Package 'mxnet'

November 15, 2016

Type Package

Title MXNet

Version 0.7

Date 2015-12-23

Author Tianqi Chen, Qiang Kou, Tong He

Maintainer Qiang Kou <qkou@umail.iu.edu>

Description MXNet is a deep learning framework designed for both efficiency and flexibility. It allows you to mix the flavours of deep learning programs together to maximize the efficiency and your productivity.

License BSD

URL https://github.com/dmlc/mxnet/R-package

BugReports https://github.com/dmlc/mxnet/issues

Imports methods,
Rcpp (>= 0.12.1),
DiagrammeR (>= 0.8.1),
data.table,
jsonlite,
magrittr,
stringr

Suggests testthat,
mlbench,
knitr,
rmarkdown,
imager,
roxygen2

LinkingTo Rcpp

RoxygenNote 5.0.1

VignetteBuilder knitr

R topics documented:

`arguments` *Get the arguments of symbol.*

Description

Get the arguments of symbol.

Usage

```
arguments(x)
```

Arguments

`x`	The input symbol

as.array.MXNDArray *as.array operator overload of mx.ndarray*

Description

as.array operator overload of mx.ndarray

Usage

```
## S3 method for class 'MXNDArray'
as.array(nd)
```

Arguments

nd	The mx.ndarray

as.matrix.MXNDArray *as.matrix operator overload of mx.ndarray*

Description

as.matrix operator overload of mx.ndarray

Usage

```
## S3 method for class 'MXNDArray'
as.matrix(nd)
```

Arguments

nd	The mx.ndarray

ctx *Get the context of mx.ndarray*

Description

Get the context of mx.ndarray

Usage

```
ctx(nd)
```

Arguments

nd	The mx.ndarray

dim.MXNDArray *Dimension operator overload of mx.ndarray*

Description

Dimension operator overload of mx.ndarray

Usage

```
## S3 method for class 'MXNDArray'
dim(nd)
```

Arguments

nd	The mx.ndarray

graph.viz *Convert symbol to dot object for visualization purpose.*

Description

Convert symbol to dot object for visualization purpose.

Usage

```
graph.viz(model, graph.title = "Computation graph",
  graph.title.font.name = "Helvetica", graph.title.font.size = 30,
  graph.width.px = 500, graph.height.px = 500)
```

Arguments

model	a string representing the path to a file containing the JSon of a model dump or the actual model dump.
graph.title	a string displayed on top of the viz.
graph.title.font.name	a string representing the font to use for the title.
graph.title.font.size	a numeric representing the size of the font to use for the title.
graph.width.px	a numeric representing the size (width) of the graph. In pixels
graph.height.px	a numeric representing the size (height) of the graph. In pixels

Value

a graph object ready to be displayed with the print function.

is.mx.context *Check if the type is mxnet context.*

Description

Check if the type is mxnet context.

Usage

```
is.mx.context(x)
```

Value

Logical indicator

is.mx.dataiter *Judge if an object is mx.dataiter*

Description

Judge if an object is mx.dataiter

Usage

```
is.mx.dataiter(x)
```

Value

Logical indicator

is.mx.ndarray *Check if src.array is mx.ndarray*

Description

Check if src.array is mx.ndarray

Usage

```
is.mx.ndarray(src.array)
```

Value

Logical indicator

Examples

```
mat = mx.nd.array(1:10)
is.mx.ndarray(mat)
mat2 = 1:10
is.mx.ndarray(mat2)
```

is.mx.symbol *Judge if an object is mx.symbol*

Description

Judge if an object is mx.symbol

Usage

```
is.mx.symbol(x)
```

Value

Logical indicator

is.num.in.vect *Top-k accuracy metric for classification*

Description

Top-k accuracy metric for classification

Usage

```
is.num.in.vect(vect, num)
```

length.MXNDArray *Length operator overload of mx.ndarray*

Description

Length operator overload of mx.ndarray

Usage

```
## S3 method for class 'MXNDArray'
length(nd)
```

Arguments

nd	The mx.ndarray

mx.apply *Apply symbol to the inputs.*

Description

Apply symbol to the inputs.

Usage

```
mx.apply(x, ...)
```

Arguments

x	The symbol to be applied
kwargs	The keyword arguments to the symbol

mx.callback.log.train.metric *Log training metric each period*

Description

Log training metric each period

Usage

```
mx.callback.log.train.metric(period, logger = NULL)
```

mx.callback.save.checkpoint *Save checkpoint to files each period iteration.*

Description

Save checkpoint to files each period iteration.

Usage

```
mx.callback.save.checkpoint(prefix, period = 1)
```

Arguments

prefix	The prefix of the model checkpoint.

mx.cpu *Create a mxnet CPU context.*

Description

Create a mxnet CPU context.

Arguments

dev.id	optional, default=0 The device ID, this is meaningless for CPU, included for interface compatiblity.

Value

The CPU context.

mx.ctx.default *Set/Get default context for array creation.*

Description

Set/Get default context for array creation.

Usage

```
mx.ctx.default(new = NULL)
```

Arguments

new,	optional takes mx.cpu() or mx.gpu(id), new default ctx.

Value

The default context.

mx.exec.backward *Peform an backward on the executors This function will MUTATE the state of exec*

Description

Peform an backward on the executors This function will MUTATE the state of exec

Usage

```
mx.exec.backward(exec, ...)
```

`mx.exec.forward` *Peform an forward on the executors This function will MUTATE the state of exec*

Description

Peform an forward on the executors This function will MUTATE the state of exec

Usage

```
mx.exec.forward(exec, is.train = TRUE)
```

`mx.exec.update.arg.arrays` *Update the executors with new arrays This function will MUTATE the state of exec*

Description

Update the executors with new arrays This function will MUTATE the state of exec

Usage

```
mx.exec.update.arg.arrays(exec, arg.arrays, match.name = FALSE,
  skip.null = FALSE)
```

`mx.exec.update.aux.arrays` *Update the executors with new arrays This function will MUTATE the state of exec*

Description

Update the executors with new arrays This function will MUTATE the state of exec

Usage

```
mx.exec.update.aux.arrays(exec, arg.arrays, match.name = FALSE,
  skip.null = FALSE)
```

`mx.exec.update.grad.arrays` *Update the executors with new arrays This function will MUTATE the state of exec*

Description

Update the executors with new arrays This function will MUTATE the state of exec

Usage

```
mx.exec.update.grad.arrays(exec, arg.arrays, match.name = FALSE,
  skip.null = FALSE)
```

`mx.gpu` *Create a mxnet GPU context.*

Description

Create a mxnet GPU context.

Arguments

`dev.id`	optional, default=0 The GPU device ID, starts from 0.

Value

The GPU context.

`mx.gru` *Training GRU Unrolled Model*

Description

Training GRU Unrolled Model

Usage

```
mx.gru(train.data, eval.data = NULL, num.gru.layer, seq.len, num.hidden,
  num.embed, num.label, batch.size, input.size, ctx = mx.ctx.default(),
  num.round = 10, update.period = 1, initializer = mx.init.uniform(0.01),
  dropout = 0, optimizer = "sgd", ...)
```

Arguments

train.data	mx.io.DataIter or list(data=R.array, label=R.array) The Training set.
eval.data	mx.io.DataIter or list(data=R.array, label=R.array), optional The validation set used for validation evaluation during the progress.
num.gru.layer	integer The number of the layer of gru.
seq.len	integer The length of the input sequence.
num.hidden	integer The number of hidden nodes.
num.embed	integer The output dim of embedding.
num.label	integer The number of labels.
batch.size	integer The batch size used for R array training.
input.size	integer The input dim of one-hot encoding of embedding
ctx	mx.context, optional The device used to perform training.
num.round	integer, default=10 The number of iterations over training data to train the model.
update.period	integer, default=1 The number of iterations to update parameters during training period.
initializer	initializer object. default=mx.init.uniform(0.01) The initialization scheme for parameters.
dropout	float, default=0 A number in [0,1) containing the dropout ratio from the last hidden layer to the output layer.
optimizer	string, default="sgd" The optimization method.
...	other parameters passing to mx.gru/.

Value

model A trained gru unrolled model.

---

mx.gru.forward *Using forward function to predict in gru inference model*

Description

Using forward function to predict in gru inference model

Usage

```
mx.gru.forward(model, input.data, new.seq = FALSE)
```

Arguments

model	gru model A gru inference model
input.data,	array.matrix The input data for forward function
new.seq	boolean, default=FALSE Whether the input is the start of a new sequence

Value

result A list(prob=prob, model=model) containing the result probability of each label and the model.

mx.gru.inference *Create a GRU Inference Model*

Description

Create a GRU Inference Model

Usage

```
mx.gru.inference(num.gru.layer, input.size, num.hidden, num.embed, num.label,
  batch.size = 1, arg.params, ctx = mx.cpu(), dropout = 0)
```

Arguments

num.gru.layer	integer The number of the layer of gru.
input.size	integer The input dim of one-hot encoding of embedding
num.hidden	integer The number of hidden nodes.
num.embed	integer The output dim of embedding.
num.label	integer The number of labels.
batch.size	integer, default=1 The batch size used for R array training.
arg.params	list The batch size used for R array training.
ctx	mx.context, optional Model parameter, list of name to NDArray of net's weights.
dropout	float, default=0 A number in [0,1) containing the dropout ratio from the last hidden layer to the output layer.

Value

model list(rnn.exec=integer, symbol=mxnet symbol, num.rnn.layer=integer, num.hidden=integer, seq.len=integer, batch.size=integer, num.embed=integer) A gru inference model.

mx.init.create *Create initialization of argument like arg.array*

Description

Create initialization of argument like arg.array

Usage

```
mx.init.create(initializer, shape.array, ctx, skip.unknown = TRUE)
```

Arguments

initializer	The initializer.
shape.array	named-list The shape of the weights
ctx	mx.context The context of the weights
skip.unknown	Whether skip the unknown weight types

`mx.init.internal.default` *Internal default value initialization scheme.*

Description

Internal default value initialization scheme.

Usage

```
mx.init.internal.default(name, shape, ctx, allow.unknown = FALSE)
```

Arguments

name	the name of the variable.
shape	the shape of the array to be generated.

`mx.init.normal` *Create a initializer that initialize the weight with normal(0, sd)*

Description

Create a initializer that initialize the weight with normal(0, sd)

Usage

```
mx.init.normal(sd)
```

Arguments

sd	The standard deviation of normal distribution

`mx.init.uniform` *Create a initializer that initialize the weight with uniform [-scale, scale]*

Description

Create a initializer that initialize the weight with uniform [-scale, scale]

Usage

```
mx.init.uniform(scale)
```

Arguments

scale	The scale of uniform distribution

mx.init.Xavier *Xavier initializer*

Description

Create a initializer which initialize weight with Xavier or similar initialization scheme.

Usage

```
mx.init.Xavier(rnd_type = "uniform", factor_type = "avg", magnitude = 3)
```

Arguments

rnd_type	A string of character indicating the type of distribution from which the weights are initialized.
factor_type	A string of character.
magnitude	A numeric number indicating the scale of random number range.

mx.io.arrayiter *Create MXDataIter compatible iterator from R's array*

Description

Create MXDataIter compatible iterator from R's array

Usage

```
mx.io.arrayiter(data, label, batch.size = 128, shuffle = FALSE)
```

Arguments

data	The data array.
label	The label array.
batch.size	The batch size used to pack the array.
shuffle	Whether shuffle the data

mx.io.CSVIter *Create iterator for dataset in csv.*

Description

Create iterator for dataset in csv.

Usage

```
mx.io.CSVIter(...)
```

Arguments

data.csv	string, required Dataset Param: Data csv path.
data.shape	Shape(tuple), required Dataset Param: Shape of the data.
label.csv	string, optional, default='NULL' Dataset Param: Label csv path. If is NULL, all labels will be returned as 0
label.shape	Shape(tuple), optional, default=(1,) Dataset Param: Shape of the label.

Value

iter The result mx.dataiter

mx.io.extract *Extract a certain field from DataIter.*

Description

Extract a certain field from DataIter.

Usage

```
mx.io.extract(iter, field)
```

`mx.io.ImageRecordIter` *Create iterator for dataset packed in recordio.*

Description

Create iterator for dataset packed in recordio.

Usage

```
mx.io.ImageRecordIter(...)
```

Arguments

`path.imglist`	string, optional, default='' Dataset Param: Path to image list.
`path.imgrec`	string, optional, default='./data/imgrec.rec' Dataset Param: Path to image record file.
`aug.seq`	string, optional, default='aug_default' Augmentation Param: the augmenter names to represent sequence of augmenters to be applied, seperated by comma. Additional keyword parameters will be seen by these augmenters.
`label.width`	int, optional, default='1' Dataset Param: How many labels for an image.
`data.shape`	Shape(tuple), required Dataset Param: Shape of each instance generated by the DataIter.
`preprocess.threads`	int, optional, default='4' Backend Param: Number of thread to do preprocessing.
`verbose`	boolean, optional, default=True Auxiliary Param: Whether to output parser information.
`num.parts`	int, optional, default='1' partition the data into multiple parts
`part.index`	int, optional, default='0' the index of the part will read
`shuffle`	boolean, optional, default=False Augmentation Param: Whether to shuffle data.
`seed`	int, optional, default='0' Augmentation Param: Random Seed.
`batch.size`	int (non-negative), required Batch Param: Batch size.
`round.batch`	boolean, optional, default=True Batch Param: Use round robin to handle overflow batch.
`prefetch.buffer`	, optional, default=4 Backend Param: Number of prefetched parameters
`rand.crop`	boolean, optional, default=False Augmentation Param: Whether to random crop on the image
`crop.y.start`	int, optional, default='-1' Augmentation Param: Where to nonrandom crop on y.
`crop.x.start`	int, optional, default='-1' Augmentation Param: Where to nonrandom crop on x.
`max.rotate.angle`	int, optional, default='0' Augmentation Param: rotated randomly in [-max_rotate_angle, max_rotate_angle].

`max.aspect.ratio`
: float, optional, default=0 Augmentation Param: denotes the max ratio of random aspect ratio augmentation.

`max.shear.ratio`
: float, optional, default=0 Augmentation Param: denotes the max random shearing ratio.

`max.crop.size`
: int, optional, default='-1' Augmentation Param: Maximum crop size.

`min.crop.size`
: int, optional, default='-1' Augmentation Param: Minimum crop size.

`max.random.scale`
: float, optional, default=1 Augmentation Param: Maxmum scale ratio.

`min.random.scale`
: float, optional, default=1 Augmentation Param: Minimum scale ratio.

`max.img.size`
: float, optional, default=1e+10 Augmentation Param: Maxmum image size after resizing.

`min.img.size`
: float, optional, default=0 Augmentation Param: Minimum image size after resizing.

`random.h`
: int, optional, default='0' Augmentation Param: Maximum value of H channel in HSL color space.

`random.s`
: int, optional, default='0' Augmentation Param: Maximum value of S channel in HSL color space.

`random.l`
: int, optional, default='0' Augmentation Param: Maximum value of L channel in HSL color space.

`rotate`
: int, optional, default='-1' Augmentation Param: Rotate angle.

`fill.value`
: int, optional, default='255' Augmentation Param: Maximum value of illumination variation.

`data.shape`
: Shape(tuple), required Dataset Param: Shape of each instance generated by the DataIter.

`inter.method`
: int, optional, default='1' Augmentation Param: 0-NN 1-bilinear 2-cubic 3-area 4-lanczos4 9-auto 10-rand.

`pad`
: int, optional, default='0' Augmentation Param: Padding size.

`mirror`
: boolean, optional, default=False Augmentation Param: Whether to mirror the image.

`rand.mirror`
: boolean, optional, default=False Augmentation Param: Whether to mirror the image randomly.

`mean.img`
: string, optional, default='' Augmentation Param: Mean Image to be subtracted.

`mean.r`
: float, optional, default=0 Augmentation Param: Mean value on R channel.

`mean.g`
: float, optional, default=0 Augmentation Param: Mean value on G channel.

`mean.b`
: float, optional, default=0 Augmentation Param: Mean value on B channel.

`mean.a`
: float, optional, default=0 Augmentation Param: Mean value on Alpha channel.

`scale`
: float, optional, default=1 Augmentation Param: Scale in color space.

`max.random.contrast`
: float, optional, default=0 Augmentation Param: Maximum ratio of contrast variation.

`max.random.illumination`
: float, optional, default=0 Augmentation Param: Maximum value of illumination variation.

Value

iter The result mx.dataiter

`mx.io.MNISTIter` *Create iterator for MNIST hand-written digit number recognition dataset.*

Description

Create iterator for MNIST hand-written digit number recognition dataset.

Usage

```
mx.io.MNISTIter(...)
```

Arguments

`image`	string, optional, default='./train-images-idx3-ubyte' Dataset Param: Mnist image path.
`label`	string, optional, default='./train-labels-idx1-ubyte' Dataset Param: Mnist label path.
`batch.size`	int, optional, default='128' Batch Param: Batch Size.
`shuffle`	boolean, optional, default=True Augmentation Param: Whether to shuffle data.
`flat`	boolean, optional, default=False Augmentation Param: Whether to flat the data into 1D.
`seed`	int, optional, default='0' Augmentation Param: Random Seed.
`silent`	boolean, optional, default=False Auxiliary Param: Whether to print out data info.
`num.parts`	int, optional, default='1' partition the data into multiple parts
`part.index`	int, optional, default='0' the index of the part will read
`prefetch.buffer`	, optional, default=4 Backend Param: Number of prefetched parameters

Value

iter The result mx.dataiter

`mx.kv.create` *Create a mxnet KVStore.*

Description

Create a mxnet KVStore.

Arguments

`type`	string(default="local") The type of kvstore.

Value

The kvstore.

`mx.lr_scheduler.FactorScheduler`

Learning rate scheduler. Reduction based on a factor value.

Description

Learning rate scheduler. Reduction based on a factor value.

Usage

```
mx.lr_scheduler.FactorScheduler(step, factor_val, stop_factor_lr = 1e-08,
  verbose = TRUE)
```

Arguments

`step`	(integer) Schedule learning rate after n updates
`factor`	(double) The factor for reducing the learning rate

Value

scheduler function

`mx.lr_scheduler.MultiFactorScheduler`

Multifactor learning rate scheduler. Reduction based on a factor value at different steps.

Description

Multifactor learning rate scheduler. Reduction based on a factor value at different steps.

Usage

```
mx.lr_scheduler.MultiFactorScheduler(step, factor_val, stop_factor_lr = 1e-08,
  verbose = TRUE)
```

Arguments

`step`	(array of integer) Schedule learning rate after n updates
`factor`	(double) The factor for reducing the learning rate

Value

scheduler function

`mx.lstm` *Training LSTM Unrolled Model*

Description

Training LSTM Unrolled Model

Usage

```
mx.lstm(train.data, eval.data = NULL, num.lstm.layer, seq.len, num.hidden,
  num.embed, num.label, batch.size, input.size, ctx = mx.ctx.default(),
  num.round = 10, update.period = 1, initializer = mx.init.uniform(0.01),
  dropout = 0, optimizer = "sgd", ...)
```

Arguments

`train.data`	mx.io.DataIter or list(data=R.array, label=R.array) The Training set.
`eval.data`	mx.io.DataIter or list(data=R.array, label=R.array), optional The validation set used for validation evaluation during the progress.
`num.lstm.layer`	integer The number of the layer of lstm.
`seq.len`	integer The length of the input sequence.
`num.hidden`	integer The number of hidden nodes.
`num.embed`	integer The output dim of embedding.
`num.label`	integer The number of labels.
`batch.size`	integer The batch size used for R array training.
`input.size`	integer The input dim of one-hot encoding of embedding
`ctx`	mx.context, optional The device used to perform training.
`num.round`	integer, default=10 The number of iterations over training data to train the model.
`update.period`	integer, default=1 The number of iterations to update parameters during training period.
`initializer`	initializer object. default=mx.init.uniform(0.01) The initialization scheme for parameters.
`dropout`	float, default=0 A number in [0,1) containing the dropout ratio from the last hidden layer to the output layer.
`optimizer`	string, default="sgd" The optimization method.
`...`	other parameters passing to `mx.lstm`/.

Value

model A trained lstm unrolled model.

mx.lstm.forward *Using forward function to predict in lstm inference model*

Description

Using forward function to predict in lstm inference model

Usage

```
mx.lstm.forward(model, input.data, new.seq = FALSE)
```

Arguments

model	lstm model A Lstm inference model
input.data,	array.matrix The input data for forward function
new.seq	boolean, default=FALSE Whether the input is the start of a new sequence

Value

result A list(prob=prob, model=model) containing the result probability of each label and the model.

mx.lstm.inference *Create a LSTM Inference Model*

Description

Create a LSTM Inference Model

Usage

```
mx.lstm.inference(num.lstm.layer, input.size, num.hidden, num.embed, num.label,
  batch.size = 1, arg.params, ctx = mx.cpu(), dropout = 0)
```

Arguments

num.lstm.layer	integer The number of the layer of lstm.
input.size	integer The input dim of one-hot encoding of embedding
num.hidden	integer The number of hidden nodes.
num.embed	integer The output dim of embedding.
num.label	integer The number of labels.
batch.size	integer, default=1 The batch size used for R array training.
arg.params	list The batch size used for R array training.
ctx	mx.context, optional Model parameter, list of name to NDArray of net's weights.
dropout	float, default=0 A number in [0,1) containing the dropout ratio from the last hidden layer to the output layer.

Value

model list(rnn.exec=integer, symbol=mxnet symbol, num.rnn.layer=integer, num.hidden=integer, seq.len=integer, batch.size=integer, num.embed=integer) A lstm inference model.

mx.metric.accuracy *Accuracy metric for classification*

Description

Accuracy metric for classification

Usage

```
mx.metric.accuracy
```

Format

An object of class `mx.metric` of length 3.

mx.metric.custom *Helper function to create a customized metric*

Description

Helper function to create a customized metric

Usage

```
mx.metric.custom(name, feval)
```

mx.metric.mae *MAE (Mean Absolute Error) metric for regression*

Description

MAE (Mean Absolute Error) metric for regression

Usage

```
mx.metric.mae
```

Format

An object of class `mx.metric` of length 3.

mx.metric.rmse *RMSE (Root Mean Squared Error) metric for regression*

Description

RMSE (Root Mean Squared Error) metric for regression

Usage

```
mx.metric.rmse
```

Format

An object of class `mx.metric` of length 3.

mx.metric.rmsle *RMSLE (Root Mean Squared Logarithmic Error) metric for regression*

Description

RMSLE (Root Mean Squared Logarithmic Error) metric for regression

Usage

```
mx.metric.rmsle
```

Format

An object of class `mx.metric` of length 3.

mx.mlp *Convenience interface for multiple layer perceptron*

Description

Convenience interface for multiple layer perceptron

Usage

```
mx.mlp(data, label, hidden_node = 1, out_node, dropout = NULL,
  activation = "tanh", out_activation = "softmax",
  device = mx.ctx.default(), ...)
```

Arguments

data	the input matrix. Only mx.io.DataIter and R array/matrix types supported.
label	the training label. Only R array type supported.
hidden_node	a vector containing number of hidden nodes on each hidden layer as well as the output layer.
out_node	the number of nodes on the output layer.
dropout	a number in [0,1) containing the dropout ratio from the last hidden layer to the output layer.
activation	either a single string or a vector containing the names of the activation functions.
out_activation	a single string containing the name of the output activation function.
device	whether train on cpu (default) or gpu.
...	other parameters passing to mx.model.FeedForward.create/
eval_metric	the evaluation metric/

Examples

```
require(mlbench)
data(Sonar, package="mlbench")
Sonar[,61] = as.numeric(Sonar[,61])-1
train.ind = c(1:50, 100:150)
train.x = data.matrix(Sonar[train.ind, 1:60])
train.y = Sonar[train.ind, 61]
test.x = data.matrix(Sonar[-train.ind, 1:60])
test.y = Sonar[-train.ind, 61]
model = mx.mlp(train.x, train.y, hidden_node = 10, out_node = 2, out_activation = "softmax",
               learning.rate = 0.1)
preds = predict(model, test.x)
```

---

mx.model.FeedForward.create

Create a MXNet Feedforward neural net model with the specified training.

Description

Create a MXNet Feedforward neural net model with the specified training.

Usage

```
mx.model.FeedForward.create(symbol, X, y = NULL, ctx = NULL,
  begin.round = 1, num.round = 10, optimizer = "sgd",
  initializer = mx.init.uniform(0.01), eval.data = NULL,
  eval.metric = NULL, epoch.end.callback = NULL,
  batch.end.callback = NULL, array.batch.size = 128,
  array.layout = "auto", kvstore = "local", verbose = TRUE,
  arg.params = NULL, aux.params = NULL, ...)
```

Arguments

symbol	The symbolic configuration of the neural network.
X	mx.io.DataIter or R array/matrix The training data.
y	R array, optional label of the data This is only used when X is R array.
ctx	mx.context or list of mx.context, optional The devices used to perform training.
begin.round	integer (default=1) The initial iteration over the training data to train the model.
num.round	integer (default=10) The number of iterations over training data to train the model.
optimizer	string, default="sgd" The optimization method.
initializer,	initializer object. default=mx.init.uniform(0.01) The initialization scheme for parameters.
eval.data	mx.io.DataIter or list(data=R.array, label=R.array), optional The validation set used for validation evaluation during the progress
eval.metric	function, optional The evaluation function on the results.
epoch.end.callback	function, optional The callback when iteration ends.
batch.end.callback	function, optional The callback when one mini-batch iteration ends.
array.batch.size	integer (default=128) The batch size used for R array training.
array.layout	can be "auto", "colmajor", "rowmajor", (detault=auto) The layout of array. "rowmajor" is only supported for two dimensional array. For matrix, "rowmajor" means dim(X) = c(nexample, nfeatures), "colmajor" means dim(X) = c(nfeatures, nexample) "auto" will auto detect the layout by match the feature size, and will report error when X is a square matrix to ask user to explicitly specify layout.
kvstore	string (default="local") The parameter synchronization scheme in multiple devices.
verbose	logical (default=TRUE) Specifies whether to print information on the iterations during training.
arg.params	list, optional Model parameter, list of name to NDArray of net's weights.
aux.params	list, optional Model parameter, list of name to NDArray of net's auxiliary states.

Value

model A trained mxnet model.

---

mx.model.load *Load model checkpoint from file.*

Description

Load model checkpoint from file.

Usage

```
mx.model.load(prefix, iteration)
```

Arguments

prefix	string prefix of the model name
iteration	integer Iteration number of model we would like to load.

---

mx.model.save *Save model checkpoint into file.*

Description

Save model checkpoint into file.

Usage

```
mx.model.save(model, prefix, iteration)
```

Arguments

model	The feedforward model to be saved.
prefix	string prefix of the model name
iteration	integer Iteration number of model we would like to load.

---

mx.nd.abs *Take absolute value of the src*

Description

Take absolute value of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

`mx.nd.argmax.channel` *Take argmax indices of each channel of the src.The result will be ndarray of shape (num_channel,) on the same device.*

Description

Take argmax indices of each channel of the src.The result will be ndarray of shape (num_channel,) on the same device.

Arguments

`src`	NDArray Source input to the function

Value

out The result mx.ndarray

`mx.nd.array` *Create a new* `mx.ndarray` *that copies the content from src on ctx.*

Description

Create a new `mx.ndarray` that copies the content from src on ctx.

Usage

```
mx.nd.array(src.array, ctx = NULL)
```

Arguments

`src.array`	Source array data of class `array`, `vector` or `matrix`.
`ctx`	optional The context device of the array. mx.ctx.default() will be used in default.

Value

An `mx.ndarray`

An Rcpp_MXNDArray object

Examples

```
mat = mx.nd.array(x)
mat = 1 - mat + (2 * mat)/(mat + 0.5)
as.array(mat)
```

`mx.nd.batch.dot` *Calculate batched dot product of two matrices. (batch, M, K) batch_dot (batch, K, N) –> (batch, M, N)*

Description

Calculate batched dot product of two matrices. (batch, M, K) batch_dot (batch, K, N) –> (batch, M, N)

Arguments

`lhs`	NDArray Left operand to the function
`rhs`	NDArray Right operand to the function

Value

out The result mx.ndarray

`mx.nd.broadcast.axis` *Broadcast data in the given axis to the given size. The original size of the broadcasting axis must be 1.*

Description

Broadcast data in the given axis to the given size. The original size of the broadcasting axis must be 1.

Arguments

`src`	NDArray Source input to the function
`axis`	Shape(tuple), optional, default=() The axes to perform the broadcasting.
`size`	Shape(tuple), optional, default=() Target sizes of the broadcasting axes.

Value

out The result mx.ndarray

mx.nd.broadcast.div *lhs divide rhs with broadcast*

Description

lhs divide rhs with broadcast

Arguments

lhs	NDArray Left operand to the function
rhs	NDArray Right operand to the function

Value

out The result mx.ndarray

mx.nd.broadcast.minus *lhs minus rhs with broadcast*

Description

lhs minus rhs with broadcast

Arguments

lhs	NDArray Left operand to the function
rhs	NDArray Right operand to the function

Value

out The result mx.ndarray

mx.nd.broadcast.mul *lhs multiple rhs with broadcast*

Description

lhs multiple rhs with broadcast

Arguments

lhs	NDArray Left operand to the function
rhs	NDArray Right operand to the function

Value

out The result mx.ndarray

mx.nd.broadcast.plus *lhs add rhs with broadcast*

Description

lhs add rhs with broadcast

Arguments

lhs	NDArray Left operand to the function
rhs	NDArray Right operand to the function

Value

out The result mx.ndarray

mx.nd.broadcast.power *lhs power rhs with broadcast*

Description

lhs power rhs with broadcast

Arguments

lhs	NDArray Left operand to the function
rhs	NDArray Right operand to the function

Value

out The result mx.ndarray

mx.nd.broadcast.to *Broadcast data to the target shape. The original size of the broadcasting axis must be 1.*

Description

Broadcast data to the target shape. The original size of the broadcasting axis must be 1.

Arguments

src	NDArray Source input to the function
shape	Shape(tuple), optional, default=() The shape of the desired array. We can set the dim to zero if it's same as the original. E.g 'A = broadcast_to(B, shape=(10, 0, 0))' has the same meaning as 'A = broadcast_axis(B, axis=0, size=10)'.

Value

out The result mx.ndarray

`mx.nd.ceil` *Take ceil value of the src*

Description

Take ceil value of the src

Arguments

`src`	NDArray Source input to the function

Value

out The result mx.ndarray

`mx.nd.choose.element.0index` *Choose one element from each line(row for python, column for R/Julia) in lhs according to index indicated by rhs. This function assume rhs uses 0-based index.*

Description

Choose one element from each line(row for python, column for R/Julia) in lhs according to index indicated by rhs. This function assume rhs uses 0-based index.

Arguments

`lhs`	NDArray Left operand to the function.
`rhs`	NDArray Right operand to the function.

Value

out The result mx.ndarray

`mx.nd.clip` *Clip ndarray elements to range (a_min, a_max)*

Description

Clip ndarray elements to range (a_min, a_max)

Arguments

`src`	NDArray Source input
`a.min`	real_t Minimum value
`a.max`	real_t Maximum value

Value

out The result mx.ndarray

mx.nd.copyto *Generate an mx.ndarray object on ctx, with data copied from src*

Description

Generate an mx.ndarray object on ctx, with data copied from src

Usage

```
mx.nd.copyto(src, ctx)
```

Arguments

src	The source mx.ndarray object.
ctx	The target context.

mx.nd.cos *Take cos of the src*

Description

Take cos of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

mx.nd.crop *Crop the input matrix and return a new one*

Description

Crop the input matrix and return a new one

Arguments

src	NDArray Source input to the function
begin	Shape(tuple), required starting coordinates
end	Shape(tuple), required ending coordinates

Value

out The result mx.ndarray

mx.nd.dot *Calculate dot product of two matrices or two vectors*

Description

Calculate dot product of two matrices or two vectors

Arguments

lhs	NDArray Left operand to the function
rhs	NDArray Right operand to the function

Value

out The result mx.ndarray

mx.nd.exp *Take exp of the src*

Description

Take exp of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

mx.nd.expand.dims *Expand the shape of array by inserting a new axis.*

Description

Expand the shape of array by inserting a new axis.

Arguments

src	NDArray Source input to the function
axis	int (non-negative), required Position (amongst axes) where new axis is to be inserted.

Value

out The result mx.ndarray

`mx.nd.fill.element.0index` *Fill one element of each line(row for python, column for R/Julia) in lhs according to index indicated by rhs and values indicated by mhs. This function assume rhs uses 0-based index.*

Description

Fill one element of each line(row for python, column for R/Julia) in lhs according to index indicated by rhs and values indicated by mhs. This function assume rhs uses 0-based index.

Arguments

`lhs`	NDArray Left operand to the function.
`mhs`	NDArray Middle operand to the function.
`rhs`	NDArray Right operand to the function.

Value

out The result mx.ndarray

`mx.nd.flip` *Flip the input matrix along axis and return a new one*

Description

Flip the input matrix along axis and return a new one

Arguments

`src`	NDArray Source input to the function
`axis`	int, required The dimension to flip

Value

out The result mx.ndarray

`mx.nd.floor` *Take floor value of the src*

Description

Take floor value of the src

Arguments

`src`	NDArray Source input to the function

Value

out The result mx.ndarray

mx.nd.load *Load an mx.nd.array object on disk*

Description

Load an mx.nd.array object on disk

Usage

```
mx.nd.load(filename)
```

Arguments

filename	the filename (including the path)

Examples

```
mat = mx.nd.array(1:3)
mx.nd.save(mat, 'temp.mat')
mat2 = mx.nd.load('temp.mat')
as.array(mat)
as.array(mat2)
```

mx.nd.log *Take log of the src*

Description

Take log of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

`mx.nd.max` *Take max of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

Take max of the src in the given axis and returns a NDArray. Follows numpy semantics.

Arguments

`src`	NDArray Source input to the function
`axis`	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
`keepdims`	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.

Value

out The result mx.ndarray

`mx.nd.max.axis` *(Depreciated! Use max instead!) Take max of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

(Depreciated! Use max instead!) Take max of the src in the given axis and returns a NDArray. Follows numpy semantics.

Arguments

`src`	NDArray Source input to the function
`axis`	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
`keepdims`	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.

Value

out The result mx.ndarray

`mx.nd.min` *Take min of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

Take min of the src in the given axis and returns a NDArray. Follows numpy semantics.

Arguments

`src`	NDArray Source input to the function
`axis`	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
`keepdims`	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.

Value

out The result mx.ndarray

`mx.nd.min.axis` *(Depreciated! Use min instead!) Take min of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

(Depreciated! Use min instead!) Take min of the src in the given axis and returns a NDArray. Follows numpy semantics.

Arguments

`src`	NDArray Source input to the function
`axis`	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
`keepdims`	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.

Value

out The result mx.ndarray

`mx.nd.norm` *Take L2 norm of the src.The result will be ndarray of shape (1,) on the same device.*

Description

Take L2 norm of the src.The result will be ndarray of shape (1,) on the same device.

Arguments

`src`	NDArray Source input to the function

Value

out The result mx.ndarray

`mx.nd.ones` *Generate an mx.ndarray object with ones*

Description

Generate an mx.ndarray object with ones

Usage

```
mx.nd.ones(shape, ctx = NULL)
```

Arguments

`shape`	the dimension of the `mx.ndarray`
`ctx`	optional The context device of the array. mx.ctx.default() will be used in default.

Examples

```
mat = mx.nd.ones(10)
as.array(mat)
mat2 = mx.nd.ones(c(5,5))
as.array(mat)
mat3 = mx.nd.ones(c(3,3,3))
as.array(mat3)
```

mx.nd.round *Take round value of the src*

Description

Take round value of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

mx.nd.rsqrt *Take rsqrt of the src*

Description

Take rsqrt of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

mx.nd.save *Save an mx.nd.array object*

Description

Save an mx.nd.array object

Usage

```
mx.nd.save(ndarray, filename)
```

Arguments

ndarray	the mx.nd.array object
filename	the filename (including the path)

Examples

```
mat = mx.nd.array(1:3)
mx.nd.save(mat, 'temp.mat')
mat2 = mx.nd.load('temp.mat')
as.array(mat)
as.array(mat2[[1]])
```

---

mx.nd.sign *Take sign value of the src*

Description

Take sign value of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

---

mx.nd.sin *Take sin of the src*

Description

Take sin of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

`mx.nd.slice.axis` *Slice the input along certain axis and return a sliced array.*

Description

Slice the input along certain axis and return a sliced array.

Arguments

`src`	NDArray Source input to the function
`axis`	int, required The axis to be sliced
`begin`	int, required The beginning index to be sliced
`end`	int, required The end index to be sliced

Value

out The result mx.ndarray

`mx.nd.smooth.l1` *Calculate Smooth L1 Loss(lhs, scalar)*

Description

Calculate Smooth L1 Loss(lhs, scalar)

Arguments

`src`	NDArray Source input to the function

Value

out The result mx.ndarray

`mx.nd.softmax.cross.entropy` *Calculate cross_entropy(lhs, one_hot(rhs))*

Description

Calculate cross_entropy(lhs, one_hot(rhs))

Arguments

`lhs`	NDArray Left operand to the function
`rhs`	NDArray Right operand to the function

Value

out The result mx.ndarray

mx.nd.sqrt *Take sqrt of the src*

Description

Take sqrt of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

mx.nd.square *Take square of the src*

Description

Take square of the src

Arguments

src	NDArray Source input to the function

Value

out The result mx.ndarray

mx.nd.sum *Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.

Arguments

src	NDArray Source input to the function
axis	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
keepdims	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.

Value

out The result mx.ndarray

`mx.nd.sum.axis` *(Depreciated! Use sum instead!) Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

(Depreciated! Use sum instead!) Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.

Arguments

`src`	NDArray Source input to the function
`axis`	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
`keepdims`	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.

Value

out The result mx.ndarray

`mx.nd.transpose` *Transpose the input matrix and return a new one*

Description

Transpose the input matrix and return a new one

Arguments

`src`	NDArray Source input to the function
`axes`	Shape(tuple), optional, default=() Target axis order. By default the axes will be inverted.

Value

out The result mx.ndarray

mx.nd.zeros *Generate an mx.nd.array object with zeros*

Description

Generate an mx.nd.array object with zeros

Usage

```
mx.nd.zeros(shape, ctx = NULL)
```

Arguments

shape	the dimension of the mx.nd.array
ctx	optional The context device of the array. mx.ctx.default() will be used in default.

Examples

```
mat = mx.nd.zeros(10)
as.array(mat)
mat2 = mx.nd.zeros(c(5,5))
as.array(mat)
mat3 = mx.nd.zeroes(c(3,3,3))
as.array(mat3)
```

mx.opt.adadelta *Create an AdaDelta optimizer with respective parameters.*

Description

AdaDelta optimizer as described in Zeiler, M. D. (2012). *ADADELTA: An adaptive learning rate method.* http://arxiv.org/abs/1212.5701

Usage

```
mx.opt.adadelta(rho = 0.9, epsilon = 1e-05, wd = 0, rescale.grad = 1,
  clip_gradient = NULL)
```

Arguments

rho	float, default=0.90 Decay rate for both squared gradients and delta x.
epsilon	float, default=1e-5 The constant as described in the thesis.
wd	float, default=0.0 L2 regularization coefficient add to all the weights.
rescale.grad	float, default=1.0 rescaling factor of gradient.
clip_gradient	float, optional clip gradient in range [-clip_gradient, clip_gradient].

`mx.opt.adagrad` *Create an AdaGrad optimizer with respective parameters. AdaGrad optimizer of Duchi et al., 2011,*

Description

This code follows the version in http://arxiv.org/pdf/1212.5701v1.pdf Eq(5) by Matthew D. Zeiler, 2012. AdaGrad will help the network to converge faster in some cases.

Usage

```
mx.opt.adagrad(learning.rate = 0.05, epsilon = 1e-08, wd = 0,
  rescale.grad = 1, clip_gradient = NULL, lr_scheduler = NULL)
```

Arguments

`learning.rate`	float, default=0.05 Step size.
`epsilon`	float, default=1e-8
`wd`	float, default=0.0 L2 regularization coefficient add to all the weights.
`rescale.grad`	float, default=1.0 rescaling factor of gradient.
`clip_gradient`	float, optional clip gradient in range [-clip_gradient, clip_gradient].
`lr_scheduler`	function, optional The learning rate scheduler.

`mx.opt.adam` *Create an Adam optimizer with respective parameters. Adam optimizer as described in [King2014].*

Description

[King2014] Diederik Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization, http://arxiv.org/abs/1412.6980

Usage

```
mx.opt.adam(learning.rate = 0.001, beta1 = 0.9, beta2 = 0.999,
  epsilon = 1e-08, wd = 0, rescale.grad = 1, clip_gradient = NULL,
  lr_scheduler = NULL)
```

Arguments

`learning.rate`	float, default=0.001 Step size.
`beta1`	float, default=0.9 Exponential decay rate for the first moment estimates.
`beta2`	float, default=0.999 Exponential decay rate for the second moment estimates.
`epsilon`	float, default=1e-8
`wd`	float, default=0.0 L2 regularization coefficient add to all the weights.
`rescale.grad`	float, default=1.0 rescaling factor of gradient.
`clip_gradient`	float, optional clip gradient in range [-clip_gradient, clip_gradient].
`lr_scheduler`	function, optional The learning rate scheduler.

`mx.opt.create` *Create an optimizer by name and parameters*

Description

Create an optimizer by name and parameters

Usage

```
mx.opt.create(name, ...)
```

Arguments

`name`	The name of the optimizer
`...`	Additional arguments

`mx.opt.get.updater` *Get an updater closure that can take list of weight and gradient and return updated list of weight.*

Description

Get an updater closure that can take list of weight and gradient and return updated list of weight.

Usage

```
mx.opt.get.updater(optimizer, weights)
```

Arguments

`optimizer`	The optimizer
`weights`	The weights to be optimized

`mx.opt.rmsprop` *Create an RMSProp optimizer with respective parameters. Reference: Tieleman T, Hinton G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude[J]. COURSERA: Neural Networks for Machine Learning, 2012, 4(2). The code follows: http://arxiv.org/pdf/1308.0850v5.pdf Eq(38) - Eq(45) by Alex Graves, 2013.*

Description

Create an RMSProp optimizer with respective parameters. Reference: Tieleman T, Hinton G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude[J]. COURSERA: Neural Networks for Machine Learning, 2012, 4(2). The code follows: http://arxiv.org/pdf/1308.0850v5.pdf Eq(38) - Eq(45) by Alex Graves, 2013.

Usage

```
mx.opt.rmsprop(learning.rate = 0.002, gamma1 = 0.95, gamma2 = 0.9,
  wd = 0, rescale.grad = 1, clip_gradient = NULL, lr_scheduler = NULL)
```

Arguments

learning.rate	float, default=0.002 Step size.
gamma1	float, default=0.95 decay factor of moving average for gradient, gradient^2.
wd	float, default=0.0 L2 regularization coefficient add to all the weights.
rescale.grad	float, default=1.0 rescaling factor of gradient.
clip_gradient	float, optional clip gradient in range [-clip_gradient, clip_gradient].
lr_scheduler	function, optional The learning rate scheduler.
gamm2	float, default=0.9 "momentum" factor.

---

mx.opt.sgd *Create an SGD optimizer with respective parameters. Perform SGD with momentum update*

Description

Create an SGD optimizer with respective parameters. Perform SGD with momentum update

Usage

```
mx.opt.sgd(learning.rate, momentum = 0, wd = 0, rescale.grad = 1,
  clip_gradient = NULL, lr_scheduler = NULL)
```

---

mx.rnn *Training RNN Unrolled Model*

Description

Training RNN Unrolled Model

Usage

```
mx.rnn(train.data, eval.data = NULL, num.rnn.layer, seq.len, num.hidden,
  num.embed, num.label, batch.size, input.size, ctx = mx.ctx.default(),
  num.round = 10, update.period = 1, initializer = mx.init.uniform(0.01),
  dropout = 0, optimizer = "sgd", batch.norm = FALSE, ...)
```

Arguments

train.data	mx.io.DataIter or list(data=R.array, label=R.array) The Training set.
eval.data	mx.io.DataIter or list(data=R.array, label=R.array), optional The validation set used for validation evaluation during the progress.
num.rnn.layer	integer The number of the layer of rnn.
seq.len	integer The length of the input sequence.
num.hidden	integer The number of hidden nodes.
num.embed	integer The output dim of embedding.
num.label	integer The number of labels.
batch.size	integer The batch size used for R array training.
input.size	integer The input dim of one-hot encoding of embedding
ctx	mx.context, optional The device used to perform training.
num.round	integer, default=10 The number of iterations over training data to train the model.
update.period	integer, default=1 The number of iterations to update parameters during training period.
initializer	initializer object. default=mx.init.uniform(0.01) The initialization scheme for parameters.
dropout	float, default=0 A number in [0,1) containing the dropout ratio from the last hidden layer to the output layer.
optimizer	string, default="sgd" The optimization method.
batch.norm	boolean, default=FALSE Whether to use batch normalization.
...	other parameters passing to mx.rnn/.

Value

model A trained rnn unrolled model.

---

mx.rnn.forward *Using forward function to predict in rnn inference model*

Description

Using forward function to predict in rnn inference model

Usage

```
mx.rnn.forward(model, input.data, new.seq = FALSE)
```

Arguments

model	rnn model A rnn inference model
input.data,	array.matrix The input data for forward function
new.seq	boolean, default=FALSE Whether the input is the start of a new sequence

Value

result A list(prob=prob, model=model) containing the result probability of each label and the model.

mx.rnn.inference *Create a RNN Inference Model*

Description

Create a RNN Inference Model

Usage

```
mx.rnn.inference(num.rnn.layer, input.size, num.hidden, num.embed, num.label,
  batch.size = 1, arg.params, ctx = mx.cpu(), dropout = 0,
  batch.norm = FALSE)
```

Arguments

num.rnn.layer	integer The number of the layer of rnn.
input.size	integer The input dim of one-hot encoding of embedding
num.hidden	integer The number of hidden nodes.
num.embed	integer The output dim of embedding.
num.label	integer The number of labels.
batch.size	integer, default=1 The batch size used for R array training.
arg.params	list The batch size used for R array training.
ctx	mx.context, optional Model parameter, list of name to NDArray of net's weights.
dropout	float, default=0 A number in [0,1) containing the dropout ratio from the last hidden layer to the output layer.
batch.norm	boolean, default=FALSE Whether to use batch normalization.

Value

model list(rnn.exec=integer, symbol=mxnet symbol, num.rnn.layer=integer, num.hidden=integer, seq.len=integer, batch.size=integer, num.embed=integer) A rnn inference model.

mx.rnorm *Generate nomal distribution with mean and sd.*

Description

Generate nomal distribution with mean and sd.

Usage

```
mx.rnorm(shape, mean = 0, sd = 1, ctx = NULL)
```

Arguments

shape	Dimension, The shape(dimension) of the result.
mean	numeric, The mean of distribution.
sd	numeric, The standard deviations.
ctx,	optional The context device of the array. mx.ctx.default() will be used in default.

Examples

```
mx.set.seed(0)
as.array(mx.runif(2))
# 0.5488135 0.5928446
mx.set.seed(0)
as.array(mx.rnorm(2))
# 2.212206 1.163079
```

---

mx.runif *Generate uniform distribution in [low, high) with specified shape.*

Description

Generate uniform distribution in [low, high) with specified shape.

Usage

```
mx.runif(shape, min = 0, max = 1, ctx = NULL)
```

Arguments

shape	Dimension, The shape(dimension) of the result.
min	numeric, The lower bound of distribution.
max	numeric, The upper bound of distribution.
ctx,	optional The context device of the array. mx.ctx.default() will be used in default.

Examples

```
mx.set.seed(0)
as.array(mx.runif(2))
# 0.5488135 0.5928446
mx.set.seed(0)
as.array(mx.rnorm(2))
# 2.212206 1.163079
```

`mx.set.seed` *Set the seed used by mxnet device-specific random number generators.*

Description

Set the seed used by mxnet device-specific random number generators.

Usage

```
mx.set.seed(seed)
```

Arguments

`seed`	the seed value to the device random number generators.

Details

We have a specific reason why `mx.set.seed` is introduced, instead of simply use `set.seed`.

The reason that is that most of mxnet random number generator can run on different devices, such as GPU. We need to use massively parallel PRNG on GPU to get fast random number generations. It can also be quite costly to seed these PRNGs. So we introduced `mx.set.seed` for mxnet specific device random numbers.

Examples

```
mx.set.seed(0)
as.array(mx.runif(2))
# 0.5488135 0.5928446
mx.set.seed(0)
as.array(mx.rnorm(2))
# 2.212206 1.163079
```

`mx.simple.bind` *Simple bind the symbol to executor, with information from input shapes.*

Description

Simple bind the symbol to executor, with information from input shapes.

Usage

```
mx.simple.bind(symbol, ctx, grad.req = "null", ...)
```

`mx.symbol.abs` *Take absolute value of the src*

Description

Take absolute value of the src

Usage

```
mx.symbol.abs(...)
```

Arguments

`src`	Symbol Left symbolic input to the function
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.Activation` *Apply activation function to input.Softmax Activation is only available with CUDNN on GPUand will be computed at each location across channel if input is 4D.*

Description

Apply activation function to input.Softmax Activation is only available with CUDNN on GPUand will be computed at each location across channel if input is 4D.

Usage

```
mx.symbol.Activation(...)
```

Arguments

`data`	Symbol Input data to activation function.
`act.type`	'relu', 'sigmoid', 'softrelu', 'tanh', required Activation function to be applied.
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.BatchNorm` *Apply batch normalization to input.*

Description

Apply batch normalization to input.

Usage

```
mx.symbol.BatchNorm(...)
```

Arguments

`data`	Symbol Input data to batch normalization
`eps`	float, optional, default=0.001 Epsilon to prevent div 0
`momentum`	float, optional, default=0.9 Momentum for moving average
`fix.gamma`	boolean, optional, default=True Fix gamma while training
`use.global.stats`	boolean, optional, default=False Whether use global moving statistics instead of local batch-norm. This will force change batch-norm into a scale shift operator.
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.batch_dot` *Calculate batched dot product of two matrices. (batch, M, K) batch_dot (batch, K, N) –> (batch, M, N)*

Description

Calculate batched dot product of two matrices. (batch, M, K) batch_dot (batch, K, N) –> (batch, M, N)

Usage

```
mx.symbol.batch_dot(...)
```

Arguments

`lhs`	Symbol Left symbolic input to the function
`rhs`	Symbol Right symbolic input to the function
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.BlockGrad *Get output from a symbol and pass 0 gradient back*

Description

Get output from a symbol and pass 0 gradient back

Usage

```
mx.symbol.BlockGrad(...)
```

Arguments

data	Symbol Input data.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.broadcast_axis *Broadcast data in the given axis to the given size. The original size of the broadcasting axis must be 1.*

Description

Broadcast data in the given axis to the given size. The original size of the broadcasting axis must be 1.

Usage

```
mx.symbol.broadcast_axis(...)
```

Arguments

src	Symbol Left symbolic input to the function
axis	Shape(tuple), optional, default=() The axes to perform the broadcasting.
size	Shape(tuple), optional, default=() Target sizes of the broadcasting axes.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.broadcast_div

lhs divide rhs with broadcast

Description

lhs divide rhs with broadcast

Usage

```
mx.symbol.broadcast_div(...)
```

Arguments

lhs	Symbol Left symbolic input to the function
rhs	Symbol Right symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.broadcast_minus

lhs minus rhs with broadcast

Description

lhs minus rhs with broadcast

Usage

```
mx.symbol.broadcast_minus(...)
```

Arguments

lhs	Symbol Left symbolic input to the function
rhs	Symbol Right symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.broadcast_mul

lhs multiple rhs with broadcast

Description

lhs multiple rhs with broadcast

Usage

```
mx.symbol.broadcast_mul(...)
```

Arguments

lhs	Symbol Left symbolic input to the function
rhs	Symbol Right symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.broadcast_plus

lhs add rhs with broadcast

Description

lhs add rhs with broadcast

Usage

```
mx.symbol.broadcast_plus(...)
```

Arguments

lhs	Symbol Left symbolic input to the function
rhs	Symbol Right symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.broadcast_power`

lhs power rhs with broadcast

Description

lhs power rhs with broadcast

Usage

```
mx.symbol.broadcast_power(...)
```

Arguments

`lhs`	Symbol Left symbolic input to the function
`rhs`	Symbol Right symbolic input to the function
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.broadcast_to`

Broadcast data to the target shape. The original size of the broadcasting axis must be 1.

Description

Broadcast data to the target shape. The original size of the broadcasting axis must be 1.

Usage

```
mx.symbol.broadcast_to(...)
```

Arguments

`src`	Symbol Left symbolic input to the function
`shape`	Shape(tuple), optional, default=() The shape of the desired array. We can set the dim to zero if it's same as the original. E.g 'A = broadcast_to(B, shape=(10, 0, 0))' has the same meaning as 'A = broadcast_axis(B, axis=0, size=10)'.
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Cast *Cast array to a different data type.*

Description

Cast array to a different data type.

Usage

```
mx.symbol.Cast(...)
```

Arguments

data	Symbol Input data to cast function.
dtype	'float16', 'float32', 'float64', 'int32', 'uint8', required Target data type.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.ceil *Take ceil value of the src*

Description

Take ceil value of the src

Usage

```
mx.symbol.ceil(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Concat *Perform an feature concat on channel dim (dim 1) over all the inputs.*

Description

Perform an feature concat on channel dim (dim 1) over all the inputs.

Usage

```
mx.symbol.Concat(data, num.args, dim = NULL, name = NULL)
```

Arguments

data	list, required List of tensors to concatenate
num.args	int, required Number of inputs to be concated.
dim	int, optional, default='1' the dimension to be concated.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Convolution *Apply convolution to input then add a bias.*

Description

Apply convolution to input then add a bias.

Usage

```
mx.symbol.Convolution(...)
```

Arguments

data	Symbol Input data to the ConvolutionOp.
weight	Symbol Weight matrix.
bias	Symbol Bias parameter.
kernel	Shape(tuple), required convolution kernel size: (y, x)
stride	Shape(tuple), optional, default=(1,1) convolution stride: (y, x)
dilate	Shape(tuple), optional, default=(1,1) convolution dilate: (y, x)
pad	Shape(tuple), optional, default=(0,0) pad for convolution: (y, x)
num.filter	int (non-negative), required convolution filter(channel) number
num.group	int (non-negative), optional, default=1 Number of groups partition. This option is not supported by CuDNN, you can use SliceChannel to num_group,apply convolution and concat instead to achieve the same need.

workspace	long (non-negative), optional, default=512 Tmp workspace for convolution (MB).
no.bias	boolean, optional, default=False Whether to disable bias parameter.
cudnn.tune	'fastest', 'limited_workspace', 'off',optional, default='limited_workspace' Whether to find convolution algo by running performance test.Leads to higher startup time but may give better speed
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.cos *Take cos of the src*

Description

Take cos of the src

Usage

```
mx.symbol.cos(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.Crop *Crop the 2nd and 3rd dim of input data, with the corresponding size of h_w or with width and height of the second input symbol, i.e., with one input, we need h_w to specify the crop height and width, otherwise the second input symbol's size will be used*

Description

Crop the 2nd and 3rd dim of input data, with the corresponding size of h_w or with width and height of the second input symbol, i.e., with one input, we need h_w to specify the crop height and width, otherwise the second input symbol's size will be used

Usage

```
mx.symbol.Crop(...)
```

Arguments

data	Symbol or Symbol[] Tensor or List of Tensors, the second input will be used as crop_like shape reference
num.args	int, required Number of inputs for crop, if equals one, then we will use the h_wfor crop height and width, else if equals two, then we will use the heightand width of the second input symbol, we name crop_like here
offset	Shape(tuple), optional, default=(0,0) crop offset coordinate: (y, x)
h.w	Shape(tuple), optional, default=(0,0) crop height and weight: (h, w)
center.crop	boolean, optional, default=False If set to true, then it will use be the center_crop,or it will crop using the shape of crop_like
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.Custom *Custom operator implemented in frontend.*

Description

Custom operator implemented in frontend.

Usage

```
mx.symbol.Custom(...)
```

Arguments

op.type	string Type of custom operator. Must be registered first.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Deconvolution

Apply deconvolution to input then add a bias.

Description

Apply deconvolution to input then add a bias.

Usage

```
mx.symbol.Deconvolution(...)
```

Arguments

data	Symbol Input data to the DeconvolutionOp.
weight	Symbol Weight matrix.
bias	Symbol Bias parameter.
kernel	Shape(tuple), required deconvolution kernel size: (y, x)
stride	Shape(tuple), optional, default=(1,1) deconvolution stride: (y, x)
pad	Shape(tuple), optional, default=(0,0) pad for deconvolution: (y, x), a good number is : (kernel-1)/2, if target_shape set, pad will be ignored and will be computed automatically
adj	Shape(tuple), optional, default=(0,0) adjustment for output shape: (y, x), if target_shape set, adj will be ignored and will be computed automatically
target.shape	Shape(tuple), optional, default=(0,0) output shape with targe shape : (y, x)
num.filter	int (non-negative), required deconvolution filter(channel) number
num.group	int (non-negative), optional, default=1 number of groups partition
workspace	long (non-negative), optional, default=512 Tmp workspace for deconvolution (MB)
no.bias	boolean, optional, default=True Whether to disable bias parameter.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.dot *Calculate dot product of two matrices or two vectors*

Description

Calculate dot product of two matrices or two vectors

Usage

```
mx.symbol.dot(...)
```

Arguments

lhs	Symbol Left symbolic input to the function
rhs	Symbol Right symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Dropout *Apply dropout to input*

Description

Apply dropout to input

Usage

```
mx.symbol.Dropout(...)
```

Arguments

data	Symbol Input data to dropout.
p	float, optional, default=0.5 Fraction of the input that gets dropped out at training time
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.ElementWiseSum` *Perform an elementwise sum over all the inputs.*

Description

Perform an elementwise sum over all the inputs.

Usage

```
mx.symbol.ElementWiseSum(...)
```

Arguments

`num.args`	int, required Number of inputs to be summed.
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.Embedding` *Get embedding for one-hot input. A n-dimensional input tensor will be trainsformed into a (n+1)-dimensional tensor, where a new dimension is added for the embedding results.*

Description

Get embedding for one-hot input. A n-dimensional input tensor will be trainsformed into a (n+1)-dimensional tensor, where a new dimension is added for the embedding results.

Usage

```
mx.symbol.Embedding(...)
```

Arguments

`data`	Symbol Input data to the EmbeddingOp.
`weight`	Symbol Enbedding weight matrix.
`input.dim`	int, required input dim of one-hot encoding
`output.dim`	int, required output dim of embedding
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.exp *Take exp of the src*

Description

Take exp of the src

Usage

```
mx.symbol.exp(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.expand_dims *Expand the shape of array by inserting a new axis.*

Description

Expand the shape of array by inserting a new axis.

Usage

```
mx.symbol.expand_dims(...)
```

Arguments

src	Symbol Left symbolic input to the function
axis	int (non-negative), required Position (amongst axes) where new axis is to be inserted.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Flatten *Flatten input*

Description

Flatten input

Usage

```
mx.symbol.Flatten(...)
```

Arguments

data	Symbol Input data to flatten.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.floor *Take floor value of the src*

Description

Take floor value of the src

Usage

```
mx.symbol.floor(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.FullyConnected

Apply matrix multiplication to input then add a bias.

Description

Apply matrix multiplication to input then add a bias.

Usage

```
mx.symbol.FullyConnected(...)
```

Arguments

data	Symbol Input data to the FullyConnectedOp.
weight	Symbol Weight matrix.
bias	Symbol Bias parameter.
num.hidden	int, required Number of hidden nodes of the output.
no.bias	boolean, optional, default=False Whether to disable bias parameter.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Group

Create a symbol that groups symbols together.

Description

Create a symbol that groups symbols together.

Usage

```
mx.symbol.Group(...)
```

Arguments

kwarg	Variable length of symbols or list of symbol.

Value

The result symbol

mx.symbol.IdentityAttachKLSparseReg

Apply a sparse regularization to the output a sigmoid activation function.

Description

Apply a sparse regularization to the output a sigmoid activation function.

Usage

```
mx.symbol.IdentityAttachKLSparseReg(...)
```

Arguments

data	Symbol Input data.
sparseness.target	float, optional, default=0.1 The sparseness target
penalty	float, optional, default=0.001 The tradeoff parameter for the sparseness penalty
momentum	float, optional, default=0.9 The momentum for running average
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.infer.shape *Inference the shape of arguments, outputs, and auxiliary states.*

Description

Inference the shape of arguments, outputs, and auxiliary states.

Usage

```
mx.symbol.infer.shape(symbol, ...)
```

Arguments

symbol	The mx.symbol object

`mx.symbol.L2Normalization` *Set the l2 norm of each instance to a constant.*

Description

Set the l2 norm of each instance to a constant.

Usage

```
mx.symbol.L2Normalization(...)
```

Arguments

`data`	Symbol Input data to the L2NormalizationOp.
`eps`	float, optional, default=1e-10 Epsilon to prevent div 0
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.LeakyReLU` *Apply activation function to input.*

Description

Apply activation function to input.

Usage

```
mx.symbol.LeakyReLU(...)
```

Arguments

`data`	Symbol Input data to activation function.
`act.type`	'elu', 'leaky', 'prelu', 'rrelu',optional, default='leaky' Activation function to be applied.
`slope`	float, optional, default=0.25 Init slope for the activation. (For leaky and elu only)
`lower.bound`	float, optional, default=0.125 Lower bound of random slope. (For rrelu only)
`upper.bound`	float, optional, default=0.334 Upper bound of random slope. (For rrelu only)
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.LinearRegressionOutput

Use linear regression for final output, this is used on final output of a net.

Description

Use linear regression for final output, this is used on final output of a net.

Usage

```
mx.symbol.LinearRegressionOutput(...)
```

Arguments

data	Symbol Input data to function.
label	Symbol Input label to function.
grad.scale	float, optional, default=1 Scale the gradient by a float factor
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.load

Load an mx.symbol object

Description

Load an mx.symbol object

Usage

```
mx.symbol.load(filename)
```

Arguments

filename	the filename (including the path)

Examples

```
data = mx.symbol.Variable('data')
mx.symbol.save(data, 'temp.symbol')
data2 = mx.symbol.load('temp.symbol')
```

`mx.symbol.load.json` *Load an mx.symbol object from a json string*

Description

Load an mx.symbol object from a json string

Arguments

`str`	the json str represent a mx.symbol

`mx.symbol.log` *Take log of the src*

Description

Take log of the src

Usage

```
mx.symbol.log(...)
```

Arguments

`src`	Symbol Left symbolic input to the function
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.LogisticRegressionOutput`
Use Logistic regression for final output, this is used on final output of a net. Logistic regression is suitable for binary classification or probability prediction tasks.

Description

Use Logistic regression for final output, this is used on final output of a net. Logistic regression is suitable for binary classification or probability prediction tasks.

Usage

```
mx.symbol.LogisticRegressionOutput(...)
```

Arguments

data	Symbol Input data to function.
label	Symbol Input label to function.
grad.scale	float, optional, default=1 Scale the gradient by a float factor
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.LRN *Apply convolution to input then add a bias.*

Description

Apply convolution to input then add a bias.

Usage

```
mx.symbol.LRN(...)
```

Arguments

data	Symbol Input data to the ConvolutionOp.
alpha	float, optional, default=0.0001 value of the alpha variance scaling parameter in the normalization formula
beta	float, optional, default=0.75 value of the beta power parameter in the normalization formula
knorm	float, optional, default=2 value of the k parameter in normalization formula
nsize	int (non-negative), required normalization window width in elements.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.MAERegressionOutput` *Use mean absolute error regression for final output, this is used on final output of a net.*

Description

Use mean absolute error regression for final output, this is used on final output of a net.

Usage

```
mx.symbol.MAERegressionOutput(...)
```

Arguments

`data`	Symbol Input data to function.
`label`	Symbol Input label to function.
`grad.scale`	float, optional, default=1 Scale the gradient by a float factor
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.MakeLoss` *Get output from a symbol and pass 1 gradient back. This is used as a terminal loss if unary and binary operator are used to composite a loss with no declaration of backward dependency*

Description

Get output from a symbol and pass 1 gradient back. This is used as a terminal loss if unary and binary operator are used to composite a loss with no declaration of backward dependency

Usage

```
mx.symbol.MakeLoss(...)
```

Arguments

`data`	Symbol Input data.
`grad.scale`	float, optional, default=1 gradient scale as a supplement to unary and binary operators
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.normal *Sample a normal distribution*

Description

Sample a normal distribution

Usage

```
mx.symbol.normal(...)
```

Arguments

loc	float, optional, default=0 Mean of the distribution.
scale	float, optional, default=1 Standard deviation of the distribution.
shape	Shape(tuple), required The shape of the output
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.Pooling *Perform spatial pooling on inputs.*

Description

Perform spatial pooling on inputs.

Usage

```
mx.symbol.Pooling(...)
```

Arguments

data	Symbol Input data to the pooling operator.
global.pool	boolean, optional, default=False Ignore kernel size, do global pooling based on current input feature map. This is useful for input with different shape
kernel	Shape(tuple), required pooling kernel size: (y, x)
pool.type	'avg', 'max', 'sum', required Pooling type to be applied.
stride	Shape(tuple), optional, default=(1,1) stride: for pooling (y, x)
pad	Shape(tuple), optional, default=(0,0) pad for pooling: (y, x)
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.Reshape` *Reshape input to target shape*

Description

Reshape input to target shape

Usage

```
mx.symbol.Reshape(...)
```

Arguments

`data`	Symbol Input data to reshape.
`target.shape`	Shape(tuple), optional, default=(0,0) (Deprecated! Use shape instead.) Target new shape. One and only one dim can be 0, in which case it will be inferred from the rest of dims
`keep.highest`	boolean, optional, default=False (Deprecated! Use shape instead.) Whether keep the highest dim unchanged.If set to yes, than the first dim in target_shape is ignored,and always fixed as input
`shape`	, optional, default=() Target new shape. If the dim is same, set it to 0. If the dim is set to be -1, it will be inferred from the rest of dims. One and only one dim can be -1
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.ROIPooling` *Performs region-of-interest pooling on inputs. Resize bounding box coordinates by spatial_scale and crop input feature maps accordingly. The cropped feature maps are pooled by max pooling to a fixed size output indicated by pooled_size. batch_size will change to the number of region bounding boxes after ROIPooling*

Description

Performs region-of-interest pooling on inputs. Resize bounding box coordinates by spatial_scale and crop input feature maps accordingly. The cropped feature maps are pooled by max pooling to a fixed size output indicated by pooled_size. batch_size will change to the number of region bounding boxes after ROIPooling

Usage

```
mx.symbol.ROIPooling(...)
```

Arguments

data	Symbol Input data to the pooling operator, a 4D Feature maps
rois	Symbol Bounding box coordinates, a 2D array of [[batch_index, x1, y1, x2, y2]]. (x1, y1) and (x2, y2) are top left and down right corners of designated region of interest. batch_index indicates the index of corresponding image in the input data
pooled.size	Shape(tuple), required fix pooled size: (h, w)
spatial.scale	float, required Ratio of input feature map height (or w) to raw image height (or w). Equals the reciprocal of total stride in convolutional layers
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.round *Take round value of the src*

Description

Take round value of the src

Usage

```
mx.symbol.round(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.rsqrt *Take rsqrt of the src*

Description

Take rsqrt of the src

Usage

```
mx.symbol.rsqrt(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.save *Save an mx.symbol object*

Description

Save an mx.symbol object

Usage

```
mx.symbol.save(symbol, filename)
```

Arguments

symbol	the mx.symbol object
filename	the filename (including the path)

Examples

```
data = mx.symbol.Variable('data')
mx.symbol.save(data, 'temp.symbol')
data2 = mx.symbol.load('temp.symbol')
```

---

mx.symbol.sign *Take sign value of the src*

Description

Take sign value of the src

Usage

```
mx.symbol.sign(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.sin *Take sin of the src*

Description

Take sin of the src

Usage

```
mx.symbol.sin(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.SliceChannel *Slice input equally along specified axis*

Description

Slice input equally along specified axis

Usage

```
mx.symbol.SliceChannel(...)
```

Arguments

num.outputs	int, required Number of outputs to be sliced.
axis	int, optional, default='1' Dimension along which to slice.
squeeze.axis	boolean, optional, default=False If true AND the sliced dimension becomes 1, squeeze that dimension.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.slice_axis *Slice the input along certain axis and return a sliced array.*

Description

Slice the input along certain axis and return a sliced array.

Usage

```
mx.symbol.slice_axis(...)
```

Arguments

src	Symbol Left symbolic input to the function
axis	int, required The axis to be sliced
begin	int, required The beginning index to be sliced
end	int, required The end index to be sliced
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.smooth_l1 *Calculate Smooth L1 Loss(lhs, scalar)*

Description

Calculate Smooth L1 Loss(lhs, scalar)

Usage

```
mx.symbol.smooth_l1(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.Softmax` *DEPRECATED: Perform a softmax transformation on input. Please use SoftmaxOutput*

Description

DEPRECATED: Perform a softmax transformation on input. Please use SoftmaxOutput

Usage

```
mx.symbol.Softmax(...)
```

Arguments

`data`	Symbol Input data to softmax.
`grad.scale`	float, optional, default=1 Scale the gradient by a float factor
`ignore.label`	float, optional, default=-1 the label value will be ignored during backward (only works if use_ignore is set to be true).
`multi.output`	boolean, optional, default=False If set to true, for a (n,k,x_1,..,x_n) dimensional input tensor, softmax will generate n*x_1*...*x_n output, each has k classes
`use.ignore`	boolean, optional, default=False If set to true, the ignore_label value will not contribute to the backward gradient
`normalization`	'batch', 'null', 'valid',optional, default='null' If set to null, op will do nothing on output gradient.If set to batch, op will normalize gradient by divide batch sizeIf set to valid, op will normalize gradient by divide sample not ignored
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.SoftmaxActivation` *Apply softmax activation to input. This is intended for internal layers. For output (loss layer) please use SoftmaxOutput. If type=instance, this operator will compute a softmax for each instance in the batch; this is the default mode. If type=channel, this operator will compute a num_channel-class softmax at each position of each instance; this can be used for fully convolutional network, image segmentation, etc.*

Description

Apply softmax activation to input. This is intended for internal layers. For output (loss layer) please use SoftmaxOutput. If type=instance, this operator will compute a softmax for each instance in the batch; this is the default mode. If type=channel, this operator will compute a num_channel-class softmax at each position of each instance; this can be used for fully convolutional network, image segmentation, etc.

Usage

```
mx.symbol.SoftmaxActivation(...)
```

Arguments

data	Symbol Input data to activation function.
mode	'channel', 'instance',optional, default='instance' Softmax Mode. If set to instance, this operator will compute a softmax for each instance in the batch; this is the default mode. If set to channel, this operator will compute a num_channel-class softmax at each position of each instance; this can be used for fully convolutional network, image segmentation, etc.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

---

mx.symbol.SoftmaxOutput

Perform a softmax transformation on input, backprop with logloss.

Description

Perform a softmax transformation on input, backprop with logloss.

Usage

```
mx.symbol.SoftmaxOutput(...)
```

Arguments

data	Symbol Input data to softmax.
label	Symbol Label data, can also be probability value with same shape as data
grad.scale	float, optional, default=1 Scale the gradient by a float factor
ignore.label	float, optional, default=-1 the label value will be ignored during backward (only works if use_ignore is set to be true).
multi.output	boolean, optional, default=False If set to true, for a (n,k,x_1,..,x_n) dimensional input tensor, softmax will generate n*x_1*...*x_n output, each has k classes
use.ignore	boolean, optional, default=False If set to true, the ignore_label value will not contribute to the backward gradient
normalization	'batch', 'null', 'valid',optional, default='null' If set to null, op will do nothing on output gradient.If set to batch, op will normalize gradient by divide batch sizeIf set to valid, op will normalize gradient by divide sample not ignored
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.softmax_cross_entropy`
Calculate cross_entropy(lhs, one_hot(rhs))

Description

Calculate cross_entropy(lhs, one_hot(rhs))

Usage

```
mx.symbol.softmax_cross_entropy(...)
```

Arguments

`lhs`	Symbol Left symbolic input to the function
`rhs`	Symbol Right symbolic input to the function
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.SpatialTransformer`
Apply spatial transformer to input feature map.

Description

Apply spatial transformer to input feature map.

Usage

```
mx.symbol.SpatialTransformer(...)
```

Arguments

`data`	Symbol Input data to the SpatialTransformerOp.
`loc`	Symbol localisation net, the output dim should be 6 when transform_type is affine, and the name of loc symbol should better starts with 'stn_loc', so that initialization it with iddentify tranform, or you shold initialize the weight and bias by yourself.
`target.shape`	Shape(tuple), optional, default=(0,0) output shape(h, w) of spatial transformer: (y, x)
`transform.type`	'affine', required transformation type
`sampler.type`	'bilinear', required sampling type
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.sqrt *Take sqrt of the src*

Description

Take sqrt of the src

Usage

```
mx.symbol.sqrt(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.square *Take square of the src*

Description

Take square of the src

Usage

```
mx.symbol.square(...)
```

Arguments

src	Symbol Left symbolic input to the function
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.sum` *Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.

Usage

```
mx.symbol.sum(...)
```

Arguments

`src`	Symbol Left symbolic input to the function
`axis`	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
`keepdims`	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.sum_axis` *(Depreciated! Use sum instead!) Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.*

Description

(Depreciated! Use sum instead!) Take sum of the src in the given axis and returns a NDArray. Follows numpy semantics.

Usage

```
mx.symbol.sum_axis(...)
```

Arguments

`src`	Symbol Left symbolic input to the function
`axis`	Shape(tuple), optional, default=() Same as Numpy. The axes to perform the reduction.If left empty, a global reduction will be performed.
`keepdims`	boolean, optional, default=False Same as Numpy. If keepdims is set to true, the axis which is reduced is left in the result as dimension with size one.
`name`	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.SwapAxis *Apply swapaxis to input.*

Description

Apply swapaxis to input.

Usage

```
mx.symbol.SwapAxis(...)
```

Arguments

data	Symbol Input data to the SwapAxisOp.
dim1	int (non-negative), optional, default=0 the first axis to be swapped.
dim2	int (non-negative), optional, default=0 the second axis to be swapped.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.transpose *Transpose the input matrix and return a new one*

Description

Transpose the input matrix and return a new one

Usage

```
mx.symbol.transpose(...)
```

Arguments

src	Symbol Left symbolic input to the function
axes	Shape(tuple), optional, default=() Target axis order. By default the axes will be inverted.
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.uniform *Sample a uniform distribution*

Description

Sample a uniform distribution

Usage

```
mx.symbol.uniform(...)
```

Arguments

low	float, optional, default=0 The lower bound of distribution
high	float, optional, default=1 The upper bound of distribution
shape	Shape(tuple), required The shape of the output
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

mx.symbol.UpSampling *Perform nearest neighboor/bilinear up sampling to inputs*

Description

Perform nearest neighboor/bilinear up sampling to inputs

Usage

```
mx.symbol.UpSampling(...)
```

Arguments

data	Symbol[] Array of tensors to upsample
scale	int (non-negative), required Up sampling scale
num.filter	int (non-negative), optional, default=0 Input filter. Only used by nearest sample_type.
sample.type	'bilinear', 'nearest', required upsampling method
multi.input.mode	'concat', 'sum',optional, default='concat' How to handle multiple input. concat means concatenate upsampled images along the channel dimension. sum means add all images together, only available for nearest neighbor upsampling.
num.args	int, required Number of inputs to be upsampled. For nearest neighbor upsampling, this can be 1-N; the size of output will be(scale*h_0,scale*w_0) and all other inputs will be upsampled to thesame size. For bilinear upsampling this must be 2; 1 input and 1 weight.

workspace	long (non-negative), optional, default=512 Tmp workspace for deconvolution (MB)
name	string, optional Name of the resulting symbol.

Value

out The result mx.symbol

`mx.symbol.Variable` *Create a symbolic variable with specified name.*

Description

Create a symbolic variable with specified name.

Arguments

name	string The name of the result symbol.

Value

The result symbol

`mxnet` *MXNet: Flexible and Efficient GPU computing and Deep Learning.*

Description

MXNet is a flexible and efficient GPU computing and deep learning framework.

Details

It enables you to write seamless tensor/matrix computation with multiple GPUs in R.

It also enables you construct and customize the state-of-art deep learning models in R, and apply them to tasks such as image classification and data science challenges.

`mxnet.export` *Internal function to generate mxnet_generated.R Users do not need to call this function.*

Description

Internal function to generate mxnet_generated.R Users do not need to call this function.

Usage

```
mxnet.export(path)
```

Arguments

path	The path to the root of the package.

Ops.MXNDArray *Binary operator overloading of mx.ndarray*

Description

Binary operator overloading of mx.ndarray

Usage

```
## S3 method for class 'MXNDArray'
Ops(e1, e2)
```

Arguments

e1	The first operand
e1	The second operand

outputs *Get the outputs of a symbol.*

Description

Get the outputs of a symbol.

Usage

```
outputs(x)
```

Arguments

x	The input symbol

predict.MXFeedForwardModel *Predict the outputs given a model and dataset.*

Description

Predict the outputs given a model and dataset.

Usage

```
## S3 method for class 'MXFeedForwardModel'
predict(model, X, ctx = NULL,
  array.batch.size = 128, array.layout = "auto")
```

Arguments

model	The MXNet Model.
X	The dataset to predict.
ctx	mx.cpu() or mx.gpu(i) The device used to generate the prediction.
array.batch.size	The batch size used in batching. Only used when X is R's array.
array.layout	can be "auto", "colmajor", "rowmajor", (detault=auto) The layout of array. "rowmajor" is only supported for two dimensional array. For matrix, "rowmajor" means dim(X) = c(nexample, nfeatures), "colmajor" means dim(X) = c(nfeatures, nexample) "auto" will auto detect the layout by match the feature size, and will report error when X is a square matrix to ask user to explicitly specify layout.

---

print.MXNDArray *print operator overload of mx.ndarray*

Description

print operator overload of mx.ndarray

Usage

```
## S3 method for class 'MXNDArray'
print(nd)
```

Arguments

nd	The mx.ndarray

Index