Package: leabRa (via r-universe)

October 30, 2024

Type Package

Title The Artificial Neural Networks Algorithm Leabra

Version 0.1.0.9000

Description The algorithm Leabra (local error driven and associative biologically realistic algorithm) allows for the construction of artificial neural networks that are biologically realistic and balance supervised and unsupervised learning within a single framework. This package is based on the 'MATLAB' version by Sergio Verduzco-Flores, which in turn was based on the description of the algorithm by Randall O'Reilly (1996) <ftp://grey.colorado.edu/pub/oreilly/thesis/oreilly_thesis.all.pdf>. For more general (not 'R' specific) information on the algorithm Leabra see <https://grey.colorado.edu/emergent/index.php/Leabra>.

Depends R (>= 2.10)

License GPL-2

Encoding UTF-8

LazyData true

Imports plyr (>= 1.8.4), R6 (>= 2.2.1)

Collate 'data.R' 'misc.R' 'unit.R' 'layer.R' 'network.R' 'leabRa.R'

RoxygenNote 6.0.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/johannes-titz/leabRa

BugReports https://github.com/johannes-titz/leabRa/issues

Repository https://johannes-titz.r-universe.dev

RemoteUrl https://github.com/johannes-titz/leabra

RemoteRef HEAD

RemoteSha 237bc5c67c81fcd24e7dac895af3106bbad9974a

Contents

`animals` *Small example data set for self-organized artificial neural networks*

Description

A small dataset describing 10 animals represented by 6 features that are either present (1) or absent (0) for demonstrating self-organized learning in artificial neural networks.

Usage

```
animals
```

Format

A data frame with 10 rows and 6 variables describing 10 different animals with 6 feature vectors that are either present (1) or absent (0).

Source

Knight, K. (1990). Connectionist ideas and algorithms. *Communications of the ACM*, *33*(11), 59–74.

`layer` *Leabra layer class*

Description

This class simulates a biologically realistic layer of neurons in the Leabra framework. It consists of several `unit` objects in the variable (field) `units` and some layer-specific variables.

Usage

```
layer
```

Format

`R6Class` object

Value

Object of R6Class with methods for calculating changes of activation in a layer of neurons.

Fields

units A list with all unit objects of the layer.

avg_act The average activation of all units in the layer (this is an active binding).

n Number of units in layer.

weights A receiving x sending weight matrix, where the receiving units (rows) has the current weight values for the sending units (columns). The weights will be set by the network object, because they depend on the connection to other layers.

ce_weights Sigmoidal contrast-enhanced version of the weight matrix weights. These weights will be set by the network object.

layer_number Layer number in network (this is 1 if you create a layer on your own, without the network class).

Methods

new(dim, g_i_gain = 2) Creates an object of this class with default parameters.

dim A pair of numbers giving the dimensions (rows and columns) of the layer.

g_i_gain Gain factor for inhibitory conductance, if you want less activation in a layer, set this higher.

get_unit_acts() Returns a vector with the activations of all units of a layer.

get_unit_scaled_acts() Returns a vector with the scaled activations of all units of a layer. Scaling is done with recip_avg_act_n, a reciprocal function of the number of active units.

cycle(intern_input, ext_input) Iterates one time step with layer object.

intern_input Vector with inputs from all other layers. Each input has already been scaled by a reciprocal function of the number of active units (recip_avg_act_n) of the sending layer and by the connection strength between the receiving and sending layer. The weight matrix ce_weights is multiplied with this input vector to get the excitatory conductance for each unit in the layer.

ext_input Vector with inputs not coming from another layer, with length equal to the number of units in this layer. If empty (NULL), no external inputs are processed. If the external inputs are not clamped, this is actually an excitatory conductance value, which is added to the conductance produced by the internal input and weight matrix.

clamp_cycle(activations) Iterates one time step with layer object with clamped activations, meaning that activations are instantaneously set without time integration.

activations Activations you want to clamp to the units in the layer.

get_unit_act_avgs() Returns a list with the short, medium and long term activation averages of all units in the layer as vectors. The super short term average is not returned, and the long term average is not updated before being returned (this is done in the function chg_wt() with the methodupdt_unit_avg_l). These averages are used by the network class to calculate weight changes.

`updt_unit_avg_l()` Updates the long-term average (`avg_l`) of all units in the layer, usually done after a plus phase.

`updt_recip_avg_act_n()` Updates the `avg_act_inert` and `recip_avg_act_n` variables, these variables update before the weights are changed instead of cycle by cycle. This version of the function assumes full connectivity between layers.

`reset(random = FALSE)` Sets the activation and activation averages of all units to 0. Used to begin trials from a stationary point.

`random` Logical variable, if TRUE the activations are set randomly between .05 and .95 for every unit instead of 0.

`set_ce_weights()` Sets contrast enhanced weight values.

`get_unit_vars(show_dynamics = TRUE, show_constants = FALSE)` Returns a data frame with the current state of all unit variables in the layer. Every row is a unit. You can choose whether you want dynamic values and / or constant values. This might be useful if you want to analyze what happens in units of a layer, which would otherwise not be possible, because most of the variables (fields) are private in the unit class.

`show_dynamics` Should dynamic values be shown? Default is TRUE.

`show_constants` Should constant values be shown? Default is FALSE.

`get_layer_vars(show_dynamics = TRUE, show_constants = FALSE)` Returns a data frame with 1 row with the current state of the variables in the layer. You can choose whether you want dynamic values and / or constant values. This might be useful if you want to analyze what happens in a layer, which would otherwise not be possible, because some of the variables (fields) are private in the layer class.

`show_dynamics` Should dynamic values be shown? Default is TRUE.

`show_constants` Should constant values be shown? Default is FALSE.

References

O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., and Contributors (2016). Computational Cognitive Neuroscience. Wiki Book, 3rd (partial) Edition. URL: http://ccnbook.colorado.edu

Have also a look at https://grey.colorado.edu/emergent/index.php/Leabra (especially the link to the 'MATLAB' code) and https://en.wikipedia.org/wiki/Leabra

Examples

```
l <- layer$new(c(5, 5)) # create a 5 x 5 layer with default leabra values

l$g_e_avg # private values cannot be accessed
# if you want to see alle variables, you need to use the function
l$get_layer_vars(show_dynamics = TRUE, show_constants = TRUE)
# if you want to see a summary of all units without constant values
l$get_unit_vars(show_dynamics = TRUE, show_constants = FALSE)

# let us clamp the activation of the 25 units to some random values between
# 0.05 and 0.95
l <- layer$new(c(5, 5))
activations <- runif(25, 0.05, .95)
l$avg_act
```

```
l$clamp_cycle(activations)
l$avg_act
# what happened to the unit activations?
l$get_unit_acts()
# compare with activations
activations
# scaled activations are scaled by the average activation of the layer and
# should be smaller
l$get_unit_scaled_acts()
```

`leabRa` *leabRa: A package for biologically realistic neural networks based on Leabra*

Description

The Leabra package provides three classes to construct artificial neural networks: unit, layer and network.

Details

Note that the classes in this package are `R6Class` classes.

For further information check out the vignette with vignette("leabRa").

`network` *Leabra network class*

Description

Class to simulate a biologically realistic network of neurons (`units`) organized in `layers`

Usage

```
network
```

Format

`R6Class` object.

Details

This class simulates a biologically realistic artificial neuronal network in the Leabra framework (e.g. O'Reilly et al., 2016). It consists of several `layer` objects in the variable (field) `layers` and some network-specific variables.

Value

Object of R6Class with methods for calculating changes of activation in a network of neurons organized in layers.

Fields

`layers` A list of layer objects.

`lrate` Learning rate, gain factor for how much the connection weights should change when the method `chg_wt()` is called.

Methods

`new(dim_lays, cxn, g_i_gain = rep(2, length(dim_lays)), w_init_fun = function(x) runif(x, 0.3, 0.7), w_in`
Creates an object of this class with default parameters.

- `dim_lays` List of number pairs for rows and columns of the layers, e.g. `list(c(5, 5), c(10, 10), c(5, 5))` for a 25 x 100 x 25 network.
- `cxn` Matrix specifying connection strength between layers, if layer j sends projections to layer i, then `cxn[i, j]` = `strength` > `0` and 0 otherwise. Strength specifies the relative strength of that connection with respect to the other projections to layer i.
- `g_i_gain` Vector of inhibitory conductance gain values for every layer. This comes in handy to control overall level of inhibition of specific layers. Default is 2 for every layer.
- `w_init_fun` Function that specifies how random weights should be created, default value is to generate weights between 0.3 and 0.7 from a uniform distribution. It is close to 0.5 because the weights are contrast enhanced internally, so will actually be in a wider range.
- `w_init` Matrix of initial weight matrices (like a cell array in 'MATLAB'), this is analogous to `cxn`, i.e. `w_init[i, j]` contains the initial weight matrix for the connection from layer j to i. If you specify a `w_init`, `w_init_fun` is ignored. You can use this if you want to have full control over the weight matrix.

`cycle(ext_inputs, clamp_inp)` Iterates one time step with the network object with external inputs.

- `ext_inputs` A list of matrices; ext_inputs[[i]] is a matrix that for layer i specifies the external input to each of its units. An empty matrix (`NULL`) denotes no input to that layer. You can also use a vector instead of a matrix, because the matrix is vectorized anyway.
- `clamp_inp` Logical variable; TRUE: external inputs are clamped to the activities of the units in the layers, FALSE: external inputs are summed to excitatory conductance values (note: not to the activation) of the units in the layers.

`chg_wt()` Changes the weights of the entire network with the XCAL learning equation.

`reset(random = F)` Sets the activation of all units in all layers to 0, and sets all activation time averages to that value. Used to begin trials from a random stationary point. The activation values may also be set to a random value.

- `random` Logical variable, if TRUE set activation randomly between .05 and .95, if FALSE set activation to 0, which is the default.

`create_inputs(which_layers, n_inputs, prop_active = 0.3)` Returns a list of length `n_inputs` with random input patterns (either 0.05, or. 0.95) for the layers specified in `which_layers`. All other layers will have an input of NULL.

which_layers Vector of layer numbers, for which you want to create random inputs.

n_inputs Single numeric value, how many inputs should be created.

prop_active Average proportion of active units in the input patterns, default is 0.3.

learn_error_driven(inputs_minus, inputs_plus, lrate = 0.1, n_cycles_minus = 50, n_cycles_plus = 25, rando
Learns to associate specific inputs with specific outputs in an error-driven fashion.

inputs_minus Inputs for the minus phase (the to be learned output is not presented).

inputs_plus Inputs for the plus phase (the to be learned output is presented).

lrate Learning rate, default is 0.1.

n_cycles_minus How many cycles to run in the minus phase, default is 50.

n_cycles_plus How many cycles to run in the plus phase, default is 25.

random_order Should the order of stimulus presentation be randomized? Default is FALSE.

show_progress Whether progress of learning should be shown. Default is TRUE.

learn_self_organized(inputs, lrate = 0.1, n_cycles = 50, random_order = FALSE, show_progress = TRUE)
Learns to categorize inputs in a self-organized fashion.

inputs Inputs for cycling.

lrate Learning rate, default is 0.1.

n_cycles How many cycles to run, default is 50.

random_order Should the order of stimulus presentation be randomized? Default is FALSE.

show_progress Whether progress of learning should be shown. Default is TRUE.

test_inputs = function(inputs, n_cycles = 50, show_progress = FALSE) Tests inputs without changing the weights (without learning). This is usually done after several learning runs.

inputs Inputs for cycling.

n_cycles How many cycles to run, default is 50.

show_progress Whether progress of learning should be shown. Default is FALSE.

mad_per_epoch(outs_per_epoch, inputs_plus, layer) Calculates mean absolute distance for two lists of activations for a specific layer. This can be used to compare whether the network has learned what it was supposed to learn.

outs_per_epoch Output activations for entire network for each trial for every epoch. This is what the network produced on its own.

inputs_plus Original inputs for the plus phase. This is what the network was supposed to learn.

layer Single numeric, for which layer to calculate the mean absolute distance. Usually, this is the "output" layer.

set_weights(weights) Sets new weights for entire network, useful to load networks that have already learned and thus very specific weights.

weights Matrix of matrices (like a cell array in 'MATLAB') with new weight values.

get_weights() Returns the complete weight matrix, w[i, j] contains the weight matrix for the projections from layer j to layer i. Note that this is a matrix of matrices (equivalent to a 'MATLAB' cell array).

get_layer_and_unit_vars(show_dynamics = T, show_constants = F) Returns a data frame with the current state of all layer and unit variables. Every row is a unit. You can choose whether you want dynamic values and / or constant values. This might be useful if you want to analyze what happens in the network overall, which would otherwise not be possible, because most of the variables (fields) are private in the layer and unit class.

show_dynamics Should dynamic values be shown? Default is TRUE.

show_constants Should constant values be shown? Default is FALSE.

get_network_vars(show_dynamics = T, show_constants = F) Returns a data frame with 1 row with the current state of the variables in the network. You can choose whether you want dynamic values and / or constant values. This might be useful if you want to analyze what happens in a network, which would otherwise not be possible, because some of the variables (fields) are private in the network class. There are some additional variables in the network class that cannot be extracted this way because they are matrices; if it is necessary to extract them, look at the source code.

show_dynamics Should dynamic values be shown? Default is TRUE.

show_constants Should constant values be shown? Default is FALSE.

References

O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., and Contributors (2016). Computational Cognitive Neuroscience. Wiki Book, 3rd (partial) Edition. URL: http://ccnbook.colorado.edu

Have also a look at https://grey.colorado.edu/emergent/index.php/Leabra (especially the link to the 'MATLAB' code) and https://en.wikipedia.org/wiki/Leabra

Examples

```
# create a small network with 3 layers
dim_lays <- list(c(2, 5), c(2, 10), c(2, 5))
cxn <- matrix(c(0, 0, 0,
                1, 0, 0.2,
                0, 1, 0), nrow = 3, byrow = TRUE)
net <- network$new(dim_lays, cxn)

net$m_in_s # private values cannot be accessed
# if you want to see alle variables, you need to use the function
net$get_network_vars(show_dynamics = TRUE, show_constants = TRUE)
# if you want to see a summary of all units (with layer information) without
# constant values
net$get_layer_and_unit_vars(show_dynamics = TRUE, show_constants = FALSE)

# let us create 10 random inputs for layer 1 and 3
inputs <- net$create_inputs(c(1, 3), 10)
inputs # a list of lists

# the input in layer 1 should be associated with the output in layer 3; we
# can use error driven learning to achieve this

# first we will need the input for the minus phase (where no correct output
# is presented; layer 3 is NULL)
inputs_minus <- lapply(inputs, function(x) replace(x, 3, list(NULL)))
inputs_minus # layer 3 is indeed NULL
# now we can learn with default parameters; we will run 10 epochs,
# inputs_plus is equivalent to inputs; the output will be activations after
# each trial for the wohle network; this might take a while depending on your
# system
```

```
n_epochs <- 10
## Not run:
output <- lapply(seq(n_epochs),
                 function(x) net$learn_error_driven(inputs_minus,
                                                    inputs,
                                                    lrate = 0.5))
# let's compare the actual output with what should have been learned we can
# use the method mad_per_epoch for this; it will calculate the mean absolute
# distance for each epoch; we are interested in layer 3
mad <- net$mad_per_epoch(output, inputs, 3)
# the error should decrease with increasing epoch number
plot(mad)
## End(Not run)
```

`unit` *Leabra unit (neuron) class*

Description

This class simulates a biologically realistic neuron (also called unit) in the Leabra framework. When you use the layer class, you will see that a layer object has a variable (field) `units`, which is a list of unit objects.

Usage

```
unit
```

Format

`R6Class` object.

Value

Object of `R6Class` with methods for calculating neuron activation changes.

Fields

`activation` Percentage activation ("firing rate") of the unit, which is sent to other units, think of it as a percentage of how many neurons are active in a microcolumn of 100 neurons.

`avg_s` Short-term running average activation, integrates over avg_ss (a private variable, which integrates over activation), represents plus phase learning signal.

`avg_m` Medium-term running average activation, integrates over avg_s, represents minus phase learning signal.

`avg_l` Long-term running average activation, integrates over avg_m, drives long-term floating average for self-organized learning.

`unit_number` Number of unit in layer, if the unit is not created within a layer, this value will be 1.

Methods

`new()` Creates an object of this class with default parameters.

`cycle(g_e_raw, g_i)` Cycles 1 ms with given excitatory conductance `g_e_raw` and inhibitory conductance `g_i`. Excitatory conductance depends on the connection weights to other units and the activity of those other units. Inhibitory conductance depends on feedforward and feedback inhibition. See layer cycle method.

`g_e_raw` Raw excitatory conductance. The actual excitatory conductance will incrementally approach this value with every cycle.

`g_i` Inhibitory conductance.

`clamp_cycle(activation)` Clamps the value of `activation` to the `activation` variable of the unit without any time integration. Then updates averages (`avg_ss`, `avg_s`, `avg_m`). This is usually done when presenting external input.

`activation` Activation to clamp.

`updt_avg_l()` Updates the variable `avg_l`. This usually happens before the weights are changed in the network (after the plus phase), and not every cycle.

`get_vars(show_dynamics = TRUE, show_constants = FALSE)` Returns a data frame with 1 row with the current state of all the variables of the unit. You can choose whether you want dynamic values and / or constant values. This might be useful if you want to analyze what happens in a unit, which would otherwise not be possible, because most of the variables (fields) are private in this class.

`show_dynamics` Should dynamic values be shown? Default is TRUE

`show_constants` Should constant values be shown? Default is FALSE

References

O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., and Contributors (2016). Computational Cognitive Neuroscience. Wiki Book, 3rd (partial) Edition. URL: http://ccnbook.colorado.edu

Have also a look at https://grey.colorado.edu/emergent/index.php/Leabra (especially the link to the 'MATLAB' code) and https://en.wikipedia.org/wiki/Leabra

Examples

```
u <- unit$new() # creates a new unit with default leabra values

print(u) # a lot of private values
u$v # private values cannot be accessed
# if you want to see alle variables, you need to use the function
u$get_vars(show_dynamics = TRUE, show_constants = TRUE)

# let us clamp the activation to 0.7
u$activation
u$clamp_cycle(0.7)
c(u$activation, u$avg_s, u$avg_m, u$avg_l)
# activation is indeed 0.7, but avg_l was not updated, this only happens
# before the weights are changed, let us update it now
u$updt_avg_l()
```

```
c(u$activation, u$avg_s, u$avg_m, u$avg_l)
# seems to work

# let us run 10 cycles with unclamped activation and output the activation
# produced because of changes in conductance
u <- unit$new()
cycle_number <- 1:10
result <- lapply(cycle_number, function(x)
                 u$cycle(g_e_raw = 0.5, g_i = 0.5)$get_vars())
# make a data frame out of the list
result <- plyr::ldply(result)
# plot activation
plot(result$activation, type = "b", xlab = "cycle", ylab = "activation")
# add conductance g_e to plot, should approach g_e_raw
lines(result$g_e, type = "b", col = "blue")
```

Index