Package: rTorch (via r-universe)

March 24, 2025

Title R Bindings to 'PyTorch'

Version 0.4.2

Description 'R' implementation and interface of the Machine Learning platform 'PyTorch' <https://pytorch.org/> developed in 'Python'. It requires a 'conda' environment with 'torch' and 'torchvision' Python packages to provide 'PyTorch' functions, methods and classes. The key object in 'PyTorch' is the tensor which is in essence a multidimensional array. These tensors are fairly flexible in performing calculations in CPUs as well as 'GPUs' to accelerate tensor operations.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.1)

Imports reticulate (>= 1.10), jsonlite (>= 1.2), utils, methods, rstudioapi (>= 0.7),

Suggests testthat, knitr, rmarkdown

SystemRequirements ``conda (python>=3.6 pytorch torchvision cpuonly numpy (>= 1.14.0) matplotlib pandas -c pytorch); python-minimal, pandoc pandoc-citeproc, qpdf; Python (>=3.6), pytorch (>=1.4), torchvision, cpuonly, numpy (>= 1.14.0); pandoc (>= 2.0), qpdf (>= 7.0) on Solaris"

RoxygenNote 7.1.1

URL https://github.com/f0nzie/rTorch

BugReports https://github.com/f0nzie/rTorch/issues

VignetteBuilder knitr

NeedsCompilation no

Author Alfonso R. Reyes [aut, cre, cph]

Maintainer Alfonso R. Reyes <alfonso.reyes@oilgainsanalytics.com>

Date/Publication 2020-10-12 15:20:02 UTC

Additional_repositories https://cranhaven.r-universe.dev

Config/pak/sysreqs libpng-dev python3

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

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/rTorch

RemoteSha de3f04e9cc27fa40722bc8599af1086ecd1bd581

RemoteSubdir rTorch

Contents

`*.torch.Tensor` *Tensor multiplication*

Description

This generic is similar to `torch$mul(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a * b
```

Arguments

a	tensor
b	tensor

Value

Another tensor representing the multiplication of two tensors.

Examples

```
## Not run:
a <- torch$Tensor(list(1, 1, 1))
b <- torch$Tensor(list(2, 2, 2))
s <- 2.0
a * b

## End(Not run)
```

`+.torch.Tensor` *Add two tensors*

Description

This generic is similar to applying `torch$add(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a + b
```

Arguments

a	tensor
b	tensor

Value

Another tensor representing the addition of two tensors.

Examples

```
## Not run:
a <- torch$Tensor(list(1, 1, 1))
b <- torch$Tensor(list(2, 2, 2))
s <- 2.0
a + b

## End(Not run)
```

-.torch.Tensor *Subtract two tensors*

Description

This generic is similar to applying `torch$sub(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a - b
```

Arguments

a	tensor
b	tensor

Value

Another tensor representing the subtraction of two tensors.

Examples

```
## Not run:
a <- torch$Tensor(list(1, 1, 1))
b <- torch$Tensor(list(2, 2, 2))
s <- 2.0
a - b

## End(Not run)
```

/.torch.Tensor *Divide two tensors*

Description

This generic is similar to `torch$div(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a / b
```

Arguments

a	tensor
b	tensor

Value

Another tensor representing the division of two tensors.

Examples

```
## Not run:
a <- torch$Tensor(list(1, 1, 1))
b <- torch$Tensor(list(2, 2, 2))
s <- 2.0
a / b

## End(Not run)
```

<.torch.Tensor *Is a tensor less than another tensor*

Description

This generic is similar to `torch$lt(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a < b
```

Arguments

a	tensor
b	tensor

Value

A tensor of booleans representing the logical result of the comparison. False to represent 0, and True to represent 1 in a tensor of data type `torch$uint8`.

Examples

```
## Not run:
A <- torch$ones(28L, 28L)
C <- A * 0.5
A < C


## End(Not run)
```

`<=.torch.Tensor` *Is a tensor less or equal than another tensor*

Description

This generic is similar to `torch$le(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a <= b
```

Arguments

a	tensor
b	tensor

Value

A tensor of booleans representing the logical result of the comparison. False to represent 0, and True to represent 1 in a tensor of data type `torch$uint8`.

Examples

```
## Not run:
A <- torch$ones(5L, 5L)
C <- torch$as_tensor(np$random$randint(2L, size=c(5L, 5L)), dtype=torch$float32)
A <= C
C <= A

## End(Not run)
```

==.torch.Tensor *Compares two tensors if equal*

Description

This generic is approximately similar to `torch$eq(a, b)`, with the difference that the generic returns a tensor of booleans instead of a tensor of data type `torch$uint8`.

Usage

```
## S3 method for class 'torch.Tensor'
x == y
```

Arguments

x	tensor
y	tensor

Value

A tensor of booleans, where False corresponds to 0, and 1 to True in a tensor of data type `torch$bool`.

Examples

```
## Not run:
a <- torch$Tensor(list(1, 1, 1))
b <- torch$Tensor(list(2, 2, 2))
a == b

## End(Not run)
```

>.torch.Tensor *A tensor greater than another tensor*

Description

This generic is similar to `torch$gt(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a > b
```

Arguments

a	tensor
b	tensor

Value

A tensor of booleans representing the logical result of the comparison. False to represent 0, and True to represent 1 in a tensor of data type `torch$uint8`.

Examples

```
## Not run:
A <- torch$ones(5L, 5L)
C <- torch$as_tensor(np$random$randint(2L, size=c(5L, 5L)), dtype=torch$float32)
A > C
C > A

## End(Not run)
```

>=.torch.Tensor *Is a tensor greater or equal than another tensor*

Description

This generic is similar to `torch$ge(a, b)`

Usage

```
## S3 method for class 'torch.Tensor'
a >= b
```

Arguments

a	tensor
b	tensor

Value

A tensor of booleans representing the logical result of the comparison. False to represent 0, and True to represent 1 in a tensor of data type `torch$uint8`.

Examples

```
## Not run:
A <- torch$ones(5L, 5L)
C <- torch$as_tensor(np$random$randint(2L, size=c(5L, 5L)), dtype=torch$float32)
A >= C
C >= A

## End(Not run)
```

`all.torch.Tensor` *all*

Description

Returns True if all elements in the tensor are non-zero, False otherwise.

Usage

```
## S3 method for class 'torch.Tensor'
all(x, dim, ...)
```

Arguments

`x`	tensor
`dim`	dimension to reduce
`...`	other parameters (yet to be developed)

Value

A tensor of type torch.uint8 representing the boolean result: 1 for TRUE and 0 for FALSE.

Examples

```
## Not run:
a <- torch$BoolTensor(list(TRUE, TRUE, TRUE, TRUE))
b <- torch$BoolTensor(list(FALSE, TRUE, TRUE, TRUE))
c <- torch$BoolTensor(list(TRUE, TRUE, TRUE, FALSE))
all(a)
all(b)
all(c)
d <- torch$tensor(list(list(0, 0),
                       list(0, 0),
                       list(0, 1),
                       list(1, 1)), dtype=torch$uint8)
all(d)
all(d, dim=0L)
all(d, dim=1L)

## End(Not run)
```

all_dims *All dims*

Description

This function returns an object that can be used when subsetting tensors with `[`. If you are familiar with Python, this is equivalent to the Python Ellipsis `...`, (not to be confused with `...` in `R`). In Python, if `x` is a numpy array or a torch tensor, in `x[..., i]` the ellipsis means "expand to match number of dimension of x". To translate the above Python expression to R, write: `x[all_dims(), i]`.

Usage

```
all_dims()
```

Examples

```
## Not run:
# Run this
d <- torch$tensor(list(list(0, 0),
                       list(0, 0),
                       list(0, 1),
                       list(1, 1)), dtype=torch$uint8)
d[all_dims(), 1]

f <- torch$arange(9L)$reshape(c(3L, 3L))
f
f[all_dims()]
f[all_dims(), 1L]

## End(Not run)
```

any.torch.Tensor *any*

Description

Returns True if any elements in the tensor are non-zero, False otherwise.

Usage

```
## S3 method for class 'torch.Tensor'
any(x, dim, ...)
```

Arguments

x	tensor
dim	dimension to reduce
...	other params (yet to be developed)

Value

A tensor of type torch.uint8 representing the boolean result: 1 for TRUE and 0 for FALSE.

Examples

```
## Not run:
a <- torch$BoolTensor(list(TRUE, TRUE, TRUE, TRUE))
b <- torch$BoolTensor(list(FALSE, TRUE, TRUE, TRUE))
c <- torch$BoolTensor(list(TRUE, TRUE, TRUE, FALSE))
any(a)
any(b)
any(c)
d <- torch$tensor(list(list(1, 0),
                       list(0, 0),
                       list(0, 1),
                       list(0, 0)), dtype=torch$uint8)
any(d)
any(d, dim=0L)
any(d, dim=1L)

## End(Not run)
```

---

as_boolean *Convert tensor to boolean type*

Description

Convert a tensor to a boolean equivalent tensor

Usage

```
as_boolean(x)
```

Arguments

x	a torch tensor

Examples

```
## Not run:
uo <- torch$ones(3L)
as_boolean(uo)              # tensor([True, True, True], dtype=torch.bool)

## End(Not run)
```

dataset_mnist_digits *MNIST database of handwritten digits*

Description

Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.

Usage

```
dataset_mnist_digits(ntrain = 60000L, ntest = 10000L, onehot = TRUE)
```

Arguments

ntrain	number of training samples
ntest	number of test samples
onehot	boolean

dim.torch.Tensor *Dimensions of a tensor*

Description

Get the dimensions of a tensor displaying it as a vector.

Usage

```
## S3 method for class 'torch.Tensor'
dim(x)
```

Arguments

x	tensor

Value

a vector of integers with the dimensions of the tensor

Examples

```
## Not run:
uo = torch$ones(3L, 5L)  # it is a 3x5 tensor
dim(uo)

## End(Not run)
```

`install_pytorch` *Install PyTorch and its dependencies*

Description

Install PyTorch and its dependencies

Usage

```
install_pytorch(
  method = c("conda", "virtualenv", "auto"),
  conda = "auto",
  version = "default",
  envname = "r-torch",
  extra_packages = NULL,
  restart_session = TRUE,
  conda_python_version = "3.6",
  pip = FALSE,
  channel = "stable",
  cuda_version = NULL,
  dry_run = FALSE,
  ...
)
```

Arguments

`method`	Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on *Windows* (as this isn't supported by *PyTorch*). Note also that since this command runs without privillege the "system" method is available only on *Windows*.
`conda`	The path to a `conda` executable. Use `"auto"` to allow `reticulate` to automatically find an appropriate `conda` binary. See **Finding Conda** for more details.
`version`	PyTorch version to install. The "default" version is **1.4**. You can specify a specific **PyTorch** version with `version="1.2"`, or `version="1.6"`.
`envname`	Name of Python or conda environment to install within. The default environment name is `r-torch`.
`extra_packages`	Additional Python packages to install along with PyTorch. If more than one package use a character vector: `c("pandas", "matplotlib")`.
`restart_session`	Restart R session after installing (note this will only occur within RStudio).
`conda_python_version`	the *Python* version installed in the created *conda* environment. Python **3.4** is installed by default. But you could specify for instance: `conda_python_version="3.7"`.
`pip`	logical

channel	conda channel. The default channel is stable. The alternative channel is nightly.
cuda_version	string for the cuda toolkit version to install. For example, to install a specific CUDA version use cuda_version="10.2".
dry_run	logical, set to TRUE for unit tests, otherwise will execute the command.
...	other arguments passed to reticulate::conda_install() or reticulate::virtualenv_install().

---

is_tensor *Is the object a tensor*

Description

Determine if the object is a tensor by looking inheritance

Usage

```
is_tensor(obj)
```

Arguments

obj	an object

---

length.torch.Tensor *Length of a tensor.*

Description

This function is equivalent to torch$numel()

Usage

```
## S3 method for class 'torch.Tensor'
length(x)
```

Arguments

x	tensor

Value

the number of elements of a tensor as an integer

Examples

```
## Not run:
uo = torch$ones(3L, 5L)   # tensor with 15 elements
length(uo)

## End(Not run)
```

`log.torch.Tensor` *Logarithm of a tensor given the tensor and the base*

Description

Logarithm of a tensor given the tensor and the base

Usage

```
## S3 method for class 'torch.Tensor'
log(x, base = exp(1L))
```

Arguments

`x`	a tensor
`base`	the base of the logarithm

Examples

```
## Not run:
x <- torch$tensor(c(512, 1024, 2048, 4096))   # tensor([ 9., 10., 11., 12.])
base <- 2
log(x, base)

x <- torch$tensor(c(1, 10, 100, 1000))         # tensor([0., 1., 2., 3.])
log(x, 10)

## End(Not run)
```

log10.torch.Tensor *Logarithm of a tensor in base 10*

Description

Logarithm of a tensor in base 10

Usage

```
## S3 method for class 'torch.Tensor'
log10(x)
```

Arguments

x	a tensor

Examples

```
## Not run:
x <- torch$tensor(c(1, 10, 100, 1000))     # tensor([0., 1., 2., 3.])

## End(Not run)
```

log2.torch.Tensor *Logarithm of a tensor in base 2*

Description

Logarithm of a tensor in base 2

Usage

```
## S3 method for class 'torch.Tensor'
log2(x)
```

Arguments

x	a tensor

Examples

```
## Not run:
x <- torch$tensor(c(512, 1024, 2048, 4096))   # tensor([ 9., 10., 11., 12.])

## End(Not run)
```

logical_and *Logical AND of two tensors*

Description

There is not equivalent function in PyTorch for this generic. To generate this generic we use the function `np$logical_and()`.

Usage

```
## S3 method for class 'torch.Tensor'
x & y
```

Arguments

x	tensor
y	tensor

Value

A tensor of booleans representing the logical result of the comparison. False to represent 0, and True to represent 1 in a tensor of data type `torch$uint8`.

Examples

```
## Not run:
A <- torch$BoolTensor(list(0L, 1L))
B <- torch$BoolTensor(list(1L, 0L))
C <- torch$BoolTensor(list(1L, 1L))
A & B
C & A
B & C

## End(Not run)
```

logical_not *Logical NOT of a tensor*

Description

There is not equivalent function in PyTorch for this generic. To generate This generic we use the function `np$logical_not(x)`.

Usage

```
## S3 method for class 'torch.Tensor'
!x
```

Arguments

x	tensor

Value

A tensor of booleans, where False corresponds to 0, and 1 to True in a tensor of data type `torch$bool`.

Examples

```
## Not run:
A <- torch$ones(5L)
!A

Z <- torch$zeros(5L)
!Z

## End(Not run)
```

---

logical_or *Logical OR of two tensors*

Description

There is not equivalent function in PyTorch for this generic. To generate this generic we use the function `np$logical_or()`.

Usage

```
## S3 method for class 'torch.Tensor'
x | y
```

Arguments

x	tensor
y	tensor

Value

A tensor of booleans representing the logical result of the comparison. False to represent 0, and True to represent 1 in a tensor of data type `torch$uint8`.

Examples

```
## Not run:
A <- torch$BoolTensor(list(0L, 1L))
B <- torch$BoolTensor(list(1L, 0L))
C <- torch$BoolTensor(list(1L, 1L))
A | B
C | A
B | C

## End(Not run)
```

`make_copy` *Make copy of tensor, numpy array or R array*

Description

A copy of an array or tensor might be needed to prevent warnings by new PyTorch versions on overwriting the numpy object

Usage

```
make_copy(object, ...)
```

Arguments

`object`	a torch tensor or numpy array or R array
`...`	additional parameters

`not_equal_to` *Compare two tensors if not equal*

Description

This generic is approximately similar to `torch$ne(a, b)`, with the difference that the generic returns a tensor of booleans instead of a tensor of data type `torch$uint8`.

Usage

```
## S3 method for class 'torch.Tensor'
x != y
```

Arguments

`x`	tensor
`y`	tensor

Value

A tensor of booleans, where False corresponds to 0, and 1 to True in a tensor of data type `torch$bool`.

Examples

```
## Not run:
a <- torch$Tensor(list(1, 1, 1))
b <- torch$Tensor(list(2, 2, 2))
a != b

## End(Not run)
```

---

one_tensor_op *One tensor operation*

Description

One tensor operation

Usage

```
one_tensor_op(x)

## S3 method for class 'torch.Tensor'
exp(x)
```

Arguments

x	tensor

Methods (by class)

- `torch.Tensor`: Exponential of a tensor

Examples

```
## Not run:
A <- torch$ones(c(60000L, 1L, 28L, 28L))
dim(A)

## End(Not run)
```

---

rTorch *PyTorch for R*

Description

PyTorch bindings for R

shape *Tensor shape*

Description

Tensor shape

Usage

```
shape(...)
```

Arguments

...	Tensor dimensions

tensor_ops *Two tensor operations*

Description

Two tensor operations

Usage

```
tensor_ops(a, b)

## S3 method for class 'torch.Tensor'
a ^ b
```

Arguments

a	tensor
b	tensor

Methods (by class)

- torch.Tensor: A tensor 'a' to the power of 'b'

Examples

```
## Not run:
a <- torch$Tensor(list(1, 1, 1))
b <- torch$Tensor(list(2, 2, 2))
s <- 2.0
a + b
b - a
a * b
a / s
a == b
a == a
a != a
x <- torch$Tensor(list(list(2, 2, 2), list(4, 4, 4)))
y <- torch$Tensor(list(list(1, 2, 1), list(3, 4, 5)))
x > y
x < y
x >= y
y <= x
diag <- torch$eye(3L)
zeros <- torch$zeros(c(3L, 3L))
diag & zeros
diag & diag
diag | diag
zeros | zeros
zeros & zeros
diag & zeros
diag | zeros

## End(Not run)
## Not run:
x <- torch$arange(1,11)
torch$pow(x, 2)      #     x^(2)
torch$pow(x, -2)     #     x^(1/2)

## End(Not run)
```

`torch` *Main PyTorch module*

Description

Interface to main PyTorch module. Provides access to top level classes

Interface to numpy module.

Interface to Torchvision module.

Usage

```
torch

np

torchvision
```

Format

PyTorch module

numpy module

Torchvision module

---

torch_extract_opts *Tensor extract options*

Description

Tensor extract options

Usage

```
torch_extract_opts(
  style = getOption("torch.extract.style"),
  ...,
  one_based = getOption("torch.extract.one_based", TRUE),
  inclusive_stop = getOption("torch.extract.inclusive_stop", TRUE),
 disallow_out_of_bounds = getOption("torch.extract.dissallow_out_of_bounds", TRUE),
 warn_tensors_passed_asis = getOption("torch.extract.warn_tensors_passed_asis", TRUE),
 warn_negatives_pythonic = getOption("torch.extract.warn_negatives_pythonic", TRUE)
)
```

Arguments

style	one of NULL (the default) "R" or "python". If supplied, this overrides all other options. "python" is equivalent to all the other arguments being FALSE. "R" is equivalent to warn_tensors_passed_asis and warn_negatives_pythonic set to FALSE
...	ignored
one_based	TRUE or FALSE, if one-based indexing should be used
inclusive_stop	TRUE or FALSE, if slices like start:stop should be inclusive of stop
disallow_out_of_bounds	TRUE or FALSE, whether checks are performed on the slicing index to ensure it is within bounds.

warn_tensors_passed_asis
: TRUE or FALSE, whether to emit a warning the first time a tensor is supplied to [that tensors are passed as-is, with no R to python translation

warn_negatives_pythonic
: TRUE or FALSE, whether to emit a warning the first time a negative number is supplied to [about the non-standard (python-style) interpretation

Value

an object with class "torch_extract_opts", suitable for passing to `[.torch.tensor()`

Examples

```
## Not run:

x <- torch$arange(1L, 10L)

opts <-  torch_extract_opts("R")
x[1, options = opts]

# or for more fine-grained control
opts <- torch_extract_opts(
    one_based = FALSE,
    warn_tensors_passed_asis = FALSE,
    warn_negatives_pythonic = FALSE
)
x[0:2, options = opts]

## End(Not run)
```

---

torch_size *Size of a torch tensor object*

Description

Get the size of a torch tensor or of torch.size object

Usage

```
torch_size(obj)
```

Arguments

obj
: a torch tensor object

`[.torch.Tensor` *Subset tensors with* `[`

Description

Subset tensors with `[`

Usage

```
## S3 method for class 'torch.Tensor'

  x[
  ...,
  drop = TRUE,
  style = getOption("torch.extract.style"),
  options = torch_extract_opts(style)
]
```

Arguments

`x`	a tensor
`...`	slicing specs. See examples and details.
`drop`	whether to drop scalar dimensions
`style`	One of `"python"` or `"R"`.
`options`	An object returned by `torch_extract_opts()`

Examples

```
## Not run:

x <- torch$arange(0L, 15L)$view(3L, 5L)

# by default, numerics supplied to `...` are interpreted R style
x[,1]    # first column
x[1:2,]  # first two rows
x[,1, drop = FALSE]

# strided steps can be specified in R syntax or python syntax
x[, seq(1, 5, by = 2)]
x[, 1:5:2]

# if you are unfamiliar with python-style strided steps, see:
# https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.indexing.html#basic-slicing-and-indexing

# missing arguments for python syntax are valid, but they must by backticked
# or supplied as NULL
x[, `::2`]
x[, NULL:NULL:2]
```

```
x[, `2:`]

# Another Python feature that is available is a Python style ellipsis `...`
# (not to be confused with R dots `...`), that in R has been defined as
# all_dims() expands to the shape of the tensor

y <- torch$arange(0L, 3L^5L)$view(3L, 3L, 3L, 3L, 3L)
as.logical((all(y[all_dims(), 1] == y[,,,,1]))$numpy()) == TRUE


# negative numbers are always interpreted Python style
# The first time a negative number is supplied to `[`, a warning is issued
# about the non-standard behavior.
x[-1,] # last row, with a warning
x[-1,] # the warning is only issued once

# specifying `style = 'python'` changes the following:
# +  zero-based indexing is used
# +  slice sequences in the form of `start:stop` do not include `stop`
#    in the returned value
# +  out-of-bounds indices in a slice are valid

# The style argument can be supplied to individual calls of `[` or set
# as a global option

# example of zero based  indexing
x[0, , style = 'python']   # first row
x[1, , style = 'python']   # second row

# example of slices with exclusive stop
# run the next options() line before the tensor operations

options(torch.extract.style = 'python')
x[, 0:1]   # just the first column
x[, 0:2]   # first and second column

# example of out-of-bounds index
x[, 0:10]
options(torch.extract.style = NULL)

# slicing with tensors is valid too, but note that tensors are never
# translated and are always interpreted Python-style.
# A warning is issued the first time a tensor is passed to `[`
# just as in Python, only scalar tensors are valid

# To silence the warnings about tensors being passed as-is and negative numbers
# being interpreted python-style, set
options(torch.extract.style = 'R')

# clean up from examples
options(torch.extract.style = NULL)

## End(Not run)
```

%.*% *#' @export "round.torch.Tensor" <- function(input) # round: Returns a new tensor with each of the elements of input rounded to the closest integer. torch$round(input) Dot product of two tensors*

Description

This generic is similar to `torch$dot(a, b)`

Usage

```
a %.*% b
```

Arguments

a	tensor
b	tensor

Value

a scalar

Examples

```
## Not run:
p <- torch$Tensor(list(2, 3))
q <- torch$Tensor(list(2, 1))
p %.*% q

## End(Not run)
```

%%.torch.Tensor *Remainder*

Description

Computes the element-wise remainder of division.

Usage

```
## S3 method for class 'torch.Tensor'
a %% b
```

Arguments

a	a tensor
b	a scalar or a tensor

Value

the reminder of the division between tensor by a scalar or tensor

Examples

```
## Not run:
x <- torch$Tensor(list(-3., -2, -1, 1, 2, 3))
y <- torch$Tensor(list(1., 2, 3, 4, 5))
torch$remainder(x, 2)
torch$remainder(y, 1.5)

x %% 2
y %% 1.5

## End(Not run)
```

%**% *Matrix/Tensor multiplication of two tensors*

Description

This generic is similar to `torch$matmul(a, b)`

Usage

```
a %**% b
```

Arguments

a	tensor
b	tensor

Value

a scalar or a tensor

Examples

```
## Not run:
p <- torch$randn(3L)
q <- torch$randn(3L)
p %**% q

## End(Not run)
```

Index