

# Package: deepregression (via r-universe)

September 11, 2024

**Title** Fitting Deep Distributional Regression

**Version** 1.0.0

**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 = ``2.10.0", pip = TRUE), list(package = ``tensorflow_probability", version = ``0.16", pip = TRUE), list(package = ``keras", version = ``2.10.0", 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

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Author** David Ruegamer [aut, cre], Florian Pfisterer [ctb], Philipp Baumann [ctb], Chris Kolb [ctb], Lucas Kook [ctb]

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

**Date/Publication** 2023-01-17 22:50:08 UTC

**Repository** <https://druegamer.r-universe.dev>

**RemoteUrl** <https://github.com/cran/deepregression>

**RemoteRef** HEAD

**RemoteSha** 338b46a35be92ef1bf704c6d08d80c7f1764034a

## Contents

check_and_install . . . . .	3
coef.drEnsemble . . . . .	4
combine_penalties . . . . .	5
create_family . . . . .	5
create_penalty . . . . .	6
cv . . . . .	6
deepregression . . . . .	7
distfun_to_dist . . . . .	9
ensemble . . . . .	10
ensemble.deepregression . . . . .	10
extractval . . . . .	11
extractvar . . . . .	12
extract_pure_gam_part . . . . .	13
extract_S . . . . .	13
family_to_tfd . . . . .	14
family_to_trafo . . . . .	14
fitted.drEnsemble . . . . .	15
form_control . . . . .	15
from_dist_to_loss . . . . .	16
from_preds_to_dist . . . . .	16
gam_plot_data . . . . .	17
get_distribution . . . . .	18
get_ensemble_distribution . . . . .	18
get_gamdata . . . . .	19
get_gamdata_reduced_nr . . . . .	19
get_gam_part . . . . .	20
get_layernr_by_opname . . . . .	20
get_layernr_trainable . . . . .	21
get_layer_by_opname . . . . .	21
get_names_pfc . . . . .	22
get_partial_effect . . . . .	22
get_processor_name . . . . .	23
get_special . . . . .	23
get_type_pfc . . . . .	24
get_weight_by_name . . . . .	24
get_weight_by_opname . . . . .	25
handle_gam_term . . . . .	25
keras_dr . . . . .	26
layer_add_identity . . . . .	27
layer_generator . . . . .	28
layer_sparse_conv_2d . . . . .	29
layer_spline . . . . .	29
log_score . . . . .	30
loop_through_pfc_and_call_trafo . . . . .	31
makeInputs . . . . .	31
makelayername . . . . .	32

make_folds . . . . .	32
make_generator . . . . .	33
make_generator_from_matrix . . . . .	33
make_tfd_dist . . . . .	34
multioptimizer . . . . .	36
names_families . . . . .	36
orthog_control . . . . .	37
orthog_P . . . . .	38
orthog_post_fitting . . . . .	38
orthog_structured_smooths_Z . . . . .	39
penalty_control . . . . .	39
plot.deepregression . . . . .	40
plot_cv . . . . .	44
precalc_gam . . . . .	44
predict_gen . . . . .	45
prepare_data . . . . .	45
prepare_newdata . . . . .	46
process_terms . . . . .	46
quant . . . . .	47
reinit_weights . . . . .	48
reinit_weights.deepregression . . . . .	48
separate_define_relation . . . . .	49
stddev . . . . .	49
stop_iter_cv_result . . . . .	50
subnetwork_init . . . . .	50
tfd_mse . . . . .	51
tfd_zinb . . . . .	52
tfd_zip . . . . .	52
tf_repeat . . . . .	52
tf_row_tensor . . . . .	53
tf_split_multiple . . . . .	53
tf_stride_cols . . . . .	54
tf_stride_last_dim_tensor . . . . .	54
tib_layer . . . . .	55
update_miniconda_deepregression . . . . .	56
weight_control . . . . .	56
<b>Index</b>	<b>58</b>

---

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)
```

**Arguments**

force                    if TRUE, forces the installations

**Value**

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

---

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

---

combine\_penalties      *Function to combine two penalties*

---

**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          *Function to create (custom) family*

---

**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 distributional layer

---

create\_penalty            *Function to create mgcv-type penalty*

---

**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                                *Generic cv function*

---

**Description**

Generic cv function

**Usage**

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

**Arguments**

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

**Description**

Fitting Semi-Structured Deep Distributional Regression

**Usage**

```

deepregression(
  y,
  list_of_formulas,
  list_of_deep_models = NULL,
  family = "normal",
  data,
  tf_seed = as.integer(1991 - 5 - 4),
  return_preproc = FALSE,
  subnetwork_builder = subnetwork_init,
  model_builder = keras_dr,
  fitting_function = utils::getFromNamespace("fit.keras.engine.training.Model",
    "keras"),
  additional_processors = list(),
  penalty_options = penalty_control(),
  orthog_options = orthog_control(),
  weight_options = weight_control(),
  formula_options = form_control(),
  output_dim = 1L,
  verbose = FALSE,
  ...
)

```

**Arguments**

`y` response variable

`list_of_formulas`

a named list of right hand side formulas, one for each parameter of the distribution specified in `family`; set to `~ 1` if the parameter should be treated as constant. Use the `s()`-notation from `mgcv` for specification of non-linear structured effects and `d(...)` for deep learning predictors (predictors in brackets are separated by commas), where `d` can be replaced by an name name of the names in `list_of_deep_models`, e.g., `~ 1 + s(x) + my_deep_mod(a, b, c)`, where `my_deep_mod` is the name of the neural net specified in `list_of_deep_models` and `a, b, c` are features modeled via this network.

`list_of_deep_models`

a named list of functions specifying a keras model. See the examples for more details.

family	a character specifying the distribution. For information on possible distribution and parameters, see <code>make_tfd_dist</code> . Can also be a custom distribution.
data	data.frame or named list with input features
tf_seed	a seed for TensorFlow (only works with R version $\geq 2.2.0$ )
return_preproc	logical; if TRUE only the pre-processed data and layers are returned (default FALSE).
subnetwork_builder	function to build each subnetwork (network for each distribution parameter; per default <code>subnetwork_init</code> ). Can also be a list of the same size as <code>list_of_formulas</code> .
model_builder	function to build the model based on additive predictors (per default <code>keras_dr</code> ). In order to work with the methods defined for the class <code>deepregression</code> , the model should behave like a keras model
fitting_function	function to fit the instantiated model when calling <code>fit</code> . Per default the keras <code>fit</code> function.
additional_processors	a named list with additional processors to convert the formula(s). Can have an attribute "controls" to pass additional controls
penalty_options	options for smoothing and penalty terms defined by <code>penalty_control</code>
orthog_options	options for the orthogonalization defined by <code>orthog_control</code>
weight_options	options for layer weights defined by <code>weight_control</code>
formula_options	options for formula parsing (mainly used to make calculation more efficiently)
output_dim	dimension of the output, per default 1L
verbose	logical; whether to print progress of model initialization to console
...	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,x3) + s(xa) + x1

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

*Function to define output distribution based on dist\_fun*


---

### 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                      *Generic deep ensemble function*

---

**Description**

Generic deep ensemble function

**Usage**

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

**Arguments**

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

---

ensemble.deepregression  
*Ensemblind deepregression models*

---

**Description**

Ensemblind 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

*Formula helpers*


---

**Description**

Formula helpers

**Usage**

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

```
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
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 *Extract the smooth term from a deepregression term specification*

---

**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                    *Convenience function to extract penalty matrix and value*

---

**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

---

fitted.drEnsemble      *Method for extracting the fitted values of an ensemble*

---

**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      *Options for formula parsing*

---

**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\_dist\_to\_loss      *Function to transform a distribution layer output into a loss function*

---

**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\_preds\_to\_dist      *Define Predictor of a Deep Distributional Regression Model*

---

**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

---

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      *Function to return the fitted distribution*

---

**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	<i>Extract property of gamdata</i>
-------------	------------------------------------

---

**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	<i>Extract number in matching table of reduced gam term</i>
------------------------	---

---

**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	<i>Extract gam part from wrapped term</i>
--------------	---

---

**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_layernr_by_opname	<i>Function to return layer number given model and name</i>
-----------------------	---

---

**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_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_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      *Extract processor name from term*

---

**Description**

Extract processor name from term

**Usage**

```
get_processor_name(term)
```

**Arguments**

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

**Value**

processor name as string

---

get\_special      *Extract terms defined by specials in formula*

---

**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                      *Function to subset parsed formulas*

---

### 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                *Function to retrieve the weights of a structured layer*

---

### 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    *Function to return weight given model and name*

---

**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    *Function to define smoothness and call mgcv's smooth constructor*

---

**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

keras\_dr

*Compile a Deep Distributional Regression Model***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      *Convenience layer function*

---

**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_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,
  layer_class = tf$keras$layers$Dense,
  without_layer = tf$identity,
  name = makelayername(term, param_nr),
  further_layer_args = NULL,
  layer_args_names = NULL,
  units = as.integer(output_dim),
  ...
)

int_processor(term, data, output_dim, param_nr, controls)

lin_processor(term, data, output_dim, param_nr, controls)

gam_processor(term, data, output_dim, param_nr, controls)
```

### 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
layer_class	a tf or keras layer function
without_layer	function to be used as layer if controls\$with_layer is FALSE
name	character; name of layer. if NULL, makelayername will be used to create layer name
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

... other keras layer parameters  
 data data frame; the data used in processors

**Value**

a basic processor list structure

---

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 *Function to define spline as TensorFlow layer*

---

**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

---

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 = function(x) tfd_independent(x),
  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)
```

**Arguments**

pfc	list of processor transformed formulas
newdata	list in the same format as the original data

**Value**

list of matrices or arrays

---

makeInputs

*Convenience layer function*


---

**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	<i>Make a DataGenerator from a data.frame or matrix</i>
----------------------------	---

---

**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

**Usage**

```
make_tfd_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)
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)
- "gammr": 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 `location^2 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 `scale = mean` and `concentration = 1/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)

---

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	<i>Returns the parameter names for a given family</i>
----------------	---

---

### 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

---

orthog\_control                      *Options for orthogonalization*

---

## 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                      *Function to compute adjusted penalty when orthogonalizing*

---

**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      *Orthogonalize a Semi-Structured Model Post-hoc*

---

**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

*Orthogonalize structured term by another matrix*


---

**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

*Options for penalty setup in the pre-processing*


---

**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    *Generic functions for deepregression models*

---

**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(),
  convertfun = function(x) tf$constant(x, dtype = "float32"),
  ...
)
```

```

)

## 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

x	a deepregression object
which	character vector or number(s) identifying the effect to plot; default plots all effects
which_param	integer, indicating for which distribution parameter coefficients should be returned (default is first parameter)
only_data	logical, if TRUE, only the data for plotting is returned
grid_length	the length of an equidistant grid at which a two-dimensional function is evaluated for plotting.
main_multiple	vector of strings; plot main titles if multiple plots are selected
type	either NULL (all types of coefficients are returned), "linear" for linear coefficients or "smooth" for coefficients of smooth terms
get_weight_fun	function to extract weight from model given x, a name and param_nr

...	arguments passed to the predict function
object	a deepregression model
newdata	optional new data, either data.frame or list
batch_size	integer, the batch size used for mini-batch training
apply_fun	function applied to fitted distribution, per default tfd_mean
convert_fun	how should the resulting tensor be converted, per default as.matrix
epochs	integer, the number of epochs to fit the model
early_stopping	logical, whether early stopping should be user.
early_stopping_metric	character, based on which metric should early stopping be triggered (default: "val_loss")
verbose	whether to print training in each fold
view_metrics	logical, whether to trigger the Viewer in RStudio / Browser.
patience	number of patience for early stopping
save_weights	logical, whether to save weights in each epoch.
validation_data	optional specified validation data
validation_split	float in [0,1] defining the amount of data used for validation
callbacks	a list of callbacks used for fitting
convertfun	function to convert R into Tensor object
plot	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"), ...)
```

**Arguments**

x	drCV object returned by cv.deepregression
what	character indicating what to plot (currently supported 'loss' or 'weights')
...	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(lof, 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_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, gamdata = NULL)
```

**Arguments**

pfc                    list of processor transformed formulas  
 gamdata              processor for gam part

**Value**

list of matrices or arrays

---

prepare\_newdata      *Function to prepare new data based on parsed formulas*

---

**Description**

Function to prepare new data based on parsed formulas

**Usage**

```
prepare_newdata(pfc, newdata, gamdata = NULL)
```

**Arguments**

pfc                    list of processor transformed formulas  
 newdata              list in the same format as the original data  
 gamdata              processor for gam part

**Value**

list of matrices or arrays

---

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,
  ...
)

```

**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
...	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



---

```
separate_define_relation
    Function to define orthogonalization connections in the formula
```

---

**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

---

```
stddev
    Generic sd function
```

---

**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 stopping iteration from CV</i>
---------------------	---

---

**Description**

Function to get the stopping 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

---

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      *Function to index tensors columns*

---

**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  
*Function to index tensors last dimension*

---

**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

---

`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**

<code>python</code>	string; version of python
<code>uninstall</code>	logical; whether to uninstall previous conda env
<code>also_packages</code>	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

check\_and\_install, 3  
coef.deepregression  
    (plot.deepregression), 40  
coef.drEnsemble, 4  
combine\_penalties, 5  
create\_family, 5  
create\_penalty, 6  
cv, 6  
cv.deepregression  
    (plot.deepregression), 40  
  
deepregression, 7  
distfun\_to\_dist, 9  
  
ensemble, 10  
ensemble.deepregression, 10  
extract\_pure\_gam\_part, 13  
extract\_S, 13  
extractlen (extractval), 11  
extractval, 11  
extractvar, 12  
  
family\_to\_tfd, 14  
family\_to\_trafo, 14  
fit.deepregression  
    (plot.deepregression), 40  
fitted.deepregression  
    (plot.deepregression), 40  
fitted.drEnsemble, 15  
form2text (extractval), 11  
form\_control, 15  
from\_dist\_to\_loss, 16  
from\_preds\_to\_dist, 16  
  
gam\_plot\_data, 17  
gam\_processor (layer\_generator), 28  
get\_distribution, 18  
get\_ensemble\_distribution, 18  
get\_gam\_part, 20  
get\_gamdata, 19  
get\_gamdata\_reduced\_nr, 19  
get\_layer\_by\_opname, 21  
get\_layernr\_by\_opname, 20  
get\_layernr\_trainable, 21  
get\_names\_pfc, 22  
get\_partial\_effect, 22  
get\_processor\_name, 23  
get\_special, 23  
get\_type\_pfc, 24  
get\_weight\_by\_name, 24  
get\_weight\_by\_opname, 25  
  
handle\_gam\_term, 25  
  
int\_processor (layer\_generator), 28  
inverse\_group\_lasso\_pen (tib\_layer), 55  
  
keras\_dr, 26  
  
layer\_add\_identity, 27  
layer\_concatenate\_identity  
    (layer\_add\_identity), 27  
layer\_generator, 28  
layer\_group\_hadamard (tib\_layer), 55  
layer\_hadamard (tib\_layer), 55  
layer\_hadamard\_diff (tib\_layer), 55  
layer\_sparse\_conv\_2d, 29  
layer\_spline, 29  
lin\_processor (layer\_generator), 28  
log\_score, 30  
loop\_through\_pfc\_and\_call\_trafo, 31  
  
make\_folds, 32  
make\_generator, 33  
make\_generator\_from\_matrix, 33  
make\_tfd\_dist, 8, 14, 34  
makeInputs, 31  
makelayername, 32  
mean.deepregression  
    (plot.deepregression), 40  
multioptimizer, 36

names\_families, 36

orthog\_control, 8, 37  
orthog\_P, 38  
orthog\_post\_fitting, 38  
orthog\_structured\_smooths\_Z, 39

penalty\_control, 8, 39  
plot.deepregression, 40  
plot\_cv, 44  
precalc\_gam, 44  
predict.deepregression  
    (plot.deepregression), 40  
predict\_gen, 45  
prepare\_data, 45  
prepare\_newdata, 46  
print.deepregression  
    (plot.deepregression), 40  
process\_terms, 46

quant, 47  
quant.deepregression  
    (plot.deepregression), 40

regularizer\_group\_lasso (tib\_layer), 55  
reinit\_weights, 48  
reinit\_weights.deepregression, 48

separate\_define\_relation, 49  
simplyconnected\_layer (tib\_layer), 55  
stddev, 49  
stddev.deepregression  
    (plot.deepregression), 40  
stop\_iter\_cv\_result, 50  
subnetwork\_init, 50

tf\_repeat, 52  
tf\_row\_tensor, 53  
tf\_split\_multiple, 53  
tf\_stride\_cols, 54  
tf\_stride\_last\_dim\_tensor, 54  
tfd\_mse, 51  
tfd\_zinb, 52  
tfd\_zip, 52  
tib\_layer, 55  
tibgroup\_layer (tib\_layer), 55

update\_miniconda\_deepregression, 56

weight\_control, 8, 56