Package: mlr3torch (via r-universe)

February 13, 2025

Title Deep Learning with 'mlr3'

Version 0.2.1

Description Deep Learning library that extends the mlr3 framework by building upon the 'torch' package. It allows to conveniently build, train, and evaluate deep learning models without having to worry about low level details. Custom architectures can be created using the graph language defined in 'mlr3pipelines'.

License LGPL (>= 3)

BugReports https://github.com/mlr-org/mlr3torch/issues

URL https://mlr3torch.mlr-org.com/, https://github.com/mlr-org/mlr3torch/

Depends mlr3 (>= 0.20.0), mlr3pipelines (>= 0.6.0), torch (>= 0.14.0), R (>= 3.5.0)

Imports backports, checkmate (>= 2.2.0), data.table, lgr, methods, mlr3misc (>= 0.14.0), paradox (>= 1.0.0), R6, withr

Suggests callr, curl, future, ggplot2, igraph, jsonlite, knitr, mlr3tuning (>= 1.0.0), progress, rmarkdown, rpart, viridis, visNetwork, testthat (>= 3.0.0), tfevents, torchvision (>= 0.6.0), waldo

Config/testthat/edition 3

NeedsCompilation no

ByteCompile yes

Encoding UTF-8

RoxygenNote 7.3.2

Collate 'CallbackSet.R' 'aaa.R' 'TorchCallback.R' 'CallbackSetCheckpoint.R' 'CallbackSetEarlyStopping.R' 'CallbackSetHistory.R' 'CallbackSetLRScheduler.R' 'CallbackSetProgress.R' 'CallbackSetTB.R' 'CallbackSetUnfreeze.R' 'ContextTorch.R' 'DataBackendLazy.R' 'utils.R' 'DataDescriptor.R' 'LearnerTorch.R' 'LearnerTorchFeatureless.R' 'LearnerTorchImage.R'

'LearnerTorchMLP.R' 'task_dataset.R' 'shape.R'
'PipeOpTorchIngress.R' 'LearnerTorchModel.R'
'LearnerTorchTabResNet.R' 'LearnerTorchVision.R'
'ModelDescriptor.R' 'PipeOpModule.R' 'PipeOpTorch.R'
'PipeOpTaskPreprocTorch.R' 'PipeOpTorchActivation.R'
'PipeOpTorchAdaptiveAvgPool.R' 'PipeOpTorchAvgPool.R'
'PipeOpTorchBatchNorm.R' 'PipeOpTorchBlock.R'
'PipeOpTorchCallbacks.R' 'PipeOpTorchConv.R'
'PipeOpTorchConvTranspose.R' 'PipeOpTorchDropout.R'
'PipeOpTorchHead.R' 'PipeOpTorchLayerNorm.R'
'PipeOpTorchLinear.R' 'TorchLoss.R' 'PipeOpTorchLoss.R'
'PipeOpTorchMaxPool.R' 'PipeOpTorchMerge.R'
'PipeOpTorchModel.R' 'PipeOpTorchOptimizer.R'
'PipeOpTorchReshape.R' 'PipeOpTorchSoftmax.R' 'Select.R'
'TaskClassif_cifar.R' 'TaskClassif_lazy_iris.R'
'TaskClassif_melanoma.R' 'TaskClassif_mnist.R'
'TaskClassif_tiny_imagenet.R' 'TorchDescriptor.R'
'TorchOptimizer.R' 'bibentries.R' 'cache.R' 'lazy_tensor.R'
'learner_torch_methods.R' 'materialize.R' 'merge_graphs.R'
'multi_tensor_dataset.R' 'nn.R' 'nn_graph.R'
'paramset_torchlearner.R' 'preprocess.R' 'rd_info.R'
'with_torch_settings.R' 'zzz.R'

Author Sebastian Fischer [cre, aut]
(<https://orcid.org/0000-0002-9609-3197>), Bernd Bischl [ctb]
(<https://orcid.org/0000-0001-6002-6980>), Lukas Burk [ctb]
(<https://orcid.org/0000-0001-7528-3795>), Martin Binder [aut],
Florian Pfisterer [ctb]
(<https://orcid.org/0000-0001-8867-762X>), Carson Zhang [ctb]

Maintainer Sebastian Fischer <sebf.fischer@gmail.com>

Repository CRAN

Date/Publication 2025-02-13 16:00:02 UTC

Contents

`mlr3torch-package` *mlr3torch: Deep Learning with 'mlr3'*

Description

Deep Learning library that extends the mlr3 framework by building upon the 'torch' package. It allows to conveniently build, train, and evaluate deep learning models without having to worry about low level details. Custom architectures can be created using the graph language defined in 'mlr3pipelines'.

Options

- `mlr3torch.cache`: Whether to cache the downloaded data (`TRUE`) or not (`FALSE`, default). This can also be set to a specific folder on the file system to be used as the cache directory.

Author(s)

Maintainer: Sebastian Fischer <sebf.fischer@gmail.com> (ORCID)

Authors:

- Martin Binder <mlr.developer@mb706.com>

Other contributors:

- Bernd Bischl <bernd_bischl@gmx.net> (ORCID) [contributor]
- Lukas Burk <github@quantenbrot.de> (ORCID) [contributor]
- Florian Pfisterer <pfistererf@googlemail.com> (ORCID) [contributor]
- Carson Zhang <carsonzhang4@gmail.com> [contributor]

See Also

Useful links:

- https://mlr3torch.mlr-org.com/
- https://github.com/mlr-org/mlr3torch/
- Report bugs at https://github.com/mlr-org/mlr3torch/issues

assert_lazy_tensor *Assert Lazy Tensor*

Description

Asserts whether something is a lazy tensor.

Usage

```
assert_lazy_tensor(x)
```

Arguments

x	(any) Object to check.

as_data_descriptor *Convert to Data Descriptor*

Description

Converts the input to a DataDescriptor.

Usage

```
as_data_descriptor(x, dataset_shapes, ...)
```

Arguments

x	(any) Object to convert.
dataset_shapes	(named list() of (integer() or NULL)) The shapes of the output. Names are the elements of the list returned by the dataset. If the shape is not NULL (unknown, e.g. for images of different sizes) the first dimension must be NA to indicate the batch dimension.
...	(any) Further arguments passed to the DataDescriptor constructor.

Examples

```
ds = dataset("example",
  initialize = function() self$iris = iris[, -5],
  .getitem = function(i) list(x = torch_tensor(as.numeric(self$iris[i, ]))),
  .length = function() nrow(self$iris)
)()
as_data_descriptor(ds, list(x = c(NA, 4L)))

# if the dataset has a .getbatch method, the shapes are inferred
ds2 = dataset("example",
  initialize = function() self$iris = iris[, -5],
  .getbatch = function(i) list(x = torch_tensor(as.matrix(self$iris[i, ]))),
  .length = function() nrow(self$iris)
)()
as_data_descriptor(ds2)
```

---

as_lazy_tensor *Convert to Lazy Tensor*

Description

Convert a object to a lazy_tensor.

Usage

```
as_lazy_tensor(x, ...)

## S3 method for class 'dataset'
as_lazy_tensor(x, dataset_shapes = NULL, ids = NULL, ...)
```

Arguments

x	(any) Object to convert to a lazy_tensor
...	(any) Additional arguments passed to the method.
dataset_shapes	(named list() of (integer() or NULL)) The shapes of the output. Names are the elements of the list returned by the dataset. If the shape is not NULL (unknown, e.g. for images of different sizes) the first dimension must be NA to indicate the batch dimension.
ids	(integer()) Which ids to include in the lazy tensor.

Examples

```
iris_ds = dataset("iris",
  initialize = function() {
    self$iris = iris[, -5]
  },
  .getbatch = function(i) {
    list(x = torch_tensor(as.matrix(self$iris[i, ])))
  },
  .length = function() nrow(self$iris)
)()
# no need to specify the dataset shapes as they can be inferred from the .getbatch method
# only first 5 observations
as_lazy_tensor(iris_ds, ids = 1:5)
# all observations
head(as_lazy_tensor(iris_ds))

iris_ds2 = dataset("iris",
  initialize = function() self$iris = iris[, -5],
  .getitem = function(i) list(x = torch_tensor(as.numeric(self$iris[i, ]))),
  .length = function() nrow(self$iris)
)()
# if .getitem is implemented we cannot infer the shapes as they might vary,
# so we have to annotate them explicitly
as_lazy_tensor(iris_ds2, dataset_shapes = list(x = c(NA, 4L)))[1:5]

# Convert a matrix
lt = as_lazy_tensor(matrix(rnorm(100), nrow = 20))
materialize(lt[1:5], rbind = TRUE)
```

---

as_lr_scheduler *Convert to CallbackSetLRScheduler*

Description

Convert a `torch` scheduler generator to a `CallbackSetLRScheduler`.

Usage

```
as_lr_scheduler(x, step_on_epoch)
```

Arguments

`x`	(`function`) The `torch` scheduler generator defined using `torch::lr_scheduler()`.
`step_on_epoch`	(`logical(1)`) Whether the scheduler steps after every epoch

as_torch_callback *Convert to a TorchCallback*

Description

Converts an object to a TorchCallback.

Usage

```
as_torch_callback(x, clone = FALSE, ...)
```

Arguments

x	(any) Object to be converted.
clone	(logical(1)) Whether to make a deep clone.
...	(any) Additional arguments

Value

TorchCallback.

See Also

Other Callback: TorchCallback, as_torch_callbacks(), callback_set(), mlr3torch_callbacks, mlr_callback_set, mlr_callback_set.checkpoint, mlr_callback_set.progress, mlr_callback_set.tb, mlr_callback_set.unfreeze, mlr_context_torch, t_clbk(), torch_callback()

as_torch_callbacks *Convert to a list of Torch Callbacks*

Description

Converts an object to a list of TorchCallback.

Usage

```
as_torch_callbacks(x, clone, ...)
```

Arguments

x	(any) Object to convert.
clone	(logical(1)) Whether to create a deep clone.
...	(any) Additional arguments.

Value

list() of TorchCallbacks

See Also

Other Callback: TorchCallback, as_torch_callback(), callback_set(), mlr3torch_callbacks, mlr_callback_set, mlr_callback_set.checkpoint, mlr_callback_set.progress, mlr_callback_set.tb, mlr_callback_set.unfreeze, mlr_context_torch, t_clbk(), torch_callback()

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchLoss, TorchOptimizer, as_torch_loss(), as_torch_optimizer(), mlr3torch_losses, mlr3torch_optimizers, t_clbk(), t_loss(), t_opt()

---

as_torch_loss *Convert to TorchLoss*

Description

Converts an object to a TorchLoss.

Usage

```
as_torch_loss(x, clone = FALSE, ...)
```

Arguments

x	(any) Object to convert to a TorchLoss.
clone	(logical(1)) Whether to make a deep clone.
...	(any) Additional arguments. Currently used to pass additional constructor arguments to TorchLoss for objects of type nn_loss.

Value

TorchLoss.

See Also

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchLoss, TorchOptimizer, as_torch_callbacks(), as_torch_optimizer(), mlr3torch_losses, mlr3torch_optimizers, t_clbk(), t_loss(), t_opt()

---

as_torch_optimizer *Convert to TorchOptimizer*

Description

Converts an object to a TorchOptimizer.

Usage

```
as_torch_optimizer(x, clone = FALSE, ...)
```

Arguments

x	(any) Object to convert to a TorchOptimizer.
clone	(logical(1)) Whether to make a deep clone. Default is FALSE.
...	(any) Additional arguments. Currently used to pass additional constructor arguments to TorchOptimizer for objects of type torch_optimizer_generator.

Value

TorchOptimizer

See Also

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchLoss, TorchOptimizer, as_torch_callbacks(), as_torch_loss(), mlr3torch_losses, mlr3torch_optimizers, t_clbk(), t_loss(), t_opt()

---

auto_device *Auto Device*

Description

First tries cuda, then cpu.

Usage

```
auto_device(device = NULL)
```

Arguments

device	(character(1)) The device. If not NULL, is returned as is.

---

batchgetter_categ *Batchgetter for Categorical data*

Description

Converts a data frame of categorical data into a long tensor by converting the data to integers. No input checks are performed.

Usage

```
batchgetter_categ(data, ...)
```

Arguments

data	(data.table) data.table to be converted to a tensor.
...	(any) Unused.

---

batchgetter_num *Batchgetter for Numeric Data*

Description

Converts a data frame of numeric data into a float tensor by calling as.matrix(). No input checks are performed

Usage

```
batchgetter_num(data, ...)
```

Arguments

data	(data.table()) data.table to be converted to a tensor.
...	(any) Unused.

callback_set *Create a Set of Callbacks for Torch*

Description

Creates an `R6ClassGenerator` inheriting from `CallbackSet`. Additionally performs checks such as that the stages are not accidentally misspelled. To create a `TorchCallback` use `torch_callback()`.

In order for the resulting class to be cloneable, the private method `$deep_clone()` must be provided.

Usage

```
callback_set(
  classname,
  on_begin = NULL,
  on_end = NULL,
  on_exit = NULL,
  on_epoch_begin = NULL,
  on_before_valid = NULL,
  on_epoch_end = NULL,
  on_batch_begin = NULL,
  on_batch_end = NULL,
  on_after_backward = NULL,
  on_batch_valid_begin = NULL,
  on_batch_valid_end = NULL,
  on_valid_end = NULL,
  state_dict = NULL,
  load_state_dict = NULL,
  initialize = NULL,
  public = NULL,
  private = NULL,
  active = NULL,
  parent_env = parent.frame(),
  inherit = CallbackSet,
  lock_objects = FALSE
)
```

Arguments

`classname` (`character(1)`)
The class name.

`on_begin`, `on_end`, `on_epoch_begin`, `on_before_valid`, `on_epoch_end`, `on_batch_begin`, `on_batch_end`, `on_after_backward`, `on_batch_valid_begin`, `on_batch_valid_end`, `on_valid_end`, `on_exit`
(`function`)
Function to execute at the given stage, see section *Stages*.

`state_dict` (`function()`)
The function that retrieves the state dict from the callback. This is what will be available in the learner after training.

`load_state_dict` (`function(state_dict)`)
Function that loads a callback state.

`initialize` (`function()`)
The initialization method of the callback.

`public`, `private`, `active` (`list()`)
Additional public, private, and active fields to add to the callback.

`parent_env` (`environment()`)
The parent environment for the `R6Class`.

`inherit` (`R6ClassGenerator`)
From which class to inherit. This class must either be `CallbackSet` (default) or inherit from it.

`lock_objects` (`logical(1)`)
Whether to lock the objects of the resulting `R6Class`. If `FALSE` (default), values can be freely assigned to `self` without declaring them in the class definition.

Value

`CallbackSet`

See Also

Other Callback: `TorchCallback`, `as_torch_callback()`, `as_torch_callbacks()`, `mlr3torch_callbacks`, `mlr_callback_set`, `mlr_callback_set.checkpoint`, `mlr_callback_set.progress`, `mlr_callback_set.tb`, `mlr_callback_set.unfreeze`, `mlr_context_torch`, `t_clbk()`, `torch_callback()`

DataDescriptor *Data Descriptor*

Description

A data descriptor is a rather internal data structure used in the `lazy_tensor` data type. In essence it is an annotated `torch::dataset` and a preprocessing graph (consisting mosty of `PipeOpModule` operators). The additional meta data (e.g. pointer, shapes) allows to preprocess `lazy_tensor`s in an `mlr3pipelines::Graph` just like any (non-lazy) data types. The preprocessing is applied when `materialize()` is called on the `lazy_tensor`.

To create a data descriptor, you can also use the `as_data_descriptor()` function.

Details

While it would be more natural to define this as an S3 class, we opted for an R6 class to avoid the usual trouble of serializing S3 objects. If each row contained a DataDescriptor as an S3 class, this would copy the object when serializing.

Public fields

`dataset` (`torch::dataset`)
: The dataset.

`graph` (`Graph`)
: The preprocessing graph.

`dataset_shapes` (named `list()` of (`integer()` or `NULL`))
: The shapes of the output.

`input_map` (`character()`)
: The input map from the dataset to the preprocessing graph.

`pointer` (`character(2)`)
: The output pointer.

`pointer_shape` (`integer()` | `NULL`)
: The shape of the output indicated by `pointer`.

`dataset_hash` (`character(1)`)
: Hash for the wrapped dataset.

`hash` (`character(1)`)
: Hash for the data descriptor.

`graph_input` (`character()`)
: The input channels of the preprocessing graph (cached to save time).

`pointer_shape_predict` (`integer()` or `NULL`)
: Internal use only.

Methods

Public methods:

- `DataDescriptor$new()`
- `DataDescriptor$print()`
- `DataDescriptor$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
DataDescriptor$new(
  dataset,
  dataset_shapes = NULL,
  graph = NULL,
  input_map = NULL,
  pointer = NULL,
  pointer_shape = NULL,
  pointer_shape_predict = NULL,
  clone_graph = TRUE
)
```

Arguments:

`dataset` (`torch::dataset`)
: The torch dataset. It should return a named `list()` of `torch_tensor` objects.

`dataset_shapes` (named `list()` of (`integer()` or `NULL`))
: The shapes of the output. Names are the elements of the list returned by the dataset. If the shape is not `NULL` (unknown, e.g. for images of different sizes) the first dimension must be `NA` to indicate the batch dimension.

`graph` (`Graph`)
: The preprocessing graph. If left `NULL`, no preprocessing is applied to the data and `input_map`, `pointer`, `pointer_shape`, and `pointer_shape_predict` are inferred in case the dataset returns only one element.

`input_map` (`character()`)
: Character vector that must have the same length as the input of the graph. Specifies how the data from the `dataset` is fed into the preprocessing graph.

`pointer` (`character(2)` | `NULL`)
: Points to an output channel within `graph`: Element 1 is the `PipeOp`'s id and element 2 is that `PipeOp`'s output channel.

`pointer_shape` (`integer()` | `NULL`)
: Shape of the output indicated by `pointer`.

`pointer_shape_predict` (`integer()` or `NULL`)
: Internal use only. Used in a `Graph` to anticipate possible mismatches between train and predict shapes.

`clone_graph` (`logical(1)`)
: Whether to clone the preprocessing graph.

Method `print()`**:** Prints the object

Usage:

```
DataDescriptor$print(...)
```

Arguments:

`...` (any)
: Unused

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DataDescriptor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

ModelDescriptor, lazy_tensor

Examples

```
# Create a dataset
ds = dataset(
  initialize = function() self$x = torch_randn(10, 3, 3),
  .getitem = function(i) list(x = self$x[i, ]),
  .length = function() nrow(self$x)
)()
```

```
dd = DataDescriptor$new(ds, list(x = c(NA, 3, 3)))
dd
# is the same as using the converter:
as_data_descriptor(ds, list(x = c(NA, 3, 3)))
```

---

is_lazy_tensor *Check for lazy tensor*

Description

Checks whether an object is a lazy tensor.

Usage

```
is_lazy_tensor(x)
```

Arguments

x	(any) Object to check.

---

lazy_tensor *Create a lazy tensor*

Description

Create a lazy tensor.

Usage

```
lazy_tensor(data_descriptor = NULL, ids = NULL)
```

Arguments

data_descriptor	(DataDescriptor or NULL) The data descriptor or NULL for a lazy tensor of length 0.
ids	(integer()) The elements of the data_descriptor to be included in the lazy tensor.

Examples

```
ds = dataset("example",
  initialize = function() self$iris = iris[, -5],
  .getitem = function(i) list(x = torch_tensor(as.numeric(self$iris[i, ]))),
  .length = function() nrow(self$iris)
)()
dd = as_data_descriptor(ds, list(x = c(NA, 4L)))
lt = as_lazy_tensor(dd)
```

`materialize` *Materialize Lazy Tensor Columns*

Description

This will materialize a `lazy_tensor()` or a `data.frame()` / `list()` containing – among other things – `lazy_tensor()` columns. I.e. the data described in the underlying `DataDescriptor`s is loaded for the indices in the `lazy_tensor()`, is preprocessed and then put unto the specified device. Because not all elements in a lazy tensor must have the same shape, a list of tensors is returned by default. If all elements have the same shape, these tensors can also be rbinded into a single tensor (parameter `rbind`).

Usage

```
materialize(x, device = "cpu", rbind = FALSE, ...)

## S3 method for class 'list'
materialize(x, device = "cpu", rbind = FALSE, cache = "auto", ...)
```

Arguments

`x`	(any) The object to materialize. Either a `lazy_tensor` or a `list()` / `data.frame()` containing `lazy_tensor` columns.
`device`	(`character(1)`) The torch device.
`rbind`	(`logical(1)`) Whether to rbind the lazy tensor columns (`TRUE`) or return them as a list of tensors (`FALSE`). In the second case, there is no batch dimension.
`...`	(any) Additional arguments.
`cache`	(`character(1)` or `environment()` or `NULL`) Optional cache for (intermediate) materialization results. Per default, caching will be enabled when the same dataset or data descriptor (with different output pointer) is used for more than one lazy tensor column.

Details

Materializing a lazy tensor consists of:

1. Loading the data from the internal dataset of the DataDescriptor.
2. Processing these batches in the preprocessing Graphs.
3. Returning the result of the PipeOp pointed to by the DataDescriptor (pointer).

With multiple lazy_tensor columns we can benefit from caching because: a) Output(s) from the dataset might be input to multiple graphs. b) Different lazy tensors might be outputs from the same graph.

For this reason it is possible to provide a cache environment. The hash key for a) is the hash of the indices and the dataset. The hash key for b) is the hash of the indices, dataset and preprocessing graph.

Value

(list() of lazy_tensors or a lazy_tensor)

Examples

```
lt1 = as_lazy_tensor(torch_randn(10, 3))
materialize(lt1, rbind = TRUE)
materialize(lt1, rbind = FALSE)
lt2 = as_lazy_tensor(torch_randn(10, 4))
d = data.table::data.table(lt1 = lt1, lt2 = lt2)
materialize(d, rbind = TRUE)
materialize(d, rbind = FALSE)
```

---

mlr3torch_callbacks *Dictionary of Torch Callbacks*

Description

A mlr3misc::Dictionary of torch callbacks. Use t_clbk() to conveniently retrieve callbacks. Can be converted to a data.table using as.data.table.

Usage

```
mlr3torch_callbacks
```

Format

An object of class DictionaryMlr3torchCallbacks (inherits from Dictionary, R6) of length 12.

See Also

Other Callback: TorchCallback, as_torch_callback(), as_torch_callbacks(), callback_set(), mlr_callback_set, mlr_callback_set.checkpoint, mlr_callback_set.progress, mlr_callback_set.tb, mlr_callback_set.unfreeze, mlr_context_torch, t_clbk(), torch_callback()

Other Dictionary: mlr3torch_losses, mlr3torch_optimizers, t_opt()

Examples

```
mlr3torch_callbacks$get("checkpoint")
# is the same as
t_clbk("checkpoint")
# convert to a data.table
as.data.table(mlr3torch_callbacks)
```

---

mlr3torch_losses *Loss Functions*

Description

Dictionary of torch loss descriptors. See t_loss() for conveniently retrieving a loss function. Can be converted to a data.table using as.data.table.

Usage

```
mlr3torch_losses
```

Format

An object of class DictionaryMlr3torchLosses (inherits from Dictionary, R6) of length 12.

Available Loss Functions

cross_entropy, l1, mse

See Also

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchLoss, TorchOptimizer, as_torch_callbacks(), as_torch_loss(), as_torch_optimizer(), mlr3torch_optimizers, t_clbk(), t_loss(), t_opt()

Other Dictionary: mlr3torch_callbacks, mlr3torch_optimizers, t_opt()

Examples

```
mlr3torch_losses$get("mse")
# is equivalent to
t_loss("mse")
# convert to a data.table
as.data.table(mlr3torch_losses)
```

mlr3torch_optimizers *Optimizers*

Description

Dictionary of torch optimizers. Use `t_opt` for conveniently retrieving optimizers. Can be converted to a `data.table` using `as.data.table`.

Usage

```
mlr3torch_optimizers
```

Format

An object of class `DictionaryMlr3torchOptimizers` (inherits from `Dictionary`, `R6`) of length 12.

Available Optimizers

adagrad, adam, adamw, rmsprop, sgd

See Also

Other Torch Descriptor: `TorchCallback`, `TorchDescriptor`, `TorchLoss`, `TorchOptimizer`, `as_torch_callbacks()`, `as_torch_loss()`, `as_torch_optimizer()`, `mlr3torch_losses`, `t_clbk()`, `t_loss()`, `t_opt()`

Other Dictionary: `mlr3torch_callbacks`, `mlr3torch_losses`, `t_opt()`

Examples

```
mlr3torch_optimizers$get("adam")
# is equivalent to
t_opt("adam")
# convert to a data.table
as.data.table(mlr3torch_optimizers)
```

mlr_backends_lazy *Lazy Data Backend*

Description

This lazy data backend wraps a constructor that lazily creates another backend, e.g. by downloading (and caching) some data from the internet. This backend should be used, when some metadata of the backend is known in advance and should be accessible before downloading the actual data. When the backend is first constructed, it is verified that the provided metadata was correct, otherwise an informative error message is thrown. After the construction of the lazily constructed backend, calls like `$data()`, `$missings()`, `$distinct()`, or `$hash()` are redirected to it.

Information that is available before the backend is constructed is:

- `nrow` - The number of rows (set as the length of the `rownames`).
- `ncol` - The number of columns (provided via the `id` column of `col_info`).
- `colnames` - The column names.
- `rownames` - The row names.
- `col_info` - The column information, which can be obtained via `mlr3::col_info()`.

Beware that accessing the backend's hash also contructs the backend.

Note that while in most cases the data contains `lazy_tensor` columns, this is not necessary and the naming of this class has nothing to do with the `lazy_tensor` data type.

Important

When the constructor generates `factor()` variables it is important that the ordering of the levels in data corresponds to the ordering of the levels in the `col_info` argument.

Super class

`mlr3::DataBackend` -> `DataBackendLazy`

Active bindings

`backend (DataBackend)`
The wrapped backend that is lazily constructed when first accessed.

`nrow (integer(1))`
Number of rows (observations).

`ncol (integer(1))`
Number of columns (variables), including the primary key column.

`rownames (integer())`
Returns vector of all distinct row identifiers, i.e. the contents of the primary key column.

`colnames (character())`
Returns vector of all column names, including the primary key column.

`is_constructed (logical(1))`
Whether the backend has already been constructed.

Methods

Public methods:

- DataBackendLazy$new()
- DataBackendLazy$data()
- DataBackendLazy$head()
- DataBackendLazy$distinct()
- DataBackendLazy$missings()
- DataBackendLazy$print()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
DataBackendLazy$new(constructor, rownames, col_info, primary_key)
```

Arguments:

`constructor` (function)
: A function with argument `backend` (the lazy backend), whose return value must be the actual backend. This function is called the first time the field `$backend` is accessed.

`rownames` (integer())
: The row names. Must be a permutation of the rownames of the lazily constructed backend.

`col_info` (data.table::data.table())
: A data.table with columns `id`, `type` and `levels` containing the column id, type and levels. Note that the levels must be provided in the correct order.

`primary_key` (character(1))
: Name of the primary key column.

Method `data()`**:** Returns a slice of the data in the specified format. The rows must be addressed as vector of primary key values, columns must be referred to via column names. Queries for rows with no matching row id and queries for columns with no matching column name are silently ignored. Rows are guaranteed to be returned in the same order as `rows`, columns may be returned in an arbitrary order. Duplicated row ids result in duplicated rows, duplicated column names lead to an exception.

Accessing the data triggers the construction of the backend.

Usage:

```
DataBackendLazy$data(rows, cols)
```

Arguments:

`rows` (integer())
: Row indices.

`cols` (character())
: Column names.

Method `head()`**:** Retrieve the first `n` rows. This triggers the construction of the backend.

Usage:

```
DataBackendLazy$head(n = 6L)
```

Arguments:

`n` (`integer(1)`)
Number of rows.

Returns: `data.table::data.table()` of the first `n` rows.

Method `distinct()`**:** Returns a named list of vectors of distinct values for each column specified. If `na_rm` is `TRUE`, missing values are removed from the returned vectors of distinct values. Non-existing rows and columns are silently ignored.

This triggers the construction of the backend.

Usage:

```
DataBackendLazy$distinct(rows, cols, na_rm = TRUE)
```

Arguments:

`rows` (`integer()`)
Row indices.

`cols` (`character()`)
Column names.

`na_rm` (`logical(1)`)
Whether to remove NAs or not.

Returns: Named `list()` of distinct values.

Method `missings()`**:** Returns the number of missing values per column in the specified slice of data. Non-existing rows and columns are silently ignored.

This triggers the construction of the backend.

Usage:

```
DataBackendLazy$missings(rows, cols)
```

Arguments:

`rows` (`integer()`)
Row indices.

`cols` (`character()`)
Column names.

Returns: Total of missing values per column (named `numeric()`).

Method `print()`**:** Printer.

Usage:

```
DataBackendLazy$print()
```

Examples

```
# We first define a backend constructor
constructor = function(backend) {
  cat("Data is constructed!\n")
  DataBackendDataTable$new(
    data.table(x = rnorm(10), y = rnorm(10), row_id = 1:10),
    primary_key = "row_id"
  )
}
```

```
# to wrap this backend constructor in a lazy backend, we need to provide the correct metadata for it
column_info = data.table(
  id = c("x", "y", "row_id"),
  type = c("numeric", "numeric", "integer"),
  levels = list(NULL, NULL, NULL)
)
backend_lazy = DataBackendLazy$new(
  constructor = constructor,
  rownames = 1:10,
  col_info = column_info,
  primary_key = "row_id"
)

# Note that the constructor is not called for the calls below
# as they can be read from the metadata
backend_lazy$nrow
backend_lazy$rownames
backend_lazy$ncol
backend_lazy$colnames
col_info(backend_lazy)

# Only now the backend is constructed
backend_lazy$data(1, "x")
# Is the same as:
backend_lazy$backend$data(1, "x")
```

`mlr_callback_set` *Base Class for Callbacks*

Description

Base class from which callbacks should inherit (see section *Inheriting*). A callback set is a collection of functions that are executed at different stages of the training loop. They can be used to gain more control over the training process of a neural network without having to write everything from scratch.

When used a in torch learner, the `CallbackSet` is wrapped in a `TorchCallback`. The latters parameter set represents the arguments of the `CallbackSet`'s `$initialize()` method.

Inheriting

For each available stage (see section *Stages*) a public method `$on_<stage>()` can be defined. The evaluation context (a `ContextTorch`) can be accessed via `self$ctx`, which contains the current state of the training loop. This context is assigned at the beginning of the training loop and removed afterwards. Different stages of a callback can communicate with each other by assigning values to `$self`.

State: To be able to store information in the `$model` slot of a `LearnerTorch`, callbacks support a state API. You can overload the `$state_dict()` public method to define what will be stored

in learner$model$callbacks$<id> after training finishes. This then also requires to implement a $load_state_dict(state_dict) method that defines how to load a previously saved callback state into a different callback. Note that the $state_dict() should not include the parameter values that were used to initialize the callback.

For creating custom callbacks, the function torch_callback() is recommended, which creates a CallbackSet and then wraps it in a TorchCallback. To create a CallbackSet the convenience function callback_set() can be used. These functions perform checks such as that the stages are not accidentally misspelled.

Stages

- begin :: Run before the training loop begins.
- epoch_begin :: Run he beginning of each epoch.
- batch_begin :: Run before the forward call.
- after_backward :: Run after the backward call.
- batch_end :: Run after the optimizer step.
- batch_valid_begin :: Run before the forward call in the validation loop.
- batch_valid_end :: Run after the forward call in the validation loop.
- valid_end :: Run at the end of validation.
- epoch_end :: Run at the end of each epoch.
- end :: Run after last epoch.
- exit :: Run at last, using on.exit().

Terminate Training

If training is to be stopped, it is possible to set the field $terminate of ContextTorch. At the end of every epoch this field is checked and if it is TRUE, training stops. This can for example be used to implement custom early stopping.

Public fields

ctx (ContextTorch or NULL)
: The evaluation context for the callback. This field should always be NULL except during the $train() call of the torch learner.

Active bindings

stages (character())
: The active stages of this callback set.

Methods

Public methods:

- CallbackSet$print()
- CallbackSet$state_dict()
- CallbackSet$load_state_dict()

- `CallbackSet$clone()`

Method `print()`**:** Prints the object.

Usage:

```
CallbackSet$print(...)
```

Arguments:

`...` (any)
Currently unused.

Method `state_dict()`**:** Returns information that is kept in the the `LearnerTorch`'s state after training. This information should be loadable into the callback using `$load_state_dict()` to be able to continue training. This returns `NULL` by default.

Usage:

```
CallbackSet$state_dict()
```

Method `load_state_dict()`**:** Loads the state dict into the callback to continue training.

Usage:

```
CallbackSet$load_state_dict(state_dict)
```

Arguments:

`state_dict` (any)
The state dict as retrieved via `$state_dict()`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CallbackSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Callback: `TorchCallback`, `as_torch_callback()`, `as_torch_callbacks()`, `callback_set()`, `mlr3torch_callbacks`, `mlr_callback_set.checkpoint`, `mlr_callback_set.progress`, `mlr_callback_set.tb`, `mlr_callback_set.unfreeze`, `mlr_context_torch`, `t_clbk()`, `torch_callback()`

`mlr_callback_set.checkpoint` *Checkpoint Callback*

Description

Saves the optimizer and network states during training. The final network and optimizer are always stored.

Details

Saving the learner itself in the callback with a trained model is impossible, as the model slot is set *after* the last callback step is executed.

Super class

`mlr3torch::CallbackSet` -> `CallbackSetCheckpoint`

Methods

Public methods:

- `CallbackSetCheckpoint$new()`
- `CallbackSetCheckpoint$on_epoch_end()`
- `CallbackSetCheckpoint$on_batch_end()`
- `CallbackSetCheckpoint$on_exit()`
- `CallbackSetCheckpoint$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
CallbackSetCheckpoint$new(path, freq, freq_type = "epoch")
```

Arguments:

`path (character(1))`
The path to a folder where the models are saved.

`freq (integer(1))`
The frequency how often the model is saved. Frequency is either per step or epoch, which can be configured through the `freq_type` parameter.

`freq_type (character(1))`
Can be be either `"epoch"` (default) or `"step"`.

Method `on_epoch_end()`**:** Saves the network and optimizer state dict. Does nothing if `freq_type` or `freq` are not met.

Usage:

```
CallbackSetCheckpoint$on_epoch_end()
```

Method `on_batch_end()`**:** Saves the selected objects defined in `save`. Does nothing if freq_type or freq are not met.

Usage:

```
CallbackSetCheckpoint$on_batch_end()
```

Method `on_exit()`**:** Saves the learner.

Usage:

```
CallbackSetCheckpoint$on_exit()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CallbackSetCheckpoint$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Callback: TorchCallback, as_torch_callback(), as_torch_callbacks(), callback_set(), mlr3torch_callbacks, mlr_callback_set, mlr_callback_set.progress, mlr_callback_set.tb, mlr_callback_set.unfreeze, mlr_context_torch, t_clbk(), torch_callback()

Examples

```
cb = t_clbk("checkpoint", freq = 1)
task = tsk("iris")

pth = tempfile()
learner = lrn("classif.mlp", epochs = 3, batch_size = 1, callbacks = cb)
learner$param_set$set_values(cb.checkpoint.path = pth)

learner$train(task)

list.files(pth)
```

---

mlr_callback_set.history

History Callback

Description

Saves the training and validation history during training. The history is saved as a data.table where the validation measures are prefixed with `"valid."` and the training measures are prefixed with `"train."`.

Super class

mlr3torch::CallbackSet -> CallbackSetHistory

Methods

Public methods:

- CallbackSetHistory$on_begin()
- CallbackSetHistory$state_dict()
- CallbackSetHistory$load_state_dict()
- CallbackSetHistory$on_before_valid()
- CallbackSetHistory$on_epoch_end()
- CallbackSetHistory$clone()

Method `on_begin()`**:** Initializes lists where the train and validation metrics are stored.

Usage:

```
CallbackSetHistory$on_begin()
```

Method `state_dict()`**:** Converts the lists to data.tables.

Usage:

```
CallbackSetHistory$state_dict()
```

Method `load_state_dict()`**:** Sets the field `$train` and `$valid` to those contained in the state dict.

Usage:

```
CallbackSetHistory$load_state_dict(state_dict)
```

Arguments:

`state_dict (callback_state_history)`
The state dict as retrieved via `$state_dict()`.

Method `on_before_valid()`**:** Add the latest training scores to the history.

Usage:

```
CallbackSetHistory$on_before_valid()
```

Method `on_epoch_end()`**:** Add the latest validation scores to the history.

Usage:

```
CallbackSetHistory$on_epoch_end()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CallbackSetHistory$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
cb = t_clbk("history")
task = tsk("iris")

learner = lrn("classif.mlp", epochs = 3, batch_size = 1,
  callbacks = t_clbk("history"), validate = 0.3)
learner$param_set$set_values(
  measures_train = msrs(c("classif.acc", "classif.ce")),
  measures_valid = msr("classif.ce")
)
learner$train(task)

print(learner$model$callbacks$history)
```

`mlr_callback_set.lr_scheduler`

Learning Rate Scheduling Callback

Description

Changes the learning rate based on the schedule specified by a `torch::lr_scheduler`.

As of this writing, the following are available:

- `torch::lr_cosine_annealing()`
- `torch::lr_lambda()`
- `torch::lr_multiplicative()`
- `torch::lr_one_cycle()`
- `torch::lr_reduce_on_plateau()`
- `torch::lr_step()`
- Custom schedulers defined with `torch::lr_scheduler()`.

Super class

`mlr3torch::CallbackSet` -> `CallbackSetLRScheduler`

Public fields

`scheduler_fn` (`lr_scheduler_generator`)
The `torch` function that creates a learning rate scheduler

`scheduler` (`LRScheduler`)
The learning rate scheduler wrapped by this callback

Methods

Public methods:

- `CallbackSetLRScheduler$new()`
- `CallbackSetLRScheduler$on_begin()`
- `CallbackSetLRScheduler$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
CallbackSetLRScheduler$new(.scheduler, step_on_epoch, ...)
```

Arguments:

`.scheduler` (`lr_scheduler_generator`)
The `torch` scheduler generator (e.g. `torch::lr_step`).

`step_on_epoch` (`logical(1)`)
Whether the scheduler steps after every epoch (otherwise every batch).

... (any)
The scheduler-specific arguments

Method `on_begin()`**:** Creates the scheduler using the optimizer from the context

Usage:

```
CallbackSetLRScheduler$on_begin()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CallbackSetLRScheduler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`mlr_callback_set.progress` *Progress Callback*

Description

Prints a progress bar and the metrics for training and validation.

Super class

`mlr3torch::CallbackSet` -> `CallbackSetProgress`

Methods

Public methods:

- `CallbackSetProgress$on_epoch_begin()`
- `CallbackSetProgress$on_batch_end()`
- `CallbackSetProgress$on_before_valid()`
- `CallbackSetProgress$on_batch_valid_end()`
- `CallbackSetProgress$on_epoch_end()`
- `CallbackSetProgress$on_end()`
- `CallbackSetProgress$clone()`

Method `on_epoch_begin()`**:** Initializes the progress bar for training.

Usage:

```
CallbackSetProgress$on_epoch_begin()
```

Method `on_batch_end()`**:** Increments the training progress bar.

Usage:

```
CallbackSetProgress$on_batch_end()
```

Method `on_before_valid()`**:** Creates the progress bar for validation.

Usage:

```
CallbackSetProgress$on_before_valid()
```

Method `on_batch_valid_end()`**:** Increments the validation progress bar.

Usage:

```
CallbackSetProgress$on_batch_valid_end()
```

Method `on_epoch_end()`**:** Prints a summary of the training and validation process.

Usage:

```
CallbackSetProgress$on_epoch_end()
```

Method `on_end()`**:** Prints the time at the end of training.

Usage:

```
CallbackSetProgress$on_end()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CallbackSetProgress$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other Callback: `TorchCallback`, `as_torch_callback()`, `as_torch_callbacks()`, `callback_set()`, `mlr3torch_callbacks`, `mlr_callback_set`, `mlr_callback_set.checkpoint`, `mlr_callback_set.tb`, `mlr_callback_set.unfreeze`, `mlr_context_torch`, `t_clbk()`, `torch_callback()`

Examples

```
task = tsk("iris")

learner = lrn("classif.mlp", epochs = 5, batch_size = 1,
  callbacks = t_clbk("progress"), validate = 0.3)
learner$param_set$set_values(
  measures_train = msrs(c("classif.acc", "classif.ce")),
  measures_valid = msr("classif.ce")
)

learner$train(task)
```

`mlr_callback_set.tb` *TensorBoard Logging Callback*

Description

Logs training loss, training measures, and validation measures as events. To view them, use TensorBoard with `tensorflow::tensorboard()` (requires `tensorflow`) or the CLI.

Details

Logs events at most every epoch.

Super class

`mlr3torch::CallbackSet` -> `CallbackSetTB`

Methods

Public methods:

- `CallbackSetTB$new()`
- `CallbackSetTB$on_epoch_end()`
- `CallbackSetTB$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
CallbackSetTB$new(path, log_train_loss)
```

Arguments:

`path (character(1))`
: The path to a folder where the events are logged. Point TensorBoard to this folder to view them.

`log_train_loss (logical(1))`
: Whether we log the training loss.

Method `on_epoch_end()`**:** Logs the training loss, training measures, and validation measures as TensorBoard events.

Usage:

```
CallbackSetTB$on_epoch_end()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CallbackSetTB$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Callback: TorchCallback, as_torch_callback(), as_torch_callbacks(), callback_set(), mlr3torch_callbacks, mlr_callback_set, mlr_callback_set.checkpoint, mlr_callback_set.progress, mlr_callback_set.unfreeze, mlr_context_torch, t_clbk(), torch_callback()

`mlr_callback_set.unfreeze` *Unfreezing Weights Callback*

Description

Unfreeze some weights (parameters of the network) after some number of steps or epochs.

Super class

`mlr3torch::CallbackSet` -> `CallbackSetUnfreeze`

Methods

Public methods:

- `CallbackSetUnfreeze$new()`
- `CallbackSetUnfreeze$on_begin()`
- `CallbackSetUnfreeze$on_epoch_begin()`
- `CallbackSetUnfreeze$on_batch_begin()`
- `CallbackSetUnfreeze$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
CallbackSetUnfreeze$new(starting_weights, unfreeze)
```

Arguments:

`starting_weights` (`Select`)
: A `Select` denoting the weights that are trainable from the start.

`unfreeze` (`data.table`)
: A `data.table` with a column `weights` (a list column of `Selects`) and a column `epoch` or `batch`. The selector indicates which parameters to unfreeze, while the `epoch` or `batch` column indicates when to do so.

Method `on_begin()`**:** Sets the starting weights

Usage:

```
CallbackSetUnfreeze$on_begin()
```

Method `on_epoch_begin()`**:** Unfreezes weights if the training is at the correct epoch

Usage:

```
CallbackSetUnfreeze$on_epoch_begin()
```

Method `on_batch_begin()`**:** Unfreezes weights if the training is at the correct batch

Usage:

```
CallbackSetUnfreeze$on_batch_begin()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CallbackSetUnfreeze$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Callback: `TorchCallback`, `as_torch_callback()`, `as_torch_callbacks()`, `callback_set()`, `mlr3torch_callbacks`, `mlr_callback_set`, `mlr_callback_set.checkpoint`, `mlr_callback_set.progress`, `mlr_callback_set.tb`, `mlr_context_torch`, `t_clbk()`, `torch_callback()`

Examples

```
task = tsk("iris")
cb = t_clbk("unfreeze")
mlp = lrn("classif.mlp", callbacks = cb,
 cb.unfreeze.starting_weights = select_invert(
   select_name(c("0.weight", "3.weight", "6.weight", "6.bias"))
 ),
 cb.unfreeze.unfreeze = data.table(
   epoch = c(2, 5),
   weights = list(select_name("0.weight"), select_name(c("3.weight", "6.weight")))
 ),
 epochs = 6, batch_size = 150, neurons = c(1, 1, 1)
)

mlp$train(task)
```

---

mlr_context_torch *Context for Torch Learner*

Description

Context for training a torch learner. This is the - mostly read-only - information callbacks have access to through the argument `ctx`. For more information on callbacks, see `CallbackSet`.

Public fields

`learner` (`Learner`)
: The torch learner.

`task_train` (`Task`)
: The training task.

`task_valid` (`Task` or `NULL`)
: The validation task.

`loader_train` (`torch::dataloader`)
: The data loader for training.

`loader_valid` (`torch::dataloader`)
: The data loader for validation.

`measures_train` (`list()` of `Measure`s)
: Measures used for training.

`measures_valid` (`list()` of `Measure`s)
: Measures used for validation.

`network` (`torch::nn_module`)
: The torch network.

`optimizer` (`torch::optimizer`)
: The optimizer.

`loss_fn` (`torch::nn_module`)
: The loss function.

`total_epochs` (`integer(1)`)
: The total number of epochs the learner is trained for.

`last_scores_train` (named `list()` or `NULL`)
: The scores from the last training batch. Names are the ids of the training measures. If `LearnerTorch` sets `eval_freq` different from `1`, this is `NULL` in all epochs that don't evaluate the model.

`last_scores_valid` (`list()`)
: The scores from the last validation batch. Names are the ids of the validation measures. If `LearnerTorch` sets `eval_freq` different from `1`, this is `NULL` in all epochs that don't evaluate the model.

`last_loss` (`numeric(1)`)
: The loss from the last trainings batch.

`epoch` (`integer(1)`)
: The current epoch.

`step` (`integer(1)`)
: The current iteration.

`prediction_encoder` (`function()`)
: The learner's prediction encoder.

`batch` (named `list()` of `torch_tensors`)
: The current batch.

`terminate` (`logical(1)`)
: If this field is set to `TRUE` at the end of an epoch, training stops.

`device` (`torch::torch_device`)
: The device.

Methods

Public methods:

- `ContextTorch$new()`
- `ContextTorch$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
ContextTorch$new(
  learner,
  task_train,
  task_valid = NULL,
  loader_train,
  loader_valid = NULL,
  measures_train = NULL,
  measures_valid = NULL,
  network,
  optimizer,
  loss_fn,
  total_epochs,
  prediction_encoder,
  eval_freq = 1L,
  device
)
```

Arguments:

`learner` (`Learner`)
: The torch learner.

`task_train` (`Task`)
: The training task.

`task_valid` (`Task` or `NULL`)
: The validation task.

`loader_train` (`torch::dataloader`)
: The data loader for training.

`loader_valid` (`torch::dataloader` or `NULL`)
: The data loader for validation.

`measures_train` (`list()` of `Measure`s or `NULL`)
: Measures used for training. Default is `NULL`.

`measures_valid` (`list()` of `Measure`s or `NULL`)
: Measures used for validation.

`network` (`torch::nn_module`)
: The torch network.

`optimizer` (`torch::optimizer`)
: The optimizer.

`loss_fn` (`torch::nn_module`)
: The loss function.

`total_epochs` (`integer(1)`)
: The total number of epochs the learner is trained for.

`prediction_encoder (function())`
The learner's prediction encoder.

`eval_freq (integer(1))`
The evaluation frequency.

`device (character(1))`
The device.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ContextTorch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Callback: `TorchCallback`, `as_torch_callback()`, `as_torch_callbacks()`, `callback_set()`, `mlr3torch_callbacks`, `mlr_callback_set`, `mlr_callback_set.checkpoint`, `mlr_callback_set.progress`, `mlr_callback_set.tb`, `mlr_callback_set.unfreeze`, `t_clbk()`, `torch_callback()`

`mlr_learners.mlp` *My Little Pony*

Description

Fully connected feed forward network with dropout after each activation function. The features can either be a single `lazy_tensor` or one or more numeric columns (but not both).

Dictionary

This Learner can be instantiated using the sugar function `lrn()`:

```
lrn("classif.mlp", ...)
lrn("regr.mlp", ...)
```

Properties

- Supported task types: 'classif', 'regr'
- Predict Types:
  - classif: 'response', 'prob'
  - regr: 'response'
- Feature Types: "integer", "numeric", "lazy_tensor"
- Required Packages: **mlr3**, **mlr3torch**, **torch**

Parameters

Parameters from LearnerTorch, as well as:

- activation :: [nn_module]
  The activation function. Is initialized to nn_relu.
- activation_args :: named list()
  A named list with initialization arguments for the activation function. This is intialized to an empty list.
- neurons :: integer()
  The number of neurons per hidden layer. By default there is no hidden layer. Setting this to c(10, 20) would have a the first hidden layer with 10 neurons and the second with 20.
- n_layers :: integer()
  The number of layers. This parameter must only be set when neurons has length 1.
- p :: numeric(1)
  The dropout probability. Is initialized to 0.5.
- shape :: integer() or NULL
  The input shape of length 2, e.g. c(NA, 5). Only needs to be present when there is a lazy tensor input with unknown shape (NULL). Otherwise the input shape is inferred from the number of numeric features.

Super classes

mlr3::Learner -> mlr3torch::LearnerTorch -> LearnerTorchMLP

Methods

Public methods:

- LearnerTorchMLP$new()
- LearnerTorchMLP$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
LearnerTorchMLP$new(
  task_type,
  optimizer = NULL,
  loss = NULL,
  callbacks = list()
)
```

Arguments:

task_type (character(1))
: The task type, either "classif" or "regr".

optimizer (TorchOptimizer)
: The optimizer to use for training. Per default, *adam* is used.

loss (TorchLoss)
: The loss used to train the network. Per default, *mse* is used for regression and *cross_entropy* for classification.

callbacks (list() of TorchCallbacks)
The callbacks. Must have unique ids.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerTorchMLP$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Gorishniy Y, Rubachev I, Khrulkov V, Babenko A (2021). "Revisiting Deep Learning for Tabular Data." *arXiv*, **2106.11959**.

See Also

Other Learner: mlr_learners.tab_resnet, mlr_learners.torch_featureless, mlr_learners_torch, mlr_learners_torch_image, mlr_learners_torch_model

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.mlp")
learner$param_set$set_values(
  epochs = 1, batch_size = 16, device = "cpu",
  neurons = 10
)

# Define a Task
task = tsk("iris")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners.tab_resnet` *Tabular ResNet*

Description

Tabular resnet.

Dictionary

This Learner can be instantiated using the sugar function `lrn()`:

```
lrn("classif.tab_resnet", ...)
lrn("regr.tab_resnet", ...)
```

Properties

- Supported task types: 'classif', 'regr'
- Predict Types:
  - classif: 'response', 'prob'
  - regr: 'response'
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3torch**, **torch**

Parameters

Parameters from `LearnerTorch`, as well as:

- `n_blocks` :: `integer(1)`
  The number of blocks.
- `d_block` :: `integer(1)`
  The input and output dimension of a block.
- `d_hidden` :: `integer(1)`
  The latent dimension of a block.
- `d_hidden_multiplier` :: `numeric(1)`
  Alternative way to specify the latent dimension as `d_block * d_hidden_multiplier`.
- `dropout1` :: `numeric(1)`
  First dropout ratio.
- `dropout2` :: `numeric(1)`
  Second dropout ratio.

Super classes

`mlr3::Learner` -> `mlr3torch::LearnerTorch` -> `LearnerTorchTabResNet`

Methods

Public methods:

- `LearnerTorchTabResNet$new()`
- `LearnerTorchTabResNet$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
LearnerTorchTabResNet$new(
  task_type,
  optimizer = NULL,
  loss = NULL,
  callbacks = list()
)
```

Arguments:

`task_type` (`character(1)`)
: The task type, either `"classif"` or `"regr"`.

`optimizer` (`TorchOptimizer`)
: The optimizer to use for training. Per default, *adam* is used.

`loss` (`TorchLoss`)
: The loss used to train the network. Per default, *mse* is used for regression and *cross_entropy* for classification.

`callbacks` (`list()` of `TorchCallbacks`)
: The callbacks. Must have unique ids.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LearnerTorchTabResNet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Gorishniy Y, Rubachev I, Khrulkov V, Babenko A (2021). "Revisiting Deep Learning for Tabular Data." *arXiv*, **2106.11959**.

See Also

Other Learner: `mlr_learners.mlp`, `mlr_learners.torch_featureless`, `mlr_learners_torch`, `mlr_learners_torch_image`, `mlr_learners_torch_model`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.tab_resnet")
learner$param_set$set_values(
  epochs = 1, batch_size = 16, device = "cpu",
```

```
  n_blocks = 2, d_block = 10, d_hidden = 20, dropout1 = 0.3, dropout2 = 0.3
)

# Define a Task
task = tsk("iris")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners.torchvision`
AlexNet Image Classifier

Description

Classic image classification networks from `torchvision`.

Parameters

Parameters from `LearnerTorchImage` and

- `pretrained` :: `logical(1)`
  Whether to use the pretrained model. The final linear layer will be replaced with a new `nn_linear` with the number of classes inferred from the `Task`.

Properties

- Supported task types: `"classif"`
- Predict Types: `"response"` and `"prob"`
- Feature Types: `"lazy_tensor"`
- Required packages: `"mlr3torch"`, `"torch"`, `"torchvision"`

Super classes

`mlr3::Learner` -> `mlr3torch::LearnerTorch` -> `mlr3torch::LearnerTorchImage` -> `LearnerTorchVision`

Methods

Public methods:

- `LearnerTorchVision$new()`
- `LearnerTorchVision$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
LearnerTorchVision$new(
  name,
  module_generator,
  label,
  optimizer = NULL,
  loss = NULL,
  callbacks = list()
)
```

Arguments:

`name` (`character(1)`)
The name of the network.

`module_generator` (`function(pretrained, num_classes)`)
Function that generates the network.

`label` (`character(1)`)
The label of the network. #' @references Krizhevsky, Alex, Sutskever, Ilya, Hinton, E. G (2017). "Imagenet classification with deep convolutional neural networks." *Communications of the ACM*, **60**(6), 84–90. Sandler, Mark, Howard, Andrew, Zhu, Menglong, Zhmoginov, Andrey, Chen, Liang-Chieh (2018). "Mobilenetv2: Inverted residuals and linear bottlenecks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520. He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, Sun, Jian (2016). "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778. Simonyan, Karen, Zisserman, Andrew (2014). "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556*.

`optimizer` (`TorchOptimizer`)
The optimizer to use for training. Per default, *adam* is used.

`loss` (`TorchLoss`)
The loss used to train the network. Per default, *mse* is used for regression and *cross_entropy* for classification.

`callbacks` (`list()` of `TorchCallback`s)
The callbacks. Must have unique ids.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LearnerTorchVision$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`mlr_learners.torch_featureless` *Featureless Torch Learner*

Description

Featureless torch learner. Output is a constant weight that is learned during training. For classification, this should (asymptoptically) result in a majority class prediction when using the standard cross-entropy loss. For regression, this should result in the median for L1 loss and in the mean for L2 loss.

Dictionary

This Learner can be instantiated using the sugar function `lrn()`:

```
lrn("classif.torch_featureless", ...)
lrn("regr.torch_featureless", ...)
```

Properties

- Supported task types: 'classif', 'regr'
- Predict Types:
  - classif: 'response', 'prob'
  - regr: 'response'
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date", "lazy_tensor"
- Required Packages: **mlr3**, **mlr3torch**, **torch**

Parameters

Only those from `LearnerTorch`.

Super classes

`mlr3::Learner` -> `mlr3torch::LearnerTorch` -> `LearnerTorchFeatureless`

Methods

Public methods:

- `LearnerTorchFeatureless$new()`
- `LearnerTorchFeatureless$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
LearnerTorchFeatureless$new(
  task_type,
  optimizer = NULL,
  loss = NULL,
  callbacks = list()
)
```

Arguments:

`task_type` (`character(1)`)
: The task type, either `"classif"` or `"regr"`.

`optimizer` (`TorchOptimizer`)
: The optimizer to use for training. Per default, *adam* is used.

`loss` (`TorchLoss`)
: The loss used to train the network. Per default, *mse* is used for regression and *cross_entropy* for classification.

`callbacks` (`list()` of `TorchCallbacks`)
: The callbacks. Must have unique ids.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LearnerTorchFeatureless$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Learner: `mlr_learners.mlp`, `mlr_learners.tab_resnet`, `mlr_learners_torch`, `mlr_learners_torch_image`, `mlr_learners_torch_model`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.torch_featureless")
learner$param_set$set_values(
  epochs = 1, batch_size = 16, device = "cpu"

)

# Define a Task
task = tsk("iris")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)
```

```
# Score the predictions
predictions$score()
```

`mlr_learners_torch` *Base Class for Torch Learners*

Description

This base class provides the basic functionality for training and prediction of a neural network. All torch learners should inherit from this class.

Validation

To specify the validation data, you can set the `$validate` field of the Learner, which can be set to:

- `NULL`: no validation
- `ratio`: only proportion `1 - ratio` of the task is used for training and `ratio` is used for validation.
- `"test"` means that the `"test"` task of a resampling is used and is not possible when calling `$train()` manually.
- `"predefined"`: This will use the predefined `$internal_valid_task` of a `mlr3::Task`.

This validation data can also be used for early stopping, see the description of the `Learner`'s parameters.

Saving a Learner

In order to save a `LearnerTorch` for later usage, it is necessary to call the `$marshal()` method on the `Learner` before writing it to disk, as the object will otherwise not be saved correctly. After loading a marshaled `LearnerTorch` into R again, you then need to call `$unmarshal()` to transform it into a useable state.

Early Stopping and Internal Tuning

In order to prevent overfitting, the `LearnerTorch` class allows to use early stopping via the `patience` and `min_delta` parameters, see the `Learner`'s parameters. When tuning a `LearnerTorch` it is also possible to combine the explicit tuning via `mlr3tuning` and the `LearnerTorch`'s internal tuning of the epochs via early stopping. To do so, you just need to include `epochs = to_tune(upper = <upper>, internal = TRUE)` in the search space, where `<upper>` is the maximally allowed number of epochs, and configure the early stopping.

Model

The Model is a list of class "learner_torch_model" with the following elements:

- network :: The trained network.
- optimizer :: The $state_dict() optimizer used to train the network.
- loss_fn :: The $state_dict() of the loss used to train the network.
- callbacks :: The callbacks used to train the network.
- seed :: The seed that was / is used for training and prediction.
- epochs :: How many epochs the model was trained for (early stopping).
- task_col_info :: A data.table() containing information about the train-task.

Parameters

General:

The parameters of the optimizer, loss and callbacks, prefixed with "opt.", "loss." and "cb.<callback id>." respectively, as well as:

- epochs :: integer(1)
  The number of epochs.
- device :: character(1)
  The device. One of "auto", "cpu", or "cuda" or other values defined in mlr_reflections$torch$devices. The value is initialized to "auto", which will select "cuda" if possible, then try "mps" and otherwise fall back to "cpu".
- num_threads :: integer(1)
  The number of threads for intraop pararallelization (if device is "cpu"). This value is initialized to 1.
- num_interop_threads :: integer(1)
  The number of threads for intraop and interop pararallelization (if device is "cpu"). This value is initialized to 1. Note that this can only be set once during a session and changing the value within an R session will raise a warning.
- seed :: integer(1) or "random" or NULL
  The torch seed that is used during training and prediction. This value is initialized to "random", which means that a random seed will be sampled at the beginning of the training phase. This seed (either set or randomly sampled) is available via $model$seed after training and used during prediction. Note that by setting the seed during the training phase this will mean that by default (i.e. when seed is "random"), clones of the learner will use a different seed. If set to NULL, no seeding will be done.
- tensor_dataset :: logical(1) | "device"
  Whether to load all batches at once at the beginning of training and stack them. This is initialized to FALSE. If set to "device", the device of the tensors will be set to the value of device, which can avoid unnecessary moving of tensors between devices. When your dataset fits into memory this will make the loading of batches faster. Note that this should not be set for datasets that contain lazy_tensors with random data augmentation, as this augmentation will only be applied once at the beginning of training.

Evaluation:

- measures_train :: Measure or list() of Measures
  Measures to be evaluated during training.
- measures_valid :: Measure or list() of Measures
  Measures to be evaluated during validation.
- eval_freq :: integer(1)
  How often the train / validation predictions are evaluated using measures_train / measures_valid. This is initialized to 1. Note that the final model is always evaluated.

Early Stopping:

- patience :: integer(1)
  This activates early stopping using the validation scores. If the performance of a model does not improve for patience evaluation steps, training is ended. Note that the final model is stored in the learner, not the best model. This is initialized to 0, which means no early stopping. The first entry from measures_valid is used as the metric. This also requires to specify the $validate field of the Learner, as well as measures_valid. If this is set, the epoch after which no improvement was observed, can be accessed via the $internal_tuned_values field of the learner.
- min_delta :: double(1)
  The minimum improvement threshold for early stopping. Is initialized to 0.

Dataloader:

- batch_size :: integer(1)
  The batch size (required).
- shuffle :: logical(1)
  Whether to shuffle the instances in the dataset. This is initialized to TRUE, which differs from the default (FALSE).
- sampler :: torch::sampler
  Object that defines how the dataloader draw samples.
- batch_sampler :: torch::sampler
  Object that defines how the dataloader draws batches.
- num_workers :: integer(1)
  The number of workers for data loading (batches are loaded in parallel). The default is 0, which means that data will be loaded in the main process.
- collate_fn :: function
  How to merge a list of samples to form a batch.
- pin_memory :: logical(1)
  Whether the dataloader copies tensors into CUDA pinned memory before returning them.
- drop_last :: logical(1)
  Whether to drop the last training batch in each epoch during training. Default is FALSE.
- timeout :: numeric(1)
  The timeout value for collecting a batch from workers. Negative values mean no timeout and the default is -1.
- worker_init_fn :: function(id)
  A function that receives the worker id (in [1, num_workers]) and is exectued after seeding on the worker but before data loading.

- `worker_globals` :: `list()` | `character()`
  When loading data in parallel, this allows to export globals to the workers. If this is a character vector, the objects in the global environment with those names are copied to the workers.
- `worker_packages` :: `character()`
  Which packages to load on the workers.

Also see `torch::dataloder` for more information.

Inheriting

There are no seperate classes for classification and regression to inherit from. Instead, the `task_type` must be specified as a construction argument. Currently, only classification and regression are supported.

When inheriting from this class, one should overload two private methods:

- `.network(task, param_vals)`
  (`Task`, `list()`) -> `nn_module`
  Construct a `torch::nn_module` object for the given task and parameter values, i.e. the neural network that is trained by the learner. For classification, the output of this network are expected to be the scores before the application of the final softmax layer.
- `.dataset(task, param_vals)`
  (`Task`, `list()`) -> `torch::dataset`
  Create the dataset for the task. The dataset must return a named list where:
  - `x` is a list of torch tensors that are the input to the network. For networks with more than one input, the names must correspond to the inputs of the network.
  - `y` is the target tensor.
  - `.index` are the indices of the batch (`integer()` or a `torch_int()`).

  Moreover, one needs to pay attention respect the row ids of the provided task.

It is also possible to overwrite the private `.dataloader()` method. This must respect the dataloader parameters from the `ParamSet`.

- `.dataloader(dataset, param_vals)`
  (`Task`, `list()`) -> `torch::dataloader`
  Create a dataloader from the task. Needs to respect at least `batch_size` and `shuffle` (otherwise predictions can be permuted).

To change the predict types, the it is possible to overwrite the method below:

- `.encode_prediction(predict_tensor, task)`
  (`torch_tensor`, `Task`) -> `list()`
  Take in the raw predictions from `self$network` (`predict_tensor`) and encode them into a format that can be converted to valid `mlr3` predictions using `mlr3::as_prediction_data()`. This method must take `self$predict_type` into account.

While it is possible to add parameters by specifying the `param_set` construction argument, it is currently not possible to remove existing parameters, i.e. those listed in section *Parameters*. None of the parameters provided in `param_set` can have an id that starts with `"loss."`, `"opt."`, or `"cb."`, as these are preserved for the dynamically constructed parameters of the optimizer, the loss function, and the callbacks.

To perform additional input checks on the task, the private `.verify_train_task(task, param_vals)` and `.verify_predict_task(task, param_vals)` can be overwritten.

For learners that have other construction arguments that should change the hash of a learner, it is required to implement the private `$.additional_phash_input()`.

Super class

`mlr3::Learner` -> `LearnerTorch`

Active bindings

`validate` How to construct the internal validation data. This parameter can be either `NULL`, a ratio in $(0, 1)$, `"test"`, or `"predefined"`.

`loss` (`TorchLoss`)
The torch loss.

`optimizer` (`TorchOptimizer`)
The torch optimizer.

`callbacks` (`list()` of `TorchCallbacks`)
List of torch callbacks. The ids will be set as the names.

`internal_valid_scores` Retrieves the internal validation scores as a named `list()`. Specify the `$validate` field and the `measures_valid` parameter to configure this. Returns `NULL` if learner is not trained yet.

`internal_tuned_values` When early stopping is activate, this returns a named list with the early-stopped epochs, otherwise an empty list is returned. Returns `NULL` if learner is not trained yet.

`marshaled` (`logical(1)`)
Whether the learner is marshaled.

`network` (`nn_module()`)
Shortcut for `learner$model$network`.

`param_set` (`ParamSet`)
The parameter set

`hash` (`character(1)`)
Hash (unique identifier) for this object.

`phash` (`character(1)`)
Hash (unique identifier) for this partial object, excluding some components which are varied systematically during tuning (parameter values).

Methods

Public methods:

- `LearnerTorch$new()`
- `LearnerTorch$format()`
- `LearnerTorch$print()`
- `LearnerTorch$marshal()`
- `LearnerTorch$unmarshal()`

- LearnerTorch$dataset()
- LearnerTorch$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
LearnerTorch$new(
  id,
  task_type,
  param_set,
  properties,
  man,
  label,
  feature_types,
  optimizer = NULL,
  loss = NULL,
  packages = character(),
  predict_types = NULL,
  callbacks = list()
)
```

Arguments:

`id` (`character(1)`)
: The id for of the new object.

`task_type` (`character(1)`)
: The task type.

`param_set` (`ParamSet` or `alist()`)
: Either a parameter set, or an `alist()` containing different values of self, e.g. `alist(private$.param_set1, private$.param_set2)`, from which a `ParamSet` collection should be created.

`properties` (`character()`)
: The properties of the object. See `mlr_reflections$learner_properties` for available values.

`man` (`character(1)`)
: String in the format `[pkg]::[topic]` pointing to a manual page for this object. The referenced help package can be opened via method `$help()`.

`label` (`character(1)`)
: Label for the new instance.

`feature_types` (`character()`)
: The feature types. See `mlr_reflections$task_feature_types` for available values, Additionally, `"lazy_tensor"` is supported.

`optimizer` (`NULL` or `TorchOptimizer`)
: The optimizer to use for training. Defaults to adam.

`loss` (`NULL` or `TorchLoss`)
: The loss to use for training. Defaults to MSE for regression and cross entropy for classification.

`packages` (`character()`)
: The R packages this object depends on.

`predict_types` (`character()`)
The predict types. See `mlr_reflections$learner_predict_types` for available values. For regression, the default is `"response"`. For classification, this defaults to `"response"` and `"prob"`. To deviate from the defaults, it is necessary to overwrite the private `$.encode_prediction()` method, see section *Inheriting*.

`callbacks` (`list()` of `TorchCallbacks`)
The callbacks to use for training. Defaults to an empty `list()`, i.e. no callbacks.

Method `format()`**:** Helper for print outputs.

Usage:

```
LearnerTorch$format(...)
```

Arguments:

`...` (ignored).

Method `print()`**:** Prints the object.

Usage:

```
LearnerTorch$print(...)
```

Arguments:

`...` (any)
Currently unused.

Method `marshal()`**:** Marshal the learner.

Usage:

```
LearnerTorch$marshal(...)
```

Arguments:

`...` (any)
Additional parameters.

Returns: self

Method `unmarshal()`**:** Unmarshal the learner.

Usage:

```
LearnerTorch$unmarshal(...)
```

Arguments:

`...` (any)
Additional parameters.

Returns: self

Method `dataset()`**:** Create the dataset for a task.

Usage:

```
LearnerTorch$dataset(task)
```

Arguments:

`task` `Task`
The task

Returns: dataset

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerTorch$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other Learner: mlr_learners.mlp, mlr_learners.tab_resnet, mlr_learners.torch_featureless, mlr_learners_torch_image, mlr_learners_torch_model

---

mlr_learners_torch_image

Image Learner

Description

Base Class for Image Learners. The features are assumed to be a single lazy_tensor column in RGB format.

Parameters

Parameters include those inherited from LearnerTorch and the param_set construction argument.

Super classes

mlr3::Learner -> mlr3torch::LearnerTorch -> LearnerTorchImage

Methods

Public methods:

- LearnerTorchImage$new()
- LearnerTorchImage$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
LearnerTorchImage$new(
  id,
  task_type,
  param_set = ps(),
  label,
  optimizer = NULL,
  loss = NULL,
  callbacks = list(),
```

```
  packages = "torchvision",
  man,
  properties = NULL,
  predict_types = NULL
)
```

Arguments:

`id` (`character(1)`)
: The id for of the new object.

`task_type` (`character(1)`)
: The task type.

`param_set` (`ParamSet`)
: The parameter set.

`label` (`character(1)`)
: Label for the new instance.

`optimizer` (`TorchOptimizer`)
: The torch optimizer.

`loss` (`TorchLoss`)
: The loss to use for training.

`callbacks` (`list()` of `TorchCallbacks`)
: The callbacks used during training. Must have unique ids. They are executed in the order in which they are provided

`packages` (`character()`)
: The R packages this object depends on.

`man` (`character(1)`)
: String in the format `[pkg]::[topic]` pointing to a manual page for this object. The referenced help package can be opened via method `$help()`.

`properties` (`character()`)
: The properties of the object. See `mlr_reflections$learner_properties` for available values.

`predict_types` (`character()`)
: The predict types. See `mlr_reflections$learner_predict_types` for available values.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LearnerTorchImage$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Learner: `mlr_learners.mlp`, `mlr_learners.tab_resnet`, `mlr_learners.torch_featureless`, `mlr_learners_torch`, `mlr_learners_torch_model`

`mlr_learners_torch_model` *Learner Torch Model*

Description

Create a torch learner from an instantiated `nn_module()`. For classification, the output of the network must be the scores (before the softmax).

Parameters

See `LearnerTorch`

Super classes

`mlr3::Learner` -> `mlr3torch::LearnerTorch` -> `LearnerTorchModel`

Active bindings

`ingress_tokens` (named `list()` with `TorchIngressToken` or `NULL`)
The ingress tokens. Must be non-`NULL` when calling `$train()`.

Methods

Public methods:

- `LearnerTorchModel$new()`
- `LearnerTorchModel$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
LearnerTorchModel$new(
  network = NULL,
  ingress_tokens = NULL,
  task_type,
  properties = NULL,
  optimizer = NULL,
  loss = NULL,
  callbacks = list(),
  packages = character(0),
  feature_types = NULL
)
```

Arguments:

`network` (`nn_module`)
An instantiated `nn_module`. Is not cloned during construction. For classification, outputs must be the scores (before the softmax).

`ingress_tokens` (`list` of `TorchIngressToken()`)
A list with ingress tokens that defines how the dataloader will be defined.

`task_type` (`character(1)`)
The task type.

`properties` (`NULL` or `character()`)
The properties of the learner. Defaults to all available properties for the given task type.

`optimizer` (`TorchOptimizer`)
The torch optimizer.

`loss` (`TorchLoss`)
The loss to use for training.

`callbacks` (`list()` of `TorchCallbacks`)
The callbacks used during training. Must have unique ids. They are executed in the order in which they are provided

`packages` (`character()`)
The R packages this object depends on.

`feature_types` (`NULL` or `character()`)
The feature types. Defaults to all available feature types.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LearnerTorchModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Learner: `mlr_learners.mlp`, `mlr_learners.tab_resnet`, `mlr_learners.torch_featureless`, `mlr_learners_torch`, `mlr_learners_torch_image`

Other Graph Network: `ModelDescriptor()`, `TorchIngressToken()`, `mlr_pipeops_module`, `mlr_pipeops_torch`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `model_descriptor_to_learner()`, `model_descriptor_to_module()`, `model_descriptor_union()`, `nn_graph()`

Examples

```
# We show the learner using a classification task

# The iris task has 4 features and 3 classes
network = nn_linear(4, 3)
task = tsk("iris")

# This defines the dataloader.
# It loads all 4 features, which are also numeric.
# The shape is (NA, 4) because the batch dimension is generally NA
ingress_tokens = list(
  input = TorchIngressToken(task$feature_names, batchgetter_num, c(NA, 4))
)
```

```
# Creating the learner and setting required parameters
learner = lrn("classif.torch_model",
  network = network,
  ingress_tokens = ingress_tokens,
  batch_size = 16,
  epochs = 1,
  device = "cpu"
)

# A simple train-predict
ids = partition(task)
learner$train(task, ids$train)
learner$predict(task, ids$test)
```

`mlr_pipeops_augment_center_crop` *Center Crop Augmentation*

Description

Calls `torchvision::transform_center_crop`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
size	untyped	-	
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

mlr_pipeops_augment_color_jitter

Color Jitter Augmentation

Description

Calls `torchvision::transform_color_jitter`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
brightness	numeric	0		$[0, \infty)$
contrast	numeric	0		$[0, \infty)$
saturation	numeric	0		$[0, \infty)$
hue	numeric	0		$[0, \infty)$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

mlr_pipeops_augment_crop

Crop Augmentation

Description

Calls `torchvision::transform_crop`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
top	integer	-		$(-\infty, \infty)$
left	integer	-		$(-\infty, \infty)$
height	integer	-		$(-\infty, \infty)$
width	integer	-		$(-\infty, \infty)$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

`mlr_pipeops_augment_hflip`
Horizontal Flip Augmentation

Description

Calls `torchvision::transform_hflip`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

`mlr_pipeops_augment_random_affine`
Random Affine Augmentation

Description

Calls `torchvision::transform_random_affine`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
degrees	untyped	-		-
translate	untyped	NULL		-
scale	untyped	NULL		-
resample	integer	0		$(-\infty, \infty)$
fillcolor	untyped	0		-
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

`mlr_pipeops_augment_random_choice`
Random Choice Augmentation

Description

Calls `torchvision::transform_random_choice`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
transforms	untyped	-	
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

`mlr_pipeops_augment_random_crop`
Random Crop Augmentation

Description

Calls `torchvision::transform_random_crop`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
size	untyped	-	
padding	untyped	NULL	
pad_if_needed	logical	FALSE	TRUE, FALSE
fill	untyped	0L	
padding_mode	character	constant	constant, edge, reflect, symmetric
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

mlr_pipeops_augment_random_horizontal_flip

Random Horizontal Flip Augmentation

Description

Calls `torchvision::transform_random_horizontal_flip`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
p	numeric	0.5		$[0, 1]$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

mlr_pipeops_augment_random_order

Random Order Augmentation

Description

Calls `torchvision::transform_random_order`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
transforms	untyped	-	
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

`mlr_pipeops_augment_random_resized_crop`
Random Resized Crop Augmentation

Description

Calls `torchvision::transform_random_resized_crop`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
size	untyped	-		-
scale	untyped	c(0.08, 1)		-
ratio	untyped	c(3/4, 4/3)		-
interpolation	integer	2		$[0, 3]$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

`mlr_pipeops_augment_random_vertical_flip`
Random Vertical Flip Augmentation

Description

Calls `torchvision::transform_random_vertical_flip`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
p	numeric	0.5		$[0, 1]$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

`mlr_pipeops_augment_resized_crop`
Resized Crop Augmentation

Description

Calls `torchvision::transform_resized_crop`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
top	integer	-		$(-\infty, \infty)$
left	integer	-		$(-\infty, \infty)$
height	integer	-		$(-\infty, \infty)$
width	integer	-		$(-\infty, \infty)$
size	untyped	-		-
interpolation	integer	2		$[0, 3]$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

mlr_pipeops_augment_rotate

Rotate Augmentation

Description

Calls `torchvision::transform_rotate`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
angle	untyped	-		-
resample	integer	0		$(-\infty, \infty)$
expand	logical	FALSE	TRUE, FALSE	-
center	untyped	NULL		-
fill	untyped	NULL		-
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

mlr_pipeops_augment_vflip

Vertical Flip Augmentation

Description

Calls `torchvision::transform_vflip`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

`mlr_pipeops_module` *Class for Torch Module Wrappers*

Description

`PipeOpModule` wraps an `nn_module` or `function` that is being called during the `train` phase of this `mlr3pipelines::PipeOp`. By doing so, this allows to assemble `PipeOpModules` in a computational `mlr3pipelines::Graph` that represents either a neural network or a preprocessing graph of a `lazy_tensor`. In most cases it is easier to create such a network by creating a graph that generates this graph.

In most cases it is easier to create such a network by creating a structurally related graph consisting of nodes of class `PipeOpTorchIngress` and `PipeOpTorch`. This graph will then generate the graph consisting of `PipeOpModules` as part of the `ModelDescriptor`.

Input and Output Channels

The number and names of the input and output channels can be set during construction. They input and output `"torch_tensor"` during training, and `NULL` during prediction as the prediction phase currently serves no meaningful purpose.

State

The state is the value calculated by the public method `shapes_out()`.

Parameters

No parameters.

Internals

During training, the wrapped `nn_module` / `function` is called with the provided inputs in the order in which the channels are defined. Arguments are **not** matched by name.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpModule`

Public fields

`module` (`nn_module`)
The torch module that is called during the training phase.

Methods

Public methods:

- `PipeOpModule$new()`
- `PipeOpModule$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpModule$new(
  id = "module",
  module = nn_identity(),
  inname = "input",
  outname = "output",
  param_vals = list(),
  packages = character(0)
)
```

Arguments:

`id` (`character(1)`)
The id for of the new object.

`module` (`nn_module` or `function()`)
The torch module or function that is being wrapped.

`inname` (`character()`)
The names of the input channels.

`outname` (`character()`)
The names of the output channels. If this parameter has length 1, the parameter module must return a tensor. Otherwise it must return a `list()` of tensors of corresponding length.

`param_vals` (named `list()`)
Parameter values to be set after construction.

`packages` (`character()`)
The R packages this object depends on.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpModule$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Graph Network: `ModelDescriptor()`, `TorchIngressToken()`, `mlr_learners_torch_model`, `mlr_pipeops_torch`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingres`

mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union(), nn_graph()

Other PipeOp: mlr_pipeops_torch_callbacks, mlr_pipeops_torch_optimizer

Examples

```
## creating an PipeOpModule manually

# one input and output channel
po_module = po("module",
  id = "linear",
  module = torch::nn_linear(10, 20),
  inname = "input",
  outname = "output"
)
x = torch::torch_randn(16, 10)
# This calls the forward function of the wrapped module.
y = po_module$train(list(input = x))
str(y)

# multiple input and output channels
nn_custom = torch::nn_module("nn_custom",
  initialize = function(in_features, out_features) {
    self$lin1 = torch::nn_linear(in_features, out_features)
    self$lin2 = torch::nn_linear(in_features, out_features)
  },
  forward = function(x, z) {
    list(out1 = self$lin1(x), out2 = torch::nnf_relu(self$lin2(z)))
  }
)

module = nn_custom(3, 2)
po_module = po("module",
  id = "custom",
  module = module,
  inname = c("x", "z"),
  outname = c("out1", "out2")
)
x = torch::torch_randn(1, 3)
z = torch::torch_randn(1, 3)
out = po_module$train(list(x = x, z = z))
str(out)

# How such a PipeOpModule is usually generated
graph = po("torch_ingress_num") %>>% po("nn_linear", out_features = 10L)
result = graph$train(tsk("iris"))
# The PipeOpTorchLinear generates a PipeOpModule and adds it to a new (module) graph
result[[1]]$graph
linear_module = result[[1L]]$graph$pipeops$nn_linear
linear_module
formalArgs(linear_module$module)
linear_module$input$name
```

```
# Constructing a PipeOpModule using a simple function
po_add1 = po("module",
  id = "add_one",
  module = function(x) x + 1
)
input = list(torch_tensor(1))
po_add1$train(input)$output
```

`mlr_pipeops_nn_adaptive_avg_pool1d` *1D Adaptive Average Pooling*

Description

Applies a 1D adaptive average pooling over an input signal composed of several input planes.

nn_module

Calls `nn_adaptive_avg_pool1d()` during training.

Parameters

- `output_size` :: `integer(1)`
  The target output size. A single number.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchAdaptiveAvgPool` -> `PipeOpTorchAdaptiveAvgPool1D`

Methods

Public methods:

- `PipeOpTorchAdaptiveAvgPool1D$new()`
- `PipeOpTorchAdaptiveAvgPool1D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchAdaptiveAvgPool1D$new(
  id = "nn_adaptive_avg_pool1d",
  param_vals = list()
)
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchAdaptiveAvgPool1D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_n`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_transpose2d`, `mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_adaptive_avg_pool1d")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_adaptive_avg_pool2d` *2D Adaptive Average Pooling*

Description

Applies a 2D adaptive average pooling over an input signal composed of several input planes.

nn_module

Calls `nn_adaptive_avg_pool2d()` during training.

Parameters

- `output_size` :: `integer()`
  The target output size. Can be a single number or a vector.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchAdaptiveAvgPool` -> `PipeOpTorchAdaptiveAvgPool2D`

Methods

Public methods:

- `PipeOpTorchAdaptiveAvgPool2D$new()`
- `PipeOpTorchAdaptiveAvgPool2D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchAdaptiveAvgPool2D$new(
  id = "nn_adaptive_avg_pool2d",
  param_vals = list()
)
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchAdaptiveAvgPool2D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_n`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_transpose2d`, `mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingres`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_adaptive_avg_pool2d")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_adaptive_avg_pool3d

3D Adaptive Average Pooling

Description

Applies a 3D adaptive average pooling over an input signal composed of several input planes.

nn_module

Calls `nn_adaptive_avg_pool3d()` during training.

Parameters

- `output_size` :: `integer()`
  The target output size. Can be a single number or a vector.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchAdaptiveAvgPool` -> `PipeOpTorchAdaptiveAvgPool3D`

Methods

Public methods:

- `PipeOpTorchAdaptiveAvgPool3D$new()`
- `PipeOpTorchAdaptiveAvgPool3D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchAdaptiveAvgPool3D$new(
  id = "nn_adaptive_avg_pool3d",
  param_vals = list()
)
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchAdaptiveAvgPool3D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_n mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoi mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingres mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_adaptive_avg_pool3d")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_avg_pool1d

1D Average Pooling

Description

Applies a 1D average pooling over an input signal composed of several input planes.

nn_module

Calls nn_avg_pool1d() during training.

Parameters

- `kernel_size` :: (`integer()`)
  The size of the window. Can be a single number or a vector.
- `stride` :: `integer()`
  The stride of the window. Can be a single number or a vector. Default: `kernel_size`.

- padding :: integer()
  Implicit zero paddings on both sides of the input. Can be a single number or a vector. Default: 0.
- ceil_mode :: integer()
  When TRUE, will use ceil instead of floor to compute the output shape. Default: FALSE.
- count_include_pad :: logical(1)
  When TRUE, will include the zero-padding in the averaging calculation. Default: TRUE.
- divisor_override :: logical(1)
  If specified, it will be used as divisor, otherwise size of the pooling region will be used. Default: NULL. Only available for dimension greater than 1.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method $shapes_out().

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> mlr3torch::PipeOpTorchAvgPool -> PipeOpTorchAvgPool1D

Methods

Public methods:

- PipeOpTorchAvgPool1D$new()
- PipeOpTorchAvgPool1D$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchAvgPool1D$new(id = "nn_avg_pool1d", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of the resulting object.

param_vals (list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchAvgPool1D$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_avg_pool1d")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_avg_pool2d

2D Average Pooling

Description

Applies 2D average-pooling operation in $kH * kW$ regions by step size $sH * sW$ steps. The number of output features is equal to the number of input planes.

nn_module

Calls nn_avg_pool2d() during training.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `kernel_size` :: `(integer())`
  The size of the window. Can be a single number or a vector.
- `stride` :: `integer()`
  The stride of the window. Can be a single number or a vector. Default: `kernel_size`.
- `padding` :: `integer()`
  Implicit zero paddings on both sides of the input. Can be a single number or a vector. Default: 0.
- `ceil_mode` :: `integer()`
  When `TRUE`, will use ceil instead of floor to compute the output shape. Default: `FALSE`.
- `count_include_pad` :: `logical(1)`
  When `TRUE`, will include the zero-padding in the averaging calculation. Default: `TRUE`.
- `divisor_override` :: `logical(1)`
  If specified, it will be used as divisor, otherwise size of the pooling region will be used. Default: NULL. Only available for dimension greater than 1.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchAvgPool` -> `PipeOpTorchAvgPool2D`

Methods

Public methods:

- `PipeOpTorchAvgPool2D$new()`
- `PipeOpTorchAvgPool2D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchAvgPool2D$new(id = "nn_avg_pool2d", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchAvgPool2D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_avg_pool2d")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_avg_pool3d`
3D Average Pooling

Description

Applies 3D average-pooling operation in $kT * kH * kW$ regions by step size $sT * sH * sW$ steps. The number of output features is equal to $\lfloor \frac{\text{input planes}}{sT} \rfloor$.

Internals

Calls `nn_avg_pool3d()` during training.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `kernel_size` :: `(integer())`
  The size of the window. Can be a single number or a vector.
- `stride` :: `integer()`
  The stride of the window. Can be a single number or a vector. Default: `kernel_size`.
- `padding` :: `integer()`
  Implicit zero paddings on both sides of the input. Can be a single number or a vector. Default: 0.
- `ceil_mode` :: `integer()`
  When `TRUE`, will use ceil instead of floor to compute the output shape. Default: `FALSE`.
- `count_include_pad` :: `logical(1)`
  When `TRUE`, will include the zero-padding in the averaging calculation. Default: `TRUE`.
- `divisor_override` :: `logical(1)`
  If specified, it will be used as divisor, otherwise size of the pooling region will be used. Default: NULL. Only available for dimension greater than 1.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchAvgPool` -> `PipeOpTorchAvgPool3D`

Methods

Public methods:

- `PipeOpTorchAvgPool3D$new()`
- `PipeOpTorchAvgPool3D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchAvgPool3D$new(id = "nn_avg_pool3d", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchAvgPool3D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_avg_pool3d")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_batch_norm1d

1D Batch Normalization

Description

Applies Batch Normalization for each channel across a batch of data.

nn_module

Calls torch::nn_batch_norm1d(). The parameter num_features is inferred as the second dimension of the input shape.

Parameters

- eps :: numeric(1)
  A value added to the denominator for numerical stability. Default: 1e-5.
- momentum :: numeric(1)
  The value used for the running_mean and running_var computation. Can be set to NULL for cumulative moving average (i.e. simple average). Default: 0.1

- `affine` :: `logical(1)`
  a boolean value that when set to `TRUE`, this module has learnable affine parameters. Default: `TRUE`
- `track_running_stats` :: `logical(1)`
  a boolean value that when set to `TRUE`, this module tracks the running mean and variance, and when set to `FALSE`, this module does not track such statistics and always uses batch statistics in both training and eval modes. Default: `TRUE`

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchBatchNorm` -> `PipeOpTorchBatchNorm1D`

Methods

Public methods:

- `PipeOpTorchBatchNorm1D$new()`
- `PipeOpTorchBatchNorm1D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchBatchNorm1D$new(id = "nn_batch_norm1d", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchBatchNorm1D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_batch_norm1d")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_batch_norm2d`

2D Batch Normalization

Description

Applies Batch Normalization for each channel across a batch of data.

nn_module

Calls `torch::nn_batch_norm2d()`. The parameter `num_features` is inferred as the second dimension of the input shape.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `eps` :: `numeric(1)`
  A value added to the denominator for numerical stability. Default: `1e-5`.
- `momentum` :: `numeric(1)`
  The value used for the running_mean and running_var computation. Can be set to `NULL` for cumulative moving average (i.e. simple average). Default: 0.1
- `affine` :: `logical(1)`
  a boolean value that when set to `TRUE`, this module has learnable affine parameters. Default: `TRUE`
- `track_running_stats` :: `logical(1)`
  a boolean value that when set to `TRUE`, this module tracks the running mean and variance, and when set to `FALSE`, this module does not track such statistics and always uses batch statistics in both training and eval modes. Default: `TRUE`

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchBatchNorm` -> `PipeOpTorchBatchNorm2D`

Methods

Public methods:

- `PipeOpTorchBatchNorm2D$new()`
- `PipeOpTorchBatchNorm2D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchBatchNorm2D$new(id = "nn_batch_norm2d", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchBatchNorm2D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_batch_norm2d")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_batch_norm3d`

3D Batch Normalization

Description

Applies Batch Normalization for each channel across a batch of data.

nn_module

Calls `torch::nn_batch_norm3d()`. The parameter `num_features` is inferred as the second dimension of the input shape.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `eps` :: `numeric(1)`
  A value added to the denominator for numerical stability. Default: `1e-5`.
- `momentum` :: `numeric(1)`
  The value used for the running_mean and running_var computation. Can be set to `NULL` for cumulative moving average (i.e. simple average). Default: 0.1
- `affine` :: `logical(1)`
  a boolean value that when set to `TRUE`, this module has learnable affine parameters. Default: `TRUE`
- `track_running_stats` :: `logical(1)`
  a boolean value that when set to `TRUE`, this module tracks the running mean and variance, and when set to `FALSE`, this module does not track such statistics and always uses batch statistics in both training and eval modes. Default: `TRUE`

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchBatchNorm` -> `PipeOpTorchBatchNorm3D`

Methods

Public methods:

- `PipeOpTorchBatchNorm3D$new()`
- `PipeOpTorchBatchNorm3D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchBatchNorm3D$new(id = "nn_batch_norm3d", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchBatchNorm3D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_batch_norm3d")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_block *Block Repetition*

Description

Repeat a block `n_blocks` times by concatenating it with itself (via `%>>%`).

Naming

For the generated module graph, the IDs of the modules are generated by prefixing the IDs of the `n_blocks` layers with the ID of the `PipeOpTorchBlock` and postfixing them with `__<layer>`.

Parameters

The parameters available for the block itself, as well as

- `n_blocks` :: `integer(1)`
  How often to repeat the block.

Input and Output Channels

The PipeOp sets its input and output channels to those from the block (Graph) it received during construction.

State

The state is the value calculated by the public method $shapes_out().

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> PipeOpTorchBlock

Active bindings

block (Graph)
The neural network segment that is repeated by this PipeOp.

Methods

Public methods:

- PipeOpTorchBlock$new()
- PipeOpTorchBlock$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchBlock$new(block, id = "nn_block", param_vals = list())
```

Arguments:

block (Graph)
A graph consisting primarily of PipeOpTorch objects that is to be repeated.

id (character(1))
The id for of the new object.

param_vals (named list())
Parameter values to be set after construction.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchBlock$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d,

mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
block = po("nn_linear") %>>% po("nn_relu")
po_block = po("nn_block", block,
nn_linear.out_features = 10L, n_blocks = 3)
network = po("torch_ingress_num") %>>%
po_block %>>%
po("nn_head") %>>%
po("torch_loss", t_loss("cross_entropy")) %>>%
po("torch_optimizer", t_opt("adam")) %>>%
po("torch_model_classif",
  batch_size = 50,
  epochs = 3)

task = tsk("iris")
network$train(task)
```

---

mlr_pipeops_nn_celu *CELU Activation Function*

Description

Applies element-wise, $CELU(x) = max(0, x) + min(0, \alpha * (exp(x\alpha) - 1))$.

nn_module

Calls `torch::nn_celu()` when trained.

Parameters

- `alpha` :: `numeric(1)`
  The alpha value for the ELU formulation. Default: 1.0
- `inplace` :: `logical(1)`
  Whether to do the operation in-place. Default: `FALSE`.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method $shapes_out().

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> PipeOpTorchCELU

Methods

Public methods:

- PipeOpTorchCELU$new()
- PipeOpTorchCELU$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchCELU$new(id = "nn_celu", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of the resulting object.

param_vals (list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchCELU$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod,

mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_celu")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_conv1d *1D Convolution*

Description

Applies a 1D convolution over an input signal composed of several input planes.

nn_module

Calls `torch::nn_conv1d()` when trained. The paramter `in_channels` is inferred from the second dimension of the input tensor.

Parameters

- `out_channels` :: `integer(1)`
  Number of channels produced by the convolution.
- `kernel_size` :: `integer()`
  Size of the convolving kernel.
- `stride` :: `integer()`
  Stride of the convolution. The default is 1.
- `padding` :: `integer()`
  'dilation * (kernel_size - 1) - padding' zero-padding will be added to both sides of the input. Default: 0.
- `groups` :: `integer()`
  Number of blocked connections from input channels to output channels. Default: 1
- `bias` :: `logical(1)`
  If 'TRUE', adds a learnable bias to the output. Default: 'TRUE'.
- `dilation` :: `integer()`
  Spacing between kernel elements. Default: 1.

- padding_mode :: character(1)
  The padding mode. One of "zeros", "reflect", "replicate", or "circular". Default is "zeros".

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method $shapes_out().

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> mlr3torch::PipeOpTorchConv -> PipeOpTorchConv1D

Methods

Public methods:

- PipeOpTorchConv1D$new()
- PipeOpTorchConv1D$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchConv1D$new(id = "nn_conv1d", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of the resulting object.

param_vals (list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchConv1D$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten,

mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_conv1d", kernel_size = 10, out_channels = 1)
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_conv2d` *2D Convolution*

Description

Applies a 2D convolution over an input image composed of several input planes.

nn_module

Calls `torch::nn_conv2d()` when trained. The paramter `in_channels` is inferred from the second dimension of the input tensor.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `out_channels` :: `integer(1)`
  Number of channels produced by the convolution.
- `kernel_size` :: `integer()`
  Size of the convolving kernel.

- `stride` :: `integer()`
  Stride of the convolution. The default is 1.
- `padding` :: `integer()`
  'dilation * (kernel_size - 1) - padding' zero-padding will be added to both sides of the input. Default: 0.
- `groups` :: `integer()`
  Number of blocked connections from input channels to output channels. Default: 1
- `bias` :: `logical(1)`
  If 'TRUE', adds a learnable bias to the output. Default: 'TRUE'.
- `dilation` :: `integer()`
  Spacing between kernel elements. Default: 1.
- `padding_mode` :: `character(1)`
  The padding mode. One of `"zeros"`, `"reflect"`, `"replicate"`, or `"circular"`. Default is `"zeros"`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchConv` -> `PipeOpTorchConv2D`

Methods

Public methods:

- `PipeOpTorchConv2D$new()`
- `PipeOpTorchConv2D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchConv2D$new(id = "nn_conv2d", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchConv2D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_conv2d", kernel_size = 10, out_channels = 1)
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_conv3d *3D Convolution*

Description

Applies a 3D convolution over an input image composed of several input planes.

nn_module

Calls `torch::nn_conv3d()` when trained. The paramter `in_channels` is inferred from the second dimension of the input tensor.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `out_channels` :: `integer(1)`
  Number of channels produced by the convolution.
- `kernel_size` :: `integer()`
  Size of the convolving kernel.
- `stride` :: `integer()`
  Stride of the convolution. The default is 1.
- `padding` :: `integer()`
  'dilation * (kernel_size - 1) - padding' zero-padding will be added to both sides of the input. Default: 0.
- `groups` :: `integer()`
  Number of blocked connections from input channels to output channels. Default: 1
- `bias` :: `logical(1)`
  If 'TRUE', adds a learnable bias to the output. Default: 'TRUE'.
- `dilation` :: `integer()`
  Spacing between kernel elements. Default: 1.
- `padding_mode` :: `character(1)`
  The padding mode. One of `"zeros"`, `"reflect"`, `"replicate"`, or `"circular"`. Default is `"zeros"`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchConv` -> `PipeOpTorchConv3D`

Methods

Public methods:

- `PipeOpTorchConv3D$new()`
- `PipeOpTorchConv3D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchConv3D$new(id = "nn_conv3d", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchConv3D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_transpose2d, mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_conv3d", kernel_size = 10, out_channels = 1)
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_conv_transpose1d

Transpose 1D Convolution

Description

Applies a 1D transposed convolution operator over an input signal composed of several input planes, sometimes also called "deconvolution".

nn_module

Calls nn_conv_transpose1d. The parameter `in_channels` is inferred as the second dimension of the input tensor.

Parameters

- `out_channels` :: `integer(1)`
  Number of output channels produce by the convolution.

- kernel_size :: integer()
  Size of the convolving kernel.
- stride :: integer()
  Stride of the convolution. Default: 1.
- padding :: integer()‘
  ‘dilation * (kernel_size - 1) - padding’ zero-padding will be added to both sides of the input. Default: 0.
- output_padding ::integer()
  Additional size added to one side of the output shape. Default: 0.
- groups :: integer()
  Number of blocked connections from input channels to output channels. Default: 1
- bias :: logical(1)
  If ‘True’, adds a learnable bias to the output. Default: ‘TRUE’.
- dilation :: integer()
  Spacing between kernel elements. Default: 1.
- padding_mode :: character(1)
  The padding mode. One of "zeros", "reflect", "replicate", or "circular". Default is "zeros".

State

The state is the value calculated by the public method $shapes_out().

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> mlr3torch::PipeOpTorchConvTranspose -> PipeOpTorchConvTranspose1D

Methods

Public methods:

- PipeOpTorchConvTranspose1D$new()
- PipeOpTorchConvTranspose1D$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchConvTranspose1D$new(id = "nn_conv_transpose1d", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchConvTranspose1D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose2d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_conv_transpose1d", kernel_size = 3, out_channels = 2)
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_conv_transpose2d`
Transpose 2D Convolution

Description

Applies a 2D transposed convolution operator over an input image composed of several input planes, sometimes also called "deconvolution".

nn_module

Calls nn_conv_transpose2d. The parameter in_channels is inferred as the second dimension of the input tensor.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method $shapes_out().

Parameters

- out_channels :: integer(1)
  Number of output channels produce by the convolution.
- kernel_size :: integer()
  Size of the convolving kernel.
- stride :: integer()
  Stride of the convolution. Default: 1.
- padding :: integer()‘
  ‘dilation * (kernel_size - 1) - padding’ zero-padding will be added to both sides of the input. Default: 0.
- output_padding ::integer()
  Additional size added to one side of the output shape. Default: 0.
- groups :: integer()
  Number of blocked connections from input channels to output channels. Default: 1
- bias :: logical(1)
  If ‘True’, adds a learnable bias to the output. Default: ‘TRUE’.
- dilation :: integer()
  Spacing between kernel elements. Default: 1.
- padding_mode :: character(1)
  The padding mode. One of "zeros", "reflect", "replicate", or "circular". Default is "zeros".

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> mlr3torch::PipeOpTorchConvTranspose -> PipeOpTorchConvTranspose2D

Methods

Public methods:

- PipeOpTorchConvTranspose2D$new()
- PipeOpTorchConvTranspose2D$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchConvTranspose2D$new(id = "nn_conv_transpose2d", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchConvTranspose2D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_conv_transpose2d", kernel_size = 3, out_channels = 2)
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_conv_transpose3d` *Transpose 3D Convolution*

Description

Applies a 3D transposed convolution operator over an input image composed of several input planes, sometimes also called "deconvolution"

nn_module

Calls `nn_conv_transpose3d`. The parameter `in_channels` is inferred as the second dimension of the input tensor.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `out_channels` :: `integer(1)`
  Number of output channels produce by the convolution.
- `kernel_size` :: `integer()`
  Size of the convolving kernel.
- `stride` :: `integer()`
  Stride of the convolution. Default: 1.
- `padding` :: integer()‘
  ‘dilation * (kernel_size - 1) - padding’ zero-padding will be added to both sides of the input. Default: 0.
- `output_padding` ::`integer()`
  Additional size added to one side of the output shape. Default: 0.
- `groups` :: `integer()`
  Number of blocked connections from input channels to output channels. Default: 1
- `bias` :: `logical(1)`
  If ‘True’, adds a learnable bias to the output. Default: ‘TRUE’.
- `dilation` :: `integer()`
  Spacing between kernel elements. Default: 1.
- `padding_mode` :: `character(1)`
  The padding mode. One of `"zeros"`, `"reflect"`, `"replicate"`, or `"circular"`. Default is `"zeros"`.

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> mlr3torch::PipeOpTorchConvTranspose -> PipeOpTorchConvTranspose3D

Methods

Public methods:

- PipeOpTorchConvTranspose3D$new()
- PipeOpTorchConvTranspose3D$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchConvTranspose3D$new(id = "nn_conv_transpose3d", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchConvTranspose3D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_conv_transpose3d", kernel_size = 3, out_channels = 2)
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_dropout` *Dropout*

Description

During training, randomly zeroes some of the elements of the input tensor with probability p using samples from a Bernoulli distribution.

nn_module

Calls `torch::nn_dropout()` when trained.

Parameters

- p :: `numeric(1)`
  Probability of an element to be zeroed. Default: 0.5.
- `inplace` :: `logical(1)`
  If set to `TRUE`, will do this operation in-place. Default: `FALSE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchDropout`

Methods

Public methods:

- PipeOpTorchDropout$new()
- PipeOpTorchDropout$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchDropout$new(id = "nn_dropout", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchDropout$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_dropout")
pipeop
```

```
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_elu` *ELU Activation Function*

Description

Applies element-wise,

$$ELU(x) = max(0, x) + min(0, \alpha * (exp(x) - 1))$$

.

nn_module

Calls `torch::nn_elu()` when trained.

Parameters

- `alpha` :: `numeric(1)`
  The alpha value for the ELU formulation. Default: 1.0
- `inplace` :: `logical(1)`
  Whether to do the operation in-place. Default: `FALSE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchELU`

Methods

Public methods:

- `PipeOpTorchELU$new()`
- `PipeOpTorchELU$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchELU$new(id = "nn_elu", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchELU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`, `mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_elu")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_flatten` *Flattens a Tensor*

Description

For use with nn_sequential.

nn_module

Calls `torch::nn_flatten()` when trained.

Parameters

`start_dim` :: `integer(1)`
At wich dimension to start flattening. Default is 2. `end_dim` :: `integer(1)`
At wich dimension to stop flattening. Default is -1.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchFlatten`

Methods

Public methods:

- `PipeOpTorchFlatten$new()`
- `PipeOpTorchFlatten$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchFlatten$new(id = "nn_flatten", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchFlatten$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_flatten")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_gelu *GELU Activation Function*

Description

Gelu

nn_module

Calls `torch::nn_gelu()` when trained.

Parameters

- `approximate` :: `character(1)`
  Whether to use an approximation algorithm. Default is `"none"`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchGELU`

Methods

Public methods:

- `PipeOpTorchGELU$new()`
- `PipeOpTorchGELU$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchGELU$new(id = "nn_gelu", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchGELU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`,

mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_gelu")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_glu` *GLU Activation Function*

Description

The gated linear unit. Computes:

nn_module

Calls `torch::nn_glu()` when trained.

Parameters

- `dim` :: `integer(1)`
  Dimension on which to split the input. Default: -1

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchGLU`

Methods

Public methods:

- PipeOpTorchGLU$new()
- PipeOpTorchGLU$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchGLU$new(id = "nn_glu", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchGLU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_glu")
pipeop
```

```
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_hardshrink` *Hard Shrink Activation Function*

Description

Applies the hard shrinkage function element-wise

nn_module

Calls `torch::nn_hardshrink()` when trained.

Parameters

- `lambd` :: `numeric(1)`
  The lambda value for the Hardshrink formulation formulation. Default 0.5.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchHardShrink`

Methods

Public methods:

- `PipeOpTorchHardShrink$new()`
- `PipeOpTorchHardShrink$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchHardShrink$new(id = "nn_hardshrink", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchHardShrink$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_hardshrink")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_hardsigmoid

Hard Sigmoid Activation Function

Description

Applies the element-wise function $\text{Hardsigmoid}(x) = \frac{ReLU6(x+3)}{6}$

nn_module

Calls `torch::nn_hardsigmoid()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchHardSigmoid`

Methods

Public methods:

- `PipeOpTorchHardSigmoid$new()`
- `PipeOpTorchHardSigmoid$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchHardSigmoid$new(id = "nn_hardsigmoid", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchHardSigmoid$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_hardsigmoid")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_hardtanh`

Hard Tanh Activation Function

Description

Applies the HardTanh function element-wise.

nn_module

Calls `torch::nn_hardtanh()` when trained.

Parameters

- `min_val` :: `numeric(1)`
  Minimum value of the linear region range. Default: -1.
- `max_val` :: `numeric(1)`
  Maximum value of the linear region range. Default: 1.
- `inplace` :: `logical(1)`
  Can optionally do the operation in-place. Default: `FALSE`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchHardTanh`

Methods

Public methods:

- `PipeOpTorchHardTanh$new()`
- `PipeOpTorchHardTanh$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchHardTanh$new(id = "nn_hardtanh", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchHardTanh$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_hardtanh")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_head` *Output Head*

Description

Output head for classification and regresssion.

Details

When the method `$shapes_out()` does not have access to the task, it returns `c(NA, NA)`. When this `PipeOp` is trained however, the model descriptor has the correct output shape.

nn_module

Calls `torch::nn_linear()` with the input and output features inferred from the input shape / task.

Parameters

- `bias` :: `logical(1)`
  Whether to use a bias. Default is `TRUE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchHead`

Methods

Public methods:

- PipeOpTorchHead$new()
- PipeOpTorchHead$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchHead$new(id = "nn_head", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchHead$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_head")
pipeop
```

```
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_layer_norm` *Layer Normalization*

Description

Applies Layer Normalization for last certain number of dimensions.

nn_module

Calls `torch::nn_layer_norm()` when trained. The parameter `normalized_shape` is inferred as the dimensions of the last `dims` dimensions of the input shape.

Parameters

- `dims` :: `integer(1)`
  The number of dimensions over which will be normalized (starting from the last dimension).
- `elementwise_affine` :: `logical(1)`
  Whether to learn affine-linear parameters initialized to `1` for weights and to `0` for biases. The default is `TRUE`.
- `eps` :: `numeric(1)`
  A value added to the denominator for numerical stability.

State

The state is the value calculated by the public method `$shapes_out()`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchLayerNorm`

Methods

Public methods:

- `PipeOpTorchLayerNorm$new()`
- `PipeOpTorchLayerNorm$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchLayerNorm$new(id = "nn_layer_norm", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchLayerNorm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`, `mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_layer_norm", dims = 1)
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_leaky_relu` *Leaky ReLU Activation Function*

Description

Applies element-wise, $LeakyReLU(x) = max(0, x) + negative_slope * min(0, x)$

nn_module

Calls `torch::nn_leaky_relu()` when trained.

Parameters

- `negative_slope` :: `numeric(1)`
  Controls the angle of the negative slope. Default: 1e-2.
- `inplace` :: `logical(1)`
  Can optionally do the operation in-place. Default: 'FALSE'.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchLeakyReLU`

Methods

Public methods:

- `PipeOpTorchLeakyReLU$new()`
- `PipeOpTorchLeakyReLU$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchLeakyReLU$new(id = "nn_leaky_relu", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchLeakyReLU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_leaky_relu")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_linear *Linear Layer*

Description

Applies a linear transformation to the incoming data: $y = xA^T + b$.

nn_module

Calls `torch::nn_linear()` when trained where the parameter `in_features` is inferred as the second to last dimension of the input tensor.

Parameters

- `out_features` :: `integer(1)`
  The output features of the linear layer.
- `bias` :: `logical(1)`
  Whether to use a bias. Default is `TRUE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchLinear`

Methods

Public methods:

- `PipeOpTorchLinear$new()`
- `PipeOpTorchLinear$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchLinear$new(id = "nn_linear", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchLinear$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_linear", out_features = 10)
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_log_sigmoid`

Log Sigmoid Activation Function

Description

Applies element-wise $LogSigmoid(x_i) = log(\frac{1}{1+exp(-x_i)})$

nn_module

Calls `torch::nn_log_sigmoid()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchLogSigmoid`

Methods

Public methods:

- `PipeOpTorchLogSigmoid$new()`
- `PipeOpTorchLogSigmoid$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchLogSigmoid$new(id = "nn_log_sigmoid", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchLogSigmoid$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_mode`
`mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_log_sigmoid")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_max_pool1d`
1D Max Pooling

Description

Applies a 1D max pooling over an input signal composed of several input planes.

nn_module

Calls `torch::nn_max_pool1d()` during training.

Parameters

- `kernel_size` :: `integer()`
  The size of the window. Can be single number or a vector.
- `stride` :: `(integer(1))`
  The stride of the window. Can be a single number or a vector. Default: `kernel_size`
- `padding` :: `integer()`
  Implicit zero paddings on both sides of the input. Can be a single number or a tuple (padW,). Default: 0
- `dilation` :: `integer()`
  Controls the spacing between the kernel points; also known as the à trous algorithm. Default: 1
- `ceil_mode` :: `logical(1)`
  When True, will use ceil instead of floor to compute the output shape. Default: `FALSE`

Input and Output Channels

If `return_indices` is `FALSE` during construction, there is one input channel 'input' and one output channel 'output'. If `return_indices` is `TRUE`, there are two output channels 'output' and 'indices'. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> mlr3torch::PipeOpTorchMaxPool -> PipeOpTorchMaxPool1D

Methods

Public methods:

- PipeOpTorchMaxPool1D$new()
- PipeOpTorchMaxPool1D$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchMaxPool1D$new(
  id = "nn_max_pool1d",
  return_indices = FALSE,
  param_vals = list()
)
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`return_indices (logical(1))`
: Whether to return the indices. If this is `TRUE`, there are two output channels `"output"` and `"indices"`.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchMaxPool1D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3 mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape,

mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_max_pool1d")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_max_pool2d`
2D Max Pooling

Description

Applies a 2D max pooling over an input signal composed of several input planes.

nn_module

Calls `torch::nn_max_pool2d()` during training.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `kernel_size` :: `integer()`
  The size of the window. Can be single number or a vector.
- `stride` :: `(integer(1))`
  The stride of the window. Can be a single number or a vector. Default: `kernel_size`
- `padding` :: `integer()`
  Implicit zero paddings on both sides of the input. Can be a single number or a tuple (padW,). Default: 0
- `dilation` :: `integer()`
  Controls the spacing between the kernel points; also known as the à trous algorithm. Default: 1
- `ceil_mode` :: `logical(1)`
  When True, will use ceil instead of floor to compute the output shape. Default: `FALSE`

Input and Output Channels

If `return_indices` is `FALSE` during construction, there is one input channel 'input' and one output channel 'output'. If `return_indices` is `TRUE`, there are two output channels 'output' and 'indices'. For an explanation see `PipeOpTorch`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchMaxPool` -> `PipeOpTorchMaxPool2D`

Methods

Public methods:

- `PipeOpTorchMaxPool2D$new()`
- `PipeOpTorchMaxPool2D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchMaxPool2D$new(
  id = "nn_max_pool2d",
  return_indices = FALSE,
  param_vals = list()
)
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`return_indices (logical(1))`
: Whether to return the indices. If this is `TRUE`, there are two output channels `"output"` and `"indices"`.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchMaxPool2D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`

mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_max_pool2d")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_max_pool3d`
3D Max Pooling

Description

Applies a 3D max pooling over an input signal composed of several input planes.

nn_module

Calls `torch::nn_max_pool3d()` during training.

State

The state is the value calculated by the public method `$shapes_out()`.

Parameters

- `kernel_size` :: `integer()`
  The size of the window. Can be single number or a vector.
- `stride` :: `(integer(1))`
  The stride of the window. Can be a single number or a vector. Default: `kernel_size`
- `padding` :: `integer()`
  Implicit zero paddings on both sides of the input. Can be a single number or a tuple (padW,). Default: 0

- `dilation` :: `integer()`
  Controls the spacing between the kernel points; also known as the à trous algorithm. Default: 1
- `ceil_mode` :: `logical(1)`
  When True, will use ceil instead of floor to compute the output shape. Default: `FALSE`

Input and Output Channels

If `return_indices` is `FALSE` during construction, there is one input channel 'input' and one output channel 'output'. If `return_indices` is `TRUE`, there are two output channels 'output' and 'indices'. For an explanation see `PipeOpTorch`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchMaxPool` -> `PipeOpTorchMaxPool3D`

Methods

Public methods:

- `PipeOpTorchMaxPool3D$new()`
- `PipeOpTorchMaxPool3D$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchMaxPool3D$new(
  id = "nn_max_pool3d",
  return_indices = FALSE,
  param_vals = list()
)
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`return_indices (logical(1))`
: Whether to return the indices. If this is `TRUE`, there are two output channels `"output"` and `"indices"`.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchMaxPool3D$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2 mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_max_pool3d")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_merge *Merge Operation*

Description

Base class for merge operations such as addition (PipeOpTorchMergeSum), multiplication (PipeOpTorchMergeProd or concatenation (PipeOpTorchMergeCat).

Parameters

See the respective child class.

State

The state is the value calculated by the public method `shapes_out()`.

Input and Output Channels

`PipeOpTorchMerges` has either a *vararg* input channel if the constructor argument `innum` is not set, or input channels `"input1"`, ..., `"input<innum>"`. There is one output channel `"output"`. For an explanation see PipeOpTorch.

Internals

Per default, the `private$.shapes_out()` method outputs the broadcasted tensors. There are two things to be aware:

1. NAs are assumed to batch (this should almost always be the batch size in the first dimension).
2. Tensors are expected to have the same number of dimensions, i.e. missing dimensions are not filled with 1s. The reason is again that the first dimension should be the batch dimension. This private method can be overwritten by PipeOpTorchs inheriting from this class.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchMerge`

Methods

Public methods:

- `PipeOpTorchMerge$new()`
- `PipeOpTorchMerge$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchMerge$new(
  id,
  module_generator,
  param_set = ps(),
  innum = 0,
  param_vals = list()
)
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`module_generator (nn_module_generator)`
: The torch module generator.

`param_set (ParamSet)`
: The parameter set.

`innum (integer(1))`
: The number of inputs. Default is 0 which means there is one *vararg* input channel.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchMerge$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_su mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

---

`mlr_pipeops_nn_merge_cat`

Merge by Concatenation

Description

Concatenates multiple tensors on a given dimension. No broadcasting rules are applied here, you must reshape the tensors before to have the same shape.

nn_module

Calls `nn_merge_cat()` when trained.

Parameters

- `dim` :: `integer(1)`
  The dimension along which to concatenate the tensors.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

`PipeOpTorchMerges` has either a *vararg* input channel if the constructor argument `innum` is not set, or input channels `"input1"`, ..., `"input<innum>"`. There is one output channel `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchMerge` -> `PipeOpTorchMergeCat`

Methods

Public methods:

- `PipeOpTorchMergeCat$new()`
- `PipeOpTorchMergeCat$speak()`
- `PipeOpTorchMergeCat$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchMergeCat$new(id = "nn_merge_cat", innum = 0, param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`innum (integer(1))`
: The number of inputs. Default is 0 which means there is one *vararg* input channel.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `speak()`**:** What does the cat say?

Usage:

```
PipeOpTorchMergeCat$speak()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchMergeCat$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`,

mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_merge_cat")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_merge_prod`
Merge by Product

Description

Calculates the product of all input tensors.

nn_module

Calls `nn_merge_prod()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

`PipeOpTorchMerges` has either a *vararg* input channel if the constructor argument `innum` is not set, or input channels `"input1"`, ..., `"input<innum>"`. There is one output channel `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> mlr3torch::PipeOpTorchMerge -> PipeOpTorchMergeProd

Methods

Public methods:

- PipeOpTorchMergeProd$new()
- PipeOpTorchMergeProd$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchMergeProd$new(id = "nn_merge_prod", innum = 0, param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`innum (integer(1))`
: The number of inputs. Default is 0 which means there is one *vararg* input channel.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchMergeProd$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2
mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_mode
mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_merge_prod")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_merge_sum` *Merge by Summation*

Description

Calculates the sum of all input tensors.

nn_module

Calls `nn_merge_sum()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

`PipeOpTorchMerges` has either a *vararg* input channel if the constructor argument `innum` is not set, or input channels `"input1"`, ..., `"input<innum>"`. There is one output channel `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `mlr3torch::PipeOpTorchMerge` -> `PipeOpTorchMergeSum`

Methods

Public methods:

- PipeOpTorchMergeSum$new()
- PipeOpTorchMergeSum$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchMergeSum$new(id = "nn_merge_sum", innum = 0, param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`innum (integer(1))`
: The number of inputs. Default is 0 which means there is one *vararg* input channel.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchMergeSum$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d,

mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_merge_sum")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_prelu` *PReLU Activation Function*

Description

Applies element-wise the function $PReLU(x) = max(0, x) + weight * min(0, x)$ where weight is a learnable parameter.

nn_module

Calls `torch::nn_prelu()` when trained.

Parameters

- `num_parameters` :: `integer(1)`: Number of a to learn. Although it takes an int as input, there is only two values are legitimate: 1, or the number of channels at input. Default: 1.
- `init` :: `numeric(1)`
  T The initial value of a. Default: 0.25.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchPReLU`

Methods

Public methods:

- `PipeOpTorchPReLU$new()`
- `PipeOpTorchPReLU$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchPReLU$new(id = "nn_prelu", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchPReLU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2`
`mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_mode`
`mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_prelu")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_relu` *ReLU Activation Function*

Description

Applies the rectified linear unit function element-wise.

nn_module

Calls `torch::nn_relu()` when trained.

Parameters

- `inplace` :: `logical(1)`
  Whether to do the operation in-place. Default: `FALSE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchReLU`

Methods

Public methods:

- `PipeOpTorchReLU$new()`
- `PipeOpTorchReLU$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchReLU$new(id = "nn_relu", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchReLU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model`
`mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_relu")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_relu6 *ReLU6 Activation Function*

Description

Applies the element-wise function $ReLU6(x) = min(max(0, x), 6)$.

nn_module

Calls `torch::nn_relu6()` when trained.

Parameters

- `inplace` :: `logical(1)`
  Whether to do the operation in-place. Default: `FALSE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchReLU6`

Methods

Public methods:

- `PipeOpTorchReLU6$new()`
- `PipeOpTorchReLU6$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchReLU6$new(id = "nn_relu6", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchReLU6$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2 mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_relu6")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_reshape

Reshape a Tensor

Description

Reshape a tensor to the given shape.

nn_module

Calls nn_reshape() when trained. This internally calls torch::torch_reshape() with the given shape.

Parameters

- shape :: integer(1)
  The desired output shape. Unknown dimension (one at most) can either be specified as -1 or NA.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method $shapes_out().

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> PipeOpTorchReshape

Methods

Public methods:

- PipeOpTorchReshape$new()
- PipeOpTorchReshape$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchReshape$new(id = "nn_reshape", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of the resulting object.

param_vals (list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchReshape$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2
mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod,

mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_reshape")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_rrelu *RReLU Activation Function*

Description

Randomized leaky ReLU.

nn_module

Calls `torch::nn_rrelu()` when trained.

Parameters

- `lower`:: `numeric(1)`
  Lower bound of the uniform distribution. Default: 1/8.
- `upper`:: `numeric(1)`
  Upper bound of the uniform distribution. Default: 1/3.
- `inplace` :: `logical(1)`
  Whether to do the operation in-place. Default: `FALSE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchRReLU`

Methods

Public methods:

- PipeOpTorchRReLU$new()
- PipeOpTorchRReLU$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchRReLU$new(id = "nn_rrelu", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchRReLU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_rrelu")
pipeop
```

```
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_selu` *SELU Activation Function*

Description

Applies element-wise,

$$SELU(x) = scale * (max(0, x) + min(0, \alpha * (exp(x) - 1)))$$

, with $\alpha = 1.6732632423543772848170429916717$ and $scale = 1.0507009873554804934193349852946$.

nn_module

Calls `torch::nn_selu()` when trained.

Parameters

- `inplace` :: `logical(1)`
  Whether to do the operation in-place. Default: `FALSE`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchSELU`

Methods

Public methods:

- `PipeOpTorchSELU$new()`
- `PipeOpTorchSELU$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchSELU$new(id = "nn_selu", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchSELU$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`, `mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_selu")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_sigmoid

Sigmoid Activation Function

Description

Applies element-wise $Sigmoid(x_i) = \frac{1}{1+exp(-x_i)}$

nn_module

Calls `torch::nn_sigmoid()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchSigmoid`

Methods

Public methods:

- `PipeOpTorchSigmoid$new()`
- `PipeOpTorchSigmoid$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchSigmoid$new(id = "nn_sigmoid", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchSigmoid$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_sigmoid")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_softmax

Softmax

Description

Applies a softmax function.

nn_module

Calls `torch::nn_softmax()` when trained.

Parameters

- `dim` :: `integer(1)`
  A dimension along which Softmax will be computed (so every slice along dim will sum to 1).

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchSoftmax`

Methods

Public methods:

- `PipeOpTorchSoftmax$new()`
- `PipeOpTorchSoftmax$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchSoftmax$new(id = "nn_softmax", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchSoftmax$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2`
`mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_mode`
`mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_softmax")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_softplus` *SoftPlus Activation Function*

Description

Applies element-wise, the function $Softplus(x) = 1/\beta * log(1 + exp(\beta * x))$.

nn_module

Calls `torch::nn_softplus()` when trained.

Parameters

- `beta` :: `numeric(1)`
  The beta value for the Softplus formulation. Default: 1
- `threshold` :: `numeric(1)`
  Values above this revert to a linear function. Default: 20

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchSoftPlus`

Methods

Public methods:

- `PipeOpTorchSoftPlus$new()`
- `PipeOpTorchSoftPlus$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchSoftPlus$new(id = "nn_softplus", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchSoftPlus$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2 mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_softplus")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_softshrink` *Soft Shrink Activation Function*

Description

Applies the soft shrinkage function elementwise

nn_module

Calls `torch::nn_softshrink()` when trained.

Parameters

- `lamd` :: `numeric(1)`
  The lambda (must be no less than zero) value for the Softshrink formulation. Default: 0.5

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchSoftShrink`

Methods

Public methods:

- `PipeOpTorchSoftShrink$new()`
- `PipeOpTorchSoftShrink$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchSoftShrink$new(id = "nn_softshrink", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchSoftShrink$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_softshrink")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_softsign`

SoftSign Activation Function

Description

Applies element-wise, the function $SoftSign(x) = x/(1 + |x|$

nn_module

Calls `torch::nn_softsign()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method $shapes_out().

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> PipeOpTorchSoftSign

Methods

Public methods:

- PipeOpTorchSoftSign$new()
- PipeOpTorchSoftSign$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchSoftSign$new(id = "nn_softsign", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of the resulting object.

param_vals (list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchSoftSign$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid,

mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_softsign")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_squeeze` *Squeeze a Tensor*

Description

Squeezes a tensor by calling `torch::torch_squeeze()` with the given dimension `dim`.

nn_module

Calls `nn_squeeze()` when trained.

Parameters

- `dim` :: `integer(1)`
  The dimension to squeeze. If `NULL`, all dimensions of size 1 will be squeezed. Negative values are interpreted downwards from the last dimension.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> PipeOpTorchSqueeze

Methods

Public methods:

- PipeOpTorchSqueeze$new()
- PipeOpTorchSqueeze$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchSqueeze$new(id = "nn_squeeze", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchSqueeze$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2
mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model
mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_squeeze")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_tanh` *Tanh Activation Function*

Description

Applies the element-wise function:

nn_module

Calls `torch::nn_tanh()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchTanh`

Methods

Public methods:

- `PipeOpTorchTanh$new()`
- `PipeOpTorchTanh$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchTanh$new(id = "nn_tanh", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchTanh$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran` `mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2` `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model` `mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_tanh")
pipeop
# The available parameters
pipeop$param_set
```

---

mlr_pipeops_nn_tanhshrink

Tanh Shrink Activation Function

Description

Applies element-wise, $Tanhshrink(x) = x - Tanh(x)$

nn_module

Calls `torch::nn_tanhshrink()` when trained.

Parameters

No parameters.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchTanhShrink`

Methods

Public methods:

- `PipeOpTorchTanhShrink$new()`
- `PipeOpTorchTanhShrink$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchTanhShrink$new(id = "nn_tanhshrink", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchTanhShrink$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_tanhshrink")
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_threshold`

Treshold Activation Function

Description

Thresholds each element of the input Tensor.

nn_module

Calls `torch::nn_threshold()` when trained.

Parameters

- `threshold` :: `numeric(1)`
  The value to threshold at.
- `value` :: `numeric(1)`
  The value to replace with.
- `inplace` :: `logical(1)`
  Can optionally do the operation in-place. Default: 'FALSE'.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method $shapes_out().

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorch -> PipeOpTorchThreshold

Methods

Public methods:

- PipeOpTorchThreshold$new()
- PipeOpTorchThreshold$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchThreshold$new(id = "nn_threshold", param_vals = list())
```

Arguments:

id (character(1))
: Identifier of the resulting object.

param_vals (list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchThreshold$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2
mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod,

mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Examples

```
# Construct the PipeOp
pipeop = po("nn_threshold", threshold = 1, value = 2)
pipeop
# The available parameters
pipeop$param_set
```

`mlr_pipeops_nn_unsqueeze` *Unqueeze a Tensor*

Description

Unsqueezes a tensor by calling `torch::torch_unsqueeze()` with the given dimension `dim`.

nn_module

Calls `nn_unsqueeze()` when trained. This internally calls `torch::torch_unsqueeze()`.

Parameters

- `dim` :: `integer(1)`
  The dimension which to unsqueeze. Negative values are interpreted downwards from the last dimension.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is the value calculated by the public method `$shapes_out()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorch` -> `PipeOpTorchUnsqueeze`

Methods

Public methods:

- `PipeOpTorchUnsqueeze$new()`
- `PipeOpTorchUnsqueeze$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchUnsqueeze$new(id = "nn_unsqueeze", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchUnsqueeze$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2`
`mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model`
`mlr_pipeops_torch_model_regr`

Examples

```
# Construct the PipeOp
pipeop = po("nn_unsqueeze")
pipeop
```

```
# The available parameters
pipeop$param_set
```

`mlr_pipeops_preproc_torch`

Base Class for Lazy Tensor Preprocessing

Description

This `PipeOp` can be used to preprocess (one or more) `lazy_tensor` columns contained in an `mlr3::Task`. The preprocessing function is specified as construction argument `fn` and additional arguments to this function can be defined through the `PipeOp`'s parameter set. The preprocessing is done per column, i.e. the number of lazy tensor output columns is equal to the number of lazy tensor input columns.

To create custom preprocessing `PipeOps` you can use `pipeop_preproc_torch`.

Inheriting

In addition to specifying the construction arguments, you can overwrite the private `.shapes_out()` method. If you don't overwrite it, the output shapes are assumed to be unknown (`NULL`).

- `.shapes_out(shapes_in, param_vals, task)`
  (`list()`, `list()`, TaskorNULL) -> list()\cr This private method calculates the output shapes of the lazy
  This private method only has the responsibility to calculate the output shapes for one input column, i.e. the input `shapes_in` can be assumed to have exactly one shape vector for which it must calculate the output shapes and return it as a `list()` of length 1. It can also be assumed that the shape is not `NULL` (i.e. unknown). Also, the first dimension can be `NA`, i.e. is unknown (as for the batch dimension).

Input and Output Channels

See `PipeOpTaskPreproc`.

State

In addition to state elements from `PipeOpTaskPreprocSimple`, the state also contains the `$param_vals` that were set during training.

Parameters

In addition to the parameters inherited from `PipeOpTaskPreproc` as well as those specified during construction as the argument `param_set` there are the following parameters:

- `stages` :: `character(1)`
  The stages during which to apply the preprocessing. Can be one of `"train"`, `"predict"` or `"both"`. The initial value of this parameter is set to `"train"` when the `PipeOp`'s id starts with `"augment_"` and to `"both"` otherwise. Note that the preprocessing that is applied during `$predict()` uses the parameters that were set during `$train()` and not those that are set when performing the prediction.

Internals

During `$train()` / `$predict()`, a `PipeOpModule` with one input and one output channel is created. The pipeop applies the function `fn` to the input tensor while additionally passing the parameter values (minus `stages` and `affect_columns`) to `fn`. The preprocessing graph of the lazy tensor columns is shallowly cloned and the `PipeOpModule` is added. This is done to avoid modifying user input and means that identical `PipeOpModules` can be part of different preprocessing graphs. This is only possible, because the created `PipeOpModule` is stateless.

At a later point in the graph, preprocessing graphs will be merged if possible to avoid unnecessary computation. This is best illustrated by example: One lazy tensor column's preprocessing graph is `A -> B`. Then, two branches are created `B -> C` and `B -> D`, creating two preprocessing graphs `A -> B -> C` and `A -> B -> D`. When loading the data, we want to run the preprocessing only once, i.e. we don't want to run the `A -> B` part twice. For this reason, `task_dataset()` will try to merge graphs and cache results from graphs. However, only graphs using the same dataset can currently be merged.

Also, the shapes created during `$train()` and `$predict()` might differ. To avoid the creation of graphs where the predict shapes are incompatible with the train shapes, the hypothetical predict shapes are already calculated during `$train()` (this is why the parameters that are set during train are also used during predict) and the `PipeOpTorchModel` will check the train and predict shapes for compatibility before starting the training.

Otherwise, this mechanism is very similar to the `ModelDescriptor` construct.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `PipeOpTaskPreprocTorch`

Active bindings

`fn` The preprocessing function.

`rowwise` Whether the preprocessing is applied rowwise.

Methods

Public methods:

- `PipeOpTaskPreprocTorch$new()`
- `PipeOpTaskPreprocTorch$shapes_out()`
- `PipeOpTaskPreprocTorch$clone()`

Method `new()`**:** Creates a new instance of this `R6` class.

Usage:

```
PipeOpTaskPreprocTorch$new(
  fn,
  id = "preproc_torch",
  param_vals = list(),
  param_set = ps(),
  packages = character(0),
  rowwise = FALSE,
```

```
  stages_init = NULL,
  tags = NULL
)
```

Arguments:

`fn` (`function` or `character(2)`)
: The preprocessing function. Must not modify its input in-place. If it is a `character(2)`, the first element should be the namespace and the second element the name. When the preprocessing function is applied to the tensor, the tensor will be passed by position as the first argument. If the `param_set` is inferred (left as `NULL`) it is assumed that the first argument is the `torch_tensor`.

`id` (`character(1)`)
: The id for of the new object.

`param_vals` (named `list()`)
: Parameter values to be set after construction.

`param_set` (`ParamSet`)
: In case the function `fn` takes additional parameter besides a `torch_tensor` they can be specfied as parameters. None of the parameters can have the `"predict"` tag. All tags should include `"train"`.

`packages` (`character()`)
: The packages the preprocessing function depends on.

`rowwise` (`logical(1)`)
: Whether the preprocessing function is applied rowwise (and then concatenated by row) or directly to the whole tensor. In the first case there is no batch dimension.

`stages_init` (`character(1)`)
: Initial value for the `stages` parameter.

`tags` (`character()`)
: Tags for the pipeop.

Method `shapes_out()`**:** Calculates the output shapes that would result in applying the preprocessing to one or more lazy tensor columns with the provided shape. Names are ignored and only order matters. It uses the parameter values that are currently set.

Usage:

```
PipeOpTaskPreprocTorch$shapes_out(shapes_in, stage = NULL, task = NULL)
```

Arguments:

`shapes_in` (`list()` of (`integer()` or `NULL`))
: The input input shapes of the lazy tensors. `NULL` indicates that the shape is unknown. First dimension must be `NA` (if it is not `NULL`).

`stage` (`character(1)`)
: The stage: either `"train"` or `"predict"`.

`task` (`Task` or `NULL`)
: The task, which is very rarely needed.

Returns: `list()` of (`integer()` or `NULL`)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTaskPreprocTorch$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Creating a simple task
d = data.table(
  x1 = as_lazy_tensor(rnorm(10)),
  x2 = as_lazy_tensor(rnorm(10)),
  x3 = as_lazy_tensor(as.double(1:10)),
  y = rnorm(10)
)

taskin = as_task_regr(d, target = "y")

# Creating a simple preprocessing pipeop
po_simple = po("preproc_torch",
  # get rid of environment baggage
  fn = mlr3misc::crate(function(x, a) x + a),
  param_set = paradox::ps(a = paradox::p_int(tags = c("train", "required")))
)

po_simple$param_set$set_values(
  a = 100,
  affect_columns = selector_name(c("x1", "x2")),
  stages = "both" # use during train and predict
)

taskout_train = po_simple$train(list(taskin))[[1L]]
materialize(taskout_train$data(cols = c("x1", "x2")), rbind = TRUE)

taskout_predict_noaug = po_simple$predict(list(taskin))[[1L]]
materialize(taskout_predict_noaug$data(cols = c("x1", "x2")), rbind = TRUE)

po_simple$param_set$set_values(
  stages = "train"
)

# transformation is not applied
taskout_predict_aug = po_simple$predict(list(taskin))[[1L]]
materialize(taskout_predict_aug$data(cols = c("x1", "x2")), rbind = TRUE)

# Creating a more complex preprocessing PipeOp
PipeOpPreprocTorchPoly = R6::R6Class("PipeOpPreprocTorchPoly",
 inherit = PipeOpTaskPreprocTorch,
 public = list(
   initialize = function(id = "preproc_poly", param_vals = list()) {
     param_set = paradox::ps(
       n_degree = paradox::p_int(lower = 1L, tags = c("train", "required"))
     )
     param_set$set_values(
```

```
        n_degree = 1L
      )
      fn = mlr3misc::crate(function(x, n_degree) {
        torch::torch_cat(
          lapply(seq_len(n_degree), function(d) torch::torch_pow(x, d)),
          dim = 2L
        )
      })

      super$initialize(
        fn = fn,
        id = id,
        packages = character(0),
        param_vals = param_vals,
        param_set = param_set,
        stages_init = "both"
      )
    }
  ),
  private = list(
    .shapes_out = function(shapes_in, param_vals, task) {
      # shapes_in is a list of length 1 containing the shapes
      checkmate::assert_true(length(shapes_in[[1L]]) == 2L)
      if (shapes_in[[1L]][2L] != 1L) {
        stop("Input shape must be (NA, 1)")
      }
      list(c(NA, param_vals$n_degree))
    }
  )
)

po_poly = PipeOpPreprocTorchPoly$new(
  param_vals = list(n_degree = 3L, affect_columns = selector_name("x3"))
)

po_poly$shapes_out(list(c(NA, 1L)), stage = "train")

taskout = po_poly$train(list(taskin))[[1L]]
materialize(taskout$data(cols = "x3"), rbind = TRUE)
```

`mlr_pipeops_torch` *Base Class for Torch Module Constructor Wrappers*

Description

`PipeOpTorch` is the base class for all `PipeOp`s that represent neural network layers in a `Graph`. During **training**, it generates a `PipeOpModule` that wraps an `nn_module` and attaches it to the architecture, which is also represented as a `Graph` consisting mostly of `PipeOpModule`s an `PipeOpNOP`s.

While the former `Graph` operates on `ModelDescriptor`s, the latter operates on tensors.

The relationship between a PipeOpTorch and a PipeOpModule is similar to the relationshop between a nn_module_generator (like nn_linear) and a nn_module (like the output of nn_linear(...)). A crucial difference is that the PipeOpTorch infers auxiliary parameters (like in_features for nn_linear) automatically from the intermediate tensor shapes that are being communicated through the ModelDescriptor.

During **prediction**, PipeOpTorch takes in a Task in each channel and outputs the same new Task resulting from their feature union in each channel. If there is only one input and output channel, the task is simply piped through.

Parameters

The ParamSet is specified by the child class inheriting from PipeOpTorch. Usually the parameters are the arguments of the wrapped nn_module minus the auxiliary parameter that can be automatically inferred from the shapes of the input tensors.

Inheriting

When inheriting from this class, one should overload either the private$.shapes_out() and the private$.shape_dependent_params() methods, or overload private$.make_module().

- .make_module(shapes_in, param_vals, task)
  (list(), list()) -> nn_module
  This private method is called to generated the nn_module that is passed as argument module to PipeOpModule. It must be overwritten, when no module_generator is provided. If left as is, it calls the provided module_generator with the arguments obtained by the private method .shape_dependent_params().
- .shapes_out(shapes_in, param_vals, task)
  (list(), list(), Task or NULL) -> named list()
  This private method gets a list of numeric vectors (shapes_in), the parameter values (param_vals), as well as an (optional) Task. The shapes_in can be assumed to be in the same order as the input names of the PipeOp. The output shapes must be in the same order as the output names of the PipeOp. In case the output shapes depends on the task (as is the case for PipeOpTorchHead), the function should return valid output shapes (possibly containing NAs) if the task argument is provided or not.
- .shape_dependent_params(shapes_in, param_vals, task)
  (list(), list()) -> named list()
  This private method has the same inputs as .shapes_out. If .make_module() is not overwritten, it constructs the arguments passed to module_generator. Usually this means that it must infer the auxiliary parameters that can be inferred from the input shapes and add it to the user-supplied parameter values (param_vals).

Input and Output Channels

During *training*, all inputs and outputs are of class ModelDescriptor. During *prediction*, all input and output channels are of class Task.

State

The state is the value calculated by the public method shapes_out().

Internals

During training, the PipeOpTorch creates a PipeOpModule for the given parameter specification and the input shapes from the incoming ModelDescriptors using the private method .make_module(). The input shapes are provided by the slot pointer_shape of the incoming ModelDescriptors. The channel names of this PipeOpModule are identical to the channel names of the generating PipeOpTorch.

A model descriptor union of all incoming ModelDescriptors is then created. Note that this modifies the graph of the first ModelDescriptor **in place** for efficiency. The PipeOpModule is added to the graph slot of this union and the the edges that connect the sending PipeOpModules to the input channel of this PipeOpModule are addeded to the graph. This is possible because every incoming ModelDescriptor contains the information about the id and the channel name of the sending PipeOp in the slot pointer.

The new graph in the model_descriptor_union represents the current state of the neural network architecture. It is structurally similar to the subgraph that consists of all pipeops of class PipeOpTorch and PipeOpTorchIngress that are ancestors of this PipeOpTorch.

For the output, a shallow copy of the ModelDescriptor is created and the pointer and pointer_shape are updated accordingly. The shallow copy means that all ModelDescriptors point to the same Graph which allows the graph to be modified by-reference in different parts of the code.

Super class

mlr3pipelines::PipeOp -> PipeOpTorch

Public fields

module_generator (nn_module_generator or NULL)
: The module generator wrapped by this PipeOpTorch. If NULL, the private method private$.make_module(shapes_in param_vals) must be overwritte, see section 'Inheriting'. Do not change this after construction.

Methods

Public methods:

- PipeOpTorch$new()
- PipeOpTorch$shapes_out()
- PipeOpTorch$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorch$new(
  id,
  module_generator,
  param_set = ps(),
  param_vals = list(),
  inname = "input",
  outname = "output",
  packages = "torch",
```

```
  tags = NULL
)
```

Arguments:

`id` (`character(1)`)
: Identifier of the resulting object.

`module_generator` (`nn_module_generator`)
: The torch module generator.

`param_set` (`ParamSet`)
: The parameter set.

`param_vals` (`list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

`inname` (`character()`)
: The names of the `PipeOp`'s input channels. These will be the input channels of the generated `PipeOpModule`. Unless the wrapped `module_generator`'s forward method (if present) has the argument `...`, `inname` must be identical to those argument names in order to avoid any ambiguity.
If the forward method has the argument `...`, the order of the input channels determines how the tensors will be passed to the wrapped `nn_module`.
If left as `NULL` (default), the argument `module_generator` must be given and the argument names of the `modue_generator`'s forward function are set as `inname`.

`outname` (`character()`)
: The names of the output channels channels. These will be the ouput channels of the generated `PipeOpModule` and therefore also the names of the list returned by its `$train()`. In case there is more than one output channel, the `nn_module` that is constructed by this `PipeOp` during training must return a named `list()`, where the names of the list are the names out the output channels. The default is `"output"`.

`packages` (`character()`)
: The R packages this object depends on.

`tags` (`character()`)
: The tags of the `PipeOp`. The tags `"torch"` is always added.

Method `shapes_out()`**:** Calculates the output shapes for the given input shapes, parameters and task.

Usage:

```
PipeOpTorch$shapes_out(shapes_in, task = NULL)
```

Arguments:

`shapes_in` (`list()` of `integer()`)
: The input input shapes, which must be in the same order as the input channel names of the `PipeOp`.

`task` (`Task` or `NULL`)
: The task, which is very rarely used (default is `NULL`). An exception is `PipeOpTorchHead`.

Returns: A named `list()` containing the output shapes. The names are the names of the output channels of the `PipeOp`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorch$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingre mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union(), nn_graph()

Examples

```
## Creating a neural network
# In torch

task = tsk("iris")

network_generator = torch::nn_module(
  initialize = function(task, d_hidden) {
    d_in = length(task$feature_names)
    self$linear = torch::nn_linear(d_in, d_hidden)
    self$output = if (task$task_type == "regr") {
      torch::nn_linear(d_hidden, 1)
    } else if (task$task_type == "classif") {
      torch::nn_linear(d_hidden, length(task$class_names))
    }
  },
  forward = function(x) {
    x = self$linear(x)
    x = torch::nnf_relu(x)
    self$output(x)
  }
)

network = network_generator(task, d_hidden = 50)
x = torch::torch_tensor(as.matrix(task$data(1, task$feature_names)))
y = torch::with_no_grad(network(x))


# In mlr3torch
network_generator = po("torch_ingress_num") %>>%
  po("nn_linear", out_features = 50) %>>%
  po("nn_head")
md = network_generator$train(task)[[1L]]
network = model_descriptor_to_module(md)
y = torch::with_no_grad(network(torch_ingress_num.input = x))
```

```
## Implementing a custom PipeOpTorch

# defining a custom module
nn_custom = nn_module("nn_custom",
  initialize = function(d_in1, d_in2, d_out1, d_out2, bias = TRUE) {
    self$linear1 = nn_linear(d_in1, d_out1, bias)
    self$linear2 = nn_linear(d_in2, d_out2, bias)
  },
  forward = function(input1, input2) {
    output1 = self$linear1(input1)
    output2 = self$linear1(input2)

    list(output1 = output1, output2 = output2)
  }
)

# wrapping the module into a custom PipeOpTorch

library(paradox)

PipeOpTorchCustom = R6::R6Class("PipeOpTorchCustom",
  inherit = PipeOpTorch,
  public = list(
    initialize = function(id = "nn_custom", param_vals = list()) {
      param_set = ps(
        d_out1 = p_int(lower = 1, tags = c("required", "train")),
        d_out2 = p_int(lower = 1, tags = c("required", "train")),
        bias = p_lgl(default = TRUE, tags = "train")
      )
      super$initialize(
        id = id,
        param_vals = param_vals,
        param_set = param_set,
        inname = c("input1", "input2"),
        outname = c("output1", "output2"),
        module_generator = nn_custom
      )
    }
  ),
  private = list(
    .shape_dependent_params = function(shapes_in, param_vals, task) {
      c(param_vals,
      list(d_in1 = tail(shapes_in[["input1"]], 1)), d_in2 = tail(shapes_in[["input2"]], 1)
      )
    },
    .shapes_out = function(shapes_in, param_vals, task) {
      list(
        input1 = c(head(shapes_in[["input1"]], -1), param_vals$d_out1),
        input2 = c(head(shapes_in[["input2"]], -1), param_vals$d_out2)
      )
    }
  )
)
```

```
## Training

# generate input
task = tsk("iris")
task1 = task$clone()$select(paste0("Sepal.", c("Length", "Width")))
task2 = task$clone()$select(paste0("Petal.", c("Length", "Width")))
graph = gunion(list(po("torch_ingress_num_1"), po("torch_ingress_num_2")))
mds_in = graph$train(list(task1, task2), single_input = FALSE)

mds_in[[1L]][c("graph", "task", "ingress", "pointer", "pointer_shape")]
mds_in[[2L]][c("graph", "task", "ingress", "pointer", "pointer_shape")]

# creating the PipeOpTorch and training it
po_torch = PipeOpTorchCustom$new()
po_torch$param_set$values = list(d_out1 = 10, d_out2 = 20)
train_input = list(input1 = mds_in[[1L]], input2 = mds_in[[2L]])
mds_out = do.call(po_torch$train, args = list(input = train_input))
po_torch$state

# the new model descriptors

# the resulting graphs are identical
identical(mds_out[[1L]]$graph, mds_out[[2L]]$graph)
# not that as a side-effect, also one of the input graphs is modified in-place for efficiency
mds_in[[1L]]$graph$edges

# The new task has both Sepal and Petal features
identical(mds_out[[1L]]$task, mds_out[[2L]]$task)
mds_out[[2L]]$task

# The new ingress slot contains all ingressors
identical(mds_out[[1L]]$ingress, mds_out[[2L]]$ingress)
mds_out[[1L]]$ingress

# The pointer and pointer_shape slots are different
mds_out[[1L]]$pointer
mds_out[[2L]]$pointer

mds_out[[1L]]$pointer_shape
mds_out[[2L]]$pointer_shape

## Prediction
predict_input = list(input1 = task1, input2 = task2)
tasks_out = do.call(po_torch$predict, args = list(input = predict_input))
identical(tasks_out[[1L]], tasks_out[[2L]])
```

`mlr_pipeops_torch_callbacks`

Callback Configuration

Description

Configures the callbacks of a deep learning model.

Parameters

The parameters are defined dynamically from the callbacks, where the id of the respective callbacks is the respective set id.

Input and Output Channels

There is one input channel `"input"` and one output channel `"output"`. During *training*, the channels are of class `ModelDescriptor`. During *prediction*, the channels are of class `Task`.

State

The state is the value calculated by the public method `shapes_out()`.

Internals

During training the callbacks are cloned and added to the `ModelDescriptor`.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpTorchCallbacks`

Methods

Public methods:

- `PipeOpTorchCallbacks$new()`
- `PipeOpTorchCallbacks$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchCallbacks$new(
  callbacks = list(),
  id = "torch_callbacks",
  param_vals = list()
)
```

Arguments:

`callbacks` (`list` of `TorchCallbacks`)
: The callbacks (or something convertible via `as_torch_callbacks()`). Must have unique ids. All callbacks are cloned during construction.

`id` (`character(1)`)
: Identifier of the resulting object.

`param_vals` (`list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchCallbacks$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Model Configuration: `ModelDescriptor()`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_optimizer`, `model_descriptor_union()`

Other PipeOp: `mlr_pipeops_module`, `mlr_pipeops_torch_optimizer`

Examples

```
po_cb = po("torch_callbacks", "checkpoint")
po_cb$param_set
mdin = po("torch_ingress_num")$train(list(tsk("iris")))
mdin[[1L]]$callbacks
mdout = po_cb$train(mdin)[[1L]]
mdout$callbacks
# Can be called again
po_cb1 = po("torch_callbacks", t_clbk("progress"))
mdout1 = po_cb1$train(list(mdout))[[1L]]
mdout1$callbacks
```

---

mlr_pipeops_torch_ingress

Entrypoint to Torch Network

Description

Use this as entry-point to mlr3torch-networks. Unless you are an advanced user, you should not need to use this directly but `PipeOpTorchIngressNumeric`, `PipeOpTorchIngressCategorical` or `PipeOpTorchIngressLazyTensor`.

Parameters

Defined by the construction argument `param_set`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is set to the input shape.

Internals

Creates an object of class TorchIngressToken for the given task. The purpuse of this is to store the information on how to construct the torch dataloader from the task for this entry point of the network.

Super class

mlr3pipelines::PipeOp -> PipeOpTorchIngress

Active bindings

feature_types (character(1))
The features types that can be consumed by this PipeOpTorchIngress.

Methods

Public methods:

- PipeOpTorchIngress$new()
- PipeOpTorchIngress$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchIngress$new(
  id,
  param_set = ps(),
  param_vals = list(),
  packages = character(0),
  feature_types
)
```

Arguments:

id (character(1))
Identifier of the resulting object.

param_set (ParamSet)
The parameter set.

param_vals (list())
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

packages (character())
The R packages this object depends on.

feature_types (character())
The feature types. See mlr_reflections$task_feature_types for available values, Additionally, "lazy_tensor" is supported.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchIngress$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2 mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model mlr_pipeops_torch_model_regr

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union(), nn_graph()

---

mlr_pipeops_torch_ingress_categ

Torch Entry Point for Categorical Features

Description

Ingress PipeOp that represents a categorical (`factor()`, `ordered()` and `logical()`) entry point to a torch network.

Parameters

- `select` :: `logical(1)`
  Whether `PipeOp` should selected the supported feature types. Otherwise it will err on receiving tasks with unsupported feature types.

Internals

Uses `batchgetter_categ()`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is set to the input shape.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorchIngress` -> `PipeOpTorchIngressCategorical`

Methods

Public methods:

- `PipeOpTorchIngressCategorical$new()`
- `PipeOpTorchIngressCategorical$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchIngressCategorical$new(
  id = "torch_ingress_categ",
  param_vals = list()
)
```

Arguments:

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchIngressCategorical$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tra`
`mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2`
`mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`,

mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union(), nn_graph()

Examples

```
graph = po("select", selector = selector_type("factor")) %>>%
  po("torch_ingress_categ")
task = tsk("german_credit")
# The output is a model descriptor
md = graph$train(task)[[1L]]
ingress = md$ingress[[1L]]
ingress$batchgetter(task$data(1, ingress$features), "cpu")
```

---

mlr_pipeops_torch_ingress_ltnsr

Ingress for Lazy Tensor

Description

Ingress for a single lazy_tensor column.

Parameters

- shape :: integer() | NULL | "infer"
  The shape of the tensor, where the first dimension (batch) must be NA. When it is not specified, the lazy tensor input column needs to have a known shape. When it is set to "infer", the shape is inferred from an example batch.

Internals

The returned batchgetter materializes the lazy tensor column to a tensor.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is set to the input shape.

Super classes

mlr3pipelines::PipeOp -> mlr3torch::PipeOpTorchIngress -> PipeOpTorchIngressLazyTensor

Methods

Public methods:

- PipeOpTorchIngressLazyTensor$new()
- PipeOpTorchIngressLazyTensor$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchIngressLazyTensor$new(
  id = "torch_ingress_ltnsr",
  param_vals = list()
)
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchIngressLazyTensor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2
mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ,

mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model
mlr_pipeops_torch_model_regr

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union(), nn_graph()

Examples

```
po_ingress = po("torch_ingress_ltnsr")
task = tsk("lazy_iris")

md = po_ingress$train(list(task))[[1L]]

ingress = md$ingress
x_batch = ingress[[1L]]$batchgetter(data = task$data(1, "x"), cache = NULL)
x_batch

# Now we try a lazy tensor with unknown shape, i.e. the shapes between the rows can differ

ds = dataset(
  initialize = function() self$x = list(torch_randn(3, 10, 10), torch_randn(3, 8, 8)),
  .getitem = function(i) list(x = self$x[[i]]),
  .length = function() 2)()

task_unknown = as_task_regr(data.table(
  x = as_lazy_tensor(ds, dataset_shapes = list(x = NULL)),
  y = rnorm(2)
), target = "y", id = "example2")

# this task (as it is) can NOT be processed by PipeOpTorchIngressLazyTensor
# It therefore needs to be preprocessed
po_resize = po("trafo_resize", size = c(6, 6))
task_unknown_resize = po_resize$train(list(task_unknown))[[1L]]

# printing the transformed column still shows unknown shapes,
# because the preprocessing pipeop cannot infer them,
# however we know that the shape is now (3, 10, 10) for all rows
task_unknown_resize$data(1:2, "x")
po_ingress$param_set$set_values(shape = c(NA, 3, 6, 6))

md2 = po_ingress$train(list(task_unknown_resize))[[1L]]

ingress2 = md2$ingress
x_batch2 = ingress2[[1L]]$batchgetter(
  data = task_unknown_resize$data(1:2, "x"),
  cache = NULL
)

x_batch2
```

`mlr_pipeops_torch_ingress_num` *Torch Entry Point for Numeric Features*

Description

Ingress PipeOp that represents a numeric (`integer()` and `numeric()`) entry point to a torch network.

Internals

Uses `batchgetter_num()`.

Input and Output Channels

One input channel called `"input"` and one output channel called `"output"`. For an explanation see `PipeOpTorch`.

State

The state is set to the input shape.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3torch::PipeOpTorchIngress` -> `PipeOpTorchIngressNumeric`

Methods

Public methods:

- `PipeOpTorchIngressNumeric$new()`
- `PipeOpTorchIngressNumeric$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchIngressNumeric$new(id = "torch_ingress_num", param_vals = list())
```

Arguments:

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchIngressNumeric$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union(), nn_graph()

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2
mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Examples

```
graph = po("select", selector = selector_type(c("numeric", "integer"))) %>>%
  po("torch_ingress_num")
task = tsk("german_credit")
# The output is a model descriptor
md = graph$train(task)[[1L]]
ingress = md$ingress[[1L]]
ingress$batchgetter(task$data(1:5, ingress$features), "cpu")
```

---

mlr_pipeops_torch_loss

Loss Configuration

Description

Configures the loss of a deep learning model.

Input and Output Channels

One input channel called "input" and one output channel called "output". For an explanation see PipeOpTorch.

State

The state is the value calculated by the public method `shapes_out()`.

Parameters

The parameters are defined dynamically from the loss set during construction.

Internals

During training the loss is cloned and added to the `ModelDescriptor`.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpTorchLoss`

Methods

Public methods:

- `PipeOpTorchLoss$new()`
- `PipeOpTorchLoss$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchLoss$new(loss, id = "torch_loss", param_vals = list())
```

Arguments:

`loss` (`TorchLoss` or `character(1)` or `nn_loss`)
The loss (or something convertible via `as_torch_loss()`).

`id` (`character(1)`)
Identifier of the resulting object.

`param_vals` (`list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchLoss$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`

mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2d, mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_model, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

Other Model Configuration: ModelDescriptor(), mlr_pipeops_torch_callbacks, mlr_pipeops_torch_optimizer, model_descriptor_union()

Examples

```
po_loss = po("torch_loss", loss = t_loss("cross_entropy"))
po_loss$param_set
mdin = po("torch_ingress_num")$train(list(tsk("iris")))
mdin[[1L]]$loss
mdout = po_loss$train(mdin)[[1L]]
mdout$loss
```

---

mlr_pipeops_torch_model

PipeOp Torch Model

Description

Builds a Torch Learner from a ModelDescriptor and trains it with the given parameter specification. The task type must be specified during construction.

Parameters

General:

The parameters of the optimizer, loss and callbacks, prefixed with "opt.", "loss." and "cb.<callback id>." respectively, as well as:

- epochs :: integer(1)
  The number of epochs.
- device :: character(1)
  The device. One of "auto", "cpu", or "cuda" or other values defined in mlr_reflections$torch$devices. The value is initialized to "auto", which will select "cuda" if possible, then try "mps" and otherwise fall back to "cpu".

- num_threads :: integer(1)
  The number of threads for intraop pararallelization (if device is "cpu"). This value is initialized to 1.
- num_interop_threads :: integer(1)
  The number of threads for intraop and interop pararallelization (if device is "cpu"). This value is initialized to 1. Note that this can only be set once during a session and changing the value within an R session will raise a warning.
- seed :: integer(1) or "random" or NULL
  The torch seed that is used during training and prediction. This value is initialized to "random", which means that a random seed will be sampled at the beginning of the training phase. This seed (either set or randomly sampled) is available via $model$seed after training and used during prediction. Note that by setting the seed during the training phase this will mean that by default (i.e. when seed is "random"), clones of the learner will use a different seed. If set to NULL, no seeding will be done.
- tensor_dataset :: logical(1) | "device"
  Whether to load all batches at once at the beginning of training and stack them. This is initialized to FALSE. If set to "device", the device of the tensors will be set to the value of device, which can avoid unnecessary moving of tensors between devices. When your dataset fits into memory this will make the loading of batches faster. Note that this should not be set for datasets that contain lazy_tensors with random data augmentation, as this augmentation will only be applied once at the beginning of training.

Evaluation:

- measures_train :: Measure or list() of Measures
  Measures to be evaluated during training.
- measures_valid :: Measure or list() of Measures
  Measures to be evaluated during validation.
- eval_freq :: integer(1)
  How often the train / validation predictions are evaluated using measures_train / measures_valid. This is initialized to 1. Note that the final model is always evaluated.

Early Stopping:

- patience :: integer(1)
  This activates early stopping using the validation scores. If the performance of a model does not improve for patience evaluation steps, training is ended. Note that the final model is stored in the learner, not the best model. This is initialized to 0, which means no early stopping. The first entry from measures_valid is used as the metric. This also requires to specify the $validate field of the Learner, as well as measures_valid. If this is set, the epoch after which no improvement was observed, can be accessed via the $internal_tuned_values field of the learner.
- min_delta :: double(1)
  The minimum improvement threshold for early stopping. Is initialized to 0.

Dataloader:

- batch_size :: integer(1)
  The batch size (required).

- shuffle :: logical(1)
  Whether to shuffle the instances in the dataset. This is initialized to TRUE, which differs from the default (FALSE).
- sampler :: torch::sampler
  Object that defines how the dataloader draw samples.
- batch_sampler :: torch::sampler
  Object that defines how the dataloader draws batches.
- num_workers :: integer(1)
  The number of workers for data loading (batches are loaded in parallel). The default is 0, which means that data will be loaded in the main process.
- collate_fn :: function
  How to merge a list of samples to form a batch.
- pin_memory :: logical(1)
  Whether the dataloader copies tensors into CUDA pinned memory before returning them.
- drop_last :: logical(1)
  Whether to drop the last training batch in each epoch during training. Default is FALSE.
- timeout :: numeric(1)
  The timeout value for collecting a batch from workers. Negative values mean no timeout and the default is -1.
- worker_init_fn :: function(id)
  A function that receives the worker id (in [1, num_workers]) and is exectued after seeding on the worker but before data loading.
- worker_globals :: list() | character()
  When loading data in parallel, this allows to export globals to the workers. If this is a character vector, the objects in the global environment with those names are copied to the workers.
- worker_packages :: character()
  Which packages to load on the workers.

Also see torch::dataloder for more information.

Input and Output Channels

There is one input channel "input" that takes in ModelDescriptor during traing and a Task of the specified task_type during prediction. The output is NULL during training and a Prediction of given task_type during prediction.

State

A trained LearnerTorchModel.

Internals

A LearnerTorchModel is created by calling model_descriptor_to_learner() on the provided ModelDescriptor that is received through the input channel. Then the parameters are set according to the parameters specified in PipeOpTorchModel and its '$train() method is called on the [Task][mlr3::Task] stored

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpLearner -> PipeOpTorchModel

Methods

Public methods:

- PipeOpTorchModel$new()
- PipeOpTorchModel$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchModel$new(task_type, id = "torch_model", param_vals = list())
```

Arguments:

`task_type (character(1))`
: The task type of the model.

`id (character(1))`
: Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tra
mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2
mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model_classif, mlr_pipeops_torch_model_regr

`mlr_pipeops_torch_model_classif` *PipeOp Torch Classifier*

Description

Builds a torch classifier and trains it.

Parameters

See `LearnerTorch`

Input and Output Channels

There is one input channel `"input"` that takes in `ModelDescriptor` during traing and a `Task` of the specified `task_type` during prediction. The output is `NULL` during training and a `Prediction` of given `task_type` during prediction.

State

A trained `LearnerTorchModel`.

Internals

A `LearnerTorchModel` is created by calling `model_descriptor_to_learner()` on the provided `ModelDescriptor` that is received through the input channel. Then the parameters are set according to the parameters specified in `PipeOpTorchModel` and its '$train() `method is called on the [Task][mlr3::Task] stored`

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpLearner` -> `mlr3torch::PipeOpTorchModel` -> `PipeOpTorchModelClassif`

Methods

Public methods:

- `PipeOpTorchModelClassif$new()`
- `PipeOpTorchModelClassif$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchModelClassif$new(id = "torch_model_classif", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchModelClassif$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: mlr_pipeops_nn_adaptive_avg_pool1d, mlr_pipeops_nn_adaptive_avg_pool2d, mlr_pipeops_nn_adaptive_avg_pool3d, mlr_pipeops_nn_avg_pool1d, mlr_pipeops_nn_avg_pool2d, mlr_pipeops_nn_avg_pool3d, mlr_pipeops_nn_batch_norm1d, mlr_pipeops_nn_batch_norm2d, mlr_pipeops_nn_batch_norm3d, mlr_pipeops_nn_block, mlr_pipeops_nn_celu, mlr_pipeops_nn_conv1d, mlr_pipeops_nn_conv2d, mlr_pipeops_nn_conv3d, mlr_pipeops_nn_conv_transpose1d, mlr_pipeops_nn_conv_tran mlr_pipeops_nn_conv_transpose3d, mlr_pipeops_nn_dropout, mlr_pipeops_nn_elu, mlr_pipeops_nn_flatten, mlr_pipeops_nn_gelu, mlr_pipeops_nn_glu, mlr_pipeops_nn_hardshrink, mlr_pipeops_nn_hardsigmoid, mlr_pipeops_nn_hardtanh, mlr_pipeops_nn_head, mlr_pipeops_nn_layer_norm, mlr_pipeops_nn_leaky_relu, mlr_pipeops_nn_linear, mlr_pipeops_nn_log_sigmoid, mlr_pipeops_nn_max_pool1d, mlr_pipeops_nn_max_pool2 mlr_pipeops_nn_max_pool3d, mlr_pipeops_nn_merge, mlr_pipeops_nn_merge_cat, mlr_pipeops_nn_merge_prod, mlr_pipeops_nn_merge_sum, mlr_pipeops_nn_prelu, mlr_pipeops_nn_relu, mlr_pipeops_nn_relu6, mlr_pipeops_nn_reshape, mlr_pipeops_nn_rrelu, mlr_pipeops_nn_selu, mlr_pipeops_nn_sigmoid, mlr_pipeops_nn_softmax, mlr_pipeops_nn_softplus, mlr_pipeops_nn_softshrink, mlr_pipeops_nn_softsign, mlr_pipeops_nn_squeeze, mlr_pipeops_nn_tanh, mlr_pipeops_nn_tanhshrink, mlr_pipeops_nn_threshold, mlr_pipeops_nn_unsqueeze, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, mlr_pipeops_torch_loss, mlr_pipeops_torch_model, mlr_pipeops_torch_model_regr

Examples

```
# simple logistic regression

# configure the model descriptor
md = as_graph(po("torch_ingress_num") %>>%
  po("nn_head") %>>%
  po("torch_loss", "cross_entropy") %>>%
  po("torch_optimizer", "adam"))$train(tsk("iris"))[[1L]]

print(md)

# build the learner from the model descriptor and train it
po_model = po("torch_model_classif", batch_size = 50, epochs = 1)
po_model$train(list(md))
po_model$state
```

mlr_pipeops_torch_model_regr *Torch Regression Model*

Description

Builds a torch regression model and trains it.

Parameters

See LearnerTorch

Input and Output Channels

There is one input channel "input" that takes in ModelDescriptor during traing and a Task of the specified task_type during prediction. The output is NULL during training and a Prediction of given task_type during prediction.

State

A trained LearnerTorchModel.

Internals

A LearnerTorchModel is created by calling model_descriptor_to_learner() on the provided ModelDescriptor that is received through the input channel. Then the parameters are set according to the parameters specified in PipeOpTorchModel and its '$train() method is called on the [Task][mlr3::Task] stored

Super classes

mlr3pipelines::PipeOp -> mlr3pipelines::PipeOpLearner -> mlr3torch::PipeOpTorchModel -> PipeOpTorchModelRegr

Methods

Public methods:

- PipeOpTorchModelRegr$new()
- PipeOpTorchModelRegr$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchModelRegr$new(id = "torch_model_regr", param_vals = list())
```

Arguments:

id (character(1))
Identifier of the resulting object.

`param_vals (list())`
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchModelRegr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOps: `mlr_pipeops_nn_adaptive_avg_pool1d`, `mlr_pipeops_nn_adaptive_avg_pool2d`, `mlr_pipeops_nn_adaptive_avg_pool3d`, `mlr_pipeops_nn_avg_pool1d`, `mlr_pipeops_nn_avg_pool2d`, `mlr_pipeops_nn_avg_pool3d`, `mlr_pipeops_nn_batch_norm1d`, `mlr_pipeops_nn_batch_norm2d`, `mlr_pipeops_nn_batch_norm3d`, `mlr_pipeops_nn_block`, `mlr_pipeops_nn_celu`, `mlr_pipeops_nn_conv1d`, `mlr_pipeops_nn_conv2d`, `mlr_pipeops_nn_conv3d`, `mlr_pipeops_nn_conv_transpose1d`, `mlr_pipeops_nn_conv_tran`, `mlr_pipeops_nn_conv_transpose3d`, `mlr_pipeops_nn_dropout`, `mlr_pipeops_nn_elu`, `mlr_pipeops_nn_flatten`, `mlr_pipeops_nn_gelu`, `mlr_pipeops_nn_glu`, `mlr_pipeops_nn_hardshrink`, `mlr_pipeops_nn_hardsigmoid`, `mlr_pipeops_nn_hardtanh`, `mlr_pipeops_nn_head`, `mlr_pipeops_nn_layer_norm`, `mlr_pipeops_nn_leaky_relu`, `mlr_pipeops_nn_linear`, `mlr_pipeops_nn_log_sigmoid`, `mlr_pipeops_nn_max_pool1d`, `mlr_pipeops_nn_max_pool2d`, `mlr_pipeops_nn_max_pool3d`, `mlr_pipeops_nn_merge`, `mlr_pipeops_nn_merge_cat`, `mlr_pipeops_nn_merge_prod`, `mlr_pipeops_nn_merge_sum`, `mlr_pipeops_nn_prelu`, `mlr_pipeops_nn_relu`, `mlr_pipeops_nn_relu6`, `mlr_pipeops_nn_reshape`, `mlr_pipeops_nn_rrelu`, `mlr_pipeops_nn_selu`, `mlr_pipeops_nn_sigmoid`, `mlr_pipeops_nn_softmax`, `mlr_pipeops_nn_softplus`, `mlr_pipeops_nn_softshrink`, `mlr_pipeops_nn_softsign`, `mlr_pipeops_nn_squeeze`, `mlr_pipeops_nn_tanh`, `mlr_pipeops_nn_tanhshrink`, `mlr_pipeops_nn_threshold`, `mlr_pipeops_nn_unsqueeze`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_model`, `mlr_pipeops_torch_model_classif`

Examples

```
# simple linear regression

# build the model descriptor
md = as_graph(po("torch_ingress_num") %>>%
  po("nn_head") %>>%
  po("torch_loss", "mse") %>>%
  po("torch_optimizer", "adam"))$train(tsk("mtcars"))[[1L]]

print(md)

# build the learner from the model descriptor and train it
po_model = po("torch_model_regr", batch_size = 20, epochs = 1)
po_model$train(list(md))
po_model$state
```

`mlr_pipeops_torch_optimizer` *Optimizer Configuration*

Description

Configures the optimizer of a deep learning model.

Parameters

The parameters are defined dynamically from the optimizer that is set during construction.

Input and Output Channels

There is one input channel `"input"` and one output channel `"output"`. During *training*, the channels are of class `ModelDescriptor`. During *prediction*, the channels are of class `Task`.

State

The state is the value calculated by the public method `shapes_out()`.

Internals

During training, the optimizer is cloned and added to the `ModelDescriptor`. Note that the parameter set of the stored `TorchOptimizer` is reference-identical to the parameter set of the pipeop itself.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpTorchOptimizer`

Methods

Public methods:

- `PipeOpTorchOptimizer$new()`
- `PipeOpTorchOptimizer$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
PipeOpTorchOptimizer$new(
  optimizer = t_opt("adam"),
  id = "torch_optimizer",
  param_vals = list()
)
```

Arguments:

`optimizer` (`TorchOptimizer` or `character(1)` or `torch_optimizer_generator`)
The optimizer (or something convertible via `as_torch_optimizer()`).

`id (character(1))`
Identifier of the resulting object.

`param_vals (list())`
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpTorchOptimizer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other PipeOp: `mlr_pipeops_module`, `mlr_pipeops_torch_callbacks`

Other Model Configuration: `ModelDescriptor()`, `mlr_pipeops_torch_callbacks`, `mlr_pipeops_torch_loss`, `model_descriptor_union()`

Examples

```
po_opt = po("torch_optimizer", "sgd", lr = 0.01)
po_opt$param_set
mdin = po("torch_ingress_num")$train(list(tsk("iris")))
mdin[[1L]]$optimizer
mdout = po_opt$train(mdin)
mdout[[1L]]$optimizer
```

---

mlr_pipeops_trafo_adjust_brightness

Adjust Brightness Transformation

Description

Calls `torchvision::transform_adjust_brightness`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
brightness_factor	numeric	-		$[0, \infty)$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

`mlr_pipeops_trafo_adjust_gamma`
Adjust Gamma Transformation

Description

Calls `torchvision::transform_adjust_gamma`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
gamma	numeric	-		$[0, \infty)$
gain	numeric	1		$(-\infty, \infty)$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

`mlr_pipeops_trafo_adjust_hue`
Adjust Hue Transformation

Description

Calls `torchvision::transform_adjust_hue`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
hue_factor	numeric	-		$[-0.5, 0.5]$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

`mlr_pipeops_trafo_adjust_saturation`
Adjust Saturation Transformation

Description

Calls `torchvision::transform_adjust_saturation`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
saturation_factor	numeric	-		$(-\infty, \infty)$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

mlr_pipeops_trafo_grayscale *Grayscale Transformation*

Description

Calls `torchvision::transform_grayscale`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels	Range
num_output_channels	integer	-		$[1, 3]$
stages	character	-	train, predict, both	-
affect_columns	untyped	selector_all()		-

mlr_pipeops_trafo_nop *No Transformation*

Description

Does nothing.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

mlr_pipeops_trafo_normalize *Normalization Transformation*

Description

Calls `torchvision::transform_normalize`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
mean	untyped	-	
std	untyped	-	
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

mlr_pipeops_trafo_pad *Padding Transformation*

Description

Calls `torchvision::transform_pad`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
padding	untyped	-	
fill	untyped	0	
padding_mode	character	constant	constant, edge, reflect, symmetric
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

mlr_pipeops_trafo_reshape

Reshaping Transformation

Description

Reshapes the tensor according to the parameter shape, by calling `torch_reshape()`. This preprocessing function is applied batch-wise.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

- `shape` :: `integer()`
  The desired output shape. The first dimension is the batch dimension and should usually be `-1`.

mlr_pipeops_trafo_resize

Resizing Transformation

Description

Calls `torchvision::transform_resize`, see there for more information on the parameters. The preprocessing is applied to the whole batch.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
size	untyped	-	
interpolation	character	2	Undefined, Bartlett, Blackman, Bohman, Box, Catrom, Cosine, Cubic, Gaussian
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

mlr_pipeops_trafo_rgb_to_grayscale *RGB to Grayscale Transformation*

Description

Calls `torchvision::transform_rgb_to_grayscale`, see there for more information on the parameters. The preprocessing is applied to each element of a batch individually.

Format

`R6Class` inheriting from `PipeOpTaskPreprocTorch`.

Parameters

Id	Type	Default	Levels
stages	character	-	train, predict, both
affect_columns	untyped	selector_all()	

mlr_tasks_cifar *CIFAR Classification Tasks*

Description

The CIFAR-10 and CIFAR-100 datasets. A subset of the 80 million tiny images dataset with noisy labels was supplied to student labelers, who were asked to filter out incorrectly labeled images. The images are have datatype `torch_long()`.

CIFAR-10 contains 10 classes. CIFAR-100 contains 100 classes, which may be partitioned into 20 superclasses of 5 classes each. The CIFAR-10 and CIFAR-100 classes are mutually exclusive. See Chapter 3.1 of the technical report for more details.

The data is obtained from `torchvision::cifar10_dataset()` (or `torchvision::cifar100_dataset()`).

Format

R6::R6Class inheriting from mlr3::TaskClassif.

Construction

```
tsk("cifar10")
tsk("cifar100")
```

Download

The task's backend is a DataBackendLazy which will download the data once it is requested. Other meta-data is already available before that. You can cache these datasets by setting the mlr3torch.cache option to TRUE or to a specific path to be used as the cache directory.

Properties

- Task type: "classif"
- Properties: "multiclass"
- Has Missings: no
- Target: "class"
- Features: "image"
- Data Dimension: 60000x4

References

Krizhevsky, Alex (2009). "Learning Multiple Layers of Features from Tiny Images." *Master's thesis, Department of Computer Science, University of Toronto*.

Examples

```
task_cifar10 = tsk("cifar10")
task_cifar100 = tsk("cifar100")
print(task_cifar10)
print(task_cifar100)
```

---

mlr_tasks_lazy_iris *Iris Classification Task*

Description

A classification task for the popular datasets::iris data set. Just like the iris task, but the features are represented as one lazy tensor column.

Format

R6::R6Class inheriting from mlr3::TaskClassif.

Construction

```
tsk("lazy_iris")
```

Properties

- Task type: "classif"
- Properties: "multiclass"
- Has Missings: no
- Target: "Species"
- Features: "x"
- Data Dimension: 150x3

Source

https://en.wikipedia.org/wiki/Iris_flower_data_set

References

Anderson E (1936). "The Species Problem in Iris." *Annals of the Missouri Botanical Garden*, **23**(3), 457. doi:10.2307/2394164.

Examples

```
task = tsk("lazy_iris")
task
df = task$data()
materialize(df$x[1:6], rbind = TRUE)
```

---

mlr_tasks_melanoma *Melanoma Image classification*

Description

Classification of melanoma tumor images. The data is a preprocessed version of the 2020 SIIM-ISIC challenge where the images have been reshaped to size $(3, 128, 128)$.

By default only the training rows are active in the task, but the test data (that has no targets) is also included. Whether an observation is part of the train or test set is indicated by the column `"test"`.

There are no labels for the test rows, so by default, these observations are inactive, which means that the task uses only 32701 of the 43683 observations that are defined in the underlying data backend.

The data backend also contains a more detailed `diagnosis` of the specific type of tumor.

Columns:

- `outcome` (factor): the target variable. Whether the tumor is benign or malignant (the positive class)
- `anatom_site_general_challenge` (factor): the location of the tumor on the patient's body
- `sex` (factor): the sex of the patient
- `age_approx` (int): approximate age of the patient at the time of imaging
- `image` (lazy_tensor): The image (shape $(3, 128, 128)$) of the tumor. ee `split` (character): Whether the observation os part of the train or test set.

Construction

```
tsk("melanoma")
```

Download

The task's backend is a DataBackendLazy which will download the data once it is requested. Other meta-data is already available before that. You can cache these datasets by setting the mlr3torch.cache option to TRUE or to a specific path to be used as the cache directory.

Properties

- Task type: "classif"
- Properties: "twoclass", "groups"
- Has Missings: no
- Target: "outcome"
- Features: "sex", "anatom_site_general_challenge", "age_approx", "image"
- Data Dimension: 43683x11

Source

https://huggingface.co/datasets/carsonzhang/ISIC_2020_small

References

Rotemberg, V., Kurtansky, N., Betz-Stablein, B., Caffery, L., Chousakos, E., Codella, N., Combalia, M., Dusza, S., Guitera, P., Gutman, D., Halpern, A., Helba, B., Kittler, H., Kose, K., Langer, S., Lioprys, K., Malvehy, J., Musthaq, S., Nanda, J., Reiter, O., Shih, G., Stratigos, A., Tschandl, P., Weber, J., Soyer, P. (2021). "A patient-centric dataset of images and metadata for identifying melanomas using clinical context." *Scientific Data*, **8**, 34. doi:10.1038/s4159702100815z.

Examples

```
task = tsk("melanoma")
task
```

---

mlr_tasks_mnist *MNIST Image classification*

Description

Classic MNIST image classification.

The underlying DataBackend contains columns "label", "image", "row_id", "split", where the last column indicates whether the row belongs to the train or test set.

The first 60000 rows belong to the training set, the last 10000 rows to the test set.

Construction

```
tsk("mnist")
```

Download

The task's backend is a DataBackendLazy which will download the data once it is requested. Other meta-data is already available before that. You can cache these datasets by setting the `mlr3torch.cache` option to `TRUE` or to a specific path to be used as the cache directory.

Properties

- Task type: "classif"
- Properties: "multiclass"
- Has Missings: no
- Target: "label"
- Features: "image"
- Data Dimension: 70000x4

Source

https://torchvision.mlverse.org/reference/mnist_dataset.html

References

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, **86**(11), 2278-2324. doi:10.1109/5.726791.

Examples

```
task = tsk("mnist")
task
```

---

mlr_tasks_tiny_imagenet

Tiny ImageNet Classification Task

Description

Subset of the famous ImageNet dataset. The data is obtained from `torchvision::tiny_imagenet_dataset()`.

The underlying DataBackend contains columns `"class"`, `"image"`, `"..row_id"`, `"split"`, where the last column indicates whether the row belongs to the train, validation or test set that are provided in torchvision.

There are no labels for the test rows, so by default, these observations are inactive, which means that the task uses only 110000 of the 120000 observations that are defined in the underlying data backend.

Construction

```
tsk("tiny_imagenet")
```

Download

The task's backend is a DataBackendLazy which will download the data once it is requested. Other meta-data is already available before that. You can cache these datasets by setting the mlr3torch.cache option to TRUE or to a specific path to be used as the cache directory.

Properties

- Task type: "classif"
- Properties: "multiclass"
- Has Missings: no
- Target: "class"
- Features: "image"
- Data Dimension: 120000x4

References

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, Fei-Fei, Li (2009). "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. IEEE.

Examples

```
task = tsk("tiny_imagenet")
task
```

ModelDescriptor *Represent a Model with Meta-Info*

Description

Represents a *model*; possibly a complete model, possibly one in the process of being built up.

This model takes input tensors of shapes shapes_in and pipes them through graph. Input shapes get mapped to input channels of graph. Output shapes are named by the output channels of graph; it is also possible to represent no-ops on tensors, in which case names of input and output should be identical.

ModelDescriptor objects typically represent partial models being built up, in which case the pointer slot indicates a specific point in the graph that produces a tensor of shape pointer_shape, on which the graph should be extended. It is allowed for the graph in this structure to be modified by-reference in different parts of the code. However, these modifications may never add edges with elements of the Graph as destination. In particular, no element of graph$input may be removed

by reference, e.g. by adding an edge to the Graph that has the input channel of a PipeOp that was previously without parent as its destination.

In most cases it is better to create a specific ModelDescriptor by training a Graph consisting (mostly) of operators PipeOpTorchIngress, PipeOpTorch, PipeOpTorchLoss, PipeOpTorchOptimizer, and PipeOpTorchCallbacks.

A ModelDescriptor can be converted to a nn_graph via model_descriptor_to_module.

Usage

```
ModelDescriptor(
  graph,
  ingress,
  task,
  optimizer = NULL,
  loss = NULL,
  callbacks = NULL,
  pointer = NULL,
  pointer_shape = NULL
)
```

Arguments

graph	(Graph) Graph of PipeOpModule and PipeOpNOP operators.
ingress	(uniquely named list of TorchIngressToken) List of inputs that go into graph. Names of this must be a subset of graph$input$name.
task	(Task) (Training)-Task for which the model is being built. May be necessary for for some aspects of what loss to use etc.
optimizer	(TorchOptimizer | NULL) Additional info: what optimizer to use.
loss	(TorchLoss | NULL) Additional info: what loss to use.
callbacks	(A list of CallbackSet or NULL) Additional info: what callbacks to use.
pointer	(character(2) | NULL) Indicating an element on which a model is. Points to an output channel within graph: Element 1 is the PipeOp's id and element 2 is that PipeOp's output channel.
pointer_shape	(integer | NULL) Shape of the output indicated by pointer.

Value

(ModelDescriptor)

See Also

Other Model Configuration: mlr_pipeops_torch_callbacks, mlr_pipeops_torch_loss, mlr_pipeops_torch_optimize
model_descriptor_union()

Other Graph Network: TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module,
mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingres
mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(),
model_descriptor_union(), nn_graph()

---

model_descriptor_to_learner

Create a Torch Learner from a ModelDescriptor

Description

First a nn_graph is created using model_descriptor_to_module and then a learner is created from this module and the remaining information from the model descriptor, which must include the optimizer and loss function and optionally callbacks.

Usage

```
model_descriptor_to_learner(model_descriptor)
```

Arguments

model_descriptor	(ModelDescriptor) The model descriptor.

Value

Learner

See Also

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model,
mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ,
mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, model_descriptor_to_module(),
model_descriptor_union(), nn_graph()

model_descriptor_to_module *Create a nn_graph from ModelDescriptor*

Description

Creates the nn_graph from a ModelDescriptor. Mostly for internal use, since the ModelDescriptor is in most circumstances harder to use than just creating nn_graph directly.

Usage

```
model_descriptor_to_module(
  model_descriptor,
  output_pointers = NULL,
  list_output = FALSE
)
```

Arguments

model_descriptor	(ModelDescriptor) Model Descriptor. pointer is ignored, instead output_pointer values are used. $graph member is modified by-reference.
output_pointers	(list of character) Collection of pointers that indicate what part of the model_descriptor$graph is being used for output. Entries have the format of ModelDescriptor$pointer.
list_output	(logical(1)) Whether output should be a list of tensors. If FALSE, then length(output_pointers) must be 1.

Value

nn_graph

See Also

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_union(), nn_graph()

`model_descriptor_union` *Union of ModelDescriptors*

Description

This is a mostly internal function that is used in `PipeOpTorch`s with multiple input channels.

It creates the union of multiple `ModelDescriptor`s:

- `graph`s are combinded (if they are not identical to begin with). The first entry's `graph` is modified by reference.
- `PipeOp`s with the same ID must be identical. No new input edges may be added to `PipeOp`s.
- Drops `pointer` / `pointer_shape` entries.
- The new task is the feature union of the two incoming tasks.
- The `optimizer` and `loss` of both `ModelDescriptor`s must be identical.
- Ingress tokens and callbacks are merged, where objects with the same `"id"` must be identical.

Usage

```
model_descriptor_union(md1, md2)
```

Arguments

`md1`	(`ModelDescriptor`) The first `ModelDescriptor`.
`md2`	(`ModelDescriptor`) The second `ModelDescriptor`.

Details

The requirement that no new input edgedes may be added to `PipeOp`s is not theoretically necessary, but since we assume that ModelDescriptor is being built from beginning to end (i.e. `PipeOp`s never get new ancestors) we can make this assumption and simplify things. Otherwise we'd need to treat "..."-inputs special.)

Value

`ModelDescriptor`

See Also

Other Graph Network: `ModelDescriptor()`, `TorchIngressToken()`, `mlr_learners_torch_model`, `mlr_pipeops_module`, `mlr_pipeops_torch`, `mlr_pipeops_torch_ingress`, `mlr_pipeops_torch_ingress_categ`, `mlr_pipeops_torch_ingress_ltnsr`, `mlr_pipeops_torch_ingress_num`, `model_descriptor_to_learner()`, `model_descriptor_to_module()`, `nn_graph()`

Other Model Configuration: `ModelDescriptor()`, `mlr_pipeops_torch_callbacks`, `mlr_pipeops_torch_loss`, `mlr_pipeops_torch_optimizer`

nn *Create a Neural Network Layer*

Description

Retrieve a neural network layer from the `mlr_pipeops` dictionary.

Usage

```
nn(.key, ...)
```

Arguments

`.key`	(`character(1)`)
`...`	(any) Additional parameters, constructor arguments or fields.

Examples

```
po1 = po("nn_linear", id = "linear")
# is the same as:
po2 = nn("linear")
```

nn_graph *Graph Network*

Description

Represents a neural network using a `Graph` that usually costains mostly `PipeOpModule`s.

Usage

```
nn_graph(graph, shapes_in, output_map = graph$output$name, list_output = FALSE)
```

Arguments

`graph`	(`Graph`) The `Graph` to wrap. Is **not** cloned.
`shapes_in`	(named `integer`) Shape info of tensors that go into `graph`. Names must be `graph$input$name`, possibly in different order.
`output_map`	(`character`) Which of `graph`'s outputs to use. Must be a subset of `graph$output$name`.
`list_output`	(`logical(1)`) Whether output should be a list of tensors. If `FALSE` (default), then `length(output_map)` must be 1.

Value

nn_graph

See Also

Other Graph Network: ModelDescriptor(), TorchIngressToken(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingress_ltnsr, mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union()

Examples

```
graph = mlr3pipelines::Graph$new()
graph$add_pipeop(po("module_1", module = nn_linear(10, 20)), clone = FALSE)
graph$add_pipeop(po("module_2", module = nn_relu()), clone = FALSE)
graph$add_pipeop(po("module_3", module = nn_linear(20, 1)), clone = FALSE)
graph$add_edge("module_1", "module_2")
graph$add_edge("module_2", "module_3")

network = nn_graph(graph, shapes_in = list(module_1.input = c(NA, 10)))

x = torch_randn(16, 10)

network(module_1.input = x)
```

---

nn_merge_cat *Concatenates multiple tensors*

Description

Concatenates multiple tensors on a given dimension. No broadcasting rules are applied here, you must reshape the tensors before to have the same shape.

Usage

```
nn_merge_cat(dim = -1)
```

Arguments

dim	(integer(1)) The dimension for the concatenation.

nn_merge_prod *Product of multiple tensors*

Description

Calculates the product of all input tensors.

Usage

```
nn_merge_prod()
```

nn_merge_sum *Sum of multiple tensors*

Description

Calculates the sum of all input tensors.

Usage

```
nn_merge_sum()
```

nn_reshape *Reshape*

Description

Reshape a tensor to the given shape.

Usage

```
nn_reshape(shape)
```

Arguments

shape	(integer()) The desired output shape.

nn_squeeze *Squeeze*

Description

Squeezes a tensor by calling `torch::torch_squeeze()` with the given dimension `dim`.

Usage

```
nn_squeeze(dim)
```

Arguments

`dim`	(`integer()`) The dimension to squeeze.

nn_unsqueeze *Unsqueeze*

Description

Unsqueezes a tensor by calling `torch::torch_unsqueeze()` with the given dimension `dim`.

Usage

```
nn_unsqueeze(dim)
```

Arguments

`dim`	(`integer(1)`) The dimension to unsqueeze.

`pipeop_preproc_torch` *Create Torch Preprocessing PipeOps*

Description

Function to create objects of class `PipeOpTaskPreprocTorch` in a more convenient way. Start by reading the documentation of `PipeOpTaskPreprocTorch`.

Usage

```
pipeop_preproc_torch(
  id,
  fn,
  shapes_out = NULL,
  param_set = NULL,
  packages = character(0),
  rowwise = FALSE,
  parent_env = parent.frame(),
  stages_init = NULL,
  tags = NULL
)
```

Arguments

`id`	(`character(1)`) The id for of the new object.
`fn`	(`function`) The preprocessing function.
`shapes_out`	(`function` or `NULL` or `"infer"`) The private `.shapes_out(shapes_in, param_vals, task)` method of `PipeOpTaskPreprocTorch` (see section Inheriting). Special values are `NULL` and `infer`: If `NULL`, the output shapes are unknown. If "infer", the output shape function is inferred and calculates the output shapes as follows: For an input shape of (NA, ...) a meta-tensor of shape (1, ...) is created and the preprocessing function is applied. Afterwards the batch dimension (1) is replaced with NA and the shape is returned. If the first dimension is not `NA`, the output shape of applying the preprocessing function is returned. Method `"infer"` should be correct in most cases, but might fail in some edge cases.
`param_set`	(`ParamSet` or `NULL`) The parameter set. If this is left as `NULL` (default) the parameter set is inferred in the following way: All parameters but the first and `...` of `fn` are set as untyped parameters with tags 'train' and those that have no default value are tagged as 'required' as well. Default values are not annotated.
`packages`	(`character()`) The R packages this object depends on.

rowwise (logical(1))
Whether the preprocessing is applied row-wise.

parent_env (environment)
The parent environment for the R6 class.

stages_init (character(1))
Initial value for the stages parameter. If NULL (default), will be set to "both" in case the id starts with "trafo" and to "train" if it starts with "augment". Otherwise it must specified.

tags (character())
Tags for the pipeop

Value

An R6Class instance inheriting from PipeOpTaskPreprocTorch

Examples

```
PipeOpPreprocExample = pipeop_preproc_torch("preproc_example", function(x, a) x + a)
po_example = PipeOpPreprocExample$new()
po_example$param_set
```

Select *Selector Functions for Character Vectors*

Description

A Select function subsets a character vector. They are used by the callback CallbackSetUnfreeze to select parameters to freeze or unfreeze during training.

Usage

```
select_all()

select_none()

select_grep(pattern, ignore.case = FALSE, perl = FALSE, fixed = FALSE)

select_name(param_names, assert_present = TRUE)

select_invert(select)
```

Arguments

pattern	See grep()
ignore.case	See grep()
perl	See grep()
fixed	See grep()
param_names	The names of the parameters that you want to select
assert_present	Whether to check that param_names is a subset of the full vector of names
select	A Select

Functions

- select_all(): select_all selects all elements
- select_none(): select_none selects no elements
- select_grep(): select_grep selects elements with names matching a regular expression
- select_name(): select_name selects elements with names matching the given names
- select_invert(): select_invert selects the elements NOT selected by the given selector

Examples

```
select_all()(c("a", "b"))
select_none()(c("a", "b"))
select_grep("b$")(c("ab", "ac"))
select_name("a")(c("a", "b"))
select_invert(select_all())(c("a", "b"))
```

---

task_dataset *Create a Dataset from a Task*

Description

Creates a torch dataset from an mlr3 Task. The resulting dataset's $.get_batch() method returns a list with elements x, y and index:

- x is a list with tensors, whose content is defined by the parameter feature_ingress_tokens.
- y is the target variable and its content is defined by the parameter target_batchgetter.
- .index is the index of the batch in the task's data.

The data is returned on the device specified by the parameter device.

Usage

```
task_dataset(task, feature_ingress_tokens, target_batchgetter = NULL)
```

Arguments

task (Task)
The task for which to build the dataset.

feature_ingress_tokens
(named list() of TorchIngressToken)
Each ingress token defines one item in the $x value of a batch with corresponding names.

target_batchgetter
(function(data, device))
A function taking in arguments data, which is a data.table containing only the target variable, and device. It must return the target as a torch tensor on the selected device.

Value

torch::dataset

Examples

```
task = tsk("iris")
sepal_ingress = TorchIngressToken(
  features = c("Sepal.Length", "Sepal.Width"),
  batchgetter = batchgetter_num,
  shape = c(NA, 2)
)
petal_ingress = TorchIngressToken(
  features = c("Petal.Length", "Petal.Width"),
  batchgetter = batchgetter_num,
  shape = c(NA, 2)
)
ingress_tokens = list(sepal = sepal_ingress, petal = petal_ingress)

target_batchgetter = function(data) {
  torch_tensor(data = data[[1L]], dtype = torch_float32())$unsqueeze(2)
}
dataset = task_dataset(task, ingress_tokens, target_batchgetter)
batch = dataset$.getbatch(1:10)
batch
```

TorchCallback *Torch Callback*

Description

This wraps a CallbackSet and annotates it with metadata, most importantly a ParamSet. The callback is created for the given parameter values by calling the $generate() method.

This class is usually used to configure the callback of a torch learner, e.g. when constructing a learner of in a ModelDescriptor.

For a list of available callbacks, see mlr3torch_callbacks. To conveniently retrieve a TorchCallback, use t_clbk().

Parameters

Defined by the constructor argument param_set. If no parameter set is provided during construction, the parameter set is constructed by creating a parameter for each argument of the wrapped loss function, where the parametes are then of type ParamUty.

Super class

mlr3torch::TorchDescriptor -> TorchCallback

Methods

Public methods:

- TorchCallback$new()
- TorchCallback$clone()

Method new(): Creates a new instance of this R6 class.

Usage:

```
TorchCallback$new(
  callback_generator,
  param_set = NULL,
  id = NULL,
  label = NULL,
  packages = NULL,
  man = NULL,
  additional_args = NULL
)
```

Arguments:

callback_generator (R6ClassGenerator)
: The class generator for the callback that is being wrapped.

param_set (ParamSet or NULL)
: The parameter set. If NULL (default) it is inferred from callback_generator.

id (character(1))
: The id for of the new object.

label (character(1))
: Label for the new instance.

packages (character())
: The R packages this object depends on.

man (character(1))
: String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

`additional_args` (any)
Additional arguments if necessary. For learning rate schedulers, this is the torch::LRScheduler.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TorchCallback$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other Callback: `as_torch_callback()`, `as_torch_callbacks()`, `callback_set()`, `mlr3torch_callbacks`, `mlr_callback_set`, `mlr_callback_set.checkpoint`, `mlr_callback_set.progress`, `mlr_callback_set.tb`, `mlr_callback_set.unfreeze`, `mlr_context_torch`, `t_clbk()`, `torch_callback()`

Other Torch Descriptor: `TorchDescriptor`, `TorchLoss`, `TorchOptimizer`, `as_torch_callbacks()`, `as_torch_loss()`, `as_torch_optimizer()`, `mlr3torch_losses`, `mlr3torch_optimizers`, `t_clbk()`, `t_loss()`, `t_opt()`

Examples

```
# Create a new torch callback from an existing callback set
torch_callback = TorchCallback$new(CallbackSetCheckpoint)
# The parameters are inferred
torch_callback$param_set

# Retrieve a torch callback from the dictionary
torch_callback = t_clbk("checkpoint",
  path = tempfile(), freq = 1
)
torch_callback
torch_callback$label
torch_callback$id

# open the help page of the wrapped callback set
# torch_callback$help()

# Create the callback set
callback = torch_callback$generate()
callback
# is the same as
CallbackSetCheckpoint$new(
  path = tempfile(), freq = 1
)

# Use in a learner
learner = lrn("regr.mlp", callbacks = t_clbk("checkpoint"))
# the parameters of the callback are added to the learner's parameter set
learner$param_set
```

TorchDescriptor *Base Class for Torch Descriptors*

Description

Abstract Base Class from which TorchLoss, TorchOptimizer, and TorchCallback inherit. This class wraps a generator (R6Class Generator or the torch version of such a generator) and annotates it with metadata such as a ParamSet, a label, an ID, packages, or a manual page.

The parameters are the construction arguments of the wrapped generator and the parameter $values are passed to the generator when calling the public method $generate().

Parameters

Defined by the constructor argument param_set. All parameters are tagged with "train", but this is done automatically during initialize.

Public fields

label (character(1))
: Label for this object. Can be used in tables, plot and text output instead of the ID.

param_set (ParamSet)
: Set of hyperparameters.

packages (character(1))
: Set of required packages. These packages are loaded, but not attached.

id (character(1))
: Identifier of the object. Used in tables, plot and text output.

generator The wrapped generator that is described.

man (character(1))
: String in the format [pkg]::[topic] pointing to a manual page for this object.

Active bindings

phash (character(1))
: Hash (unique identifier) for this partial object, excluding some components which are varied systematically (e.g. the parameter values).

Methods

Public methods:

- TorchDescriptor$new()
- TorchDescriptor$print()
- TorchDescriptor$generate()
- TorchDescriptor$help()
- TorchDescriptor$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
TorchDescriptor$new(
  generator,
  id = NULL,
  param_set = NULL,
  packages = NULL,
  label = NULL,
  man = NULL,
  additional_args = NULL
)
```

Arguments:

`generator` The wrapped generator that is described.

`id (character(1))`
: The id for of the new object.

`param_set (ParamSet)`
: The parameter set.

`packages (character())`
: The R packages this object depends on.

`label (character(1))`
: Label for the new instance.

`man (character(1))`
: String in the format `[pkg]::[topic]` pointing to a manual page for this object. The referenced help package can be opened via method `$help()`.

`additional_args (list())`
: Additional arguments if necessary. For learning rate schedulers, this is the torch::LRScheduler.

Method `print()`**:** Prints the object

Usage:

```
TorchDescriptor$print(...)
```

Arguments:

`...` any

Method `generate()`**:** Calls the generator with the given parameter values.

Usage:

```
TorchDescriptor$generate()
```

Method `help()`**:** Displays the help file of the wrapped object.

Usage:

```
TorchDescriptor$help()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TorchDescriptor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Torch Descriptor: TorchCallback, TorchLoss, TorchOptimizer, as_torch_callbacks(), as_torch_loss(), as_torch_optimizer(), mlr3torch_losses, mlr3torch_optimizers, t_clbk(), t_loss(), t_opt()

TorchIngressToken *Torch Ingress Token*

Description

This function creates an S3 class of class "TorchIngressToken", which is an internal data structure. It contains the (meta-)information of how a batch is generated from a Task and fed into an entry point of the neural network. It is stored as the ingress field in a ModelDescriptor.

Usage

```
TorchIngressToken(features, batchgetter, shape)
```

Arguments

features	(character) Features on which the batchgetter will operate.
batchgetter	(function) Function with two arguments: data and device. This function is given the output of Task$data(rows = batch_indices, cols = features) and it should produce a tensor of shape shape_out.
shape	(integer) Shape that batchgetter will produce. Batch-dimension should be included as NA.

Value

TorchIngressToken object.

See Also

Other Graph Network: ModelDescriptor(), mlr_learners_torch_model, mlr_pipeops_module, mlr_pipeops_torch, mlr_pipeops_torch_ingress, mlr_pipeops_torch_ingress_categ, mlr_pipeops_torch_ingres
mlr_pipeops_torch_ingress_num, model_descriptor_to_learner(), model_descriptor_to_module(), model_descriptor_union(), nn_graph()

Examples

```
# Define a task for which we want to define an ingress token
task = tsk("iris")

# We create an ingress token for two feature Sepal.Length and Petal.Length:
# We have to specify the features, the batchgetter and the shape
features = c("Sepal.Length", "Petal.Length")
# As a batchgetter we use batchgetter_num

batch_dt = task$data(rows = 1:10, cols =features)
batch_dt
batch_tensor = batchgetter_num(batch_dt, "cpu")
batch_tensor

# The shape is unknown in the first dimension (batch dimension)

ingress_token = TorchIngressToken(
  features = features,
  batchgetter = batchgetter_num,
  shape = c(NA, 2)
)
ingress_token
```

TorchLoss *Torch Loss*

Description

This wraps a `torch::nn_loss` and annotates it with metadata, most importantly a `ParamSet`. The loss function is created for the given parameter values by calling the `$generate()` method.

This class is usually used to configure the loss function of a torch learner, e.g. when construcing a learner or in a `ModelDescriptor`.

For a list of available losses, see `mlr3torch_losses`. Items from this dictionary can be retrieved using `t_loss()`.

Parameters

Defined by the constructor argument `param_set`. If no parameter set is provided during construction, the parameter set is constructed by creating a parameter for each argument of the wrapped loss function, where the parametes are then of type `ParamUty`.

Super class

`mlr3torch::TorchDescriptor` -> `TorchLoss`

Public fields

`task_types (character())`
The task types this loss supports.

Methods

Public methods:

- `TorchLoss$new()`
- `TorchLoss$print()`
- `TorchLoss$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
TorchLoss$new(
  torch_loss,
  task_types = NULL,
  param_set = NULL,
  id = NULL,
  label = NULL,
  packages = NULL,
  man = NULL
)
```

Arguments:

`torch_loss (nn_loss)`
The loss module.

`task_types (character())`
The task types supported by this loss.

`param_set (ParamSet or NULL)`
The parameter set. If `NULL` (default) it is inferred from `torch_loss`.

`id (character(1))`
The id for of the new object.

`label (character(1))`
Label for the new instance.

`packages (character())`
The R packages this object depends on.

`man (character(1))`
String in the format `[pkg]::[topic]` pointing to a manual page for this object. The referenced help package can be opened via method `$help()`.

Method `print()`**:** Prints the object

Usage:

```
TorchLoss$print(...)
```

Arguments:

`...` any

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TorchLoss$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchOptimizer, as_torch_callbacks(), as_torch_loss(), as_torch_optimizer(), mlr3torch_losses, mlr3torch_optimizers, t_clbk(), t_loss(), t_opt()

Examples

```
# Create a new torch loss
torch_loss = TorchLoss$new(torch_loss = nn_mse_loss, task_types = "regr")
torch_loss
# the parameters are inferred
torch_loss$param_set

# Retrieve a loss from the dictionary:
torch_loss = t_loss("mse", reduction = "mean")
# is the same as
torch_loss
torch_loss$param_set
torch_loss$label
torch_loss$task_types
torch_loss$id

# Create the loss function
loss_fn = torch_loss$generate()
loss_fn
# Is the same as
nn_mse_loss(reduction = "mean")

# open the help page of the wrapped loss function
# torch_loss$help()

# Use in a learner
learner = lrn("regr.mlp", loss = t_loss("mse"))
# The parameters of the loss are added to the learner's parameter set
learner$param_set
```

`TorchOptimizer` *Torch Optimizer*

Description

This wraps a `torch::torch_optimizer_generatora` and annotates it with metadata, most importantly a `ParamSet`. The optimizer is created for the given parameter values by calling the `$generate()` method.

This class is usually used to configure the optimizer of a torch learner, e.g. when construcing a learner or in a `ModelDescriptor`.

For a list of available optimizers, see `mlr3torch_optimizers`. Items from this dictionary can be retrieved using `t_opt()`.

Parameters

Defined by the constructor argument `param_set`. If no parameter set is provided during construction, the parameter set is constructed by creating a parameter for each argument of the wrapped loss function, where the parametes are then of type `ParamUty`.

Super class

`mlr3torch::TorchDescriptor` -> `TorchOptimizer`

Methods

Public methods:

- `TorchOptimizer$new()`
- `TorchOptimizer$generate()`
- `TorchOptimizer$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
TorchOptimizer$new(
  torch_optimizer,
  param_set = NULL,
  id = NULL,
  label = NULL,
  packages = NULL,
  man = NULL
)
```

Arguments:

`torch_optimizer` (`torch_optimizer_generator`)
The torch optimizer.

`param_set` (`ParamSet` or `NULL`)
The parameter set. If `NULL` (default) it is inferred from `torch_optimizer`.

`id` (`character(1)`)
The id for of the new object.

`label` (`character(1)`)
Label for the new instance.

`packages` (`character()`)
The R packages this object depends on.

`man` (`character(1)`)
String in the format `[pkg]::[topic]` pointing to a manual page for this object. The referenced help package can be opened via method `$help()`.

Method `generate()`**:** Instantiates the optimizer.

Usage:

```
TorchOptimizer$generate(params)
```

Arguments:

`params` (named `list()` of `torch_tensor`s)
The parameters of the network.

Returns: `torch_optimizer`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TorchOptimizer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Torch Descriptor: `TorchCallback`, `TorchDescriptor`, `TorchLoss`, `as_torch_callbacks()`, `as_torch_loss()`, `as_torch_optimizer()`, `mlr3torch_losses`, `mlr3torch_optimizers`, `t_clbk()`, `t_loss()`, `t_opt()`

Examples

```
# Create a new torch loss
torch_opt = TorchOptimizer$new(optim_ignite_adam, label = "adam")
torch_opt
# If the param set is not specified, parameters are inferred but are of class ParamUty
torch_opt$param_set

# open the help page of the wrapped optimizer
# torch_opt$help()

# Retrieve an optimizer from the dictionary
torch_opt = t_opt("sgd", lr = 0.1)
torch_opt
torch_opt$param_set
torch_opt$label
torch_opt$id

# Create the optimizer for a network
net = nn_linear(10, 1)
opt = torch_opt$generate(net$parameters)
```

```
# is the same as
optim_sgd(net$parameters, lr = 0.1)

# Use in a learner
learner = lrn("regr.mlp", optimizer = t_opt("sgd"))
# The parameters of the optimizer are added to the learner's parameter set
learner$param_set
```

---

torch_callback *Create a Callback Descriptor*

Description

Convenience function to create a custom TorchCallback. All arguments that are available in callback_set() are also available here. For more information on how to correctly implement a new callback, see CallbackSet.

Usage

```
torch_callback(
  id,
  classname = paste0("CallbackSet", capitalize(id)),
  param_set = NULL,
  packages = NULL,
  label = capitalize(id),
  man = NULL,
  on_begin = NULL,
  on_end = NULL,
  on_exit = NULL,
  on_epoch_begin = NULL,
  on_before_valid = NULL,
  on_epoch_end = NULL,
  on_batch_begin = NULL,
  on_batch_end = NULL,
  on_after_backward = NULL,
  on_batch_valid_begin = NULL,
  on_batch_valid_end = NULL,
  on_valid_end = NULL,
  state_dict = NULL,
  load_state_dict = NULL,
  initialize = NULL,
  public = NULL,
  private = NULL,
  active = NULL,
  parent_env = parent.frame(),
  inherit = CallbackSet,
  lock_objects = FALSE
)
```

Arguments

id	(character(1)) ‘ The id for the torch callback.
classname	(character(1)) The class name.
param_set	(ParamSet) The parameter set, if not present it is inferred from the $initialize() method.
packages	(character()) The packages the callback depends on. Default isNULL‘.
label	(character(1)) The label for the torch callback. Defaults to the capitalized id.
man	(character(1)) String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help(). The default is NULL.
on_begin, on_end, on_epoch_begin, on_before_valid, on_epoch_end, on_batch_begin, on_batch_end, on_after_backward, on_batch_valid_begin, on_batch_valid_end, on_valid_end, on_exit	(function) Function to execute at the given stage, see section *Stages*.
state_dict	(function()) The function that retrieves the state dict from the callback. This is what will be available in the learner after training.
load_state_dict	(function(state_dict)) Function that loads a callback state.
initialize	(function()) The initialization method of the callback.
public, private, active	(list()) Additional public, private, and active fields to add to the callback.
parent_env	(environment()) The parent environment for the R6Class.
inherit	(R6ClassGenerator) From which class to inherit. This class must either be CallbackSet (default) or inherit from it.
lock_objects	(logical(1)) Whether to lock the objects of the resulting R6Class. If FALSE (default), values can be freely assigned to self without declaring them in the class definition.

Value

TorchCallback

Internals

It first creates an R6 class inheriting from CallbackSet (using callback_set()) and then wraps this generator in a TorchCallback that can be passed to a torch learner.

Stages

- begin :: Run before the training loop begins.
- epoch_begin :: Run he beginning of each epoch.
- batch_begin :: Run before the forward call.
- after_backward :: Run after the backward call.
- batch_end :: Run after the optimizer step.
- batch_valid_begin :: Run before the forward call in the validation loop.
- batch_valid_end :: Run after the forward call in the validation loop.
- valid_end :: Run at the end of validation.
- epoch_end :: Run at the end of each epoch.
- end :: Run after last epoch.
- exit :: Run at last, using on.exit().

See Also

Other Callback: TorchCallback, as_torch_callback(), as_torch_callbacks(), callback_set(), mlr3torch_callbacks, mlr_callback_set, mlr_callback_set.checkpoint, mlr_callback_set.progress, mlr_callback_set.tb, mlr_callback_set.unfreeze, mlr_context_torch, t_clbk()

Examples

```
custom_tcb = torch_callback("custom",
  initialize = function(name) {
    self$name = name
  },
  on_begin = function() {
    cat("Hello", self$name, ", we will train for ", self$ctx$total_epochs, "epochs.\n")
  },
  on_end = function() {
    cat("Training is done.")
  }
)

learner = lrn("classif.torch_featureless",
  batch_size = 16,
  epochs = 1,
  callbacks = custom_tcb,
  cb.custom.name = "Marie",
  device = "cpu"
)
task = tsk("iris")
learner$train(task)
```

t_clbk *Sugar Function for Torch Callback*

Description

Retrieves one or more TorchCallbacks from mlr3torch_callbacks. Works like mlr3::lrn() and mlr3::lrns().

Usage

```
t_clbk(.key, ...)

t_clbks(.keys)
```

Arguments

.key	(character(1)) The key of the torch callback.
...	(any) See description of dictionary_sugar_get().
.keys	(character()) The keys of the callbacks.

Value

TorchCallback

list() of TorchCallbacks

See Also

Other Callback: TorchCallback, as_torch_callback(), as_torch_callbacks(), callback_set(), mlr3torch_callbacks, mlr_callback_set, mlr_callback_set.checkpoint, mlr_callback_set.progress, mlr_callback_set.tb, mlr_callback_set.unfreeze, mlr_context_torch, torch_callback()

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchLoss, TorchOptimizer, as_torch_callbacks(), as_torch_loss(), as_torch_optimizer(), mlr3torch_losses, mlr3torch_optimizers, t_loss(), t_opt()

Examples

```
t_clbk("progress")
```

t_loss *Loss Function Quick Access*

Description

Retrieve one or more TorchLosses from mlr3torch_losses. Works like mlr3::lrn() and mlr3::lrns().

Usage

```
t_loss(.key, ...)

t_losses(.keys, ...)
```

Arguments

.key	(character(1)) Key of the object to retrieve.
...	(any) See description of dictionary_sugar_get.
.keys	(character()) The keys of the losses.

Value

A TorchLoss

See Also

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchLoss, TorchOptimizer, as_torch_callbacks(), as_torch_loss(), as_torch_optimizer(), mlr3torch_losses, mlr3torch_optimizers, t_clbk(), t_opt()

Examples

```
t_loss("mse", reduction = "mean")
# get the dictionary
t_loss()


t_losses(c("mse", "l1"))
# get the dictionary
t_losses()
```

t_opt *Optimizers Quick Access*

Description

Retrieves one or more TorchOptimizers from mlr3torch_optimizers. Works like mlr3::lrn() and mlr3::lrns().

Usage

```
t_opt(.key, ...)

t_opts(.keys, ...)
```

Arguments

.key	(character(1)) Key of the object to retrieve.
...	(any) See description of dictionary_sugar_get.
.keys	(character()) The keys of the optimizers.

Value

A TorchOptimizer

See Also

Other Torch Descriptor: TorchCallback, TorchDescriptor, TorchLoss, TorchOptimizer, as_torch_callbacks(), as_torch_loss(), as_torch_optimizer(), mlr3torch_losses, mlr3torch_optimizers, t_clbk(), t_loss()

Other Dictionary: mlr3torch_callbacks, mlr3torch_losses, mlr3torch_optimizers

Examples

```
t_opt("adam", lr = 0.1)
# get the dictionary
t_opt()


t_opts(c("adam", "sgd"))
# get the dictionary
t_opts()
```

Index