Package 'edl'

April 12, 2021

Version 1.0

Date 2021-04-01

Title Toolbox for Error-Driven Learning Simulations with Two-Layer Networks

Maintainer Jacolien van Rij <j.c.van.rij@rug.nl>

Description Error-driven learning (based on the Widrow & Hoff (1960)<https://isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf> learning rule, and essentially the same as Rescorla-Wagner's learning equations (Rescorla & Wagner, 1972, ISBN: 0390718017), which are also at the core of Naive Discrimination Learning, (Baayen et al, 2011, <doi:10.1037/a0023851>) can be used to explain bottom-up human learning (Hoppe et al, <doi:10.31234/osf.io/py5kd>), but is also at the core of artificial neural networks applications in the form of the Delta rule. This package provides a set of functions for building small-scale simulations to investigate the dynamics of error-driven learning and it's interaction with the structure of the input. For modeling error-driven learning using the Rescorla-Wagner equations the package 'ndl' (Baayen et al, 2011, <doi:10.1037/a0023851>) is available on CRAN at <https://cran.r-project.org/package=ndl>. However, the package currently only allows tracing of a cue-outcome combination, rather than returning the learned networks. To fill this gap, we implemented a new package with a few functions that facilitate inspection of the networks for small error driven learning simulations. Note that our functions are not optimized for training large data sets (no parallel processing), as they are intended for small scale simulations and course examples. (Consider the python implementation 'pyndl' <https://pyndl.readthedocs.io/en/latest/> for that purpose.)

License GPL (>= 2)

LazyData true

Depends R (>= 4.0.0), plotfunctions (>= 1.4), data.table

VignetteBuilder knitr

Suggests knitr, rmarkdown

RoxygenNote 7.1.0

Encoding UTF-8

NeedsCompilation no

Author Jacolien van Rij [aut, cre] (<https://orcid.org/0000-0001-7445-5647>), Dorothée Hoppe [aut]

Repository CRAN

Date/Publication 2021-04-12 07:50:02 UTC

R topics documented:

activationsCueSet *Calculate the change in activation for a specific cue or set of cues.*

Description

Calculate the change in activation for a specific cue or set of cues for all outcomes (or a subset) in the weightmatrices.

Usage

```
activationsCueSet(
  wmlist,
  cueset,
  split = "_",
  select.outcomes = NULL,
  init.value = 0,
  normalize = FALSE
)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights, or a single weightmatrix (matrix).
cueset	String, specifying the cue set for which to calculate change in activation.
split	String, separator between cues and/or outcomes.
select.outcomes	Optional selection of outcomes to limit (or expand) the number of activations that are returned. The value of NULL (default) will return all activations (for each outcome in wmlist). Note that specified values that are not in the weightmatrix will return the initial value without error or warning. Please use getValues for returning all outcomes in the data.
init.value	Value of activations for non-existing connections. Typically set to 0.
normalize	Logical: whether or not the activation is normalized by dividing the total activation by the number of cues. Default is FALSE. If set to TRUE, the activation reflects the average activation per cue.

Value

List of data frames. For each cueset defined in cueset, a dataframe of activation values is provided. These are returned as a list, with the cuesets as names.

Notes

The outcomes are selected based on the weightmatrices, and not necessarily all outcomes present in the training data.

Author(s)

Jacolien van Rij

See Also

getWeightsByCue, getWeightsByOutcome

Other functions for calculating activations: activationsEvents(), activationsMatrix(), activationsOutcomes(), getActivations()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)
head(train)

# this training data can actually be used train network:
wm <- RWlearning(train)

# Now we calculate the activations for all outcomes
# per event:
activations <- activationsCueSet(wm, cueset="BG_bicycle_red")
names(activations)
head(activations[[1]])

# plot:
a1 <- activations[[1]]
emptyPlot(nrow(a1), range(a1),
    xlab="Learning events", ylab="Activations",
    xmark=TRUE, ymark=TRUE, las=1)
for(i in 1:ncol(a1)){
    lines(a1[,i], col=i, lty=i)
}
legend_margin('topleft', legend=colnames(a1),
    col=1:ncol(a1), lty=1:ncol(a1),
    bty='n', cex=.75)
```

`activationsEvents` *Calculate the activations for each learning event.*

Description

Calculate the activations for each learning event. The values are returned as data frame or as a list of data frames.

Usage

```
activationsEvents(
  wmlist,
  data,
  split = "_",
  fun = NULL,
  return.list = FALSE,
  init.value = 0,
  normalize = FALSE
)
```

Arguments

`wmlist`	A list with weightmatrices, generated by `RWlearning` or `updateWeights`, or a single weightmatrix (matrix).
`data`	Data frame with columns `Cues` and `Outcomes`. Number of rows should be the same as the number of weightmatrices in `wmlist`.
`split`	String, separator between cues and/or outcomes.
`fun`	Function to apply to the activations for events with multiple outcomes. By default (`fun=NULL`) the activation values for each outcome are returned. If there are learning events with multiple outcomes, the argument `return.list` will be automatically set to TRUE.
`return.list`	Logical: whether or not the activation values are returned as list or as vector. Defaults to the value FALSE, returning a vector of activation values. But this also depends on the argument `fun` (see more info above).
`init.value`	Value of activations for non-existing connections. Typically set to 0.
`normalize`	Logical: whether or not the activation is normalized by dividing the total activation by the number of cues. Default is FALSE. If set to TRUE, the activation reflects the average activation per cue.

Value

Vector or list of activation values (see `return.list` and `fun` for the specific conditions, and the examples below).

Author(s)

Jacolien van Rij

See Also

getWeightsByCue, getWeightsByOutcome

Other functions for calculating activations: activationsCueSet(), activationsMatrix(), activationsOutcomes(), getActivations()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)
head(train)

# this training data can actually be used train network:
wm <- RWlearning(train)

# Now we calculate the activations for each event:
train$Activation <- activationsEvents(wm, train)

# With multiple outcomes per event, it is better not
# to directly assign to a new column, as a list will
# return. See the example below:
dat$Outcomes <- paste(dat$Shape, dat$Color, sep="_")
dat$Cues <- paste("BG", dat$Category, sep="_")
dat$Frequency <- dat$Frequency1
head(dat)
train <- createTrainingData(dat)
wm <- RWlearning(train)
# This code will elicit a warning message:
## Not run:
    act <- activationsEvents(wm, train)

## End(Not run)
# this code will not elicit a warning:
act <- activationsEvents(wm, train, return.list=TRUE)
head(act)
# to assign one single activation value to each event,
# we could instead apply a function, for example, by
# taking the max activation per event:
train$maxAct <- activationsEvents(wm, train, fun="max")
```

`activationsMatrix` *Calculate the activations for one or a set of cues.*

Description

Calculate the activations for one or a set of cues. The values are returned as vector or data frame.

Usage

```
activationsMatrix(
  wm,
  cues,
  split = "_",
  select.outcomes = NULL,
  init.value = 0,
  normalize = FALSE
)
```

Arguments

`wm`	A weightmatrix, generated by `RWlearning` or `updateWeights`.
`cues`	String or vector of strings. Each string represents a set of cues, separated by `split`, for which the activations will be calculated. Note: the activations will be calculated for all provided cues together, assuming these occurred in one learning event.
`split`	String, separator between cues.
`select.outcomes`	Optional selection of outcomes to limit the number of activations that are returned. The value of NULL (default) will return all activations (for each outcome in `wm`). Note that specified values that are not in the weightmatrix will return the initial value without error or warning. Please use `getValues` for returning all outcomes in the data.
`init.value`	Value of activations for non-existing connections. Typically set to 0.
`normalize`	Logical: whether or not the activation is normalized by dividing the total activation by the number of cues. Default is FALSE. If set to TRUE, the activation reflects the average activation per cue.

Value

Vector or data frame.

Author(s)

Jacolien van Rij

See Also

getWeightsByCue, getWeightsByOutcome

Other functions for calculating activations: activationsCueSet(), activationsEvents(), activationsOutcomes(), getActivations()

Examples

```
# load example data:
data(dat)

# setup data:
newdat <- data.frame(Cues =paste("BG", dat$Shape, dat$Color, sep="_"),
   Outcomes  = dat$Category,
   Frequency = dat$Frequency2)
train  <- createTrainingData(newdat)
# learning:
wm     <- RWlearning(train)

# calculate activations for all outcomes:
mat    <- getWM(wm)
activationsMatrix(mat, cues="BG_tree_green")
# only accepts one set of cues - in this case all cues
# are combined:
activationsMatrix(mat, cues=c("BG_tree", "BG_tree_brown"))
# ... which is the same as this:
activationsMatrix(mat, cues=c("BG", "BG", "tree", "tree", "brown"))
# now select one outcome:
activationsMatrix(mat, cues=c("BG", "tree"), select.outcomes="vehicle")
# cues/outcomes not in matrix:
activationsMatrix(mat, cues=c("na"), select.outcomes="new")
```

---

activationsOutcomes *Calculate the activations for all outcomes in the data.*

Description

Calculate the activations for all outcomes in the data per learning event. The activation values are returned as data frame.

Usage

```
activationsOutcomes(
  wmlist,
  data,
  split = "_",
  select.outcomes = NULL,
  init.value = 0,
  normalize = FALSE
)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights, or a single weightmatrix (matrix).
data	Data frame with columns Cues and Outcomes. Number of rows should be the same as the number of weightmatrices in wmlist.
split	String, separator between cues and/or outcomes.
select.outcomes	Optional selection of outcomes to limit (or expand) the number of activations that are returned. The value of NULL (default) will return all activations (for each outcome in data). Note that specified values that are not in the weightmatrix will return the initial value without error or warning. Please use getValues for returning all outcomes in the data.
init.value	Value of activations for non-existing connections. Typically set to 0.
normalize	Logical: whether or not the activation is normalized by dividing the total activation by the number of cues. Default is FALSE. If set to TRUE, the activation reflects the average activation per cue.

Value

Vector or list of activation values (see return.list and fun for the specific conditions, and the examples below).

Notes

The outcomes are selected based on the data with events, and not necessarily all outcomes present in the weightmatrices. For example, when the weightmatrices were first trained on another data set, some outcomes may be present in the weightmatrices but not in the current training data. To include these as well, the user can specify these extra outcomes with the argument select.outcomes.

Author(s)

Jacolien van Rij

See Also

getWeightsByCue, getWeightsByOutcome

Other functions for calculating activations: activationsCueSet(), activationsEvents(), activationsMatrix(), getActivations()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
```

```
head(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)
head(train)

# this training data can actually be used train network:
wm <- RWlearning(train)

# Now we calculate the activations for all outcomes
# per event:
activations <- activationsOutcomes(wm, train)
head(activations)

# Now with selection of outcomes (note that 'dog' does
# not occur as outcome in the data):
activations2 <- activationsOutcomes(wm, train,
    select.outcomes = c("plant", "vehicle", "dog"))
head(activations2)
tail(activations2)
```

---

check *Remove empty cues and/or outcomes.*

Description

Remove empty cues and/or outcomes.

Usage

```
check(data, rm = TRUE)
```

Arguments

data	Data frame with columns Cues and Outcomes.
rm	Logical: whether or not to remove empty strings. (Default TRUE).

Details

When rm=FALSE the function returns a code for each row of the data frame indicating whether an empty cue or outcome was detected. The function may return the following values:

0 No empty cues and outcomes were detected in this row.

1 Empty cue(s) but not empty outcomes were detected in this row.

2 Empty outcome(s) but not empty cues were detected in this row.

3 Empty cue(s) AND empty outcome(s) were detected in this row.

Value

data frame or numeric vector (see details)

Author(s)

Jacolien van Rij

Examples

```
test1 <- c("a_b", "a__b", "_a_b", "a_b_", "_a__b_", "___")
## Not run:
    # this returns an error:
    check(test1)

## End(Not run)
# data frame with cues and outcomes:
(dat <- data.frame(Cues=test1, Outcomes=sample(test1), stringsAsFactors=TRUE))
# remove empty:
check(dat)
# only indicating which rows contain empty cues/outcomes:
(test <- check(dat, rm=FALSE))
# check empty cues:
dat[test %in% c(1,3),]
# check empty outcomes:
dat[test %in% c(2,3),]
```

---

checkWM *Check whether cues and outcomes exist in a weight matrix and optionally add.*

Description

Check whether cues and outcomes exist in a weight matrix and optionally add.

Usage

```
checkWM(cues, outcomes, wm)
```

Arguments

cues	A vector with cues.
outcomes	A vector with outcomes.
wm	A matrix with connection weights between cues and outcomes.

Value

A weightmatrix (matrix)

Author(s)

Jacolien van Rij

Examples

```
data(dat)
# create training data:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- 1
train <- createTrainingData(dat)
# train network:
wm <- RWlearning(train)
# inspect weight matrix:
wm[[1]]
# retrieve cues and outcomes from data:
c <- getCues(wm)
o <- getOutcomes(wm)
# add missing cues to initial weight matrix:
checkWM(c, o, wm=wm[[1]])
```

---

createTrainingData *Create event training data from a frequency data frame.*

Description

Create event training data from a frequency data frame.

Usage

```
createTrainingData(
  data,
  nruns = 1,
  random = TRUE,
  within.runs = FALSE,
  add.id = TRUE,
  check = TRUE
)
```

Arguments

data	Data frame with columns Cues and Outcomes, and optionally Frequency.
nruns	Numeric: number of times to run through the data.
random	Logical: randomize the data or not (defaults to TRUE).

within.runs	Logical: apply setting of random to the data _within_ each run (if set to TRUE) or over all data (if set to FALSE). Default setting is FALSE. Note that to randomize the data within seprate runs, both random and within.runs should be set to TRUE.
add.id	Logical: whether or not to add columns that identify events (default is TRUE). The column Item is added to describe each type of event (unless this column already exists in data), the column Run is added when within.runs=TRUE, and the column Trial indicates the order of events within the data frame or within the run (when within.runs=TRUE).
check	Logical: check for empty strings ("") or not (defaults to TRUE). If empty strings are found, they will be removed.

Value

data frame

Author(s)

Jacolien van Rij

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)
head(train)
dim(train)
# the rows should be equal to the sum of frequencies in dat:
sum(dat$Frequency)

# this training data can actually be used train network:
wm <- RWlearning(train)
# inspect weight matrix:
wm[[1]]

# retrieve cues and outcomes from data:
c <- getCues(wm)
o <- getOutcomes(wm)
# add missing cues to initial weight matrix:
checkWM(c, o, wm=wm[[1]])

# ------------------
```

```
# additional possibility for
# simulating experimental designs:
# -------------------
dat$Frequency <- dat$Frequency2
train2 <- createTrainingData(dat, nruns=5)
head(train2)
# items are completely randomized,
# and not equally distributed over the experiment:
train2$Run <- rep(1:5, each=(nrow(train2)/5))
table(train2$Run, train2$Item)
# in this way the items are randomized within each run:
train3 <- createTrainingData(dat, nruns=5, within.runs=TRUE)
head(train3)
table(train3$Run, train3$Item)
# difference in learning (may take some time):
## Not run:
wm2 <- RWlearning(train2)
plotCueWeights(wm2, cue="brown")
wm3 <- RWlearning(train3)
plotCueWeights(wm3, cue="brown")
plotOutcomeWeights(wm3, outcome="animal")

## End(Not run)
```

`createWM` *Create empty weight matrix based on a set of cues and outcomes.*

Description

Create empty weight matrix based on a set of cues and outcomes.

Usage

```
createWM(cues, outcomes, background = NULL, init.value = 0)
```

Arguments

`cues`	A vector with cues.
`outcomes`	A vector with outcomes.
`background`	A string specifying the background cue. Sets this as the value of the background cue for all functions in this R session. If NULL, the current value of the background cue will be used.
`init.value`	Initial value for all connections, typically set to 0.

Value

A weightmatrix (matrix)

Author(s)

Jacolien van Rij

See Also

link{RWlearning}

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)

# the function RWlearning uses createWM to construct a weight matrix:
cues <- getValues(dat$Cues, unique=TRUE)
outcomes <- getValues(dat$Outcomes, unique=TRUE)
createWM(cues=cues, outcomes=outcomes)
# add background cue:
createWM(cues=cues, outcomes=outcomes, background=TRUE)
```

---

cueWindow *Create a 'cue window', for overlapping or continuous cues.*

Description

Create a 'cue window', for overlapping or continuous cues.

Usage

```
cueWindow(
  x,
  n = 1,
  step = 1,
  weights = NULL,
  min = 1,
  max = 100,
  round.values = TRUE,
  split = "_",
  premark = "",
  postmark = "",
  as.numeric = FALSE,
  dec = NULL
)
```

Arguments

x	A vector with numeric cues.
n	Numeric value specifying the window size. If n has two values, the first value indicates the left window size, and the second value the size of the right window.
step	Numeric value, indicating the difference between adjacent cues values. Set to 1 by default.
weights	A vector with weights (round numbers) for multiplying the elements within the window. Defaults to NULL (which will give all cues the same weight).
min	Numeric value specifying the lowest value on the scale. Defaults to 1.
max	Numeric value specifying the maximum value on the scale. Defaults to 100.
round.values	Logical, whether or not to round the values of x to multiples of step on the continuum between min and max. Defaults to TRUE.
split	String, specifying the cue separator. Default value is "_".
premark	String, specifying a character to add before each cue.
postmark	String, specifying a character to add after each cue.
as.numeric	Logical, whether or not to return the numeric values of the window as a list. Default is FALSE (return cue sets as a vector of strings).
dec	Number of decimals for rounding. Defaults to NULL (automatically determined).

Value

A vector of strings (default), or a list with vectors of numbers.

Author(s)

Jacolien van Rij

Examples

```
# generate random sample of cues on continuum of 1-10,
# with sep=1:
set.seed(123)
cues <- round(runif(20, min=0.5, max=10.5),1)

# Note that cues will be converted to rounded numbers
# as round.values=TRUE. With cue window of 3:
cueWindow(cues, n=3, max=10)
# step of 0.5 increases number of neighboring cues:
cueWindow(cues, n=3, max=10, step=.5)
# cue window of 5:
cueWindow(cues, n=5, max=10)
# asymmetrical window:
cueWindow(cues, n=c(2,1), max=10, step=.5)

# non-uniform weights:
cueWindow(cues, n=5, max=10, weights=c(1,2,3,2,1))
```

```
cueWindow(cues, n=2.5, max=10, step=.5, weights=c(1,2,3,2,1))
# left cues have stronger weights:
cueWindow(cues, n=5, max=10, weights=c(3,3,2,1,1))
# adjust weights, so that cue itself is not included:
cueWindow(cues, n=c(2,1), max=10, weights=c(1,1,0,1))
# premarking:
cueWindow(cues, n=2, max=10, weights=c(1,1,1), premark="stimulus")
# numeric output:
cueWindow(cues, n=2, max=10, weights=c(1,2,1), as.numeric=TRUE)
```

`dat` *Simulated learning data.*

Description

Data set for illustrating discrimination learning.

Usage

```
dat
```

Format

A data frame with 36 rows and 5 variables:

`Shape` Shape is the discriminative cue. 6 shapes: cat, rabbit, flower, tree, car, bicycle.

`Color` Color is the nondiscriminative cue. 6 colors: brown, gray, white, yellow, red, blue.

`Category` Three categories: animal, plant, vehicle.

`Frequency1` Different frequency values assigned to the shapes, no difference between colors.

`Frequency2` Different frequency values assigned to the color-shape combinations, no difference between categories.

Author(s)

Jacolien van Rij

edl *Toolbox for Error-Driven Learning Simulations with Two-Layer Networks*

Description

The package 'edl' provides a set of functions that facilitate the evaluation, interpretation, and visualization of small error-driven learning simulations.

Details

Error-driven learning is based on the Widrow & Hoff (1960) learning rule and the Rescorla-Wagner's learning equations (Rescorla & Wagner, 1972), which are also at the core of Naive Discrimination Learning (Baayen et al, 2011). Error-driven can be used to explain bottom-up human learning (Hoppe et al, under revision), but is also at the core of artificial neural networks applications in the form of the Delta rule. This package provides a set of functions for building small-scale simulations to investigate the dynamics of error-driven learning and it's interaction with the structure of the input. For modeling error-driven learning using the Rescorla-Wagner equations the package 'ndl' (Baayen et al, 2011) is available on CRAN at https://cran.r-project.org/package=ndl. However, the package currently only allows tracing of a cue-outcome combination, rather than returning the learned networks. To fill this gap, we implemented a new package with a few functions that facilitate inspection of the networks for small error driven learning simulations. Note that our functions are not optimized for training large data sets (no parallel processing), as they are intended for small scale simulations and course examples. (Consider the python implementation pyndl https://pyndl.readthedocs.io/en/latest/ for that purpose.)

Getting started

- `vignette("edl",package="edl")` - summarizes the core functions for training and visualization of results.

Also available online: https://jacolienvanrij.com/Rpackages/edl/.

References

Dorothée Hoppe, Petra Hendriks, Michael Ramscar, & Jacolien van Rij (under revision): An exploration of error-driven learning in simple two-layer networks from a discriminative learning perspective. Accepted with minor revisions.

Author(s)

Jacolien van Rij and Dorothée Hoppe, originally based on the package 'ndl'.

Maintainer: Jacolien van Rij (<j.c.van.rij@rug.nl>)

University of Groningen, The Netherlands

getActivations *Function to calculate the activations.*

Description

Calculate the activations for all or specific outcomes on the basis of a set of cues. This function combines the various functions to calculate the activations.

Usage

```
getActivations(
  wmlist,
  data = NULL,
  cueset = NULL,
  split = "_",
  select.outcomes = NULL,
  init.value = 0,
  normalize = FALSE
)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights. Or, alternatively, wmlist can be a single weightmatrix.
data	Data frame with columns Cues and Outcomes, specifying one learning event per row (i.e., assuming Frequency=1, as data generated with createTrainingData). Optional argument: when data is provided, the activations will be calculated for each learning event in data (i.e., for the combination of cues and outcomes). When data is set to NULL (no data frame provided), this function will use the cue sets specified in cueset. Use the argument select.outcomes to specify a set of outcomes for which to calculate the activations instead of for the observed outcome(s) only. See examples.
cueset	String, specifying the cue set for which to calculate change in activation. Only will be used when data is set to NULL.
split	String, separator between cues and/or outcomes.
select.outcomes	Optional selection of outcomes to limit (or expand) the number of activations that are returned. See examples for how to use this argument in combination with data and cueset. When data is provided, the value of NULL (default) will only return the activations for each learning event (i.e., only for the observed cues and outcomes). When data is provided, the value TRUE will return the activations for all outcomes in data given the cues observed in the learning events. When cueset is specified, the values of NULL (default) or TRUE will return the activations for all outcomes in wmlist. Note that specified values that are not in the weightmatrix will return the initial value without error or warning. Please use getValues for returning all outcomes in the data.

init.value	Value of activations for non-existing connections. Typically set to 0.
normalize	Logical: whether or not the activation is normalized by dividing the total activation by the number of cues. Default is FALSE. If set to TRUE, the activation reflects the average activation per cue.

Value

List: when data is provided, a list is returned with the outcome activations for each learning event; when cueset is provided, a list is returned with data frames of outcome activations. See examples.

Author(s)

Jacolien van Rij

See Also

getWeightsByCue, getWeightsByOutcome

Other functions for calculating activations: activationsCueSet(), activationsEvents(), activationsMatrix(), activationsOutcomes()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat <- droplevels(dat[1:3,])
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)


# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)
head(train)

# this training data can actually be used train network:
wm <- RWlearning(train)

# With this data we illustrate four different
# ways to retrieve activation changes.

# Situation I: return activations for each event
act1 <- getActivations(wm, data=train)
head(act1)
# plotting activations for each event doesn't provide very
# useful info:
plot(act1$Activation, type='l', ylim=c(0,1), col=alpha(1),
    ylab='Activation')
# these lines may be more interpretable:
```

```
n <- which(act1$Outcomes=="animal")
lines(n, act1$Activation[n], col=alpha(2), lwd=2)
n <- which(act1$Outcomes=="plant")
lines(n, act1$Activation[n], col=alpha(3), lwd=2)

# Situation II: return activations for each events
# for all outcomes
act2 <- getActivations(wm, data=train, select.outcomes=TRUE)
head(act2)

plot(act2$plant, type='l', ylim=c(0,1), col=alpha(1),
    ylab='Activation')
n <- which(act2$Outcomes=="plant")
rug(n, side=1)
lines(n, act2$plant[n], lwd=2, col=alpha(3))
n <- which(act2$Outcomes!="plant")
lines(n, act2$plant[n], lwd=2, col=alpha(2))
legend('topright',
    legend=c("all events", "outcome present", "outcome absent"),
   col=c(1,alpha(3),alpha(2)), lwd=c(1,2,2),
    bty='n')

# Situation III: return activations for specific cuesets
# for all outcomes
act3 <- getActivations(wm, cueset=c("BG_cat_brown", "BG_flower_brown"))
str(act3)

a31 <- act3[["BG_flower_brown"]] # or act3[[1]]
plot(a31$plant, type='l', ylim=c(0,1), col=alpha(1),
    main="BG_flower_brown", ylab='Activation')
lines(a31$animal,col=2)
rug(which(train$Cues == "BG_flower_brown"), side=1)
legend('topright',
    legend=c("plant", "animal"),
   col=c(1,2), lwd=1, bty='n')

# Situation IV: return activations for a static weight matrix
# Note: only with cueset
(final <- getWM(wm))
act4 <- getActivations(final, cueset=unique(train$Cues))
act4
```

---

getCues *Extract cues from list of weightmatrices.*

Description

Extract cues from list of weightmatrices.

Usage

```
getCues(wmlist, extra.check = FALSE)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
extra.check	Logical: whether or not to collect all cues from all weightmatrices in the list. Note that this slows down the process and should not result in different findings. Default is FALSE.

Value

Vector with cues.

Author(s)

Jacolien van Rij

See Also

getOutcomes, getValues

Examples

```
# load example data:
data(dat)
# prepare training data:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
train <- createTrainingData(dat)
# learning:
wm <- RWlearning(train)
# retrieve cues from wm list:
getCues(wm)
# or this version (which takes more time):
system.time({getCues(wm, extra.check=TRUE)})
system.time({getCues(wm)})
```

---

getLambda *Retrieve the lambda values for all or specific outcomes for each learning event.*

Description

For a given set of training data, the lambda values are returned for each or specific outcomes. The values are returned as data frame.

Usage

```
getLambda(data, lambda = 1, split = "_", select.outcomes = NULL)
```

Arguments

data	Data with columns Cues and Outcomes, as generated with createTrainingData.
lambda	Numeric, value of lambda parameter. Defaults to 1.
split	String, separator between cues or outcomes.
select.outcomes	Optional selection of outcomes to limit the number of activations that are returned. The value of NULL (default) will return all activations. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getValues for returning all outcomes in the data.

Value

Data frame.

Author(s)

Jacolien van Rij

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
head(dat)
dim(dat)
test <- getLambda(dat)
# only outcomes that do not occur in data results in 0:
test2 <- getLambda(dat, select.outcomes=c("a", "b", "C"))
```

---

getOutcomes *Extract outcomes from list of weightmatrices.*

Description

Extract outcomes from list of weightmatrices.

Usage

```
getOutcomes(wmlist, extra.check = FALSE)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
extra.check	Logical: whether or not to collect all cues from all weightmatrices in the list. Note that this slows down the process and should not result in different findings. Default is FALSE.

Value

Vector with outcomes.

Author(s)

Jacolien van Rij

See Also

getCues, getValues

Examples

```
# load example data:
data(dat)
# prepare training data:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
train <- createTrainingData(dat)
# learning:
wm <- RWlearning(train)
# retrieve cues from wm list:
getOutcomes(wm)
# or this version (which takes more time):
system.time({getOutcomes(wm, extra.check=TRUE)})
system.time({getOutcomes(wm)})
```

---

getUpdate *Retrieve the weight updates and their change for each learning event.*

Description

For a given set of training data, the weight updating values are returned for each or specific outcomes. The values are returned as data frame.

Usage

```
getUpdate(
  wmlist,
  data,
  select.outcomes = NULL,
  split = "_",
  present.outcome = FALSE
)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
data	Data with columns Cues and Outcomes, as generated with createTrainingData.
select.outcomes	Optional selection of outcomes to limit the number of activations that are returned. The value of NULL (default) will return all activations. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getValues for returning all outcomes in the data.
split	String, separator between cues or outcomes.
present.outcome	Logical: whether or not to output the update for the present output only. Defaults to FALSE. Note that if set to true, this parameter cancels the effect of select.outcomes.

Value

Data frame.

Author(s)

Jacolien van Rij

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat <- droplevels(dat[1:3,])
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)


# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)
head(train)
```

```
# this training data can actually be used train network:
wm <- RWlearning(train)

# retrieve update values for all outcomes:
updates1 <- getUpdate(data=train, wmlist=wm)
head(updates1)

# retrieve update values for observed outcomes:
updates2 <- getUpdate(data=train, wmlist=wm, present.outcome=TRUE)
head(updates2)

# plot:
n <- which("animal" == train$Outcomes)
plot(n, updates2[n], type='l',
    ylim=c(0,.1),
    ylab="Weight updates", xlab="Learning event")
```

`getValues` *Retrieve all cues from a vector of text strings.*

Description

Retrieve all cues from a vector of text strings.

Usage

```
getValues(text, split = "_", unique = FALSE, decreasing = FALSE)
```

Arguments

`text`	A vector with text strings containing cues or outcomes, separated by a symbol specified by `split`.
`split`	separator between cues.
`unique`	Logical: only return unique values (TRUE) or all values (FALSE, default). When unique values are bein returned, they are sorted.
`decreasing`	Logical: sorting in alphabetical order (FALSE, default) or the reverse order (TRUE)? Only applies when `unique` is set to TRUE.

Value

A vector with strings

Author(s)

Jacolien van Rij

See Also

strsplit, sort, unique, getOutcomes, getCues

Examples

```
# load example data:
data(dat)
# prepare training data:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
train <- createTrainingData(dat)

# find all cues in trainingdata:
cues <- getValues(train$Cues)
table(cues)
# find all outcomes in data:
out <- getValues(train$Outcomes)
table(out)
# find (sorted) unique cues and outcomes:
getValues(dat$Cues, unique=TRUE)
getValues(dat$Outcomes, unique=TRUE)
```

---

getWeightsByCue *Extract the change of connection weights between a specific cue and all outcomes.*

Description

Extract the change of connection weights between all cues and a specific outcome. The values are returned as data frame.

Usage

```
getWeightsByCue(wmlist, cue, select.outcomes = NULL, init.value = 0)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
cue	String: cue for which to extract the connection weights.
select.outcomes	Optional selection of outcomes to limit the number of connection weights that are returned. The value of NULL (default) will return all connection weights. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getOutcomes for returning all outcomes from the data, and getValues for returning all outcomes in the data.
init.value	Value of connection weights for non-existing connections. Typically set to 0.

Value

Data frame.

Author(s)

Jacolien van Rij

See Also

plotCueWeights, plotOutcomeWeights, getWeightsByOutcome

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)

# final weight matrix:
getWM(wm)

# Inspect the change in connection weights
# for cue=car
cueweights <- getWeightsByCue(wm, cue='car')
head(cueweights)
emptyPlot(nrow(cueweights), c(-.5,1), h0=0,
    main="Cue='car'", ylab='connection weights', xlab='learning events')
lines(cueweights$vehicle)
lines(cueweights$plant, col='red', lty=4)
lines(cueweights$animal, col='red', lty=2)
legend_margin('topright', legend=c('animal', 'plant', 'vehicle'),
    col=c(2,2,1), lty=c(2,4,1), lwd=1, bty='n')
```

`getWeightsByOutcome` *Extract the change of connection weights between all cues and a specific outcome.*

Description

Extract the change of connection weights between all cues and a specific outcome. The values are returned as data frame.

Usage

```
getWeightsByOutcome(wmlist, outcome, select.cues = NULL, init.value = 0)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
outcome	String: outcome for which to extract the connection weights.
select.cues	Optional selection of cues to limit the number of connection weights that are returned. The value of NULL (default) will return all connection weights. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getCues for returning all cues from the data, and getValues for returning all cues in the data.
init.value	Value of connection weights for non-existing connections. Typically set to 0.

Value

Data frame.

Author(s)

Jacolien van Rij

See Also

plotCueWeights, plotOutcomeWeights, getWeightsByCue

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)

# final weight matrix:
```

```
getWM(wm)

# Inspect the change in connection weights
# for cue=car
outweights <- getWeightsByOutcome(wm, outcome='vehicle')
head(outweights)
emptyPlot(nrow(outweights), range(outweights), h0=0,
    main="Outcome='vehicle'", ylab='connection weights', xlab='learning events')
lines(outweights$BG)
lines(outweights$car, lty=4)
lines(outweights$bicycle, lty=2)
lines(outweights$cat, col=2)
lines(outweights$red, col='blue', lty=4)
lines(outweights$gray, col='blue', lty=2)
legend('bottomright', legend=c('BG', 'car', 'bicycle', 'cat', 'red', 'gray'),
    col=c(1,1,1,2,'blue', 'blue'), lty=c(1,4,2,1,4,2), lwd=1)
```

`getWM` *Retrieve all cues from a vector of text strings.*

Description

Retrieve all cues from a vector of text strings.

Usage

```
getWM(wmlist, event = NULL)
```

Arguments

`wmlist`	A list with weightmatrices for each learning event, generated by `RWlearning`.
`event`	Numeric: for which event to return the weight matrix. Defaults to NULL, which wil return the last weight matrix.

Value

A matrix with connection weights between cues (rows) and outcomes.

Author(s)

Jacolien van Rij

See Also

`RWlearning`, `getWeightsByCue`, `getWeightsByOutcome`

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)

# final weight matrix:
getWM(wm)
# ... which is the same as:
wm[[length(wm)]]
# 25th learning event:
getWM(wm, event=25)
# ... which is the same as:
wm[[25]]
```

---

luceChoice *Function implementing the Luce choice rule.*

Description

Function implementing the Luce choice rule.

Usage

```
luceChoice(value, all)
```

Arguments

value	A positive value specifying a weight or activation (or comparable measure) of the choice option for which the choice probability is calculated
all	A positive array of the weights or activations of all possible choice options, including value

Value

A value between [0,1]

Author(s)

Dorothee Hoppe

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)

# caculate activations of outcomes given the cue set blue_car
red_rabbit <- getActivations(getWM(wm), cueset = "red_rabbit")$red_rabbit

# caculate choice probability of outcomes given the cue set blue_car after
# normalizing with rectified linear unit
luceChoice(red_rabbit["vehicle"], red_rabbit)
luceChoice(red_rabbit["plant"], red_rabbit)
luceChoice(red_rabbit["animal"], red_rabbit)

# note that when some activations are negative, this rule either should not be
# applied, or negative values have to be corrected for, e.g., with applying a
# rectified linear unit (relu)
blue_car <- getActivations(getWM(wm), cueset = "blue_car")$blue_car

## Not run:
# this is should not be done without correction
luceChoice(blue_car["vehicle"], blue_car)
# use, e.g., function relu() on the raw values

## End(Not run)
```

---

plotActivations *Visualize the change of connection weights between a specific outcome and all cues.*

Description

Visualize the activation or the change of activation per event.

Usage

```
plotActivations(
  wmlist,
  cueset,
  split = "_",
  select.outcomes = NULL,
  init.value = 0,
  add.labels = TRUE,
  add = FALSE,
  ...
)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
cueset	String, which contains the combination of cues for which to calculate the activations for per learning event.
split	String, separator between cues.
select.outcomes	Optional selection of outcomes to limit the number of activations that are returned. The value of NULL (default) will return all activations. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getValues for returning all outcomes in the data.
init.value	Value of connection weights for non-existing connections. Typically set to 0.
add.labels	Logical: whether or not to add labels for the lines. Defaults to TRUE, see examples.
add	Logical: whether or not to add the lines to an existing plot. Defaults to FALSE (starting a new plot).
...	Optional graphical arguments, as specified in par. These parameters are forwarded to the functions emptyPlot, lines, and text.

Value

Optionally a list with label specifications is returned, which allows to plot your own labels. This may be helpful for very long labels, and for overlapping lines.

Author(s)

Jacolien van Rij

See Also

plotCueWeights, getWeightsByOutcome, getWeightsByCue

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can be used train network:
wm <- RWlearning(train)

# plot connection weights for cue = 'cat':
plotActivations(wm, cueset="BG_cat_brown")
plotActivations(wm, cueset="BG_cat")

# plot your own labels:
labels <- plotActivations(wm, cues="BG_cat", add.labels=FALSE)
legend_margin('topright', legend=labels$labels, col=labels$col,
    lwd=1, bty='n')

# change color and select outcomes:
out <- getValues(train$Outcomes, unique=TRUE)
out <- out[! out %in% "animal"]
labels <- plotActivations(wm, cues="BG_cat",
    select.outcome=out, add.labels=FALSE,
   ylim=c(-.25,1),col=alpha(1))
lab2 <- plotActivations(wm, cues="BG_cat", add.labels=FALSE,
    select.outcomes="animal", add=TRUE, col=2, lwd=2, xpd=TRUE)
legend('topright', legend=c("animal", labels$labels),
    col=c(lab2$col, labels$col), lwd=c(lab2$lwd, labels$lwd),
    lty=c(lab2$lty, labels$lty), bty="n")
```

`plotCueWeights` *Visualize the change of connection weights between a specific cue and all outcomes.*

Description

Visualize the change of connection weights between a specific cue and all outcomes.

Usage

```
plotCueWeights(
  wmlist,
  cue,
```

```
  select.outcomes = NULL,
  init.value = 0,
  add.labels = TRUE,
  add = FALSE,
  ...
)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
cue	String: cue for which to extract the connection weights.
select.outcomes	Optional selection of outcomes to limit the number of connection weights that are returned. The value of NULL (default) will return all connection weights. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getOutcomes for returning all outcomes from the data, and getValues for returning all outcomes in the data.
init.value	Value of connection weights for non-existing connections. Typically set to 0.
add.labels	Logical: whether or not to add labels for the lines. Defaults to TRUE, see examples.
add	Logical: whether or not to add the lines to an existing plot. Defaults to FALSE (starting a new plot).
...	Optional graphical arguments, as specified in par. These parameters are forwarded to the functions emptyPlot, lines, and text.

Value

Optionally a list with label specifications is returned, which allows to plot your own labels. This may be helpful for very long labels, and for overlapping lines.

Author(s)

Jacolien van Rij

See Also

plotOutcomeWeights, getWeightsByOutcome, getWeightsByCue

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)
```

```
# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)

# plot connection weights for cue = 'car':
plotCueWeights(wm, cue="car")

# plot your own labels:
labels <- plotCueWeights(wm, cue="car", add.labels=FALSE)
legend_margin('topright', legend=labels$labels, col=labels$col,
    lwd=1, bty='n')

# change color and select outcomes:
out <- getValues(train$Outcomes, unique=TRUE)
out <- out[out != "vehicle"]
labels <- plotCueWeights(wm, cue="car", add.labels=FALSE,
    col=alphaPalette(c(1,2), f.seq=rep(.5,length(out))),
    select.outcomes=out)
lab2 <- plotCueWeights(wm, cue="car", add.labels=FALSE,
    select.outcomes="vehicle", add=TRUE, col=1, lwd=2)
legend_margin('topright', legend=c(labels$labels, "vehicle"),
    col=c(labels$col, lab2$col), lwd=c(labels$lwd, lab2$lwd),
    lty=c(labels$lty, lab2$lty))
```

---

plotNetwork *Return strong weights.*

Description

Return strong weights.

Usage

```
plotNetwork(
  wm,
  select.outcomes = NULL,
  select.cues = NULL,
  color = NULL,
  zlim = NULL,
  add.color.legend = TRUE,
  ...
)
```

Arguments

wm	a weightmatrix (matrix, not list) with connection weights between cues (rows) and outcomes (columns).
select.outcomes	Optional selection of outcomes to limit the number of connection weights that are returned. The value of NULL (default) will return all connection weights. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getOutcomes for returning all cues from the data, and getValues for returning all cues in the data.
select.cues	Optional selection of cues to limit the number of connection weights that are returned. The value of NULL (default) will return all connection weights. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getCues for returning all cues from the data, and getValues for returning all cues in the data.
color	The color scheme to use for plots, a list of colors such as that generated by rainbow, heat.colors, colors, topo.colors, terrain.colors or similar functions. Alternatively a vector with some colors can be provided for a custom color palette.
zlim	z-limits for the plot.
add.color.legend	Logical: whether or not to add a color legend (see also gradientLegend).
...	Optional graphical arguments, as specified in par. These parameters are forwarded to the functions emptyPlot, lines, and text.

Value

No return value

Author(s)

Jacolien van Rij

---

plotOutcomeWeights *Visualize the change of connection weights between a specific outcome and all cues.*

Description

Visualize the change of connection weights between a specific outcome and all cues.

Usage

```
plotOutcomeWeights(
  wmlist,
  outcome,
  select.cues = NULL,
  init.value = 0,
  add.labels = TRUE,
  add = FALSE,
  ...
)
```

Arguments

wmlist	A list with weightmatrices, generated by RWlearning or updateWeights.
outcome	String: outcome for which to extract the connection weights.
select.cues	Optional selection of outcomes to limit the number of connection weights that are returned. The value of NULL (default) will return all connection weights. Note that specified values that are not in the weightmatrices will return the initial value without error or warning. Please use getOutcomes for returning all outcomes from the data, and getValues for returning all outcomes in the data.
init.value	Value of connection weights for non-existing connections. Typically set to 0.
add.labels	Logical: whether or not to add labels for the lines. Defaults to TRUE, see examples.
add	Logical: whether or not to add the lines to an existing plot. Defaults to FALSE (starting a new plot).
...	Optional graphical arguments, as specified in par. These parameters are forwarded to the functions emptyPlot, lines, and text.

Value

Optionally a list with label specifications is returned, which allows to plot your own labels. This may be helpful for very long labels, and for overlapping lines.

Author(s)

Jacolien van Rij

See Also

plotCueWeights, getWeightsByOutcome, getWeightsByCue

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
```

```
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)

# plot connection weights for cue = 'car':
plotOutcomeWeights(wm, outcome="vehicle")


# plot your own labels:
labels <- plotOutcomeWeights(wm, outcome="vehicle", add.labels=FALSE)
legend_margin('topright', legend=labels$labels, col=labels$col,
    lwd=1, bty='n')

# change color and select outcomes:
out <- getValues(train$Cues, unique=TRUE)
out <- out[! out %in% c("car", "bicycle")]
labels <- plotOutcomeWeights(wm, outcome="vehicle", add.labels=FALSE,
   ylim=c(-.5,1),col=alpha(1), select.cues=out)
lab2 <- plotOutcomeWeights(wm, outcome="vehicle", add.labels=FALSE,
    select.cues=c("car", "bicycle"), add=TRUE, col=2, lwd=2, xpd=TRUE)
legend_margin('topright', legend=c(labels$labels, c("car", "bicycle")),
    col=c(labels$col, lab2$col), lwd=c(labels$lwd, lab2$lwd),
    lty=c(labels$lty, lab2$lty))
```

`RWlearning` *Function implementing the Rescorla-Wagner learning.*

Description

Function implementing the Rescorla-Wagner learning.

Usage

```
RWlearning(
  data,
  wm = NULL,
  eta = 0.01,
  lambda = 1,
  alpha = 0.1,
  beta1 = 0.1,
  beta2 = 0.1,
```

```
  progress = TRUE,
  ...
)
```

Arguments

data	A data frame with columns Cues and Outcomes.
wm	A weightmatrix of class matrix, or a list with weight matrices. If not provided a new weightmatrix is returned. Note that the cues and outcomes do not necessarily need to be available as cues and outcomes in the weightmatrix: if not present, they will be added.
eta	Learning parameter, typically set to 0.01. If eta is not specified and set to the value NULL, the values of alpha, beta1, and beta2 determine the learning rate. However, changing these settings is generally not very useful (see Hoppe et al, submitted).
lambda	Constant constraining the connection strength.
alpha	Learning parameter (scaling both positive and negative evidence adjustments), typically set to 0.1. Only used if eta=NULL.
beta1	Learning parameter for positive evidence, typically set to 0.1. Only used if eta=NULL.
beta2	Learning parameter for negative evidence, typically set to 0.1. Only used if eta=NULL.
progress	Logical: whether or not showing a progress bar (may slow down the process).
...	Parameters for the function getValues.

Value

A list with weightmatrices for each learning event.

Author(s)

Jacolien van Rij

See Also

RescorlaWagner, updateWeights

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)
```

```
# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)

# in R markdown or knitr reports the progress bar should be turned off:
wm <- RWlearning(train, progress=FALSE)

# Learning in steps is also possible:
wm <- RWlearning(train[1:20,])
getWM(wm)
length(wm)

train[21,c("Cues", "Outcomes")]
wm <- RWlearning(train[21,], wm=wm)
getWM(wm)
length(wm)
```

`RWlearningMatrix` *Function implementing the Rescorla-Wagner learning.*

Description

Function implementing the Rescorla-Wagner learning.

Usage

```
RWlearningMatrix(
  data,
  wm = NULL,
  alpha = 0.1,
  lambda = 1,
  beta1 = 0.1,
  beta2 = 0.1,
  progress = TRUE,
  ...
)
```

Arguments

`data`	A data frame with columns `Cues` and `Outcomes`.
`wm`	A weightmatrix of class matrix, or a list with weight matrices. If not provided a new weightmatrix is returned. Note that the cues and outcomes do not necessarily need to be available as cues and outcomes in the weightmatrix: if not present, they will be added.

alpha	Learning parameter (scaling both positive and negative evidence adjustments), typically set to 0.1.
lambda	Constant constraining the connection strength.
beta1	Learning parameter for positive evidence, typically set to 0.1.
beta2	Learning parameter for negative evidence, typically set to 0.1.
progress	Logical: whether or not showing a progress bar (may slow down the process).
...	Parameters for the function getValues.

Value

A weightmatrix.

Author(s)

Jacolien van Rij

See Also

RescorlaWagner, updateWeights

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm1 <- RWlearningMatrix(train)
# comparison with a list:
wm2 <- RWlearning(train)
length(wm2)
getWM(wm2)

# in R markdown or knitr reports the progress bar should be turned off:
wm <- RWlearningMatrix(train, progress=FALSE)

# Learning in steps is also possible:
wm <- RWlearningMatrix(train[1:20,])

train[21,c("Cues", "Outcomes")]
wm <- RWlearningMatrix(train[21,], wm=wm)
```

RWlearningNoCueCompetition *Function implementing the Rescorla-Wagner learning equations without cue competition (for illustration purposes).*

Description

Function implementing the Rescorla-Wagner learning equations without cue competition (for illustration purposes).

Usage

```
RWlearningNoCueCompetition(
  data,
  wm = NULL,
  eta = 0.01,
  lambda = 1,
  alpha = 0.1,
  beta1 = 0.1,
  beta2 = 0.1,
  progress = TRUE,
  ...
)
```

Arguments

data	A data frame with columns Cues and Outcomes.
wm	A weightmatrix of class matrix, or a list with weight matrices. If not provided a new weightmatrix is returned. Note that the cues and outcomes do not necessarily need to be available as cues and outcomes in the weightmatrix: if not present, they will be added.
eta	Learning parameter, typically set to 0.01. If eta is not specified and set to the value NULL, the values of alpha, beta1, and beta2 determine the learning rate. However, changing these settings is generally not very useful (see Hoppe et al, submitted).
lambda	Constant constraining the connection strength.
alpha	Learning parameter (scaling both positive and negative evidence adjustments), typically set to 0.1. Only used if eta=NULL.
beta1	Learning parameter for positive evidence, typically set to 0.1. Only used if eta=NULL.
beta2	Learning parameter for negative evidence, typically set to 0.1. Only used if eta=NULL.
progress	Logical: whether or not showing a progress bar (may slow down the process).
...	Parameters for the function getValues.

Value

A list with weightmatrices for each learning event.

Author(s)

Dorothee Hoppe

See Also

RescorlaWagner, updateWeightsNoCueCompetition

Other functions for explaining error-driven learning: RWlearningNoOutcomeCompetition(), updateWeightsNoCueCompeti
updateWeightsNoOutcomeCompetition()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearningNoCueCompetition(train)

# in R markdown or knitr reports the progress bar should be turned off:
wm <- RWlearningNoCueCompetition(train, progress=FALSE)

# Learning in steps is also possible:
wm <- RWlearningNoCueCompetition(train[1:20,])
getWM(wm)
length(wm)

train[21,c("Cues", "Outcomes")]
wm <- RWlearningNoCueCompetition(train[21,], wm=wm)
getWM(wm)
length(wm)
```

RWlearningNoOutcomeCompetition *Function implementing the Rescorla-Wagner learning equetions without outcome competition (for illustration purposes).*

Description

Function implementing the Rescorla-Wagner learning equetions without outcome competition (for illustration purposes).

Usage

```
RWlearningNoOutcomeCompetition(
  data,
  wm = NULL,
  eta = 0.01,
  lambda = 1,
  alpha = 0.1,
  beta1 = 0.1,
  beta2 = 0.1,
  progress = TRUE,
  ...
)
```

Arguments

data	A data frame with columns Cues and Outcomes.
wm	A weightmatrix of class matrix, or a list with weight matrices. If not provided a new weightmatrix is returned. Note that the cues and outcomes do not necessarily need to be available as cues and outcomes in the weightmatrix: if not present, they will be added.
eta	Learning parameter, typically set to 0.01. If eta is not specified and set to the value NULL, the values of alpha, beta1, and beta2 determine the learning rate. However, changing these settings is generally not very useful (see Hoppe et al, submitted).
lambda	Constant constraining the connection strength.
alpha	Learning parameter (scaling both positive and negative evidence adjustments), typically set to 0.1. Only used if eta=NULL.
beta1	Learning parameter for positive evidence, typically set to 0.1. Only used if eta=NULL.
beta2	Learning parameter for negative evidence, typically set to 0.1. Only used if eta=NULL.
progress	Logical: whether or not showing a progress bar (may slow down the process).
...	Parameters for the function getValues.

Value

A list with weightmatrices for each learning event.

Author(s)

Dorothee Hoppe

See Also

RescorlaWagner, updateWeightsNoOutcomeCompetition

Other functions for explaining error-driven learning: RWlearningNoCueCompetition(), updateWeightsNoCueCompetitio
updateWeightsNoOutcomeCompetition()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearningNoOutcomeCompetition(train)

# in R markdown or knitr reports the progress bar should be turned off:
wm <- RWlearningNoOutcomeCompetition(train, progress=FALSE)

# Learning in steps is also possible:
wm <- RWlearningNoOutcomeCompetition(train[1:20,])
getWM(wm)
length(wm)

train[21,c("Cues", "Outcomes")]
wm <- RWlearningNoOutcomeCompetition(train[21,], wm=wm)
getWM(wm)
length(wm)
```

setBackground *Set value background cue.*

Description

Set value background cue.

Usage

```
setBackground(value)
```

Arguments

value	A string specifying the background cue. NULL or FALSE or a number < 0 indicates to remove the background cue, whereas the values TRUE or a number > 0 set the value to "***", and a string can specify the value for the background cue.

Value

No return value

updateWeights *Function implementing the Rescorla-Wagner learning for a single learning event.*

Description

Function implementing the Rescorla-Wagner learning for a single learning event. A set of cues and outcomes are provided, and a weightmatrix that needs to be updated.

Usage

```
updateWeights(
  cur.cues,
  cur.outcomes,
  wm = NULL,
  eta = 0.01,
  lambda = 1,
  alpha = 0.1,
  beta1 = 0.1,
  beta2 = 0.1
)
```

Arguments

cur.cues	A vector with cues.
cur.outcomes	A vector with outcomes.
wm	A weightmatrix of class matrix. If not provided a new weightmatrix is returned. Note that the cues and outcomes do not necessarily need to be available as cues and outcomes in the weightmatrix: if not present, they will be added.
eta	Learning parameter, typically set to 0.01. If eta is not specified and set to the value NULL, the values of alpha, beta1, and beta2 determine the learning rate. However, changing these settings is generally not very useful (see Hoppe et al, submitted).
lambda	Constant constraining the connection strength.
alpha	Learning parameter (scaling both positive and negative evidence adjustments), typically set to 0.1.
beta1	Learning parameter for positive evidence, typically set to 0.1.
beta2	Learning parameter for negative evidence, typically set to 0.1.

Value

A weightmatrix (matrix)

Author(s)

Jacolien van Rij, based on RescorlaWagner

See Also

RescorlaWagner, RWlearning

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearning(train)
# retrieve trained network:
new <- getWM(wm)
```

```
train2 <- createTrainingData(dat)
updateWeights(getValues(train2$Cues[1]),
    getValues(train2$Outcomes[1]), wm=new)

# comparison between eta and alpha, beta1, beta2:
check.cues <- c("BG", "car", "red")
new[check.cues,]
tmp1 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new)
tmp2 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new, eta=NULL)
tmp3 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new, beta1=0.2)
tmp4 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new, eta=NULL, beta1=0.2)
# these two should be the same:
tmp1[check.cues,]
tmp2[check.cues,]
# now we change beta2, but this does not change anything,
# because eta is being used:
tmp3[check.cues,]
# when we turn eta off, beta2 changes the values:
tmp4[check.cues,]
```

`updateWeightsNoCueCompetition` *Function implementing the Rescorla-Wagner learning equations without cue competition for a single learning event.*

Description

Function implementing the Rescorla-Wagner learning equations without cue competition (for illustration purposes) for a single learning event. A set of cues and outcomes are provided, and a weightmatrix that needs to be updated.

Usage

```
updateWeightsNoCueCompetition(
  cur.cues,
  cur.outcomes,
  wm = NULL,
  eta = 0.01,
  lambda = 1,
  alpha = 0.1,
  beta1 = 0.1,
  beta2 = 0.1
)
```

Arguments

cur.cues	A vector with cues.
cur.outcomes	A vector with outcomes.
wm	A weightmatrix of class matrix. If not provided a new weightmatrix is returned. Note that the cues and outcomes do not necessarily need to be available as cues and outcomes in the weightmatrix: if not present, they will be added.
eta	Learning parameter, typically set to 0.01. If eta is not specified and set to the value NULL, the values of alpha, beta1, and beta2 determine the learning rate. However, changing these settings is generally not very useful (see Hoppe et al, submitted).
lambda	Constant constraining the connection strength.
alpha	Learning parameter (scaling both positive and negative evidence adjustments), typically set to 0.1.
beta1	Learning parameter for positive evidence, typically set to 0.1.
beta2	Learning parameter for negative evidence, typically set to 0.1.

Value

A weightmatrix (matrix)

Author(s)

Dorothee Hoppe, based on RescorlaWagner

See Also

RescorlaWagner, RWlearning

Other functions for explaining error-driven learning: RWlearningNoCueCompetition(), RWlearningNoOutcomeCompetitio
updateWeightsNoOutcomeCompetition()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)

# this training data can actually be used train network:
wm <- RWlearningNoCueCompetition(train)
# retrieve trained network:
```

```
new <- getWM(wm)

train2 <- createTrainingData(dat)
updateWeightsNoCueCompetition(getValues(train2$Cues[1]),
    getValues(train2$Outcomes[1]), wm=new)

# comparison between eta and alpha, beta1, beta2:
check.cues <- c("BG", "car", "red")
new[check.cues,]
tmp1 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new)
tmp2 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new, eta=NULL)
tmp3 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new, beta1=0.2)
tmp4 <- updateWeights(check.cues,
    c("vehicle", "animal"), wm=new, eta=NULL, beta1=0.2)
# these two should be the same:
tmp1[check.cues,]
tmp2[check.cues,]
# now we change beta2, but this does not change anything,
# because eta is being used:
tmp3[check.cues,]
# when we turn eta off, beta2 changes the values:
tmp4[check.cues,]
```

`updateWeightsNoOutcomeCompetition` *Function implementing the Rescorla-Wagner learning equations without outcome competition (for illustration purposes) for a single learning event.*

Description

Function implementing the Rescorla-Wagner learning equations without outcome competition (for illustration purposes) for a single learning event. A set of cues and outcomes are provided, and a weightmatrix that needs to be updated.

Usage

```
updateWeightsNoOutcomeCompetition(
  cur.cues,
  cur.outcomes,
  wm = NULL,
  eta = 0.01,
  lambda = 1,
  alpha = 0.1,
  beta1 = 0.1,
```

```
  beta2 = 0.1
)
```

Arguments

cur.cues	A vector with cues.
cur.outcomes	A vector with outcomes.
wm	A weightmatrix of class matrix. If not provided a new weightmatrix is returned. Note that the cues and outcomes do not necessarily need to be available as cues and outcomes in the weightmatrix: if not present, they will be added.
eta	Learning parameter, typically set to 0.01. If eta is not specified and set to the value NULL, the values of alpha, beta1, and beta2 determine the learning rate. However, changing these settings is generally not very useful (see Hoppe et al, submitted).
lambda	Constant constraining the connection strength.
alpha	Learning parameter (scaling both positive and negative evidence adjustments), typically set to 0.1.
beta1	Learning parameter for positive evidence, typically set to 0.1.
beta2	Learning parameter for negative evidence, typically set to 0.1.

Value

A weightmatrix (matrix)

Author(s)

Dorothee Hoppe, based on RescorlaWagner

See Also

RescorlaWagner, RWlearning

Other functions for explaining error-driven learning: RWlearningNoCueCompetition(), RWlearningNoOutcomeCompetitio
updateWeightsNoCueCompetition()

Examples

```
# load example data:
data(dat)

# add obligatory columns Cues, Outcomes, and Frequency:
dat$Cues <- paste("BG", dat$Shape, dat$Color, sep="_")
dat$Outcomes <- dat$Category
dat$Frequency <- dat$Frequency1
head(dat)
dim(dat)

# now use createTrainingData to sample from the specified frequencies:
train <- createTrainingData(dat)
```

```
# this training data can actually be used train network:
wm <- RWlearningNoOutcomeCompetition(train)
# retrieve trained network:
new <- getWM(wm)

train2 <- createTrainingData(dat)
updateWeightsNoOutcomeCompetition(getValues(train2$Cues[1]),
    getValues(train2$Outcomes[1]), wm=new)

# comparison between eta and alpha, beta1, beta2:
check.cues <- c("BG", "car", "red")
new[check.cues,]
tmp1 <- updateWeightsNoOutcomeCompetition(check.cues,
    c("vehicle", "animal"), wm=new)
tmp2 <- updateWeightsNoOutcomeCompetition(check.cues,
    c("vehicle", "animal"), wm=new, eta=NULL)
tmp3 <- updateWeightsNoOutcomeCompetition(check.cues,
    c("vehicle", "animal"), wm=new, beta1=0.2)
tmp4 <- updateWeightsNoOutcomeCompetition(check.cues,
    c("vehicle", "animal"), wm=new, eta=NULL, beta1=0.2)
# these two should be the same:
tmp1[check.cues,]
tmp2[check.cues,]
# now we change beta2, but this does not change anything,
# because eta is being used:
tmp3[check.cues,]
# when we turn eta off, beta2 changes the values:
tmp4[check.cues,]
```

Index