

# Package: tabnet (via r-universe)

June 12, 2026

**Title** Fit 'TabNet' Models for Classification and Regression

**Version** 0.9.0

**Description** Implements the 'TabNet' model by Sercan O. Arik et al. (2019) <[doi:10.48550/arXiv.1908.07442](https://doi.org/10.48550/arXiv.1908.07442)> with 'Coherent Hierarchical Multi-label Classification Networks' by Giunchiglia et al. <[doi:10.48550/arXiv.2010.10151](https://doi.org/10.48550/arXiv.2010.10151)> and provides a consistent interface for fitting and creating predictions. It's also fully compatible with the 'tidymodels' ecosystem.

**License** MIT + file LICENSE

**URL** <https://mlverse.github.io/tabnet/>,  
<https://github.com/mlverse/tabnet>

**BugReports** <https://github.com/mlverse/tabnet/issues>

**Depends** R (>= 3.6)

**Imports** coro, data.tree, dials, dplyr, generics, ggplot2, hardhat (>= 1.3.0), magrittr, methods, parsnip, progress, purrr, rlang, stats, stringr, tibble, tidyr, torch (>= 0.4.0), tune, utils, vctrs, withr, zeallot

**Suggests** cli, knitr, modeldata, patchwork, quarto, recipes, rmarkdown, rsample, spelling, testthat (>= 3.0.0), tidymodels, tidyverse, vip, visdat, workflows, xgboost, yardstick

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/testthat/parallel** false

**Config/testthat/start-first** interface, explain, params

**Encoding** UTF-8

**Language** en-US

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Daniel Falbel [aut], RStudio [cph], Christophe Regouby [cre, ctb], Egill Fridgeirsson [ctb], Philipp Haarmeyer [ctb], Sven Verweij [ctb] (ORCID: <<https://orcid.org/0000-0002-5573-3952>>)

**Maintainer** Christophe Regouby <[christophe.regouby@free.fr](mailto:christophe.regouby@free.fr)>

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

**Date/Publication** 2026-06-12 16:30:13 UTC

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

**RemoteRef** HEAD

**RemoteSha** 6f51ecf80f26a18f9d72b024176df43b3904ab96

## Contents

attention_width . . . . .	3
autoplot.tabnet_explain . . . . .	4
autoplot.tabnet_fit . . . . .	5
build_ancestor_matrix_from_outcomes . . . . .	6
cat_emb_dim . . . . .	6
check_compliant_node . . . . .	7
entmax . . . . .	8
get_constr_output . . . . .	9
get_tau . . . . .	9
nn_aum_loss . . . . .	10
nn_mc_loss . . . . .	11
nn_prune_head.tabnet_fit . . . . .	12
nnf_mc_loss . . . . .	13
nnf_multilabel_one_hot . . . . .	14
node_to_df . . . . .	14
predict.tabnet_fit . . . . .	15
sparsemax . . . . .	16
tabnet . . . . .	17
tabnet_config . . . . .	20
tabnet_explain . . . . .	23
tabnet_fit . . . . .	25
tabnet_nn . . . . .	28
tabnet_pretrain . . . . .	29

<b>Index</b>	<b>32</b>
--------------	-----------

---

attention\_width      *Parameters for the tabnet model*

---

### Description

Parameters for the tabnet model

### Usage

```
attention_width(range = c(8L, 64L), trans = NULL)
decision_width(range = c(8L, 64L), trans = NULL)
feature_reusage(range = c(1, 2), trans = NULL)
momentum(range = c(0.01, 0.4), trans = NULL)
mask_type(values = c("sparsemax", "entmax"))
num_independent(range = c(1L, 5L), trans = NULL)
num_shared(range = c(1L, 5L), trans = NULL)
num_steps(range = c(3L, 10L), trans = NULL)
```

### Arguments

range	the default range for the parameter value
trans	whether to apply a transformation to the parameter
values	possible values for factor parameters

These functions are used with tune grid functions to generate candidates.

### Value

A dials parameter to be used when tuning TabNet models.

### Examples

```
model <- tabnet(attention_width = tune(), feature_reusage = tune(),
momentum = tune(), penalty = tune(), rate_step_size = tune()) %>%
parsnip::set_mode("regression") %>%
parsnip::set_engine("torch")
```

---

 autoplot.tabnet\_explain

*Plot tabnet\_explain mask importance heatmap*


---

## Description

Plot tabnet\_explain mask importance heatmap

## Usage

```
autoplot.tabnet_explain(
  object,
  type = c("mask_agg", "steps"),
  quantile = 1,
  ...
)
```

## Arguments

object	A tabnet_explain object as a result of <code>tabnet_explain()</code> .
type	a character value. Either "mask_agg" the default, for a single heatmap of aggregated mask importance per predictor along the dataset, or "steps" for one heatmap at each mask step.
quantile	numerical value between 0 and 1. Provides quantile clipping of the mask values
...	not used.

## Details

Plot the tabnet\_explain object mask importance per variable along the predicted dataset. `type="mask_agg"` output a single heatmap of mask aggregated values, `type="steps"` provides a plot faceted along the `n_steps` mask present in the model. `quantile=.995` may be used for strong outlier clipping, in order to better highlight low values. `quantile=1`, the default, do not clip any values.

## Value

A ggplot object.

## Examples

```
## Not run:
library(ggplot2)
data("attrition", package = "modeldata")

## Single-outcome binary classification of `Attrition` in `attrition` dataset
attrition_fit <- tabnet_fit(Attrition ~ ., data=attrition, epoch=11)
attrition_explain <- tabnet_explain(attrition_fit, attrition)
# Plot the model aggregated mask interpretation heatmap
```

```
autoplot(attrition_explain)

## Multi-outcome regression on `Sale_Price` and `Pool_Area` in `ames` dataset,
data("ames", package = "modeldata")
x <- ames[,-which(names(ames) %in% c("Sale_Price", "Pool_Area"))]
y <- ames[, c("Sale_Price", "Pool_Area")]
ames_fit <- tabnet_fit(x, y, epochs = 1, verbose=TRUE)
ames_explain <- tabnet_explain(ames_fit, x)
autoplot(ames_explain, quantile = 0.99)

## End(Not run)
```

---

autoplot.tabnet\_fit *Plot tabnet\_fit model loss along epochs*

---

## Description

Plot tabnet\_fit model loss along epochs

## Usage

```
autoplot.tabnet_fit(object, ...)
```

```
autoplot.tabnet_pretrain(object, ...)
```

## Arguments

object	A tabnet_fit or tabnet_pretrain object as a result of <code>tabnet_fit()</code> or <code>tabnet_pretrain()</code> .
...	not used.

## Details

Plot the training loss along epochs, and validation loss along epochs if any. A dot is added on epochs where model snapshot is available, helping the choice of `from_epoch` value for later model training resume.

## Value

A ggplot object.

## Examples

```
## Not run:
library(ggplot2)
data("attrition", package = "modeldata")
attrition_fit <- tabnet_fit(Attrition ~. , data=attrition, valid_split=0.2, epoch=11)
```

```
# Plot the model loss over epochs
autoplot(attrition_fit)

## End(Not run)
```

---

```
build_ancestor_matrix_from_outcomes
```

*Build ancestor matrix aligned with observed outcome classes*

---

### Description

Extracts class names from the outcome tibble (factor levels) and builds the ancestor matrix only for classes that actually appear in the data.

### Usage

```
build_ancestor_matrix_from_outcomes(x, outcomes, device = "cpu")
```

### Arguments

x	A <code>data.tree::Node</code> object.
outcomes	A tibble with factor columns (one per hierarchy level), as returned by <code>hardhat::mold()\$outcomes</code> .
device	Torch device ("cpu" or "cuda").

### Value

A `torch_tensor` of shape (1, n\_classes, n\_classes).

---

```
cat_emb_dim
```

*Non-tunable parameters for the tabnet model*

---

### Description

Non-tunable parameters for the tabnet model

### Usage

```
cat_emb_dim(range = NULL, trans = NULL)

checkpoint_epochs(range = NULL, trans = NULL)

drop_last(range = NULL, trans = NULL)

encoder_activation(range = NULL, trans = NULL)
```

```
lr_scheduler(range = NULL, trans = NULL)
mlp_activation(range = NULL, trans = NULL)
mlp_hidden_multiplier(range = NULL, trans = NULL)
num_independent_decoder(range = NULL, trans = NULL)
num_shared_decoder(range = NULL, trans = NULL)
optimizer(range = NULL, trans = NULL)
penalty(range = NULL, trans = NULL)
verbose(range = NULL, trans = NULL)
virtual_batch_size(range = NULL, trans = NULL)
```

**Arguments**

range	unused
trans	unused

---

check\_compliant\_node *Check that Node object names are compliant*

---

**Description**

Check that Node object names are compliant

**Usage**

```
check_compliant_node(node)
```

**Arguments**

node	the Node object, or a dataframe ready to be parsed by <code>data.tree::as.Node()</code>
------	---

**Value**

node if it is compliant, else an Error with the column names to fix

**Examples**

```

library(dplyr)
library(data.tree)
data(starwars)
starwars_tree <- starwars %>%
  mutate(pathString = paste("tree", species, homeworld, `name`, sep = "/"))

# pre as.Node() check
try(check_compliant_node(starwars_tree))

# post as.Node() check
check_compliant_node(as.Node(starwars_tree))

```

---

entmax

*Alpha-entmax*


---

**Description**

With  $\alpha = 1.5$  and normalizing sparse transform (a la softmax).

**Usage**

```

entmax(dim = -1)

entmax15(dim = -1L, k = NULL)

```

**Arguments**

dim	The dimension along which to apply 1.5-entmax.
k	The number of largest elements to partial-sort input over. For optimal performance, should be slightly bigger than the expected number of non-zeros in the solution. If the solution is more than k-sparse, this function is recursively called with a 2*k schedule. If NULL, full sorting is performed from the beginning.

**Details**

Solves the optimization problem:  $\max_p \langle input, P \rangle - H_{1.5}(P)$  s.t.  $P \geq 0, \sum(P) == 1$  where  $H_{1.5}(P)$  is the Tsallis alpha-entropy with  $\alpha = 1.5$ .

**Value**

The projection result P of the same shape as input, such that  $\sum_{dim} P = 1 \forall dim$  elementwise.

**Examples**

```
## Not run:
input <- torch::torch_randn(10,5, requires_grad = TRUE)
# create a top3 alpha=1.5 entmax on last input dimension
nn_entmax <- entmax15(dim=-1L, k = 3)
result <- nn_entmax(input)

## End(Not run)
```

---

get\_constr\_output      *Apply hierarchy constraints via max-pooling over descendants (MCM)*

---

**Description**

Given neural network outputs  $x$  and ancestor matrix  $R$ , enforces that if a class is predicted positive, all its ancestors must also be positive. Implements:  $\text{final\_out}[i] = \max\{x[j] : R[i, j] = 1\}$

**Usage**

```
get_constr_output(x, R)
```

**Arguments**

$x$                     A torch\_tensor of shape (batch\_size, n\_classes).

$R$                      A torch\_tensor of shape (1, n\_classes, n\_classes) where  $R[1, i, j] = 1$  iff class  $i$  is a descendant of class  $j$ .

**Value**

A torch\_tensor of shape (batch\_size, n\_classes) with constrained outputs.

---

get\_tau                    *Optimal threshold (tau) computation for 1.5-entmax*

---

**Description**

Optimal threshold (tau) computation for 1.5-entmax

**Usage**

```
get_tau(input, dim = -1L, k = NULL)
```

## Arguments

input	The input tensor to compute thresholds over.
dim	The dimension along which to apply 1.5-entmax. Default is -1.
k	The number of largest elements to partial-sort over. For optimal performance, should be slightly bigger than the expected number of non-zeros in the solution. If the solution is more than k-sparse, this function is recursively called with a 2*k schedule. If NULL, full sorting is performed from the beginning. Default is NULL.

## Value

The threshold value for each vector, with all but the dim dimension intact.

---

nn_aum_loss	<i>AUM loss</i>
-------------	-----------------

---

## Description

Creates a criterion that measures the Area under the  $\text{Min}(FPR, FNR)$  (AUM) between each element in the input  $\text{pred}_t\text{ensor}$  and target  $\text{label}_t\text{ensor}$ .

## Usage

```
nn_aum_loss()
```

## Details

This is used for measuring the error of a binary reconstruction within highly unbalanced dataset, where the goal is optimizing the ROC curve. Note that the targets  $\text{label}_t\text{ensor}$  should be factor level of the binary outcome, i.e. with values 1L and 2L.

## Examples

```
loss <- nn_aum_loss()
input <- torch::torch_randn(4, 6, requires_grad = TRUE)
target <- input > 1.5
output <- loss(input, target)
output$backward()
```

---

nn_mc_loss	<i>Max-Constraint Margin Loss (module)</i>
------------	--

---

### Description

Module wrapper for `nnf_mc_loss()` with configurable parameters. Stores the ancestor matrix `R` and evaluation mask for reuse across batches.

### Usage

```
nn_mc_loss(
  R,
  to_eval = NULL,
  criterion = nnf_binary_cross_entropy_with_logits,
  reduction = "mean"
)
```

### Arguments

<code>R</code>	Ancestor matrix tensor of shape $(1, n\_classes, n\_classes)$ .
<code>to_eval</code>	Optional logical tensor of shape $(n\_classes,)$ indicating which classes to include in loss computation.
<code>criterion</code>	Loss function module or functional to apply after constraint propagation. Default: <code>nnf_binary_cross_entropy_with_logits()</code> .
<code>reduction</code>	(string, optional): Reduction method: 'none'   'mean'   'sum'.

### Shape

- Input output:  $(N, C)$  where  $N$  = batch size,  $C$  = number of classes
- Input target:  $(N, C)$ , same shape as output, binary values
- Output: scalar by default. If `reduction = "none"`, then  $(N, C')$  where  $C'$  is the number of evaluated classes

### See Also

[nnf\\_mc\\_loss\(\)](#), [build\\_ancestor\\_matrix\\_from\\_outcomes\(\)](#), [get\\_constr\\_output\(\)](#)

### Examples

```
## Not run:
# Build ancestor matrix from hierarchy
R <- build_ancestor_matrix_from_outcomes(my_tree, processed$outcomes, device = "cuda")

# Create loss module
loss_fn <- nn_mc_loss(R = R, reduction = "mean")

# Forward pass
```

```

output <- model(x) # (batch, n_classes)
loss <- loss_fn(output, labels)
loss$backward()

## End(Not run)

```

---

```
nn_prune_head.tabnet_fit
```

*Prune top layer(s) of a tabnet network*

---

### Description

Prune head\_size last layers of a tabnet network in order to use the pruned module as a sequential embedding module.

### Usage

```

## S3 method for class 'tabnet_fit'
nn_prune_head(x, head_size)

## S3 method for class 'tabnet_pretrain'
nn_prune_head(x, head_size)

```

### Arguments

x	nn_network to prune
head_size	number of nn_layers to prune, should be less than 2

### Value

a tabnet network with the top nn\_layer removed

### Examples

```

data("ames", package = "modeldata")
x <- ames[,-which(names(ames) == "Sale_Price")]
y <- ames$Sale_Price
# pretrain a tabnet model on ames dataset
ames_pretrain <- tabnet_pretrain(x, y, epoch = 2, checkpoint_epochs = 1)
# prune classification head to get an embedding model
pruned_pretrain <- torch::nn_prune_head(ames_pretrain, 1)

```

---

nnf_mc_loss	<i>Max-Constraint Margin Loss (functional)</i>
-------------	--

---

### Description

Computes the hierarchy-constrained loss for multi-label classification. Enforces that if a class is predicted positive, all its ancestors must also be positive, using the ancestor matrix R.

### Usage

```
nnf_mc_loss(
    output,
    target,
    R,
    to_eval = NULL,
    criterion = nnf_binary_cross_entropy_with_logits
)
```

### Arguments

output	A torch_tensor of raw network outputs (pre-sigmoid), shape (batch_size, n_classes).
target	Binary target labels, shape (batch_size, n_classes).
R	Ancestor matrix tensor of shape (1, n_classes, n_classes) where $R[1, i, j] = 1$ iff class i is a descendant of class j.
to_eval	Optional logical tensor of shape (n_classes,) indicating which classes to include in the loss computation. If NULL, all classes are evaluated.
criterion	Loss function to apply after constraint propagation. Default: nnf_binary_cross_entropy_with_logits (expects raw logits).

### Details

The loss combines constrained outputs differently for positive and negative labels:

- For positive labels: uses constrained output of label-weighted predictions
- For negative labels: uses constrained raw predictions (penalizes ancestor violations)

### Value

A scalar torch\_tensor containing the computed loss, or a tensor of shape (batch\_size, n\_classes) if reduction = "none".

### See Also

[nn\\_mc\\_loss\(\)](#), [get\\_constr\\_output\(\)](#)

---

 nnf\_multilabel\_one\_hot

*Convert class\_id tensor to binary one-hot tensor*


---

### Description

Transforms a tensor of class indices (one column per hierarchy level) into a binary tensor where each column corresponds to a class.

### Usage

```
nnf_multilabel_one_hot(y, outcomes, device = "cpu")
```

### Arguments

y	A torch_tensor of shape (batch_size, n_levels) containing 1-based class indices.
outcomes	A tibble with factor columns (as from hardhat::mold())\$outcomes).
device	Torch device.

### Value

A torch\_tensor of shape (batch\_size, n\_classes) with binary values.

---

 node\_to\_df

*Turn a Node object into predictor and outcome.*


---

### Description

Turn a Node object into predictor and outcome.

### Usage

```
node_to_df(x, drop_last_level = TRUE)
```

### Arguments

x	Node object
drop_last_level	TRUE unused

### Value

a named list of x and y, being respectively the predictor data-frame and the outcomes data-frame, as expected inputs for hardhat::mold() function.

**Examples**

```
library(dplyr)
library(data.tree)
data(starwars)
starwars_tree <- starwars %>%
  mutate(pathString = paste("tree", species, homeworld, `name`, sep = "/")) %>%
  as.Node()
node_to_df(starwars_tree)$x %>% head()
node_to_df(starwars_tree)$y %>% head()
```

---

predict.tabnet\_fit      *Predict using tabnet*

---

**Description**

Predict using tabnet

**Usage**

```
## S3 method for class 'tabnet_fit'
predict(object, new_data, type = NULL, ..., epoch = NULL)

## S3 method for class 'tabnet_fit'
augment(x, new_data, ...)
```

**Arguments**

object, x	A tabnet_fit object.
new_data	A data frame or matrix of new predictors.
type	expected outcome type within c("numeric", "prob", "class").
...	Not used, but required for extensibility.
epoch	the epoch of an existing checkpoint to infer from.

**Value**

`predict()` returns a tibble of predictions and `augment()` appends the columns in `new_data`. In either case, the number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`.

For regression data, the prediction is in the column `.pred`. For classification, the class predictions are in `.pred_class` and the probability estimates are in columns with the pattern `.pred_{level}` where `level` is the levels of the outcome factor vector.

**Examples**

```

# Minimal example for quick execution
car_split <- rsample::initial_split(mtcars[ 1:6, ])

## Not run:
# Fit
if (torch_is_installed() & interactive()) {
  mod <- tabnet_fit(mpg ~ cyl + log(drat), training(car_split))

  # Predict
  predict(mod, testing(car_split))
  augment(mod, testing(car_split))
}

## End(Not run)

```

---

 sparsemax

*Sparsemax*


---

**Description**

Normalizing sparse transform (a la softmax).

**Usage**

```
sparsemax(dim = -1L)
```

```
sparsemax15(dim = -1L, k = NULL)
```

**Arguments**

dim	The dimension along which to apply sparsemax.
k	The number of largest elements to partial-sort input over. For optimal performance, k should be slightly bigger than the expected number of non-zeros in the solution. If the solution is more than k-sparse, this function is recursively called with a 2*k schedule. If NULL, full sorting is performed from the beginning.

**Details**

Solves the projection:

$$\min_P \|input - P\|_2 \text{ s.t. } P \geq 0, \sum(P) == 1$$

**Value**

The projection result, such that  $\sum_{dim} P = 1 \forall dim$  elementwise.

## Examples

```
## Not run:
input <- torch::torch_randn(10, 5, requires_grad = TRUE)
# create a top3 alpha=1.5 sparsemax on last input dimension
nn_sparsemax <- sparsemax15(dim=1, k=3)
result <- nn_sparsemax(input)
print(result)

## End(Not run)
```

---

tabnet

*Parsnip compatible tabnet model*

---

## Description

Parsnip compatible tabnet model

## Usage

```
tabnet(
  mode = "unknown",
  cat_emb_dim = NULL,
  decision_width = NULL,
  attention_width = NULL,
  num_steps = NULL,
  mask_type = NULL,
  mask_topk = NULL,
  num_independent = NULL,
  num_shared = NULL,
  num_independent_decoder = NULL,
  num_shared_decoder = NULL,
  penalty = NULL,
  feature_reusage = NULL,
  momentum = NULL,
  epochs = NULL,
  batch_size = NULL,
  virtual_batch_size = NULL,
  learn_rate = NULL,
  optimizer = NULL,
  loss = NULL,
  clip_value = NULL,
  drop_last = NULL,
  lr_scheduler = NULL,
  rate_decay = NULL,
  rate_step_size = NULL,
  checkpoint_epochs = NULL,
```

```

    verbose = NULL,
    importance_sample_size = NULL,
    early_stopping_monitor = NULL,
    early_stopping_tolerance = NULL,
    early_stopping_patience = NULL,
    skip_importance = NULL,
    tabnet_model = NULL,
    from_epoch = NULL
)

```

## Arguments

mode	A single character string for the type of model. Possible values for this model are "unknown", "regression", or "classification".
cat_emb_dim	Size of the embedding of categorical features. If int, all categorical features will have same embedding size, if list of int, every corresponding feature will have specific embedding size.
decision_width	(int) Width of the decision prediction layer. Bigger values gives more capacity to the model with the risk of overfitting. Values typically range from 8 to 64.
attention_width	(int) Width of the attention embedding for each mask. According to the paper $n_d = n_a$ is usually a good choice. (default=8)
num_steps	(int) Number of steps in the architecture (usually between 3 and 10)
mask_type	(character) Final layer of feature selector in the attentive_transformer block, either "sparsemax", "entmax" or "entmax15". Defaults to "sparsemax".
mask_topk	(int) mask sparsity top-k for sparsemax15 and entmax15. See <a href="#">entmax15()</a> for detail.
num_independent	Number of independent Gated Linear Units layers at each step of the encoder. Usual values range from 1 to 5.
num_shared	Number of shared Gated Linear Units at each step of the encoder. Usual values at each step of the decoder. range from 1 to 5
num_independent_decoder	For pretraining, number of independent Gated Linear Units layers Usual values range from 1 to 5.
num_shared_decoder	For pretraining, number of shared Gated Linear Units at each step of the decoder. Usual values range from 1 to 5.
penalty	This is the extra sparsity loss coefficient as proposed in the original paper. The bigger this coefficient is, the sparser your model will be in terms of feature selection. Depending on the difficulty of your problem, reducing this value could help (default 1e-3).
feature_reusage	(num) This is the coefficient for feature reusage in the masks. A value close to 1 will make mask selection least correlated between layers. Values range from 1 to 2.

momentum	Momentum for batch normalization, typically ranges from 0.01 to 0.4 (default=0.02)
epochs	(int) Number of training epochs.
batch_size	(int) Number of examples per batch, large batch sizes are recommended. (default: 1024^2)
virtual_batch_size	(int) Size of the mini batches used for "Ghost Batch Normalization" (default=256^2)
learn_rate	initial learning rate for the optimizer.
optimizer	the optimization method. currently only "adam" is supported, you can also pass any torch optimizer function.
loss	(character or function) Loss function for training (default to mse for regression and cross entropy for classification)
clip_value	If a num is given this will clip the gradient at clip_value. Pass NULL to not clip.
drop_last	(logical) Whether to drop last batch if not complete during training
lr_scheduler	if NULL, no learning rate decay is used. If "step" decays the learning rate by lr_decay every step_size epochs. If "reduce_on_plateau" decays the learning rate by lr_decay when no improvement after step_size epochs. It can also be a <a href="#">torch:lr_scheduler</a> function that only takes the optimizer as parameter. The step method is called once per epoch.
rate_decay	multiplies the initial learning rate by rate_decay every rate_step_size epochs. Unused if lr_scheduler is a torch::lr_scheduler or NULL.
rate_step_size	the learning rate scheduler step size. Unused if lr_scheduler is a torch::lr_scheduler or NULL.
checkpoint_epochs	checkpoint model weights and architecture every checkpoint_epochs. (default is 10). This may cause large memory usage. Use 0 to disable checkpoints.
verbose	(logical) Whether to print progress and loss values during training.
importance_sample_size	sample of the dataset to compute importance metrics. If the dataset is larger than 1e5 obs we will use a sample of size 1e5 and display a warning.
early_stopping_monitor	Metric to monitor for early_stopping. One of "valid_loss", "train_loss" or "auto" (defaults to "auto").
early_stopping_tolerance	Minimum relative improvement to reset the patience counter. 0.01 for 1% tolerance (default 0)
early_stopping_patience	Number of epochs without improving until stopping training. (default=5)
skip_importance	if feature importance calculation should be skipped (default: FALSE)
tabnet_model	A previously fitted tabnet_model object to continue the fitting on. if NULL (the default) a brand new model is initialized.
from_epoch	When a tabnet_model is provided, restore the network weights from a specific epoch. Default is last available checkpoint for restored model, or last epoch for in-memory model.

**Value**

A TabNet parsnip instance. It can be used to fit tabnet models using parsnip machinery.

**Threading**

TabNet uses torch as its backend for computation and torch uses all available threads by default.

You can control the number of threads used by torch with:

```
torch::torch_set_num_threads(1)
torch::torch_set_num_interop_threads(1)
```

**See Also**

tabnet\_fit

**Examples**

```
library(parsnip)
data("ames", package = "modeldata")
model <- tabnet() %>%
  set_mode("regression") %>%
  set_engine("torch")
model %>%
  fit(Sale_Price ~ ., data = ames)
```

---

tabnet\_config

*Configuration for TabNet models*

---

**Description**

Configuration for TabNet models

**Usage**

```
tabnet_config(
  batch_size = 1024^2,
  penalty = 0.001,
  clip_value = NULL,
  loss = "auto",
  epochs = 5,
  drop_last = FALSE,
  decision_width = NULL,
  attention_width = NULL,
  num_steps = 3,
  feature_reusage = 1.3,
  mask_type = "sparsemax",
```

```

mask_topk = NULL,
virtual_batch_size = 256^2,
valid_split = 0,
learn_rate = 0.02,
optimizer = "adam",
lr_scheduler = NULL,
lr_decay = 0.1,
step_size = 30,
checkpoint_epochs = 10,
cat_emb_dim = 1,
num_independent = 2,
num_shared = 2,
num_independent_decoder = 1,
num_shared_decoder = 1,
momentum = 0.02,
pretraining_ratio = 0.5,
verbose = FALSE,
device = "auto",
importance_sample_size = NULL,
early_stopping_monitor = "auto",
early_stopping_tolerance = 0,
early_stopping_patience = 0L,
num_workers = 0L,
skip_importance = FALSE
)

```

### Arguments

batch_size	(int) Number of examples per batch, large batch sizes are recommended. (default: 1024^2)
penalty	This is the extra sparsity loss coefficient as proposed in the original paper. The bigger this coefficient is, the sparser your model will be in terms of feature selection. Depending on the difficulty of your problem, reducing this value could help (default 1e-3).
clip_value	If a num is given this will clip the gradient at clip_value. Pass NULL to not clip.
loss	(character or function) Loss function for training (default to mse for regression and cross entropy for classification)
epochs	(int) Number of training epochs.
drop_last	(logical) Whether to drop last batch if not complete during training
decision_width	(int) Width of the decision prediction layer. Bigger values gives more capacity to the model with the risk of overfitting. Values typically range from 8 to 64.
attention_width	(int) Width of the attention embedding for each mask. According to the paper $n_d = n_a$ is usually a good choice. (default=8)
num_steps	(int) Number of steps in the architecture (usually between 3 and 10)

feature_reusage	(num) This is the coefficient for feature reusage in the masks. A value close to 1 will make mask selection least correlated between layers. Values range from 1 to 2.
mask_type	(character) Final layer of feature selector in the attentive_transformer block, either "sparsemax", "entmax" or "entmax15". Defaults to "sparsemax".
mask_topk	(int) mask sparsity top-k for sparsemax15 and entmax15. See <a href="#">entmax15()</a> for detail.
virtual_batch_size	(int) Size of the mini batches used for "Ghost Batch Normalization" (default=256^2)
valid_split	In [0, 1). The fraction of the dataset used for validation. (default = 0 means no split)
learn_rate	initial learning rate for the optimizer.
optimizer	the optimization method. currently only "adam" is supported, you can also pass any torch optimizer function.
lr_scheduler	if NULL, no learning rate decay is used. If "step" decays the learning rate by lr_decay every step_size epochs. If "reduce_on_plateau" decays the learning rate by lr_decay when no improvement after step_size epochs. It can also be a <a href="#">torch::lr_scheduler</a> function that only takes the optimizer as parameter. The step method is called once per epoch.
lr_decay	multiplies the initial learning rate by lr_decay every step_size epochs. Unused if lr_scheduler is a torch::lr_scheduler or NULL.
step_size	the learning rate scheduler step size. Unused if lr_scheduler is a torch::lr_scheduler or NULL.
checkpoint_epochs	checkpoint model weights and architecture every checkpoint_epochs. (default is 10). This may cause large memory usage. Use 0 to disable checkpoints.
cat_emb_dim	Size of the embedding of categorical features. If int, all categorical features will have same embedding size, if list of int, every corresponding feature will have specific embedding size.
num_independent	Number of independent Gated Linear Units layers at each step of the encoder. Usual values range from 1 to 5.
num_shared	Number of shared Gated Linear Units at each step of the encoder. Usual values at each step of the decoder. range from 1 to 5
num_independent_decoder	For pretraining, number of independent Gated Linear Units layers Usual values range from 1 to 5.
num_shared_decoder	For pretraining, number of shared Gated Linear Units at each step of the decoder. Usual values range from 1 to 5.
momentum	Momentum for batch normalization, typically ranges from 0.01 to 0.4 (default=0.02)
pretraining_ratio	Ratio of features to mask for reconstruction during pretraining. Ranges from 0 to 1 (default=0.5)

verbose	(logical) Whether to print progress and loss values during training.
device	the device to use for training. "cpu" or "cuda". The default ("auto") uses to "cuda" if it's available, otherwise uses "cpu".
importance_sample_size	sample of the dataset to compute importance metrics. If the dataset is larger than 1e5 obs we will use a sample of size 1e5 and display a warning.
early_stopping_monitor	Metric to monitor for early_stopping. One of "valid_loss", "train_loss" or "auto" (defaults to "auto").
early_stopping_tolerance	Minimum relative improvement to reset the patience counter. 0.01 for 1% tolerance (default 0)
early_stopping_patience	Number of epochs without improving until stopping training. (default=5)
num_workers	(int, optional): how many subprocesses to use for data loading. 0 means that the data will be loaded in the main process. (default: 0)
skip_importance	if feature importance calculation should be skipped (default: FALSE)

**Value**

A named list with all hyperparameters of the TabNet implementation.

**Examples**

```
data("ames", package = "modeldata")

# change the model config for an faster ignite optimizer
config <- tabnet_config(optimizer = torch::optim_ignite_adamw)

## Single-outcome regression using formula specification
fit <- tabnet_fit(Sale_Price ~ ., data = ames, epochs = 1, config = config)
```

---

tabnet_explain	<i>Interpretation metrics from a TabNet model</i>
----------------	---

---

**Description**

Interpretation metrics from a TabNet model

**Usage**

```
tabnet_explain(object, new_data)

## Default S3 method:
tabnet_explain(object, new_data)

## S3 method for class 'tabnet_fit'
tabnet_explain(object, new_data)

## S3 method for class 'tabnet_pretrain'
tabnet_explain(object, new_data)

## S3 method for class 'model_fit'
tabnet_explain(object, new_data)
```

**Arguments**

object	a TabNet fit object
new_data	a data.frame to obtain interpretation metrics.

**Value**

Returns a list with

- `M_explain`: the aggregated feature importance masks as detailed in TabNet's paper.
- `masks` a list containing the masks for each step.

**Examples**

```
set.seed(2021)

n <- 256
x <- data.frame(
  x = rnorm(n),
  y = rnorm(n),
  z = rnorm(n)
)

y <- x$x

fit <- tabnet_fit(x, y, epochs = 10,
                 num_steps = 1,
                 batch_size = 512,
                 attention_width = 1,
                 num_shared = 1,
                 num_independent = 1)

ex <- tabnet_explain(fit, x)
```

---

tabnet_fit	<i>Tabnet model</i>
------------	---------------------

---

## Description

Fits the **TabNet: Attentive Interpretable Tabular Learning** model

## Usage

```
tabnet_fit(x, ...)  
  
## Default S3 method:  
tabnet_fit(x, ...)  
  
## S3 method for class 'data.frame'  
tabnet_fit(  
  x,  
  y,  
  tabnet_model = NULL,  
  config = tabnet_config(),  
  ...,  
  from_epoch = NULL,  
  weights = NULL  
)  
  
## S3 method for class 'formula'  
tabnet_fit(  
  formula,  
  data,  
  tabnet_model = NULL,  
  config = tabnet_config(),  
  ...,  
  from_epoch = NULL,  
  weights = NULL  
)  
  
## S3 method for class 'recipe'  
tabnet_fit(  
  x,  
  data,  
  tabnet_model = NULL,  
  config = tabnet_config(),  
  ...,  
  from_epoch = NULL,
```

```

    weights = NULL
  )

  ## S3 method for class 'Node'
  tabnet_fit(
    x,
    tabnet_model = NULL,
    config = tabnet_config(),
    ...,
    from_epoch = NULL
  )

```

### Arguments

x	<p>Depending on the context:</p> <ul style="list-style-type: none"> <li>• A <b>data frame</b> of predictors.</li> <li>• A <b>matrix</b> of predictors.</li> <li>• A <b>recipe</b> specifying a set of preprocessing steps created from <code>recipes::recipe()</code>.</li> <li>• A <b>Node</b> where tree will be used as hierarchical outcome, and attributes will be used as predictors.</li> </ul> <p>The predictor data should be standardized (e.g. centered or scaled). The model treats categorical predictors internally thus, you don't need to make any treatment. The model treats missing values internally thus, you don't need to make any treatment.</p>
...	Model hyperparameters. Any hyperparameters set here will update those set by the config argument. See <code>tabnet_config()</code> for a list of all possible hyperparameters.
y	<p>When x is a <b>data frame</b> or <b>matrix</b>, y is the outcome specified as:</p> <ul style="list-style-type: none"> <li>• A <b>data frame</b> with 1 or many numeric column (regression) or 1 or many categorical columns (classification) .</li> <li>• A <b>matrix</b> with 1 column.</li> <li>• A <b>vector</b>, either numeric or categorical.</li> </ul>
tabnet_model	A previously fitted <code>tabnet_model</code> object to continue the fitting on. if NULL (the default) a brand new model is initialized.
config	A set of hyperparameters created using the <code>tabnet_config</code> function. If no argument is supplied, this will use the default values in <code>tabnet_config()</code> .
from_epoch	When a <code>tabnet_model</code> is provided, restore the network weights from a specific epoch. Default is last available checkpoint for restored model, or last epoch for in-memory model.
weights	Unused. Placeholder for <code>hardhat::importance_weight()</code> variables.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	<p>When a <b>recipe</b> or <b>formula</b> is used, data is specified as:</p> <ul style="list-style-type: none"> <li>• A <b>data frame</b> containing both the predictors and the outcome.</li> </ul>

**Value**

A TabNet model object. It can be used for serialization, predictions, or further fitting.

**Fitting a pre-trained model**

When providing a parent `tabnet_model` parameter, the model fitting resumes from that model weights at the following epoch:

- last fitted epoch for a model already in torch context
  - Last model checkpoint epoch for a model loaded from file
  - the epoch related to a checkpoint matching or preceding the `from_epoch` value if provided
- The model fitting metrics append on top of the parent metrics in the returned TabNet model.

**Multi-outcome**

TabNet allows multi-outcome prediction, which is usually named **multi-label classification** or multi-output regression when outcomes are numerical. Multi-outcome currently expect outcomes to be either all numeric or all categorical.

**Threading**

TabNet uses torch as its backend for computation and torch uses all available threads by default.

You can control the number of threads used by torch with:

```
torch::torch_set_num_threads(1)
torch::torch_set_num_interop_threads(1)
```

**Examples**

```
## Not run:
data("ames", package = "modeldata")
data("attrition", package = "modeldata")

## Single-outcome regression using formula specification
fit <- tabnet_fit(Sale_Price ~ ., data = ames, epochs = 4)

## Single-outcome classification using data-frame specification
attrition_x <- attrition[ids, -which(names(attrition) == "Attrition")]
fit <- tabnet_fit(attrition_x, attrition$Attrition, epochs = 4, verbose = TRUE)

## Multi-outcome regression on `Sale_Price` and `Pool_Area` in `ames` dataset using formula,
ames_fit <- tabnet_fit(Sale_Price + Pool_Area ~ ., data = ames, epochs = 4, valid_split = 0.2)

## Multi-label classification on `Attrition` and `JobSatisfaction` in
## `attrition` dataset using recipe
library(recipes)
rec <- recipe(Attrition + JobSatisfaction ~ ., data = attrition) %>%
  step_normalize(all_numeric(), -all_outcomes())

attrition_fit <- tabnet_fit(rec, data = attrition, epochs = 4, valid_split = 0.2)
```

```
## Hierarchical classification on `acme`
data(acme, package = "data.tree")

acme_fit <- tabnet_fit(acme, epochs = 4, verbose = TRUE)

# Note: Model's number of epochs should be increased for publication-level results.

## End(Not run)
```

---

tabnet\_nn

*TabNet Model Architecture*


---

### Description

This is a nn\_module representing the TabNet architecture from [Attentive Interpretable Tabular Deep Learning](#).

### Usage

```
tabnet_nn(
  input_dim,
  output_dim,
  n_d = 8,
  n_a = 8,
  n_steps = 3,
  gamma = 1.3,
  cat_idxes = c(),
  cat_dims = c(),
  cat_emb_dim = 1,
  n_independent = 2,
  n_shared = 2,
  epsilon = 1e-15,
  virtual_batch_size = 128,
  momentum = 0.02,
  mask_type = "sparsemax",
  mask_topk = NULL
)
```

### Arguments

input_dim	Initial number of features.
output_dim	Dimension of network output. Examples : one for regression, 2 for binary classification etc.. Vector of those dimensions in case of multi-output.
n_d	Dimension of the prediction layer (usually between 4 and 64).
n_a	Dimension of the attention layer (usually between 4 and 64).

n_steps	Number of successive steps in the network (usually between 3 and 10).
gamma	Scaling factor for attention updates (usually between 1 and 2).
cat_idxes	Index of each categorical column in the dataset.
cat_dims	Number of categories in each categorical column.
cat_emb_dim	Size of the embedding of categorical features if int, all categorical features will have same embedding size if list of int, every corresponding feature will have specific size.
n_independent	Number of independent GLU layer in each GLU block of the encoder.
n_shared	Number of shared GLU layer in each GLU block of the encoder.
epsilon	Avoid log(0), this should be kept very low.
virtual_batch_size	Batch size for Ghost Batch Normalization.
momentum	Numerical value between 0 and 1 which will be used for momentum in all batch norm.
mask_type	Either "sparsemax", "entmax" or "entmax15": the sparse masking function to use.
mask_topk	the mask top-k value for k-sparsity selection in the mask for sparsemax and entmax15. defaults to 1/4 of last input_dim if NULL. See <a href="#">entmax15</a> for details.

---

tabnet_pretrain	<i>Tabnet model</i>
-----------------	---------------------

---

## Description

Pretrain the **TabNet: Attentive Interpretable Tabular Learning** model on the predictor data exclusively (unsupervised training).

## Usage

```

tabnet_pretrain(x, ...)

## Default S3 method:
tabnet_pretrain(x, ...)

## S3 method for class 'data.frame'
tabnet_pretrain(
  x,
  y = NULL,
  tabnet_model = NULL,
  config = tabnet_config(),
  ...,
  from_epoch = NULL
)

```

```

## S3 method for class 'formula'
tabnet_pretrain(
  formula,
  data,
  tabnet_model = NULL,
  config = tabnet_config(),
  ...,
  from_epoch = NULL
)

## S3 method for class 'recipe'
tabnet_pretrain(
  x,
  data,
  tabnet_model = NULL,
  config = tabnet_config(),
  ...,
  from_epoch = NULL
)

## S3 method for class 'Node'
tabnet_pretrain(
  x,
  tabnet_model = NULL,
  config = tabnet_config(),
  ...,
  from_epoch = NULL
)

```

## Arguments

x	<p>Depending on the context:</p> <ul style="list-style-type: none"> <li>• A <b>data frame</b> of predictors.</li> <li>• A <b>matrix</b> of predictors.</li> <li>• A <b>recipe</b> specifying a set of preprocessing steps created from <code>recipes::recipe()</code>.</li> <li>• A <b>Node</b> where tree leaves will be left out, and attributes will be used as predictors.</li> </ul> <p>The predictor data should be standardized (e.g. centered or scaled). The model treats categorical predictors internally thus, you don't need to make any treatment. The model treats missing values internally thus, you don't need to make any treatment.</p>
...	Model hyperparameters. Any hyperparameters set here will update those set by the config argument. See <code>tabnet_config()</code> for a list of all possible hyperparameters.
y	(optional) When x is a <b>data frame</b> or <b>matrix</b> , y is the outcome
tabnet_model	A pretrained tabnet_model object to continue the fitting on. if NULL (the default) a brand new model is initialized.

config	A set of hyperparameters created using the <code>tabnet_config</code> function. If no argument is supplied, this will use the default values in <code>tabnet_config()</code> .
from_epoch	When a <code>tabnet_model</code> is provided, restore the network weights from a specific epoch. Default is last available checkpoint for restored model, or last epoch for in-memory model.
formula	A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
data	When a <b>recipe</b> or <b>formula</b> is used, data is specified as: <ul style="list-style-type: none"> <li>• A <b>data frame</b> containing both the predictors and the outcome.</li> </ul>

**Value**

A TabNet model object. It can be used for serialization, predictions, or further fitting.

**outcome**

Outcome value are accepted here only for consistent syntax with `tabnet_fit`, but by design the outcome, if present, is ignored during pre-training.

**pre-training from a previous model**

When providing a parent `tabnet_model` parameter, the model pretraining resumes from that model weights at the following epoch:

- last pretrained epoch for a model already in torch context
  - Last model checkpoint epoch for a model loaded from file
  - the epoch related to a checkpoint matching or preceding the `from_epoch` value if provided
- The model pretraining metrics append on top of the parent metrics in the returned TabNet model.

**Threading**

TabNet uses torch as its backend for computation and torch uses all available threads by default.

You can control the number of threads used by torch with:

```
torch::torch_set_num_threads(1)
torch::torch_set_num_interop_threads(1)
```

**Examples**

```
data("ames", package = "modeldata")
pretrained <- tabnet_pretrain(Sale_Price ~ ., data = ames, epochs = 1)
```

# Index

attention\_width, 3  
augment(), 15  
augment.tabnet\_fit  
    (predict.tabnet\_fit), 15  
autoplot.tabnet\_explain, 4  
autoplot.tabnet\_fit, 5  
autoplot.tabnet\_pretrain  
    (autoplot.tabnet\_fit), 5  
  
build\_ancestor\_matrix\_from\_outcomes, 6  
build\_ancestor\_matrix\_from\_outcomes(),  
    11  
  
cat\_emb\_dim, 6  
check\_compliant\_node, 7  
checkpoint\_epochs (cat\_emb\_dim), 6  
  
decision\_width (attention\_width), 3  
drop\_last (cat\_emb\_dim), 6  
  
encoder\_activation (cat\_emb\_dim), 6  
entmax, 8  
entmax15, 29  
entmax15 (entmax), 8  
entmax15(), 18, 22  
  
feature\_reusage (attention\_width), 3  
  
get\_constr\_output, 9  
get\_constr\_output(), 11, 13  
get\_tau, 9  
  
lr\_scheduler (cat\_emb\_dim), 6  
  
mask\_type (attention\_width), 3  
mlp\_activation (cat\_emb\_dim), 6  
mlp\_hidden\_multiplier (cat\_emb\_dim), 6  
momentum (attention\_width), 3  
  
nn\_aum\_loss, 10  
nn\_mc\_loss, 11  
nn\_mc\_loss(), 13  
nn\_prune\_head.tabnet\_fit, 12  
nn\_prune\_head.tabnet\_pretrain  
    (nn\_prune\_head.tabnet\_fit), 12  
nnf\_mc\_loss, 13  
nnf\_mc\_loss(), 11  
nnf\_multilabel\_one\_hot, 14  
node\_to\_df, 14  
num\_independent (attention\_width), 3  
num\_independent\_decoder (cat\_emb\_dim), 6  
num\_shared (attention\_width), 3  
num\_shared\_decoder (cat\_emb\_dim), 6  
num\_steps (attention\_width), 3  
  
optimizer (cat\_emb\_dim), 6  
  
penalty (cat\_emb\_dim), 6  
predict(), 15  
predict.tabnet\_fit, 15  
  
recipes::recipe(), 26, 30  
  
sparsemax, 16  
sparsemax15 (sparsemax), 16  
  
tabnet, 17  
tabnet\_config, 20  
tabnet\_config(), 26, 30, 31  
tabnet\_explain, 23  
tabnet\_explain(), 4  
tabnet\_fit, 25  
tabnet\_fit(), 5  
tabnet\_nn, 28  
tabnet\_pretrain, 29  
tabnet\_pretrain(), 5  
torch::lr\_scheduler, 19, 22  
  
verbose (cat\_emb\_dim), 6  
virtual\_batch\_size (cat\_emb\_dim), 6