Package 'einops'

September 3, 2025

Title Flexible, Powerful, and Readable Tensor Operations

Version 0.2.1

Maintainer Qile Yang <qile.yang@berkeley.edu>

Description Perform tensor operations using a concise yet expressive syntax inspired by the Python library of the same name.
Reshape, rearrange, and combine multidimensional arrays for scientific computing, machine learning, and data analysis.
Einops simplifies complex manipulations, making code more maintainable and intuitive.
The original implementation is demonstrated in Rogozhnikov (2022) <https://openreview.net/forum?id=oapKSVM2bcj>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

URL https://github.com/Qile0317/einops,
https://qile0317.github.io/einops/

BugReports https://github.com/Qile0317/einops/issues

Imports assertthat, FastUtils, glue, magrittr, r2r, R6, roperators

Suggests abind, grid, imager, knitr, lifecycle, lintr, lobstr, rmarkdown, spelling, testthat (>= 3.0.0), torch, zeallot

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 3.5)

LazyData true

Language en-US

NeedsCompilation no

Author Qile Yang [cre, aut, cph] (ORCID: <https://orcid.org/0009-0005-0148-2499>)

Repository CRAN

Date/Publication 2025-09-03 21:40:08 UTC

Contents

`einop` *Perform Einstein-style tensor operations*

Description

A unified interface for rearranging, reducing, and repeating tensor dimensions using Einstein notation-inspired expressions. This was directly adapted from the python `einop` package: https://github.com/cgarciae/einop

`einop()` auto detects the operation type based on the provided expression:

- `rearrange()` when all input dimensions are present in output
- `reduce()` when some input dimensions are missing from output
- `einops.repeat()` when output contains dimensions not in input

Note that there are ongoing debates about the use of this function purely from the perspective of code readability and maintainability: https://github.com/arogozhnikov/einops/issues/84. Generally, some argue that the descriptive names of `rearrange`, `reduce`, and `repeat` encourage good practices, while others think that semantically `einop()` actually makes it *clearer* what the operation is doing, as opposed to mandating the use of these commonly used function names across many packages.

Usage

```
einop(
  x,
  expr,
  reduction = NULL,
  ...,
  .row_major = getOption("einops_row_major", FALSE)
)
```

Arguments

`x`	tensor: array, matrix, or list of arrays of the same shape and type
`expr`	string: reduction pattern
`reduction`	A string specifying the reduction operation (e.g., "mean", "sum", "max"). Required for reduce operations, ignored for rearrange and repeat operations.

...	either corresponding axes lengths or a single list of them.
.row_major	**[Experimental]** logical: whether to use row-major order for the output tensor. If TRUE, the *operation* is performed in row-major order, but the output will be in whatever order the parent framework uses (e.g. column-major for base::array()).

Value

A tensor with dimensions transformed according to the expression

Examples

```
if (requireNamespace("abind", quietly = TRUE)) {

# load a 3d tensor representing an rgb image
x <- get(data("einops_image"))[1, , , ]

# Rearrange dimensions
einop(x, "h w c -> c h w")

# Reduce dimensions
einop(x, "h w c -> h w", "mean")

# Repeat dimensions
einop(x[, , 1], "h w -> h w c", c = 3)

}
```

---

einops.repeat *Allows reordering elements and repeating them in arbitrary combinations.*

Description

This operation includes functionality of repeat, tile, and broadcast functions.

Usage

```
einops.repeat(x, expr, ..., .row_major = getOption("einops_row_major", FALSE))

`repeat`(x, expr, ..., .row_major = getOption("einops_row_major", FALSE))
```

Arguments

x	tensor: array, matrix, or list of arrays of the same shape and type
expr	string: reduction pattern
...	either corresponding axes lengths or a single list of them.

`.row_major` **[Experimental]** logical: whether to use row-major order for the output tensor. If `TRUE`, the *operation* is performed in row-major order, but the output will be in whatever order the parent framework uses (e.g. column-major for `base::array()`).

Details

When composing axes, C-order enumeration is used (consecutive elements have different last axis). Find more examples in the vignettes.

Value

tensor of the same type as input, with dimensions according to output pattern

Why can't the function be called as `repeat()`?

`repeat` is a reserved keyword in R that acts the same as `while(TRUE)`, and has no way of being overridden. Hence, this function can only be called as `einops.repeat()` or using backticks as `` `repeat`() ``.

Examples

```
if (requireNamespace("abind", quietly = TRUE)) {

set.seed(42)
# a grayscale image (of shape height x width)
image <- array(rnorm(30 * 40), dim = c(30, 40))

# change it to RGB format by repeating in each channel
output <- einops.repeat(image, 'h w -> h w c', c = 3)
# Visualize the output
as_image_tensor(output)

# repeat image 2 times along height (vertical axis)
output <- einops.repeat(image, 'h w -> (r h) w', r = 2)

# repeat image 2 times along height and 3 times along width
output <- einops.repeat(image, 'h w -> (h2 h) (w3 w)', h2 = 2, w3 = 3)

# convert each pixel to a small square 2x2, i.e. upsample an image by 2x
output <- einops.repeat(image, 'h w -> (h h2) (w w2)', h2 = 2, w2 = 2)

# 'pixelate' an image first by downsampling by 2x, then upsampling
downsampled <- reduce(image, '(h h2) (w w2) -> h w', 'mean', h2 = 2, w2 = 2)
output <- einops.repeat(downsampled, 'h w -> (h h2) (w w2)', h2 = 2, w2 = 2)
as_image_tensor(einops.repeat(output, 'h w -> h w 3'))

}
```

einops_image *Example 4D Image Tensor for Einops*

Description

An `image_tensor()` object that is a super thin wrapper around a 4D `base::array()`, representing image data in the format "b h w c" (batch, height, width, channels). The actual image data is 6 black letters on differently colored backgrounds, spelling out the word "einops". When printing this object in the terminal it will automatically plot the images. To subset, use the `[` operator, and when used with a single index, it will return a single image tensor.

Usage

```
einops_image
```

Format

An `image_tensor()` object with 6 images, each of size 96 x 96 pixels, with 3 color channels (RGB). The images are stored in a 4D array with dimensions (6, 96, 96, 3).

Examples

```
data("einops_image")
for (i in seq_len(6)) print(einops_image[i, , , ])
```

image_tensor *Image Tensor: A thin wrapper around 2-4D arrays*

Description

The `image_tensor` class provides a convenient way to work with image data in tensor format. It extends the base `array` class and provides methods for conversion to/from various image formats, plotting, and printing.

An `image_tensor` object represents image data in the format "h w c" (height, width, channels) for single images, or "b h w c" (batch, height, width, channels) for batches of images, which is a common format for deep learning frameworks. It also can be a 2D array, in which case it is treated as a black and white image and shown as such.

The main utility of wrapping image data in the `image_tensor` class is that printing of the object will automatically display the image as a plot, as long as the `imager` package is installed. Otherwise it will simply print the dimension of the image.

Usage

```
as_image_tensor(x)

image_tensor(x)
```

Arguments

x	An object to convert to or from image_tensor format.

Details

The image_tensor class provides the following methods (and more):

- as_image_tensor(): Generic function to convert objects to image_tensor format. Takes in array-like objects of 2-4 dimensions. for 2 dimensional objects, it will convert them to 3D by repeating the data across 3 channels, essentially converting grayscale images to RGB.
- as_image_tensor.default(): Default method that converts arrays to image_tensor
- as_image_tensor.cimg(): Method to convert cimg objects (from imager package) to image_tensor
- as.cimg.image_tensor(): Method to convert image_tensor objects back to cimg format
- [.image_tensor(): Subset method for image_tensor objects
- plot.image_tensor(): Plot method for image_tensor objects
- print.image_tensor(): Print method for image_tensor objects

Value

- as_image_tensor(): An object of class image_tensor
- as.cimg(): A cimg object (from imager package)
- [.image_tensor(): A subset of the image_tensor object. For 4D arrays with single index, returns a 3D slice without the batch dimension.
- plot(): Invisibly returns the input object
- print(): Invisibly returns the input object

Format

An image_tensor object is an array with dimensions in "h w c" format for single images, or "b h w c" format for batches of images:

- **h**: height dimension (image height in pixels)
- **w**: width dimension (image width in pixels)
- **c**: channel dimension (RGB, only for 3D & 4D arrays)
- **b**: batch dimension (number of images, only for 4D arrays)

Options

The following options control the default behavior of image_tensor methods:

- plot_image_tensor_axes: Whether to show axes in plots (default: FALSE)
- print_image_tensor_as_plot: Whether to print images as plots (default: TRUE)

Examples

```
# create from a matrix (grayscale)
img <- image_tensor(matrix(1:9, 3, 3))
print(img)
print(img[1:2, 1:2])

# create from a 3D array (RGB image)
img_rgb <- as_image_tensor(array(runif(27), dim = c(3, 3, 3)))
print(img_rgb)
```

---

parse_shape *Parse a tensor shape to dictionary mapping axes names to their lengths.*

Description

Use underscore to skip the dimension in parsing.

Usage

```
parse_shape(x, expr, ...)
```

Arguments

x	tensor of any supported framework
expr	character of length 1, space separated names for axes, underscore means skip axis
...	additional arguments passed to methods

Value

named list, maps axes names to their lengths

Examples

```
if (requireNamespace("abind", quietly = TRUE)) {

# Use underscore to skip the dimension in parsing.
x <- array(0, dim = c(2, 3, 5, 7))
parse_shape(x, 'batch _ h w')

# `parse_shape` output can be used to specify axes_lengths for other
# operations:
y <- array(0, dim = 700)
shape_info <- parse_shape(x, 'b _ h w')
# rearrange(y, '(b c h w) -> b c h w', shape_info) would give shape
# (2, 10, 5, 7)
```

```
}
```

`rearrange` *Reader-friendly smart element reordering for multidimensional tensors.*

Description

This operation includes functionality of transpose (axes permutation), reshape (view), squeeze, unsqueeze, stack, concatenate and other operations.

Usage

```
rearrange(x, expr, ..., .row_major = getOption("einops_row_major", FALSE))

einops.rearrange(
  x,
  expr,
  ...,
  .row_major = getOption("einops_row_major", FALSE)
)
```

Arguments

`x`	tensor: array, matrix, or list of arrays of the same shape and type
`expr`	string: reduction pattern
`...`	either corresponding axes lengths or a single list of them.
`.row_major`	**[Experimental]** logical: whether to use row-major order for the output tensor. If `TRUE`, the *operation* is performed in row-major order, but the output will be in whatever order the parent framework uses (e.g. column-major for `base::array()`).

Details

When composing axes, C-order enumeration is used (consecutive elements have different last axis). Find more examples in the vignettes.

Value

tensor of the same type as input, with dimensions according to output pattern

Examples

```
if (requireNamespace("abind", quietly = TRUE)) {

# suppose we have a set of 32 images in "h w c" format (height-width-channel)
images <- lapply(1:32, function(i) {
    as_image_tensor(array(rnorm(30*40*3), dim = c(30, 40, 3)))
})

# stacked and reordered axes to "b c h w" format
y <- rearrange(images, 'b h w c -> b c h w')

# concatenate images along height (vertical axis), 960 = 32 * 30
y <- rearrange(images, 'b h w c -> (b h) w c')

# concatenated images along horizontal axis, 1280 = 32 * 40
y <- rearrange(images, 'b h w c -> h (b w) c')

# flattened each image into a vector, 3600 = 30 * 40 * 3
y <- rearrange(images, 'b h w c -> b (c h w)')

# split each image into 4 smaller quadrants, 128 = 32 * 2 * 2
y <- rearrange(
    images, 'b (h1 h) (w1 w) c -> (b h1 w1) h w c', h1 = 2, w1 = 2
)

# space-to-depth operation
y <- rearrange(
    images, 'b (h h1) (w w1) c -> b h w (c h1 w1)', h1 = 2, w1 = 2
)

}
```

---

reduce *Rearrangement and reduction in one step*

Description

reduce() combines rearrangement and reduction using reader-friendly notation.

Usage

```
reduce(x, expr, func, ..., .row_major = getOption("einops_row_major", FALSE))

einops.reduce(
  x,
  expr,
  func,
  ...,
```

```
  .row_major = getOption("einops_row_major", FALSE)
)
```

Arguments

x	tensor: array, matrix, or list of arrays of the same shape and type
expr	string: reduction pattern
func	string or function: one of available reductions ('min', 'max', 'sum', 'mean', 'prod', 'any', 'all'), or an R 2 argument function which takes in two arguments, with the first being the tensor, and the second being an integer array indicating the dimensions to reduce over.
...	either corresponding axes lengths or a single list of them.
.row_major	**[Experimental]** logical: whether to use row-major order for the output tensor. If TRUE, the *operation* is performed in row-major order, but the output will be in whatever order the parent framework uses (e.g. column-major for base::array()).

Value

tensor of the same type as input, with dimensions according to output pattern

Examples

```
if (requireNamespace("abind", quietly = TRUE)) {

set.seed(42)
# Suppose x is a 3D array: 100 x 32 x 64
x <- array(rnorm(100 * 32 * 64), dim = c(100, 32, 64))

# perform max-reduction on the first axis
# Axis t does not appear on RHS - thus we reduced over t
y <- reduce(x, 't b c -> b c', 'max')

# same as previous, but using verbose names for axes
y <- reduce(x, 'time batch channel -> batch channel', 'max')

# let's pretend now that x is a batch of images
# with 4 dims: batch=10height = 20width = 30channel = 40
x <- array(rnorm(10 * 20 * 30 * 40), dim = c(10, 20, 30, 40))

# 2d max-pooling with kernel size = 2 * 2 for image processing
y1 <- reduce(x, 'b c (h1 h2) (w1 w2) -> b c h1 w1', 'max', h2 = 2, w2 = 2)
as_image_tensor(y1)

# same as previous, using anonymous axes,
# note: only reduced axes can be anonymous
y1 <- reduce(x, 'b c (h1 2) (w1 2) -> b c h1 w1', 'max')
as_image_tensor(y1)

# adaptive 2d max-pooling to 3 * 4 grid,
```

```
# each element is max of 10x10 tile in the original tensor.
dim(reduce(x, 'b c (h1 h2) (w1 w2) -> b c h1 w1', 'max', h1 = 3, w1 = 4))
# (10, 20, 3, 4)

# Global average pooling
dim(reduce(x, 'b c h w -> b c', 'mean'))
# (10, 20)

}
```

Index