linear coefficients or "smooth" for coefficients of smooth terms
...	further arguments supplied to coef.deepregression

Value

list of coefficient estimates of all ensemble members

collect_distribution_parameters	<i>Character-to-parameter collection function needed for mixture of same distribution (torch)</i>
---------------------------------	---

Description

Character-to-parameter collection function needed for mixture of same distribution (torch)

Usage

```
collect_distribution_parameters(family)
```

Arguments

family	character defining the distribution
--------	-------------------------------------

Value

a list of extractions for each supported distribution

combine_penalties	<i>Function to combine two penalties</i>
-------------------	--

Description

Function to combine two penalties

Usage

```
combine_penalties(penalties, dims)
```

Arguments

penalties	a list of penalties
dims	dimensions of the parameters to penalize

Value

a TensorFlow penalty combining the two penalties

create_family	<i>Function to create (custom) family</i>
---------------	---

Description

Function to create (custom) family

Usage

```
create_family(tfd_dist, trafo_list, output_dim = 1L)
```

Arguments

tfd_dist	a tensorflow probability distribution
trafo_list	list of transformations h for each parameter (e.g, exp for a variance parameter)
output_dim	integer defining the size of the response

Value

a function that can be used by `tfp$layers$DistributionLambda` to create a new distribuional layer

create_family_torch	<i>Function to create (custom) family</i>
---------------------	---

Description

Function to create (custom) family

Usage

```
create_family_torch(torch_dist, trafo_list, output_dim = 1L)
```

Arguments

torch_dist	a torch probability distribution
trafo_list	list of transformations h for each parameter (e.g, exp for a variance parameter)
output_dim	integer defining the size of the response

Value

a function that can be used to train a distribution learning model in torch

create_penalty	<i>Function to create mgcv-type penalty</i>
----------------	---

Description

Function to create mgcv-type penalty

Usage

```
create_penalty(evaluated_gam_term, df, controls, Z = NULL)
```

Arguments

evaluated_gam_term	a list resulting from a smoothConstruct call
df	integer; specified degrees-of-freedom for the gam term
controls	list; further arguments defining the smooth
Z	matrix; matrix for constraint(s)

Value

a list with penalty parameter and penalty matrix

cv	<i>Generic cv function</i>
----	----------------------------

Description

Generic cv function

Usage

```
cv(x, ...)
```

Arguments

x	model to do cv on
...	further arguments passed to the class-specific function

deepregression	<i>Fitting Semi-Structured Deep Distributional Regression</i>
----------------	---

Description

Fitting Semi-Structured Deep Distributional Regression

Usage

```
deepregression(
  y,
  list_of_formulas,
  list_of_deep_models = NULL,
  family = "normal",
  data,
  seed = as.integer(1991 - 5 - 4),
  return_prepoc = FALSE,
  subnetwork_builder = NULL,
  model_builder = NULL,
  fitting_function = NULL,
  additional_processors = list(),
  penalty_options = penalty_control(),
  orthog_options = orthog_control(),
  weight_options = weight_control(),
  formula_options = form_control(),
  output_dim = 1L,
  verbose = FALSE,
  engine = "tf",
  ...
)
```

Arguments

<code>y</code>	response variable
<code>list_of_formulas</code>	a named list of right hand side formulas, one for each parameter of the distribution specified in <code>family</code> ; set to <code>~ 1</code> if the parameter should be treated as constant. Use the <code>s()</code> -notation from <code>mgcv</code> for specification of non-linear structured effects and <code>d(...)</code> for deep learning predictors (predictors in brackets are separated by commas), where <code>d</code> can be replaced by an name name of the names in <code>list_of_deep_models</code> , e.g., <code>~ 1 + s(x) + my_deep_mod(a,b,c)</code> , where <code>my_deep_mod</code> is the name of the neural net specified in <code>list_of_deep_models</code> and <code>a,b,c</code> are features modeled via this network.
<code>list_of_deep_models</code>	a named list of functions specifying a keras model. See the examples for more details.
<code>family</code>	a character specifying the distribution. For information on possible distribution and parameters, see make_tfd_dist . Can also be a custom distribution.
<code>data</code>	data.frame or named list with input features
<code>seed</code>	a seed for TensorFlow or Torch (only works with R version $\geq 2.2.0$)
<code>return_prepoc</code>	logical; if TRUE only the pre-processed data and layers are returned (default FALSE).
<code>subnetwork_builder</code>	function to build each subnetwork (network for each distribution parameter; per default NULL). subnetwork builder will be chosen depending on the engine. Can also be a list of the same size as <code>list_of_formulas</code> .
<code>model_builder</code>	function to build the model based on additive predictors (per default NULL). model builder will be chosen depending on the engine. In order to work with the methods defined for the class <code>deepregression</code> , the model should behave like a keras model
<code>fitting_function</code>	function to fit the instantiated model when calling <code>fit</code> . Per default the keras NULL function. <code>fit</code> will be chosen depending on the engine.
<code>additional_processors</code>	a named list with additional processors to convert the formula(s). Can have an attribute "controls" to pass additional controls
<code>penalty_options</code>	options for smoothing and penalty terms defined by penalty_control
<code>orthog_options</code>	options for the orthogonalization defined by orthog_control
<code>weight_options</code>	options for layer weights defined by weight_control
<code>formula_options</code>	options for formula parsing (mainly used to make calculation more efficiently)
<code>output_dim</code>	dimension of the output, per default 1L
<code>verbose</code>	logical; whether to print progress of model initialization to console
<code>engine</code>	character; the engine which is used to setup the NN (tf or torch)
<code>...</code>	further arguments passed to the <code>model_builder</code> function

References

Ruegamer, D. et al. (2023): deepregression: a Flexible Neural Network Framework for Semi-Structured Deep Distributional Regression. [doi:10.18637/jss.v105.i02](https://doi.org/10.18637/jss.v105.i02).

Examples

```
library(deepregression)

n <- 1000
data = data.frame(matrix(rnorm(4*n), c(n,4)))
colnames(data) <- c("x1", "x2", "x3", "xa")
formula <- ~ 1 + deep_model(x1,x2) + s(xa) + x1 +
  node(x3, n_trees = 2, n_layers = 2, tree_depth = 1)

deep_model <- function(x) x %>%
  layer_dense(units = 32, activation = "relu", use_bias = FALSE) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 8, activation = "relu") %>%
  layer_dense(units = 1, activation = "linear")

y <- rnorm(n) + data$xa^2 + data$x1

mod <- deepregression(
  list_of_formulas = list(loc = formula, scale = ~ 1),
  data = data, y = y,
  list_of_deep_models = list(deep_model = deep_model)
)

if(!is.null(mod)){
  # train for more than 10 epochs to get a better model
  mod %>% fit(epochs = 10, early_stopping = TRUE)
  mod %>% fitted() %>% head()
  cvres <- mod %>% cv()
  mod %>% get_partial_effect(name = "s(xa)")
  mod %>% coef()
  mod %>% plot()
}

mod <- deepregression(
  list_of_formulas = list(loc = ~ 1 + s(xa) + x1, scale = ~ 1,
    dummy = ~ -1 + deep_model(x1,x2,x3) %OZ% 1),
  data = data, y = y,
  list_of_deep_models = list(deep_model = deep_model),
  mapping = list(1,2,1:2)
)
```

distfun_to_dist	<i>Function to define output distribution based on dist_fun</i>
-----------------	---

Description

Function to define output distribution based on dist_fun

Usage

```
distfun_to_dist(dist_fun, preds)
```

Arguments

dist_fun	a distribution function as defined by make_tfd_dist
preds	tensors with predictions

Value

a symbolic tfp distribution

ensemble	<i>Generic deep ensemble function</i>
----------	---------------------------------------

Description

Generic deep ensemble function

Usage

```
ensemble(x, ...)
```

Arguments

x	model to ensemble
...	further arguments passed to the class-specific function

ensemble.deepregression

Ensembling deepregression models

Description

Ensembling deepregression models

Usage

```
## S3 method for class 'deepregression'
ensemble(
  x,
  n_ensemble = 5,
  reinitialize = TRUE,
  mylapply = lapply,
  verbose = FALSE,
  patience = 20,
  plot = TRUE,
  print_members = TRUE,
  stop_if_nan = TRUE,
  save_weights = TRUE,
  callbacks = list(),
  save_fun = NULL,
  seed = seq_len(n_ensemble),
  ...
)
```

Arguments

x	object of class "deepregression" to ensemble
n_ensemble	numeric; number of ensemble members to fit
reinitialize	logical; if TRUE (default), model weights are initialized randomly prior to fitting each member. Fixed weights are not affected
mylapply	lapply function to be used; defaults to lapply
verbose	whether to print training in each fold
patience	number of patience for early stopping
plot	whether to plot the resulting losses in each fold
print_members	logical; print results for each member
stop_if_nan	logical; whether to stop CV if NaN values occur
save_weights	whether to save final weights of each ensemble member; defaults to TRUE
callbacks	a list of callbacks used for fitting
save_fun	function applied to the model in each fold to be stored in the final result
seed	seed for reproducibility
...	further arguments passed to object\$fit_fun

Value

object of class "drEnsemble", containing the original "deepregression" model together with a list of ensembling results (training history and, if save_weights is TRUE, the trained weights of each ensemble member)

extractval	<i>Formula helpers</i>
------------	------------------------

Description

Formula helpers

Extractval with multiple options

Usage

```
extractval(term, name, default_for_missing = FALSE, default = NULL)
```

```
extractvals(term, names)
```

```
extractlen(term, data)
```

```
form2text(form)
```

Arguments

term	formula term
name	character; the value to extract
default_for_missing	logical; if TRUE, returns default if argument is missing
default	value returned when missing
names	character vector of names
data	a data.frame or list
form	formula that is converted to a character string

Value

the value used for name

Examples

```
extractval("s(a, la = 2)", "la")
```

extractvar	<i>Extract variable from term</i>
------------	-----------------------------------

Description

Extract variable from term

Usage

```
extractvar(term, allow_ia = FALSE)
```

Arguments

term	term specified in formula
allow_ia	logical; whether to allow interaction of terms using the : notation

Value

variable as string

extract_pure_gam_part	<i>Extract the smooth term from a deepregression term specification</i>
-----------------------	---

Description

Extract the smooth term from a deepregression term specification

Usage

```
extract_pure_gam_part(term, remove_other_options = TRUE)
```

Arguments

term	term specified in a formula
remove_other_options	logical; whether to remove other options withing the smooth term

Value

pure gam part of term

extract_S	<i>Convenience function to extract penalty matrix and value</i>
-----------	---

Description

Convenience function to extract penalty matrix and value

Usage

```
extract_S(x)
```

Arguments

x	evaluated smooth term object
---	------------------------------

family_to_tfd	<i>Character-tfd mapping function</i>
---------------	---------------------------------------

Description

Character-tfd mapping function

Usage

```
family_to_tfd(family)
```

Arguments

family	character defining the distribution
--------	-------------------------------------

Value

a tfp distribution

family_to_trafo	<i>Character-to-transformation mapping function</i>
-----------------	---

Description

Character-to-transformation mapping function

Usage

```
family_to_trafo(family, add_const = 1e-08)
```

Arguments

family	character defining the distribution
add_const	see make_tfd_dist

Value

a list of transformation for each distribution parameter

family_to_trafo_torch	<i>Character-to-transformation mapping function</i>
-----------------------	---

Description

Character-to-transformation mapping function

Usage

```
family_to_trafo_torch(family, add_const = 1e-08)
```

Arguments

family	character defining the distribution
add_const	see make_torch_dist

Value

a list of transformation for each distribution parameter

family_to_trochd	<i>Character-torch mapping function</i>
------------------	---

Description

Character-torch mapping function

Usage

```
family_to_trochd(family)
```

Arguments

family	character defining the distribution
--------	-------------------------------------

Value

a torch distribution

fitted.drEnsemble	<i>Method for extracting the fitted values of an ensemble</i>
-------------------	---

Description

Method for extracting the fitted values of an ensemble

Usage

```
## S3 method for class 'drEnsemble'
fitted(object, apply_fun = tfd_mean, ...)
```

Arguments

object	a deepregression model
apply_fun	function applied to fitted distribution, per default tfd_mean
...	arguments passed to the predict function

Value

list of fitted values for each ensemble member

form_control	<i>Options for formula parsing</i>
--------------	------------------------------------

Description

Options for formula parsing

Usage

```
form_control(precalculate_gamparts = TRUE, check_form = TRUE)
```

Arguments

precalculate_gamparts	logical; if TRUE (default), additive parts are pre-calculated and can later be used more efficiently. Set to FALSE only if no smooth effects are in the formula(s) and a formula is very large so that extracting all terms takes long or might fail
check_form	logical; if TRUE (default), the formula is checked in process_terms

Value

Returns a list with options

from_distfun_to_dist_torch	<i>Function to define output distribution based on dist_fun</i>
----------------------------	---

Description

Function to define output distribution based on dist_fun

Usage

```
from_distfun_to_dist_torch(dist_fun, preds)
```

Arguments

dist_fun	a distribution function as defined by make_torch_dist
preds	tensors with predictions

Value

a symbolic torch distribution

from_dist_to_loss	<i>Function to transform a distribution layer output into a loss function</i>
-------------------	---

Description

Function to transform a distribution layer output into a loss function

Usage

```
from_dist_to_loss(  
  family,  
  ind_fun = function(x) tfd_independent(x),  
  weights = NULL  
)
```

Arguments

family	see ?deepregression
ind_fun	function applied to the model output before calculating the log-likelihood. Per default independence is assumed by applying tfd_independent.
weights	sample weights

Value

loss function

from_dist_to_loss_torch	<i>Function to transform a distribution layer output into a loss function</i>
-------------------------	---

Description

Function to transform a distribution layer output into a loss function

Usage

```
from_dist_to_loss_torch(family, weights = NULL)
```

Arguments

family	see ?deepregression
weights	sample weights

Value

loss function

from_preds_to_dist	<i>Define Predictor of a Deep Distributional Regression Model</i>
--------------------	---

Description

Define Predictor of a Deep Distributional Regression Model

Usage

```
from_preds_to_dist(
  list_pred_param,
  family = NULL,
  output_dim = 1L,
  mapping = NULL,
  from_family_to_distfun = make_tfd_dist,
  from_distfun_to_dist = distfun_to_dist,
  add_layer_shared_pred = function(x, units) layer_dense(x, units = units, use_bias =
    FALSE),
  trafo_list = NULL
)
```

Arguments

list_pred_param	list of input-output(-lists) generated from subnetwork_init
family	see ?deepregression; if NULL, concatenated list_pred_param entries are returned (after applying mapping if provided)
output_dim	dimension of the output
mapping	a list of integers. The i-th list item defines which element elements of list_pred_param are used for the i-th parameter. For example, mapping = list(1,2,1:2) means that list_pred_param[[1]] is used for the first distribution parameter, list_pred_param[[2]] for the second distribution parameter and list_pred_param[[3]] for both distribution parameters (and then added once to list_pred_param[[1]] and once to list_pred_param[[2]])
from_family_to_distfun	function to create a dist_fun (see ?distfun_to_dist) from the given character family
from_distfun_to_dist	function creating a tfp distribution based on the prediction tensors and dist_fun. See ?distfun_to_dist
add_layer_shared_pred	layer to extend shared layers defined in mapping
trafo_list	a list of transformation function to convert the scale of the additive predictors to the respective distribution parameter

Value

a list with input tensors and output tensors that can be passed to, e.g., `keras_model`

`from_preds_to_dist_torch`

Define Predictor of a Deep Distributional Regression Model

Description

Define Predictor of a Deep Distributional Regression Model

Usage

```
from_preds_to_dist_torch(
    list_pred_param,
    family = NULL,
    output_dim = 1L,
    mapping = NULL,
    from_family_to_distfun = make_torch_dist,
    from_distfun_to_dist = from_distfun_to_dist_torch,
    add_layer_shared_pred = function(input_shape, units) layer_dense_torch(input_shape =
        input_shape, units = units, use_bias = FALSE),
    trafo_list = NULL
)
```

Arguments

<code>list_pred_param</code>	list of output(-lists) generated from <code>subnetwork_init</code>
<code>family</code>	see <code>?deepregression</code> ; if <code>NULL</code> , concatenated <code>list_pred_param</code> entries are returned (after applying mapping if provided)
<code>output_dim</code>	dimension of the output
<code>mapping</code>	a list of integers. The <i>i</i> -th list item defines which element elements of <code>list_pred_param</code> are used for the <i>i</i> -th parameter. For example, <code>mapping = list(1, 2, 1:2)</code> means that <code>list_pred_param[[1]]</code> is used for the first distribution parameter, <code>list_pred_param[[2]]</code> for the second distribution parameter and <code>list_pred_param[[3]]</code> for both distribution parameters (and then added once to <code>list_pred_param[[1]]</code> and once to <code>list_pred_param[[2]]</code>)
<code>from_family_to_distfun</code>	function to create a <code>dist_fun</code> (see <code>?distfun_to_dist</code>) from the given character family
<code>from_distfun_to_dist</code>	function creating a torch distribution based on the prediction tensors and <code>dist_fun</code> . See <code>?distfun_to_dist</code>
<code>add_layer_shared_pred</code>	layer to extend shared layers defined in mapping

trafo_list a list of transformation function to convert the scale of the additive predictors to the respective distribution parameter

Value

a list with input tensors and output tensors that can be passed to, e.g., torch_model

gam_plot_data	<i>used by gam_processor</i>
---------------	------------------------------

Description

used by gam_processor

Usage

```
gam_plot_data(pp, weights, grid_length = 40, pe_fun = pe_gen)
```

Arguments

pp	processed term
weights	layer weights
grid_length	length for grid for evaluating basis
pe_fun	function used to generate partial effects

get_distribution	<i>Function to return the fitted distribution</i>
------------------	---

Description

Function to return the fitted distribution

Usage

```
get_distribution(x, data = NULL, force_float = FALSE)
```

Arguments

x	the fitted deepregression object
data	an optional data set
force_float	forces conversion into float tensors

```
get_ensemble_distribution
```

Obtain the conditional ensemble distribution

Description

Obtain the conditional ensemble distribution

Usage

```
get_ensemble_distribution(object, data = NULL, topK = NULL, ...)
```

Arguments

object	object of class "drEnsemble"
data	data for which to return the fitted distribution
topK	not implemented yet
...	further arguments currently ignored

Value

tfd_distribution of the ensemble, i.e., a mixture of the ensemble member's predicted distributions conditional on data

```
get_gamdata
```

Extract property of gamdata

Description

Extract property of gamdata

Usage

```
get_gamdata(
  term,
  param_nr,
  gamdata,
  what = c("data_trafo", "predict_trafo", "input_dim", "partial_effect", "sp_and_S",
    "df")
)
```


Arguments

term	term in formula
param_nr	integer; number of the distribution parameter
gamdata	list as returned by precalc_gam
what	string specifying what to return

Value

property of the gamdata object as defined by what

get_gamdata_reduced_nr

Extract number in matching table of reduced gam term

Description

Extract number in matching table of reduced gam term

Usage

```
get_gamdata_reduced_nr(term, param_nr, gamdata)
```

Arguments

term	term in formula
param_nr	integer; number of the distribution parameter
gamdata	list as returned by precalc_gam

Value

integer with number of gam term in matching table

get_gam_part

Extract gam part from wrapped term

Description

Extract gam part from wrapped term

Usage

```
get_gam_part(term, wrapper = "vc")
```

Arguments

term	character; gam model term
wrapper	character; function name that is wrapped around the gam part

get_help_forward_torch

Helper function to calculate amount of layers Needed when shared layers are used, because of layers have same names

Description

Helper function to calculate amount of layers Needed when shared layers are used, because of layers have same names

Usage

```
get_help_forward_torch(list_pred_param)
```

Arguments

list_pred_param
list; subnetworks

Value

layers

get_layernr_by_opname *Function to return layer number given model and name*

Description

Function to return layer number given model and name

Usage

```
get_layernr_by_opname(mod, name, partial_match = FALSE)
```

Arguments

mod deepregression model
name character
partial_match logical; whether to also check for a partial match

get_layernr_trainable *Function to return layer numbers with trainable weights*

Description

Function to return layer numbers with trainable weights

Usage

```
get_layernr_trainable(mod, logic = FALSE)
```

Arguments

mod	deepregression model
logic	logical; TRUE: return logical vector; FALSE (default) index

get_layer_by_opname *Function to return layer given model and name*

Description

Function to return layer given model and name

Usage

```
get_layer_by_opname(mod, name, partial_match = FALSE)
```

Arguments

mod	deepregression model
name	character
partial_match	logical; whether to also check for a partial match

get_luz_dataset	<i>Helper function to create an function that generates R6 instances of class dataset</i>
-----------------	---

Description

Helper function to create an function that generates R6 instances of class dataset

Usage

```
get_luz_dataset(df_list, target = NULL, length = NULL, object)
```

Arguments

df_list	list; data for the distribution learning model (data for every distributional parameter)
target	vector; target value
length	amount of inputs
object	deepregression object

Value

R6 instances of class dataset

get_names_pfc	<i>Extract term names from the parsed formula content</i>
---------------	---

Description

Extract term names from the parsed formula content

Usage

```
get_names_pfc(pfc)
```

Arguments

pfc	parsed formula content
-----	------------------------

Value

vector of term names

get_nodedata	<i>Extract attributes/hyper-parameters of the node term</i>
--------------	---

Description

Extract attributes/hyper-parameters of the node term

Usage

```
get_nodedata(term, what)
```

Arguments

term	term in formula
what	string specifying what to return

Value

property of the node specification as defined by what

get_node_term	<i>Extract variables from wrapped node term</i>
---------------	---

Description

Extract variables from wrapped node term

Usage

```
get_node_term(term)
```

Arguments

term	character; node model term
------	----------------------------

Value

reduced variable node model term

get_partial_effect	<i>Return partial effect of one smooth term</i>
--------------------	---

Description

Return partial effect of one smooth term

Usage

```
get_partial_effect(  
  object,  
  names = NULL,  
  return_matrix = FALSE,  
  which_param = 1,  
  newdata = NULL,  
  ...  
)
```

Arguments

object	deepregression object
names	string; for partial match with smooth term
return_matrix	logical; whether to return the design matrix or
which_param	integer; which distribution parameter the partial effect (FALSE, default)
newdata	data.frame; new data (optional)
...	arguments passed to get_weight_by_name

get_processor_name	<i>Extract processor name from term</i>
--------------------	---

Description

Extract processor name from term

Usage

```
get_processor_name(term)
```

Arguments

term	term in formula
------	-----------------

Value

processor name as string

get_special	<i>Extract terms defined by specials in formula</i>
-------------	---

Description

Extract terms defined by specials in formula

Usage

```
get_special(term, specials, simplify = FALSE)
```

Arguments

term	term in formula
specials	string(s); special name(s)
simplify	logical; shortcut for returning only the name of the special in term

Value

specials in formula

get_type_pfc	<i>Function to subset parsed formulas</i>
--------------	---

Description

Function to subset parsed formulas

Usage

```
get_type_pfc(pfc, type = NULL)
```

Arguments

pfc	list of parsed formulas
type	either NULL (all types of coefficients are returned), "linear" for linear coefficients or "smooth" for coefficients of

get_weight_by_name	<i>Function to retrieve the weights of a structured layer</i>
--------------------	---

Description

Function to retrieve the weights of a structured layer

Usage

```
get_weight_by_name(mod, name, param_nr = 1, postfixes = "")
```

Arguments

mod	fitted deepregression object
name	name of partial effect
param_nr	distribution parameter number
postfixes	character (vector) appended to layer name

Value

weight matrix

get_weight_by_opname	<i>Function to return weight given model and name</i>
----------------------	---

Description

Function to return weight given model and name

Usage

```
get_weight_by_opname(mod, name, partial_match = FALSE)
```

Arguments

mod	deepregression model
name	character
partial_match	logical; whether to also check for a partial match

handle_gam_term	<i>Function to define smoothness and call mgcv's smooth constructor</i>
-----------------	---

Description

Function to define smoothness and call mgcv's smooth constructor

Usage

```
handle_gam_term(object, data, controls)
```

Arguments

object	character defining the model term
data	data.frame or list
controls	controls for penalization

Value

constructed smooth term

import_packages	<i>Function to import required packages</i>
-----------------	---

Description

Function to import required packages

Usage

```
import_packages(engine)
```

Arguments

engine	tensorflow or torch
--------	---------------------

<code>import_tf_dependings</code>	<i>Function to import required packages for tensorflow @import tensorflow tfprobability keras</i>
-----------------------------------	---

Description

Function to import required packages for tensorflow
 @import tensorflow tfprobability keras

Usage

```
import_tf_dependings()
```

<code>import_torch_dependings</code>	<i>Function to import required packages for torch @import torch torchvision luz</i>
--------------------------------------	---

Description

Function to import required packages for torch
 @import torch torchvision luz

Usage

```
import_torch_dependings()
```

<code>keras_dr</code>	<i>Compile a Deep Distributional Regression Model</i>
-----------------------	---

Description

Compile a Deep Distributional Regression Model

Usage

```
keras_dr(
  list_pred_param,
  weights = NULL,
  optimizer = tf$keras$optimizers$Adam(),
  model_fun = keras_model,
  monitor_metrics = list(),
  from_preds_to_output = from_preds_to_dist,
  loss = from_dist_to_loss(family = list(...)$family, weights = weights),
  additional_penalty = NULL,
  ...
)
```

Arguments

<code>list_pred_param</code>	list of input-output(-lists) generated from <code>subnetwork_init</code>
<code>weights</code>	vector of positive values; optional (default = 1 for all observations)
<code>optimizer</code>	optimizer used. Per default Adam
<code>model_fun</code>	which function to use for model building (default <code>keras_model</code>)
<code>monitor_metrics</code>	Further metrics to monitor
<code>from_preds_to_output</code>	function taking the <code>list_pred_param</code> outputs and transforms it into a single network output
<code>loss</code>	the model's loss function; per default evaluated based on the arguments <code>family</code> and <code>weights</code> using <code>from_dist_to_loss</code>
<code>additional_penalty</code>	a penalty that is added to the negative log-likelihood; must be a function of <code>model\$trainable_weights</code> with suitable subsetting
<code>...</code>	arguments passed to <code>from_preds_to_output</code>

Value

a list with input tensors and output tensors that can be passed to, e.g., `keras_model`

Examples

```
set.seed(24)
n <- 500
x <- runif(n) %>% as.matrix()
z <- runif(n) %>% as.matrix()

y <- x - z
data <- data.frame(x = x, z = z, y = y)

# change loss to mse and adapt
# \code{from_preds_to_output} to work
```

```
# only on the first output column
mod <- deepregression(
  y = y,
  data = data,
  list_of_formulas = list(loc = ~ 1 + x + z, scale = ~ 1),
  list_of_deep_models = NULL,
  family = "normal",
  from_preds_to_output = function(x, ...) x[[1]],
  loss = "mse"
)
```

layer_add_identity	<i>Convenience layer function</i>
--------------------	-----------------------------------

Description

Convenience layer function

Usage

```
layer_add_identity(inputs)
```

```
layer_concatenate_identity(inputs)
```

Arguments

inputs	list of tensors
--------	-----------------

Details

convenience layers to work with list of inputs where inputs can also have length one

Value

tensor

layer_dense_module	<i>Function to create custom nn_linear module to overwrite reset_parameters</i>
--------------------	---

Description

Function to create custom nn_linear module to overwrite reset_parameters

Usage

```
layer_dense_module(kernel_initializer)
```

Arguments

kernel_initializer
string; initializer used to reset_parameters

Value

nn module

layer_dense_torch	<i>Function to define a torch layer similar to a tf dense layer</i>
-------------------	---

Description

Function to define a torch layer similar to a tf dense layer

Usage

```
layer_dense_torch(
    input_shape,
    units = 1L,
    name,
    trainable = TRUE,
    kernel_initializer = "glorot_uniform",
    use_bias = FALSE,
    kernel_regularizer = NULL,
    ...
)
```

Arguments

input_shape	integer; number of input units
units	integer; number of output units
name	string; string defining the layer's name
trainable	logical; whether layer is trainable
kernel_initializer	initializer; for coefficients
use_bias	logical; whether bias is used (default no)
kernel_regularizer	regularizer; for coefficients
...	arguments used in choose_kernel_initializer_torch

Value

torch layer

layer_generator	<i>Function that creates layer for each processor</i>
-----------------	---

Description

Function that creates layer for each processor

Usage

```
layer_generator(
  term,
  output_dim,
  param_nr,
  controls,
  name = makelayername(term, param_nr),
  layer_class = tf$keras$layers$Dense,
  without_layer = tf$identity,
  further_layer_args = NULL,
  layer_args_names = NULL,
  units = as.integer(output_dim),
  engine = "tf",
  ...
)

int_processor(term, data, output_dim, param_nr, controls, engine = "tf")

lin_processor(term, data, output_dim, param_nr, controls, engine = "tf")

ri_processor(term, data, output_dim, param_nr, controls, engine)
```

```

gam_processor(term, data, output_dim, param_nr, controls, engine = "tf")

autogam_processor(term, data, output_dim, param_nr, controls, engine = "tf")

node_processor(
  term,
  data,
  output_dim,
  param_nr,
  controls = NULL,
  engine = "tf"
)

```

Arguments

term	character; term in the formula
output_dim	integer; number of units in the layer
param_nr	integer; identifier for models with more than one additive predictor
controls	list; control arguments which allow to pass further information
name	character; name of layer. if NULL, makelayername will be used to create layer name
layer_class	a tf or keras layer function
without_layer	function to be used as layer if controls\$with_layer is FALSE
further_layer_args	named list; further arguments passed to the layer
layer_args_names	character vector; if NULL, default layer args will be used. Needs to be set for layers that do not provide the arguments of a default Dense layer.
units	integer; number of units for layer
engine	character; the engine which is used to setup the NN (tf or torch)
...	other keras layer parameters
data	data frame; the data used in processors

Value

a basic processor list structure

layer_node	<i>NODE/ODTs Layer</i>
------------	------------------------

Description

NODE/ODTs Layer

Usage

```
layer_node(
  name,
  units,
  n_layers = 1L,
  n_trees = 1L,
  tree_depth = 1L,
  threshold_init_beta = 1
)
```

Arguments

name	name of the layer
units	number of output dimensions, for regression and binary classification: 1, for mc-classification simply the number of classes
n_layers	number of layers consisting of ODTs in NODE
n_trees	number of trees per layer
tree_depth	depth of tree per layer
threshold_init_beta	parameter(s) for Beta-distribution used for initializing feature thresholds

Value

layer/model object

Examples

```
n <- 1000
data_regr <- data.frame(matrix(rnorm(4 * n), c(n, 4)))
colnames(data_regr) <- c("x0", "x1", "x2", "x3")
y_regr <- rnorm(n) + data_regr$x0^2 + data_regr$x1 +
  data_regr$x2*data_regr$x3 + data_regr$x2 + data_regr$x3

library(deepregression)

formula_node <- ~ node(x1, x2, x3, x0, n_trees = 2, n_layers = 2, tree_depth = 2)

mod_node_regr <- deepregression(
  list_of_formulas = list(loc = formula_node, scale = ~ 1),
```



```

data = data_regr,
y = y_regr
)

if(!is.null(mod_node_regr)){
  mod_node_regr %>% fit(epochs = 15, batch_size = 64, verbose = TRUE,
    validation_split = 0.1, early_stopping = TRUE)
  mod_node_regr %>% predict()
}

```

layer_sparse_batch_normalization

Sparse Batch Normalization layer

Description

Sparse Batch Normalization layer

Usage

```
layer_sparse_batch_normalization(lam = NULL, ...)
```

Arguments

lam	regularization strength
...	arguments passed to TensorFlow layer

Value

layer object

layer_sparse_conv_2d *Sparse 2D Convolutional layer*

Description

Sparse 2D Convolutional layer

Usage

```
layer_sparse_conv_2d(filters, kernel_size, lam = NULL, depth = 2, ...)
```

Arguments

filters	number of filters
kernel_size	size of convolutional filter
lam	regularization strength
depth	depth of weight factorization
...	arguments passed to TensorFlow layer

Value

layer object

layer_spline	<i>Function to define spline as TensorFlow layer</i>
--------------	--

Description

Function to define spline as TensorFlow layer

Usage

```
layer_spline(
    units = 1L,
    P,
    name,
    trainable = TRUE,
    kernel_initializer = "glorot_uniform"
)
```

Arguments

units	integer; number of output units
P	matrix; penalty matrix
name	string; string defining the layer's name
trainable	logical; whether layer is trainable
kernel_initializer	initializer; for basis coefficients

Value

TensorFlow layer

layer_spline_torch	<i>Function to define spline as Torch layer</i>
--------------------	---

Description

Function to define spline as Torch layer

Usage

```
layer_spline_torch(  
    P,  
    units = 1L,  
    name,  
    trainable = TRUE,  
    kernel_initializer = "glorot_uniform",  
    ...  
)
```

Arguments

P	matrix; penalty matrix
units	integer; number of output units
name	string; string defining the layer's name
trainable	logical; whether layer is trainable
kernel_initializer	initializer; for basis coefficients
...	value used for constant kernel initializer

Value

Torch spline layer

log_score	<i>Function to return the log_score</i>
-----------	---

Description

Function to return the log_score

Usage

```
log_score(
  x,
  data = NULL,
  this_y = NULL,
  ind_fun = NULL,
  convert_fun = as.matrix,
  summary_fun = function(x) x
)
```

Arguments

x	the fitted deepregression object
data	an optional data set
this_y	new y for optional data
ind_fun	function indicating the dependency; per default (iid assumption) tfd_independent is used.
convert_fun	function that converts Tensor; per default as.matrix
summary_fun	function summarizing the output; per default the identity

loop_through_pfc_and_call_trafo

Function to loop through parsed formulas and apply data trafo

Description

Function to loop through parsed formulas and apply data trafo

Usage

```
loop_through_pfc_and_call_trafo(pfc, newdata = NULL, engine = "tf")
```

Arguments

pfc	list of processor transformed formulas
newdata	list in the same format as the original data
engine	character; the engine which is used to setup the NN (tf or torch)

Value

list of matrices or arrays

makeInputs	<i>Convenience layer function</i>
------------	-----------------------------------

Description

Convenience layer function

Usage

makeInputs(pp, param_nr)

Arguments

pp processed predictors
param_nr integer for the parameter

Value

input tensors with appropriate names

makelayername	<i>Function that takes term and create layer name</i>
---------------	---

Description

Function that takes term and create layer name

Usage

makelayername(term, param_nr, truncate = 60)

Arguments

term term in formula
param_nr integer; defining number of the distribution's parameter
truncate integer; value from which on names are truncated

Value

name (string) for layer

make_folds	<i>Generate folds for CV out of one hot encoded matrix</i>
------------	--

Description

Generate folds for CV out of one hot encoded matrix

Usage

```
make_folds(mat, val_train = 0, val_test = 1)
```

Arguments

mat	matrix with columns corresponding to folds and entries corresponding to a one hot encoding
val_train	the value corresponding to train, per default 0
val_test	the value corresponding to test, per default 1

Details

val_train and val_test can both be a set of value

make_generator	<i>creates a generator for training</i>
----------------	---

Description

creates a generator for training

Usage

```
make_generator(  
  input_x,  
  input_y = NULL,  
  batch_size,  
  sizes,  
  shuffle = TRUE,  
  seed = 42L  
)
```

Arguments

input_x	list of matrices
input_y	list of matrix
batch_size	integer
sizes	sizes of the image including colour channel
shuffle	logical for shuffling data
seed	seed for shuffling in generators

Value

generator for all x and y

make_generator_from_matrix

Make a DataGenerator from a data.frame or matrix

Description

Creates a Python Class that internally iterates over the data.

Usage

```
make_generator_from_matrix(
  x,
  y = NULL,
  generator = image_data_generator(),
  batch_size = 32L,
  shuffle = TRUE,
  seed = 1L
)
```

Arguments

x	matrix;
y	vector;
generator	generator as e.g. obtained from ‘keras::image_data_generator’. Used for consistent train-test splits.
batch_size	integer
shuffle	logical; Should data be shuffled?
seed	integer; seed for shuffling data.

make_tfd_dist	<i>Families for deepregression</i>
---------------	------------------------------------

Description

Families for deepregression

Families for deepregression

Usage

```
make_tfd_dist(family, add_const = 1e-08, output_dim = 1L, trafo_list = NULL)
```

```
make_torch_dist(family, add_const = 1e-08, output_dim = 1L, trafo_list = NULL)
```

Arguments

family	character vector
add_const	small positive constant to stabilize calculations
output_dim	number of output dimensions of the response (larger 1 for multivariate case) (not implemented yet)
trafo_list	list of transformations for each distribution parameter. Per default the transformation listed in details is applied.

Details

To specify a custom distribution, define the a function as follows `function(x) do.call(your_tfd_dist, lapply(1:ncol(x)[[1]], function(i) your_trafo_list_on_inputs[[i]]( x[,i,drop=FALSE])))` and pass it to deepregression via the `dist_fun` argument. Currently the following distributions are supported with parameters (and corresponding inverse link function in brackets):

- "normal" : normal distribution with location (identity), scale (exp)
- "bernoulli" : bernoulli distribution with logits (identity)
- "bernoulli_prob" : bernoulli distribution with probabilities (sigmoid)
- "beta" : beta with concentration 1 = alpha (exp) and concentration 0 = beta (exp)
- "betar" : beta with mean (sigmoid) and scale (sigmoid)
- "cauchy" : location (identity), scale (exp)
- "chi2" : cauchy with df (exp)
- "chi" : cauchy with df (exp)
- "exponential" : exponential with lambda (exp)
- "gamma" : gamma with concentration (exp) and rate (exp)
- "gammarr" : gamma with location (exp) and scale (exp), following `gamlss.dist::GA`, which implies that the expectation is the location, and the variance of the distribution is the $\text{location}^2 \text{scale}^2$

- "gumbel" : gumbel with location (identity), scale (exp)
- "half_cauchy" : half cauchy with location (identity), scale (exp)
- "half_normal" : half normal with scale (exp)
- "horseshoe" : horseshoe with scale (exp)
- "inverse_gamma" : inverse gamma with concentration (exp) and rate (exp)
- "inverse_gamma_ls" : inverse gamma with location (exp) and variance (1/exp)
- "inverse_gaussian" : inverse Gaussian with location (exp) and concentration (exp)
- "laplace" : Laplace with location (identity) and scale (exp)
- "log_normal" : Log-normal with location (identity) and scale (exp) of underlying normal distribution
- "logistic" : logistic with location (identity) and scale (exp)
- "negbinom" : neg. binomial with count (exp) and prob (sigmoid)
- "negbinom_ls" : neg. binomial with mean (exp) and clutter factor (exp)
- "pareto" : Pareto with concentration (exp) and scale (1/exp)
- "pareto_ls" : Pareto location scale version with mean (exp) and scale (exp), which corresponds to a Pareto distribution with parameters $\text{scale} = \text{mean}$ and $\text{concentration} = 1/\text{sigma}$, where sigma is the scale in the pareto_ls version
- "poisson" : poisson with rate (exp)
- "poisson_lograte" : poisson with lograte (identity)
- "student_t" : Student's t with df (exp)
- "student_t_ls" : Student's t with df (exp), location (identity) and scale (exp)
- "uniform" : uniform with upper and lower (both identity)
- "zinb" : Zero-inflated negative binomial with mean (exp), variance (exp) and prob (sigmoid)
- "zip" : Zero-inflated poisson distribution with mean (exp) and prob (sigmoid)

To specify a custom distribution, define the a function as follows `function(x) do.call(your_tfd_dist, lapply(1:ncol(x)[[1]], function(i) your_trafo_list_on_inputs[[i]]( x[,i,drop=FALSE])))` and pass it to deepregression via the `dist_fun` argument. Currently the following distributions are supported with parameters (and corresponding inverse link function in brackets):

- "normal" : normal distribution with location (identity), scale (exp)
- "bernoulli" : bernoulli distribution with logits (identity)
- "exponential" : exponential with lambda (exp)
- "gamma" : gamma with concentration (exp) and rate (exp)
- "poisson" : poisson with rate (exp)

model_torch	<i>Function to initialize a nn_module Forward functions works with a list. The entries of the list are the input of the subnetworks</i>
-------------	---

Description

Function to initialize a nn_module Forward functions works with a list. The entries of the list are the input of the subnetworks

Usage

```
model_torch(submodules_list)
```

Arguments

submodules_list
list; subnetworks

Value

nn_module

multioptimizer	<i>Function to define an optimizer combining multiple optimizers</i>
----------------	--

Description

Function to define an optimizer combining multiple optimizers

Usage

```
multioptimizer(optimizers_and_layers)
```

Arguments

optimizers_and_layers
a list if tuples of optimizer and respective layers

Value

an optimizer

names_families	Returns the parameter names for a given family
----------------	--

Description

Returns the parameter names for a given family

Usage

```
names_families(family)
```

Arguments

family character specifying the family as defined by deepregression

Value

vector of parameter names

na_omit_list	Function to exclude NA values
--------------	-------------------------------

Description

Function to exclude NA values

Usage

```
na_omit_list(datalist)
```

Arguments

datalist list of data as returned by prepare_data and prepare_newdata

Value

list with NA values excluded and locations of original NA positions as attributes

nn_init_no_grad_constant_deepreg	<i>custom nn_linear module to overwrite reset_parameters # nn_init_constant works only if value is scalar; so warmstarts for gam does'not work</i>
----------------------------------	--

Description

custom nn_linear module to overwrite reset_parameters # nn_init_constant works only if value is scalar; so warmstarts for gam does'not work

Usage

nn_init_no_grad_constant_deepreg(tensor, value)

Arguments

- tensor scalar or vector
- value value used for constant initialization

Value

tensor

orthog_control	<i>Options for orthogonalization</i>
----------------	--------------------------------------

Description

Options for orthogonalization

Usage

```
orthog_control(  
  split_fun = split_model,  
  orthog_type = c("tf", "manual"),  
  orthogonalize = options()$orthogonalize,  
  identify_intercept = options()$identify_intercept,  
  deep_top = NULL,  
  orthog_fun = NULL,  
  deactivate_oz_at_test = TRUE  
)
```

Arguments

split_fun	a function separating the deep neural network in two parts so that the orthogonalization can be applied to the first part before applying the second network part; per default, the function <code>split_model</code> is used which assumes a dense layer as penultimate layer and separates the network into a first part without this last layer and a second part only consisting of a single dense layer that is fed into the output layer
orthog_type	one of two options; If "manual", the QR decomposition is calculated before model fitting, otherwise ("tf") a QR is calculated in each batch iteration via TF. The first only works well for larger batch sizes or ideally <code>batch_size == NROW(y)</code> .
orthogonalize	logical; if set to TRUE, automatic orthogonalization is activated
identify_intercept	whether to orthogonalize the deep network w.r.t. the intercept to make the intercept identifiable
deep_top	function; optional function to put on top of the deep network instead of splitting the function using <code>split_fun</code>
orthog_fun	function; for custom orthogonalization. if NULL, <code>orthog_type</code> is used to define the function that computes the orthogonalization
deactivate_oz_at_test	logical; whether to deactivate the orthogonalization cell at test time when using <code>orthog_tf</code> for <code>orthog_fun</code> (the default).

Value

Returns a list with options

orthog_P	<i>Function to compute adjusted penalty when orthogonalizing</i>
----------	--

Description

Function to compute adjusted penalty when orthogonalizing

Usage

```
orthog_P(P, Z)
```

Arguments

P	matrix; original penalty matrix
Z	matrix; constraint matrix

Value

adjusted penalty matrix

orthog_post_fitting	<i>Orthogonalize a Semi-Structured Model Post-hoc</i>
---------------------	---

Description

Orthogonalize a Semi-Structured Model Post-hoc

Usage

```
orthog_post_fitting(mod, name_penult, param_nr = 1)
```

Arguments

mod	deepregression model
name_penult	character name of the penultimate layer of the deep part part
param_nr	integer; number of the parameter to be returned

Value

a deepregression object with weights frozen and deep part specified by name_penult orthogonalized

orthog_structured_smooths_Z	<i>Orthogonalize structured term by another matrix</i>
-----------------------------	--

Description

Orthogonalize structured term by another matrix

Usage

```
orthog_structured_smooths_Z(S, L)
```

Arguments

S	matrix; matrix to orthogonalize
L	matrix; matrix which defines the projection and its orthogonal complement, in which S is projected

Value

constraint matrix

penalty_control	<i>Options for penalty setup in the pre-processing</i>
-----------------	--

Description

Options for penalty setup in the pre-processing

Usage

```
penalty_control(
  defaultSmoothing = NULL,
  df = 10,
  null_space_penalty = FALSE,
  absorb_cons = FALSE,
  anisotropic = TRUE,
  zero_constraint_for_smooths = TRUE,
  no_linear_trend_for_smooths = FALSE,
  hat1 = FALSE,
  sp_scale = function(x) ifelse(is.list(x) | is.data.frame(x), 1/NROW(x)[[1]]), 1/NROW(x))
)
```

Arguments

defaultSmoothing	function applied to all s-terms, per default (NULL) the minimum df of all possible terms is used. Must be a function the smooth term from mgcv's smoothCon and an argument df.
df	degrees of freedom for all non-linear structural terms (default = 7); either one common value or a list of the same length as number of parameters; if different df values need to be assigned to different smooth terms, use df as an argument for s(), te() or ti()
null_space_penalty	logical value; if TRUE, the null space will also be penalized for smooth effects. Per default, this is equal to the value give in variational.
absorb_cons	logical; adds identifiability constraint to the basis. See ?mgcv::smoothCon for more details.
anisotropic	whether or not use anisotropic smoothing (default is TRUE)
zero_constraint_for_smooths	logical; the same as absorb_cons, but done explicitly. If true a constraint is put on each smooth to have zero mean. Can be a vector of length(list_of_formulas) for each distribution parameter.
no_linear_trend_for_smooths	logical; see zero_constraint_for_smooths, but this removes the linear trend from splines
hat1	logical; if TRUE, the smoothing parameter is defined by the trace of the hat matrix $\text{sum}(\text{diag}(H))$, else $\text{sum}(\text{diag}(2*H-HH))$
sp_scale	function of response; for scaling the penalty (1/n per default)

Value

Returns a list with options

plot.deepregression	<i>Generic functions for deepregression models</i>
---------------------	--

Description

Generic functions for deepregression models
 Predict based on a deepregression object
 Function to extract fitted distribution
 Fit a deepregression model (pendant to fit for keras)
 Extract layer weights / coefficients from model
 Print function for deepregression model
 Cross-validation for deepregression objects
 mean of model fit
 Standard deviation of fit distribution
 Calculate the distribution quantiles

Usage

```
## S3 method for class 'deepregression'
plot(
  x,
  which = NULL,
  which_param = 1,
  only_data = FALSE,
  grid_length = 40,
  main_multiple = NULL,
  type = "b",
  get_weight_fun = get_weight_by_name,
  ...
)

## S3 method for class 'deepregression'
predict(
  object,
  newdata = NULL,
  batch_size = NULL,
  apply_fun = tfd_mean,
  convert_fun = as.matrix,
  ...
)
```



```
## S3 method for class 'deepregression'
fitted(object, apply_fun = tfd_mean, ...)

## S3 method for class 'deepregression'
fit(
  object,
  batch_size = 32,
  epochs = 10,
  early_stopping = FALSE,
  early_stopping_metric = "val_loss",
  verbose = TRUE,
  view_metrics = FALSE,
  patience = 20,
  save_weights = FALSE,
  validation_data = NULL,
  validation_split = ifelse(is.null(validation_data), 0.1, 0),
  callbacks = list(),
  na_handler = na_omit_list,
  ...
)

## S3 method for class 'deepregression'
coef(object, which_param = 1, type = NULL, ...)

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

## S3 method for class 'deepregression'
cv(
  x,
  verbose = FALSE,
  patience = 20,
  plot = TRUE,
  print_folds = TRUE,
  cv_folds = 5,
  stop_if_nan = TRUE,
  mylapply = lapply,
  save_weights = FALSE,
  callbacks = list(),
  save_fun = NULL,
  ...
)

## S3 method for class 'deepregression'
mean(x, data = NULL, ...)

## S3 method for class 'deepregression'
```

```

stddev(x, data = NULL, ...)

## S3 method for class 'deepregression'
quant(x, data = NULL, probs, ...)

```

Arguments

<code>x</code>	a deepregression object
<code>which</code>	character vector or number(s) identifying the effect to plot; default plots all effects
<code>which_param</code>	integer, indicating for which distribution parameter coefficients should be returned (default is first parameter)
<code>only_data</code>	logical, if TRUE, only the data for plotting is returned
<code>grid_length</code>	the length of an equidistant grid at which a two-dimensional function is evaluated for plotting.
<code>main_multiple</code>	vector of strings; plot main titles if multiple plots are selected
<code>type</code>	either NULL (all types of coefficients are returned), "linear" for linear coefficients or "smooth" for coefficients of smooth terms
<code>get_weight_fun</code>	function to extract weight from model given x, a name and param_nr
<code>...</code>	arguments passed to the predict function
<code>object</code>	a deepregression model
<code>newdata</code>	optional new data, either data.frame or list
<code>batch_size</code>	integer, the batch size used for mini-batch training
<code>apply_fun</code>	function applied to fitted distribution, per default <code>tfd_mean</code>
<code>convert_fun</code>	how should the resulting tensor be converted, per default <code>as.matrix</code>
<code>epochs</code>	integer, the number of epochs to fit the model
<code>early_stopping</code>	logical, whether early stopping should be user.
<code>early_stopping_metric</code>	character, based on which metric should early stopping be triggered (default: "val_loss")
<code>verbose</code>	whether to print training in each fold
<code>view_metrics</code>	logical, whether to trigger the Viewer in RStudio / Browser.
<code>patience</code>	number of patience for early stopping
<code>save_weights</code>	logical, whether to save weights in each epoch.
<code>validation_data</code>	optional specified validation data
<code>validation_split</code>	float in [0,1] defining the amount of data used for validation
<code>callbacks</code>	a list of callbacks used for fitting
<code>na_handler</code>	function to deal with NAs
<code>plot</code>	whether to plot the resulting losses in each fold

print_folds	whether to print the current fold
cv_folds	an integer; can also be a list of lists with train and test data sets per fold
stop_if_nan	logical; whether to stop CV if NaN values occur
mylapply	lapply function to be used; defaults to lapply
save_fun	function applied to the model in each fold to be stored in the final result
data	either NULL or a new data set
probs	the quantile value(s)

Value

Returns an object drCV, a list, one list element for each fold containing the model fit and the weighthistory.

plot_cv	<i>Plot CV results from deepregression</i>
---------	--

Description

Plot CV results from deepregression

Usage

```
plot_cv(x, what = c("loss", "weight"), engine = "tf", ...)
```

Arguments

x	drCV object returned by cv.deepregression
what	character indicating what to plot (currently supported 'loss' or 'weights')
engine	character indicating which engine was used to setup the NN
...	further arguments passed to matplot

precalc_gam	<i>Pre-calculate all gam parts from the list of formulas</i>
-------------	--

Description

Pre-calculate all gam parts from the list of formulas

Usage

```
precalc_gam(lob, data, controls)
```

Arguments

lof	list of formulas
data	the data list
controls	controls from deepregression

Value

a list of length 2 with a matching table to link every unique gam term to formula entries and the respective data transformation functions

predict_gam_handler	<i>Handler for prediction with gam terms</i>
---------------------	--

Description

Handler for prediction with gam terms

Usage

```
predict_gam_handler(object, newdata)
```

Arguments

object	sterm
newdata	data.frame or list

predict_gen	<i>Generator function for deepregression objects</i>
-------------	--

Description

Generator function for deepregression objects

Usage

```
predict_gen(
  object,
  newdata = NULL,
  batch_size = NULL,
  apply_fun = tfd_mean,
  convert_fun = as.matrix,
  ret_dist = FALSE
)
```

Arguments

object	deepregression model;
newdata	data.frame or list; for (optional) new data
batch_size	integer; NULL will use the default (20)
apply_fun	see ?predict.deepregression
convert_fun	see ?predict.deepregression
ret_dist	logical; whether to return the whole distribution or only the (mean) prediction

Value

matrix or list of distributions

prepare_data	<i>Function to prepare data based on parsed formulas</i>
--------------	--

Description

Function to prepare data based on parsed formulas

Usage

```
prepare_data(pfc, na_handler = na_omit_list, gamdata = NULL, engine = "tf")
```

Arguments

pfc	list of processor transformed formulas
na_handler	function to deal with NAs
gamdata	processor for gam part
engine	the engine which is used to setup the NN (tf or torch)

Value

list of matrices or arrays

prepare_data_torch	<i>Function to additionally prepare data for fit process (torch)</i>
--------------------	--

Description

Function to additionally prepare data for fit process (torch)

Usage

```
prepare_data_torch(pfc, input_x, target = NULL, object)
```

Arguments

pfc	list of processor transformed formulas
input_x	output of prepare_data()
target	target values
object	a deepregression object

Value

list of matrices or arrays for predict or a dataloader for fit process

prepare_input_list_model	<i>Function to prepare input list for fit process, due to different approaches</i>
--------------------------	--

Description

Function to prepare input list for fit process, due to different approaches

Usage

```
prepare_input_list_model(
  input_x,
  input_y,
  object,
  epochs = 10,
  batch_size = 32,
  validation_split = 0,
  validation_data = NULL,
  callbacks = NULL,
  verbose,
  view_metrics,
  early_stopping
)
```

Arguments

input_x	output of prepare_data()
input_y	target
object	a deepregression object
epochs	integer, the number of epochs to fit the model
batch_size	integer, the batch size used for mini-batch training
validation_split	float in [0,1] defining the amount of data used for validation
validation_data	optional specified validation data
callbacks	a list of callbacks for fitting
verbose	logical, whether to print losses during training.
view_metrics	logical, whether to trigger the Viewer in RStudio / Browser.
early_stopping	logical, whether early stopping should be user.

Value

list of arguments used in fit function

prepare_newdata	<i>Function to prepare new data based on parsed formulas</i>
-----------------	--

Description

Function to prepare new data based on parsed formulas

Usage

```
prepare_newdata(
  pfc,
  newdata,
  na_handler = na_omit_list,
  gamdata = NULL,
  engine = "tf"
)
```

Arguments

pfc	list of processor transformed formulas
newdata	list in the same format as the original data
na_handler	function to deal with NAs
gamdata	processor for gam part
engine	character; the engine which is used to setup the NN (tf or torch)

Value

list of matrices or arrays

```
prepare_torch_distr_mixdistr
```

Prepares distributions for mixture process

Description

Prepares distributions for mixture process

Usage

```
prepare_torch_distr_mixdistr(object, dists)
```

Arguments

object	object of class "drEnsemble"
dists	fitted distributions

Value

distribution parameters used for mixture of same distribution

```
process_terms
```

Control function to define the processor for terms in the formula

Description

Control function to define the processor for terms in the formula

Usage

```
process_terms(
  form,
  data,
  controls,
  output_dim,
  param_nr,
  parsing_options,
  specials_to_oz = c(),
  automatic_oz_check = TRUE,
  identify_intercept = FALSE,
  engine = "tf",
  ...
)
```


Arguments

form	the formula to be processed
data	the data for the terms in the formula
controls	controls for gam terms
output_dim	the output dimension of the response
param_nr	integer; identifier for the distribution parameter
parsing_options	options
specials_to_oz	specials that should be automatically checked for
automatic_oz_check	logical; whether to automatically check for DNNs to be orthogonalized
identify_intercept	logical; whether to make the intercept automatically identifiable
engine	character; the engine which is used to setup the NN (tf or torch)
...	further processors

Value

returns a processor function

quant	<i>Generic quantile function</i>
-------	----------------------------------

Description

Generic quantile function

Usage

```
quant(x, ...)
```

Arguments

x	object
...	further arguments passed to the class-specific function

reinit_weights	<i>Generic function to re-initialize model weights</i>
----------------	--

Description

Generic function to re-initialize model weights

Usage

```
reinit_weights(object, seed)
```

Arguments

object	model to re-initialize
seed	seed for reproducibility

reinit_weights.deepregression	<i>Method to re-initialize weights of a "deepregression" model</i>
-------------------------------	--

Description

Method to re-initialize weights of a "deepregression" model

Usage

```
## S3 method for class 'deepregression'  
reinit_weights(object, seed)
```

Arguments

object	object of class "deepregression"
seed	seed for reproducibility

Value

invisible NULL

re_layer	<i>random effect layer</i>
----------	----------------------------

Description

random effect layer
trainable penalty layer

Usage

```
re_layer(units, ...)

pen_layer(units, P, ...)
```

Arguments

units	integer; number of units
...	arguments passed to TensorFlow layer
P	penalty matrix

Value

layer object
layer object

separate_define_relation	<i>Function to define orthogonalization connections in the formula</i>
--------------------------	--

Description

Function to define orthogonalization connections in the formula

Usage

```
separate_define_relation(
  form,
  specials,
  specials_to_oz,
  automatic_oz_check = TRUE,
  identify_intercept = FALSE,
  simplify = FALSE
)
```

Arguments

form a formula for one distribution parameter
 specials specials in formula to handle separately
 specials_to_oz parts of the formula to orthogonalize
 automatic_oz_check logical; automatically check if terms must be orthogonalized
 identify_intercept logical; whether to make the intercept identifiable
 simplify logical; if FALSE, formulas are parsed more carefully.

Value

Returns a list of formula components with ids and assignments for orthogonalization

simplyconnected_layer_torch
Hadamard-type layers torch

Description

Hadamard-type layers torch

Usage

```

simplyconnected_layer_torch(
  la = la,
  multifac_initializer = torch_ones,
  input_shape
)

tiblinlasso_layer_torch(
  la,
  input_shape = 1,
  units = 1,
  kernel_initializer = "he_normal"
)

tib_layer_torch(units, la, input_shape, multifac_initializer = torch_ones)

tibgroup_layer_torch(
  units,
  group_idx = NULL,
  la = 0,
  input_shape,
  kernel_initializer = "torch_ones",
  multifac_initializer = "he_normal"
)

```

Arguments

- la numeric; regularization value (> 0)
- multfac_initializer initializer for parameters
- input_shape integer; number of input dimension
- units integer; number of units
- kernel_initializer initializer
- group_idx list of group indices

Value

- nn_module
- torch layer object
- nn_module
- nn_module

stddev	<i>Generic sd function</i>
--------	----------------------------

Description

Generic sd function

Usage

stddev(x, ...)

Arguments

- x object
- ... further arguments passed to the class-specific function

stop_iter_cv_result	<i>Function to get the stoppting iteration from CV</i>
---------------------	--

Description

Function to get the stoppting iteration from CV

Usage

```
stop_iter_cv_result(
    res,
    thisFUN = mean,
    loss = "validloss",
    whichFUN = which.min
)
```

Arguments

res	result of cv call
thisFUN	aggregating function applied over folds
loss	which loss to use for decision
whichFUN	which function to use for decision

subnetwork_init	<i>Initializes a Subnetwork based on the Processed Additive Predictor</i>
-----------------	---

Description

Initializes a Subnetwork based on the Processed Additive Predictor

Usage

```
subnetwork_init(
    pp,
    deep_top = NULL,
    orthog_fun = orthog_tf,
    split_fun = split_model,
    shared_layers = NULL,
    param_nr = 1,
    selectfun_in = function(pp) pp[[param_nr]],
    selectfun_lay = function(pp) pp[[param_nr]],
    gaminputs,
    summary_layer = layer_add_identity
)
```

Arguments

pp	list of processed predictor lists from processor
deep_top	keras layer if the top part of the deep network after orthogonalization is different to the one extracted from the provided network
orthog_fun	function used for orthogonalization
split_fun	function to split the network to extract head
shared_layers	list defining shared weights within one predictor; each list item is a vector of characters of terms as given in the parameter formula
param_nr	integer number for the distribution parameter
selectfun_in, selectfun_lay	functions defining which subset of pp to take as inputs and layers for this sub-network; per default the param_nr's entry
gaminputs	input tensors for gam terms
summary_layer	keras layer that combines inputs (typically adding or concatenating)

Value

returns a list of input and output for this additive predictor

subnetwork_init_torch *Initializes a Subnetwork based on the Processed Additive Predictor*

Description

Initializes a Subnetwork based on the Processed Additive Predictor

Usage

```
subnetwork_init_torch(
  pp,
  deep_top = NULL,
  orthog_fun = NULL,
  split_fun = split_model,
  shared_layers = NULL,
  param_nr = 1,
  selectfun_in = function(pp) pp[[param_nr]],
  selectfun_lay = function(pp) pp[[param_nr]],
  gaminputs,
  summary_layer = model_torch
)
```

Arguments

pp	list of processed predictor lists from processor
deep_top	In tf approach: keras layer if the top part of the deep network after orthogonalization; Not yet implemented for torch is different to the one extracted from the provided network
orthog_fun	function used for orthogonalization; Not yet implemented for torch
split_fun	function to split the network to extract head
shared_layers	list defining shared weights within one predictor; each list item is a vector of characters of terms as given in the parameter formula
param_nr	integer number for the distribution parameter
selectfun_in, selectfun_lay	functions defining which subset of pp to take as inputs and layers for this sub-network; per default the param_nr's entry
gaminputs	input tensors for gam terms
summary_layer	torch layer that combines inputs (typically adding or concatenating)

Value

returns a list of input and output for this additive predictor

tfd_mse	<i>For using mean squared error via TFP</i>
---------	---

Description

For using mean squared error via TFP

Usage

```
tfd_mse(mean)
```

Arguments

mean	parameter for the mean
------	------------------------

Details

deepregression allows to train based on the MSE by using loss = "mse" as argument to deepregression. This tfd function just provides a dummy family

Value

a TFP distribution

tfd_zinb	<i>Implementation of a zero-inflated negbinom distribution for TFP</i>
----------	--

Description

Implementation of a zero-inflated negbinom distribution for TFP

Usage

```
tfd_zinb(mu, r, probs)
```

Arguments

mu, r	parameter of the negbin_ls distribution
probs	vector of probabilities of length 2 (probability for poisson and probability for 0s)

tfd_zip	<i>Implementation of a zero-inflated poisson distribution for TFP</i>
---------	---

Description

Implementation of a zero-inflated poisson distribution for TFP

Usage

```
tfd_zip(lambda, probs)
```

Arguments

lambda	scalar value for rate of poisson distribution
probs	vector of probabilities of length 2 (probability for poisson and probability for 0s)

tf_repeat	<i>TensorFlow repeat function which is not available for TF 2.0</i>
-----------	---

Description

TensorFlow repeat function which is not available for TF 2.0

Usage

```
tf_repeat(a, dim)
```

Arguments

a	tensor
dim	dimension for repeating

tf_row_tensor	<i>Row-wise tensor product using TensorFlow</i>
---------------	---

Description

Row-wise tensor product using TensorFlow

Usage

tf_row_tensor(a, b, ...)

Arguments

- a, b tensor
- ... arguments passed to TensorFlow layer

Value

a TensorFlow layer

tf_split_multiple	<i>Split tensor in multiple parts</i>
-------------------	---------------------------------------

Description

Split tensor in multiple parts

Usage

tf_split_multiple(A, len)

Arguments

- A tensor
- len integer; defines the split lengths

Value

list of tensors

tf_stride_cols	<i>Function to index tensors columns</i>
----------------	--

Description

Function to index tensors columns

Usage

```
tf_stride_cols(A, start, end = NULL)
```

Arguments

- | | |
|-------|---|
| A | tensor |
| start | first index |
| end | last index (equals start index if NULL) |

Value

sliced tensor

tf_stride_last_dim_tensor	<i>Function to index tensors last dimension</i>
---------------------------	---

Description

Function to index tensors last dimension

Usage

```
tf_stride_last_dim_tensor(A, start, end = NULL)
```

Arguments

- | | |
|-------|---|
| A | tensor |
| start | first index |
| end | last index (equals start index if NULL) |

Value

sliced tensor

tib_layer	<i>Hadamard-type layers</i>
-----------	-----------------------------

Description

Hadamard-type layers

Usage

```
tib_layer(units, la, ...)

simplyconnected_layer(la, ...)

inverse_group_lasso_pen(la)

regularizer_group_lasso(la, group_idx)

tibgroup_layer(units, group_idx, la, ...)

layer_hadamard(units = 1, la = 0, depth = 3, ...)

layer_group_hadamard(units, la, group_idx, depth, ...)

layer_hadamard_diff(
  units,
  la,
  initu = "glorot_uniform",
  initv = "glorot_uniform",
  ...
)

layer_hadamard(units = 1, la = 0, depth = 3, ...)
```

Arguments

units	integer; number of units
la	numeric; regularization value (> 0)
...	arguments passed to TensorFlow layer
group_idx	list of group indices
depth	integer; depth of weight factorization
initu, initv	initializers for parameters

Value

layer object

torch_dr

*Compile a Deep Distributional Regression Model (Torch)***Description**

Compile a Deep Distributional Regression Model (Torch)

Usage

```
torch_dr(
  list_pred_param,
  optimizer = torch::optim_adam,
  model_fun = NULL,
  monitor_metrics = list(),
  from_preds_to_output = from_preds_to_dist_torch,
  loss = from_dist_to_loss_torch(family = list(...)$family, weights = NULL),
  additional_penalty = NULL,
  ...
)
```

Arguments

list_pred_param	list of output(-lists) generated from subnetwork_init
optimizer	optimizer used. Per default Adam
model_fun	NULL not needed for torch
monitor_metrics	Further metrics to monitor
from_preds_to_output	function taking the list_pred_param outputs and transforms it into a single network output
loss	the model's loss function; per default evaluated based on the arguments family and weights using from_dist_to_loss
additional_penalty	a penalty that is added to the negative log-likelihood; must be a function of model\$trainable_weights with suitable subsetting (not implemented for torch)
...	arguments passed to from_preds_to_output
weights	vector of positive values; optional (default = 1 for all observations)

Value

a luz_module_generator

update_miniconda_deepregression

Function to update miniconda and packages

Description

Function to update miniconda and packages

Usage

```
update_miniconda_deepregression(
    python = VERSIONPY,
    uninstall = TRUE,
    also_packages = TRUE
)
```

Arguments

python	string; version of python
uninstall	logical; whether to uninstall previous conda env
also_packages	logical; whether to install also all required packages

weight_control

Options for weights of layers

Description

Options for weights of layers

Usage

```
weight_control(
    specific_weight_options = NULL,
    general_weight_options = list(activation = NULL, use_bias = FALSE, trainable = TRUE,
    kernel_initializer = "glorot_uniform", bias_initializer = "zeros", kernel_regularizer
    = NULL, bias_regularizer = NULL, activity_regularizer = NULL, kernel_constraint =
    NULL, bias_constraint = NULL),
    warmstart_weights = NULL,
    shared_layers = NULL
)
```

Arguments

- `specific_weight_options`
 - specific options for certain weight terms; must be a list of length `length(list_of_formulas)` and each element in turn a named list (names are term names as in the formula) with specific options in a list
- `general_weight_options`
 - default options for layers
- `warmstart_weights`
 - While all keras layer options are available, the user can further specify a list for each distribution parameter with list elements corresponding to term names with values as vectors corresponding to start weights of the respective weights
- `shared_layers`
 - list for each distribution parameter; each list item can be again a list of character vectors specifying terms which share layers

Value

Returns a list with options

Index

autogam_processor (layer_generator), 38

check_and_install, 4

check_input_args_fit, 5

choose_kernel_initializer_torch, 5

coef.deepregression
 (plot.deepregression), 56

coef.drEnsemble, 6

collect_distribution_parameters, 6

combine_penalties, 7

create_family, 7

create_family_torch, 8

create_penalty, 8

cv, 9

cv.deepregression
 (plot.deepregression), 56

deepregression, 9

distfun_to_dist, 12

ensemble, 12

ensemble.deepregression, 13

extract_pure_gam_part, 15

extract_S, 16

extractlen (extractval), 14

extractval, 14

extractvals (extractval), 14

extractvar, 15

family_to_tfd, 16

family_to_trafo, 17

family_to_trafo_torch, 17

family_to_trochd, 18

fit.deepregression
 (plot.deepregression), 56

fitted.deepregression
 (plot.deepregression), 56

fitted.drEnsemble, 18

form2text (extractval), 14

form_control, 19

from_dist_to_loss, 20

from_dist_to_loss_torch, 20

from_distfun_to_dist_torch, 19

from_preds_to_dist, 21

from_preds_to_dist_torch, 22

gam_plot_data, 23

gam_processor (layer_generator), 38

get_distribution, 23

get_ensemble_distribution, 24

get_gam_part, 25

get_gamdata, 24

get_gamdata_reduced_nr, 25

get_help_forward_torch, 26

get_layer_by_opname, 27

get_layernr_by_opname, 26

get_layernr_trainable, 27

get_luz_dataset, 28

get_names_pfc, 28

get_node_term, 29

get_nodedata, 29

get_partial_effect, 30

get_processor_name, 30

get_special, 31

get_type_pfc, 31

get_weight_by_name, 32

get_weight_by_opname, 32

handle_gam_term, 33

import_packages, 33

import_tf_dependings, 34

import_torch_dependings, 34

int_processor (layer_generator), 38

inverse_group_lasso_pen (tib_layer), 76

keras_dr, 34

layer_add_identity, 36

layer_concatenate_identity
 (layer_add_identity), 36

layer_dense_module, 37
 layer_dense_torch, 37
 layer_generator, 38
 layer_group_hadamard (tib_layer), 76
 layer_hadamard (tib_layer), 76
 layer_hadamard_diff (tib_layer), 76
 layer_node, 40
 layer_sparse_batch_normalization, 41
 layer_sparse_conv_2d, 41
 layer_spline, 42
 layer_spline_torch, 43
 lin_processor (layer_generator), 38
 log_score, 43
 loop_through_pfc_and_call_trafo, 44

 make_folds, 46
 make_generator, 46
 make_generator_from_matrix, 47
 make_tfd_dist, 10, 17, 48
 make_torch_dist, 17
 make_torch_dist (make_tfd_dist), 48
 makeInputs, 45
 makelayername, 45
 mean.deepregression
 (plot.deepregression), 56
 model_torch, 50
 multioptimizer, 50

 na_omit_list, 51
 names_families, 51
 nn_init_no_grad_constant_deepreg, 52
 node_processor (layer_generator), 38

 orthog_control, 10, 52
 orthog_P, 53
 orthog_post_fitting, 54
 orthog_structured_smooths_Z, 54

 pen_layer (re_layer), 67
 penalty_control, 10, 55
 plot.deepregression, 56
 plot_cv, 59
 precalc_gam, 59
 predict.deepregression
 (plot.deepregression), 56
 predict_gam_handler, 60
 predict_gen, 60
 prepare_data, 61
 prepare_data_torch, 62

 prepare_input_list_model, 62
 prepare_newdata, 63
 prepare_torch_distr_mixdistr, 64
 print.deepregression
 (plot.deepregression), 56
 process_terms, 64

 quant, 65
 quant.deepregression
 (plot.deepregression), 56

 re_layer, 67
 regularizer_group_lasso (tib_layer), 76
 reinit_weights, 66
 reinit_weights.deepregression, 66
 ri_processor (layer_generator), 38

 separate_define_relation, 67
 simplyconnected_layer (tib_layer), 76
 simplyconnected_layer_torch, 68
 stddev, 69
 stddev.deepregression
 (plot.deepregression), 56
 stop_iter_cv_result, 70
 subnetwork_init, 70
 subnetwork_init_torch, 71

 tf_repeat, 73
 tf_row_tensor, 74
 tf_split_multiple, 74
 tf_stride_cols, 75
 tf_stride_last_dim_tensor, 75
 tfd_mse, 72
 tfd_zinb, 73
 tfd_zip, 73
 tib_layer, 76
 tib_layer_torch
 (simplyconnected_layer_torch),
 68
 tibgroup_layer (tib_layer), 76
 tibgroup_layer_torch
 (simplyconnected_layer_torch),
 68
 tiblinlasso_layer_torch
 (simplyconnected_layer_torch),
 68
 torch_dr, 77

 update_miniconda_deepregression, 78
 weight_control, 10, 78