

# Package ‘rnn’

June 21, 2018

**Title** Recurrent Neural Network

**Version** 0.8.1

**Description** Implementation of a Recurrent Neural Network in R.

**Depends** R (>= 3.2.2)

**License** GPL-3

**LazyData** true

**RoxygenNote** 6.0.1

**URL** <http://qua.st/rnn>, <https://github.com/bquast/rnn>

**BugReports** <https://github.com/bquast/rnn/issues>

**Imports** sigmoid, shiny

**Suggests** testthat, knitr, rmarkdown

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Bastiaan Quast [aut, cre],  
Dimitri Fichou [aut]

**Maintainer** Bastiaan Quast <bquast@gmail.com>

**Repository** CRAN

**Date/Publication** 2018-06-21 18:23:56 UTC

## R topics documented:

backprop_gru . . . . .	2
backprop_lstm . . . . .	3
backprop_r . . . . .	3
backprop_rnn . . . . .	4
bin2int . . . . .	4
clean_lstm . . . . .	5
clean_r . . . . .	5
clean_rnn . . . . .	6
epoch_annealing . . . . .	6

epoch_print . . . . .	7
init_gru . . . . .	7
init_lstm . . . . .	8
init_r . . . . .	8
init_rnn . . . . .	9
int2bin . . . . .	9
loss_L1 . . . . .	10
predictr . . . . .	10
predict_gru . . . . .	11
predict_lstm . . . . .	12
predict_rnn . . . . .	13
rnn . . . . .	13
run.finance_demo . . . . .	14
run.rnn_demo . . . . .	14
trainr . . . . .	15
update_adagrad . . . . .	17
update_r . . . . .	17
update_sgd . . . . .	18
<b>Index</b>	<b>19</b>

---

backprop_gru	<i>backprop_gru</i>
--------------	---------------------

---

## Description

backpropagate the error in a model object of type gru

## Usage

```
backprop_gru(model, a, c, j, ...)
```

## Arguments

model	the output model object
a	the input of this learning batch
c	the output of this learning batch
j	the indexes of the sample in the current batch
...	argument to be passed to method

## Value

the updated model

---

backprop_lstm	<i>backprop_lstm</i>
---------------	----------------------

---

**Description**

backpropagate the error in a model object of type rlstm

**Usage**

```
backprop_lstm(model, a, c, j, ...)
```

**Arguments**

model	the output model object
a	the input of this learning batch
c	the output of this learning batch
j	the indexes of the sample in the current batch
...	argument to be passed to method

**Value**

the updated model

---

backprop_r	<i>backprop_r</i>
------------	-------------------

---

**Description**

backpropagate the error in a model object

**Usage**

```
backprop_r(model, a, c, j, ...)
```

**Arguments**

model	the output model object
a	the input of this learning batch
c	the output of this learning batch
j	the indexes of the sample in the current batch
...	argument to be passed to method

**Value**

the updated model

---

backprop_rnn	<i>backprop_rnn</i>
--------------	---------------------

---

**Description**

backpropagate the error in a model object of type rnn

**Usage**

```
backprop_rnn(model, a, c, j, ...)
```

**Arguments**

model	the output model object
a	the input of this learning batch
c	the output of this learning batch
j	the indexes of the sample in the current batch
...	argument to be passed to method

**Value**

the updated model

---

bin2int	<i>Binary to Integer</i>
---------	--------------------------

---

**Description**

Binary to Integer

**Usage**

```
bin2int(binary)
```

```
b2i(binary)
```

**Arguments**

binary	input binary
--------	--------------

**Value**

integer representation

**Functions**

- b2i: individual Binary to Integer

---

`clean_lstm`                      *clean\_lstm*

---

**Description**

clean the model for lighter output

**Usage**

`clean_lstm(model)`

**Arguments**

`model`                      the output model object

**Value**

the updated model

---

`clean_r`                      *init\_r*

---

**Description**

Initialize the weight parameters

**Usage**

`clean_r(model)`

**Arguments**

`model`                      the output model object

**Value**

the updated model

---

`clean_rnn`*clean\_rnn*

---

**Description**

clean the model for lighter output

**Usage**

```
clean_rnn(model)
```

**Arguments**

`model`            the output model object

**Value**

the updated model

---

`epoch_annealing`*epoch\_annealing*

---

**Description**

Apply the learning rate decay to the learning rate, called in `epoch_model_function`

**Usage**

```
epoch_annealing(model)
```

**Arguments**

`model`            the output model object

**Value**

the updated model

---

`epoch_print`                      *epoch printing for trainr*

---

**Description**

Print the error and learning rate at each epoch of the trainr learning, called in `epoch_function`

**Usage**

`epoch_print(model)`

**Arguments**

`model`                      the output model object

**Value**

nothing

---

`init_gru`                      *init\_gru*

---

**Description**

Initialize the weight parameter for a gru

**Usage**

`init_gru(model)`

**Arguments**

`model`                      the output model object

**Value**

the updated model

---

`init_lstm`*init\_lstm*

---

**Description**

Initialize the weight parameter for a lstm

**Usage**

```
init_lstm(model)
```

**Arguments**

`model`            the output model object

**Value**

the updated model

---

`init_r`*init\_r*

---

**Description**

Initialize the weight parameters

**Usage**

```
init_r(model)
```

**Arguments**

`model`            the output model object

**Value**

the updated model



---

init_rnn	<i>init_rnn</i>
----------	-----------------

---

**Description**

Initialize the weight parameter for a rnn

**Usage**

```
init_rnn(model)
```

**Arguments**

model            the output model object

**Value**

the updated model

---

int2bin	<i>Integer to Binary</i>
---------	--------------------------

---

**Description**

Integer to Binary

**Usage**

```
int2bin(integer, length = 8)
```

```
i2b(integer, length = 8)
```

**Arguments**

integer            input integer  
length            binary representation length

**Value**

binary representation

**Functions**

- i2b: individual Integer to Binary

---

loss_L1	<i>L1 loss</i>
---------	----------------

---

**Description**

Apply the learning rate to the weight update, vocabulary to verify !!

**Usage**

```
loss_L1(model)
```

**Arguments**

model	the output model object
-------	-------------------------

**Value**

the updated model

---

predictr	<i>Recurrent Neural Network</i>
----------	---------------------------------

---

**Description**

predict the output of a RNN model

**Usage**

```
predictr(model, X, hidden = FALSE, real_output = T, ...)
```

**Arguments**

model	output of the trainr function
X	array of input values, dim 1: samples, dim 2: time, dim 3: variables (could be 1 or more, if a matrix, will be coerced to array)
hidden	should the function output the hidden units states
real_output	option used when the function is called inside trainr, do not drop factor for 2 dimension array output and other actions. Let it to TRUE, the default, to let the function take care of the data.
...	arguments to pass on to sigmoid function

**Value**

array or matrix of predicted values

**Examples**

```
## Not run:
# create training numbers
X1 = sample(0:127, 10000, replace=TRUE)
X2 = sample(0:127, 10000, replace=TRUE)

# create training response numbers
Y <- X1 + X2

# convert to binary
X1 <- int2bin(X1)
X2 <- int2bin(X2)
Y <- int2bin(Y)

# Create 3d array: dim 1: samples; dim 2: time; dim 3: variables.
X <- array( c(X1,X2), dim=c(dim(X1),2) )

# train the model
model <- trainr(Y=Y[,dim(Y)[2]:1],
                X=X[,dim(X)[2]:1,],
                learningrate = 1,
                hidden_dim    = 16 )

# create test inputs
A1 = int2bin( sample(0:127, 7000, replace=TRUE) )
A2 = int2bin( sample(0:127, 7000, replace=TRUE) )

# create 3d array: dim 1: samples; dim 2: time; dim 3: variables
A <- array( c(A1,A2), dim=c(dim(A1),2) )

# predict
B <- predictr(model,
              A[,dim(A)[2]:1,] )
B = B[,dim(B)[2]:1]
# convert back to integers
A1 <- bin2int(A1)
A2 <- bin2int(A2)
B <- bin2int(B)

# inspect the differences
table( B-(A1+A2) )

# plot the difference
hist( B-(A1+A2) )

## End(Not run)
```

**Description**

predict the output of a gru model

**Usage**

```
predict_gru(model, X, hidden = FALSE, real_output = T, ...)
```

**Arguments**

model	output of the trainr function
X	array of input values, dim 1: samples, dim 2: time, dim 3: variables (could be 1 or more, if a matrix, will be coerce to array)
hidden	should the function output the hidden units states
real_output	option used when the function in called inside trainr, do not drop factor for 2 dimension array output
...	arguments to pass on to sigmoid function

**Value**

array or matrix of predicted values

---

predict_lstm	<i>gru prediction function</i>
--------------	--------------------------------

---

**Description**

predict the output of a lstm model

**Usage**

```
predict_lstm(model, X, hidden = FALSE, real_output = T, ...)
```

**Arguments**

model	output of the trainr function
X	array of input values, dim 1: samples, dim 2: time, dim 3: variables (could be 1 or more, if a matrix, will be coerce to array)
hidden	should the function output the hidden units states
real_output	option used when the function in called inside trainr, do not drop factor for 2 dimension array output
...	arguments to pass on to sigmoid function

**Value**

array or matrix of predicted values

---

predict_rnn	<i>Recurrent Neural Network</i>
-------------	---------------------------------

---

**Description**

predict the output of a RNN model

**Usage**

```
predict_rnn(model, X, hidden = FALSE, real_output = T, ...)
```

**Arguments**

model	output of the trainr function
X	array of input values, dim 1: samples, dim 2: time, dim 3: variables (could be 1 or more, if a matrix, will be coerce to array)
hidden	should the function output the hidden units states
real_output	option used when the function in called inside trainr, do not drop factor for 2 dimension array output
...	arguments to pass on to sigmoid function

**Value**

array or matrix of predicted values

---

rnn	<i>Recurrent Neural Network</i>
-----	---------------------------------

---

**Description**

A Recurrent Neural Network in native R, transforms numbers to binaries before adding bit by bit, teaching itself how to carry.

**Author(s)**

Bastiaan Quast <bquast@gmail.com>

**References**

<http://qua.st/rnn>

**See Also**

[trainr](#) for training a model and [predictr](#) for using a model to make predictions. <http://qua.st/rnn>

---

run.finance\_demo      *Demo app on finance data*

---

**Description**

Function to launch the finance\_demo app

**Usage**

```
run.finance_demo(port = NULL)
```

**Arguments**

port                    if not NULL will deploy on the local network on this port, just look for the IP of the host and go for example on 192.168.1.20:port to access the app from anywhere in the network

**Author(s)**

Dimitri Fichou

**Examples**

```
## Not run:  
run.finance_demo()  
  
## End(Not run)
```

---

run.rnn\_demo            *Demo app*

---

**Description**

Function to launch the rnn\_demo app

**Usage**

```
run.rnn_demo(port = NULL)
```

**Arguments**

port                    if not NULL will deploy on the local network on this port, just look for the IP of the host and go for example on 192.168.1.20:port to access the app from anywhere in the network

**Author(s)**

Dimitri Fichou

**Examples**

```
## Not run:
run.rnn_demo()

## End(Not run)
```

---

trainr

*Recurrent Neural Network*


---

**Description**

Trains a Recurrent Neural Network.

**Usage**

```
trainr(Y, X, model = NULL, learningrate, learningrate_decay = 1,
       momentum = 0, hidden_dim = c(10), network_type = "rnn", numepochs = 1,
       sigmoid = c("logistic", "Gompertz", "tanh"), use_bias = F,
       batch_size = 1, seq_to_seq_unsync = F, update_rule = "sgd",
       epoch_function = c(epoch_print, epoch_annealing), loss_function = loss_L1,
       ...)
```

**Arguments**

Y	array of output values, dim 1: samples (must be equal to dim 1 of X), dim 2: time (must be equal to dim 2 of X), dim 3: variables (could be 1 or more, if a matrix, will be coerce to array)
X	array of input values, dim 1: samples, dim 2: time, dim 3: variables (could be 1 or more, if a matrix, will be coerce to array)
model	a model trained before, used for retraining purpose.
learningrate	learning rate to be applied for weight iteration
learningrate_decay	coefficient to apply to the learning rate at each epoch, via the epoch_annealing function
momentum	coefficient of the last weight iteration to keep for faster learning
hidden_dim	dimension(s) of hidden layer(s)
network_type	type of network, could be rnn, gru or lstm. gru and lstm are experimentale.
numepochs	number of iteration, i.e. number of time the whole dataset is presented to the network
sigmoid	method to be passed to the sigmoid function

use_bias	should the network use bias
batch_size	batch size: number of samples used at each weight iteration, only 1 supported for the moment
seq_to_seq_unsync	if TRUE, the network will be trained to backpropagate only the second half of the output error. If many to one is the target, just make Y have a time dim of 1. The X and Y data are modify at first to fit a classic learning, error are set to 0 during back propagation, input for the second part is also set to 0.
update_rule	rule to update the weight, "sgd", the default, is stochastic gradient descent, other available options are "adagrad" (experimentale, do not learn yet)
epoch_function	vector of functions to applied at each epoch loop. Use it to intereact with the objects inside the list model or to print and plot at each epoch. Should return the model.
loss_function	loss function, applied in each sample loop, vocabulary to verify.
...	Arguments to be passed to methods, to be used in user defined functions

### Value

a model to be used by the predictr function

### Examples

```
## Not run:
# create training numbers
X1 = sample(0:127, 10000, replace=TRUE)
X2 = sample(0:127, 10000, replace=TRUE)

# create training response numbers
Y <- X1 + X2

# convert to binary
X1 <- int2bin(X1, length=8)
X2 <- int2bin(X2, length=8)
Y <- int2bin(Y, length=8)

# create 3d array: dim 1: samples; dim 2: time; dim 3: variables
X <- array( c(X1,X2), dim=c(dim(X1),2) )

# train the model
model <- trainr(Y=Y,
               X=X,
               learningrate = 1,
               hidden_dim   = 16 )

## End(Not run)
```



---

<code>update_adagrad</code>	<i>update_adagrad</i>
-----------------------------	-----------------------

---

**Description**

Apply the update with adagrad, not working yet

**Usage**

```
update_adagrad(model)
```

**Arguments**

`model`            the output model object

**Value**

the updated model

---

<code>update_r</code>	<i>update_r</i>
-----------------------	-----------------

---

**Description**

Apply the update

**Usage**

```
update_r(model)
```

**Arguments**

`model`            the output model object

**Value**

the updated model

---

`update_sgd`*update\_sgd*

---

**Description**

Apply the update with stochastic gradient descent

**Usage**

```
update_sgd(model)
```

**Arguments**

`model` the output model object

**Value**

the updated model

# Index

b2i (bin2int), 4  
backprop\_gru, 2  
backprop\_lstm, 3  
backprop\_r, 3  
backprop\_rnn, 4  
bin2int, 4

clean\_lstm, 5  
clean\_r, 5  
clean\_rnn, 6

epoch\_annealing, 6  
epoch\_print, 7

i2b (int2bin), 9  
init\_gru, 7  
init\_lstm, 8  
init\_r, 8  
init\_rnn, 9  
int2bin, 9

loss\_L1, 10

predict\_gru, 11  
predict\_lstm, 12  
predict\_rnn, 13  
predictr, 10, 13

rnn, 13  
run.finance\_demo, 14  
run.rnn\_demo, 14

trainr, 13, 15

update\_adagrad, 17  
update\_r, 17  
update\_sgd, 18