Package 'hann'

May 8, 2026

Version 1.2

Date 2026-01-26

Title Hopfield Artificial Neural Networks

Description Builds and optimizes Hopfield artificial neural networks (Hopfield, 1982, <doi:10.1073/pnas.79.8.2554>). One-layer and three-layer models are implemented. The energy of the Hopfield network is minimized with formula from Krotov and Hopfield (2016, <doi:10.48550/ARXIV.1606.01164>). Optimization (supervised learning) is done through a gradient-based method. Classification is done with S3 methods predict(). Parallelization with 'OpenMP' is used if available during compilation.

Suggests lattice

Imports graphics, stats

URL https://github.com/emmanuelparadis/hann

BugReports https://github.com/emmanuelparadis/hann/issues

License GPL-3

NeedsCompilation yes

Author Emmanuel Paradis [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3092-2199>)

Maintainer Emmanuel Paradis <Emmanuel.Paradis@ird.fr>

Repository CRAN

Date/Publication 2026-01-26 16:20:32 UTC

Contents

`hann-package` *Hopfield Artificial Neural Networks*

Description

hann provides tools to build Hopfield-based artificial neural networks. Two types of networks can be built: one-layer Hopfield network, and three-layer network with a hidden (convoluted) layer.

The complete list of functions can be displayed with `library(help = hann)`. See the vignette "IntroductionHopfieldNetworks" for several examples.

More information on **hann** can be found at https://github.com/emmanuelparadis/hann.

Author(s)

Emmanuel Paradis

Maintainer: Emmanuel Paradis <Emmanuel.Paradis@ird.fr>

`binarize` *Helper Function to Prepare Data From Images*

Description

Take one or more images coded as numeric values (pixels) and return them coded as -1 or +1 depending on a threshold value.

Usage

```
binarize(x, threshold = median(x))
```

Arguments

`x`	a vector or a matrix of numeric values.
`threshold`	a numeric value: values in `x` below (or equal) to this threshold are coded -1, values above are coded +1.

Details

It is recommended to check that the default of the argument `threshold` is appropriate or not. For instance, if an image has many pixels dark and a few light, this default might not be a good choice.

If the default is a good choice (e.g., good balance between the dark and light pixels), it can be applied globally or separately for each image. Suppose the images (patterns) are arranged along the rows of a matrix `X`, the binarization can be done either with:

`xi <- t(apply(X, 1, binarize))`

where the threshold might be different for each image, or with `xi <- binarize(X)` in which case the threshold will be the same for all images.

Value

an object with integers (-1/1) with the same attributes (dim, names, etc.) than the input object `x`.

See Also

hann

Examples

```
## a plus (+) sign on a 3x3 grid:
x <- matrix(runif(9, 150, 255), 3, 3)
## make the corners lighter to draw the "+":
x[c(1, 3, 7, 9)] <- runif(4, 0, 20)
x <- round(x)
xi <- binarize(x)
## compare:
xi; x
layout(matrix(1:2, 1))
image(x, asp = 1, main = "Original image")
image(xi, asp = 1, main = "Binarized image")
layout(1)
```

---

buildSigma *Hopfield Network Energy*

Description

Minimize the energy of the Hopfield network.

Usage

```
buildSigma(xi, n = 20, nrep = 100, quiet = FALSE)
```

Arguments

xi	a matrix of patterns coded with 1 and -1.
n	the parameter of the energy function (integer).
nrep	the number of attempts.
quiet	a logical value indicating whether to print the details for each attempt.

Details

The number of columns in xi is equal to the size of the Hopfield network (the number of input neurons denoted as N), whereas the number of columns is the number of memories denoted as K (Krotov and Hopfield, 2016).

A random vector 'sigma' is first generated and then updated in order to minimize the energy level of the Hopfield network. The convergence to a low energy level depends on the initial values in 'sigma', so the procedure is repeated several times. The vector with the lowest energy level is returned.

Value

a vector of integers (-1/1). The length of this vector (N) is equal to the number of columns in xi.

References

Krotov, D. and Hopfield, J. J. (2016) Dense associative memory for pattern recognition. doi:10.48550/ARXIV.1606.01164.

See Also

hann

Examples

```
xi <- matrix(NA, K <- 1000, N <- 60)
xi[] <- sample(c(1L, -1L), K * N, TRUE)
(sigma <- buildSigma(xi))
```

---

combine *Combine Several Neural Nets for Prediction*

Description

Combine several neural networks optimized with different sets of (overlapping) classes. The patterns (xi) are classified with all networks and the signal are combined to make a final prediction.

Usage

```
combine(nets, xi)
```

Arguments

nets	a list of objects inheriting class "hann".
xi	a matrix of patterns.

Details

Suppose we want to classify (predict) patterns labelled A, B, and C, then we can optimize two networks: one classifying A and B, and another one classifying B and C. The present function combines the predictions from both networks to classify patterns into the three classes.

Since optimizing a network with many classes can be complicated, it might be a better strategy to optimize separate networks with two classes each, and then combine them to make predictions over all classes.

Value

a vector of predicted classes (as coded by the labels).

See Also

hann, predict.hann1

Examples

```
## see ?hann for explanation on rpat():
rpat <- function(type, nr = 9L, nc = 9L,
                 signal = c(200, 255), noise = c(0, 50))
{
    ij <- switch(type,
                 "V" = cbind(1:nr, ceiling(nc/2)),
                 "H" = cbind(ceiling(nr/2), 1:nc),
                 "U" = cbind(nr:1, 1:nc),
                 "D" = cbind(1:nr, 1:nc))
    x <- matrix(runif(nr * nc, noise[1], noise[2]), nr, nc)
    x[ij] <- runif(nr, signal[1], signal[2])
    round(x)
}

## take 3 patterns:
labs <- c("V", "H", "U")
## repeat each label 40 times:
cl <- rep(labs, each = 40)
## simulate a pattern for each label:
xi <- t(sapply(cl, rpat))
## binarize the patterns:
xi <- binarize(xi)

## we take only 2 patterns in turn
## S: list with the indices for each pair
S <- list(1:80, c(1:40, 81:120), 41:120)
P <- list(c("V", "H"), c("V", "U"), c("H", "U"))
```

```
cl2 <- gl(2, 40) # only 2 classes for each net
## so the class (1 or 2) will be the same for each net
## but the labels will be different (from the list 'P')

ctr <- control.hann(quiet = TRUE)

NT <- vector("list", 3) # to store the optimized nets
for (i in 1:3) {
    s <- S[[i]]
    sig <- buildSigma(xi[s, ], quiet = TRUE)
    NT[[i]] <- hann3(xi[s, ], sig, cl2, H = 10,
                     labels = P[[i]], control = ctr)
}

## final predictions:
table(cl, combine(NT, xi))
```

`control.hann` *Parameters for Neural Network Optimization*

Description

Set the control parameters for the Hopfield artificial neural network optimization.

Usage

```
control.hann(...)
```

Arguments

`...`	named arguments to be modified (see examples).

Details

When the user modifies one or several parameters by giving them as named arguments, if some names are incorrect they are ignored with a warning.

The parameters with their default values are:

- `iterlim` = `100`: an integer giving the number of iterations.
- `quiet` = `FALSE`: a logical controlling whether to print the value of the objective function at each iteration.
- `quasinewton` = `FALSE`: a logical. If `TRUE`, quasi-Newton steps are performed (not recommended unless the network has a small number of parameters and/or for a small number of iterations).
- `fullhessian` = `FALSE`: (ignored if `quasinewton` = `FALSE`) a logical, by default only some blocks of the Hessian matrix are computed. If `TRUE`, the full Hessian matrix is computed (very time consuming).

- `trace.error = FALSE`: a logical. If `TRUE`, the error rate is printed at each iteration of the optimization process.
- `wolfe = FALSE`: a logical. If `TRUE`, Wolfe's conditions are tested at each iteration.
- `target = 0.001`: the target value of the loss function to stop the optimization.
- `beta = 0.2`: the hyperparameter of the activation function.
- `mc.cores = 1`: an integer. The number of cores used when computing the objective function.

If `mc.cores` is greater than one, the optimization process calls a multithreaded code using OMP. So, do *not* do this together with functions from the package **parallel**. On the other hand, if you leave this parameter to its default value, you should be able to run several optimizations in parallel, for instance with `mclapply`.

See the vignette for applications.

Value

a list with named elements as detailed above.

Note

For the moment, the parameter `mc.cores` is accepted only by `hann1`.

References

https://en.wikipedia.org/wiki/Wolfe_conditions

See Also

`hann`

Examples

```
control.hann() # default values
ctrl <- control.hann(iterlim = 1000)
ctrl

## verbose is not a parameter:
ctrl <- control.hann(iterlim = 1000, verbose = TRUE)
```

---

hann.R *Method Top-Level Functions*

Description

Functions to fit Hopfield artificial neural networks or to access the results.

Usage

```
hann(xi, sigma, classes, H = NULL, labels = NULL, net = NULL,
     control = control.hann())
## S3 method for class 'hann'
print(x, ...)
## S3 method for class 'hann'
summary(object, ...)
## S3 method for class 'hann'
str(object, ...)
## S3 method for class 'hann'
plot(x, y, type = "h", ...)
## S3 method for class 'hann'
coef(object, ...)
## S3 method for class 'hann'
fitted(object, ...)
## S3 method for class 'hann'
labels(object, ...)
## S3 method for class 'hann'
predict(object, ...)
```

Arguments

xi	a matrix of patterns with K rows and N columns.
sigma	a vector coding the Hopfield network (length N).
classes	the classes of the patterns (vector of length K).
H	the number of neurons in the hidden layer (can be 0).
labels	a vector of labels used for the classes.
net, x, object	an object inheriting class "hann".
control	the control parameters.
y	(unused).
type	the type of plot for vectors of parameters (biases).
...	options passed to other methods.

Details

hann() calls either hann1() or hann3() depending on the value given to the argument H (the number of hidden neurons).

The other functions are (standard) methods for accessing the results.

Value

hann returns an object of class c("hann", "hann1") or c("hann", "hann3"); see the links below for their description.

See Also

hann1, hann3

Examples

```
## function to create 'images' default size is 9x9 pixels,
## with 4 possible shapes ("V"ertical, "H"orizontal, "U"p-diag.
## or "D"own-diag)
rpat <- function(type, nr = 9L, nc = 9L,
                 signal = c(200, 255), noise = c(0, 50))
{
    ij <- switch(type,
                 "V" = cbind(1:nr, ceiling(nc/2)),
                 "H" = cbind(ceiling(nr/2), 1:nc),
                 "U" = cbind(nr:1, 1:nc),
                 "D" = cbind(1:nr, 1:nc))
    x <- matrix(runif(nr * nc, noise[1], noise[2]), nr, nc)
    x[ij] <- runif(nr, signal[1], signal[2])
    round(x)
}

## the 4 types of patterns to simulate:
labs <- c("V", "H", "U", "D")
## repeat them 40 times, this will be used as class-vector:
cl <- rep(labs, each = 40)
## simulate the images:
xi <- t(sapply(cl, rpat))
## binarize the patterns (-1/+1):
xi <- binarize(xi)
## build the sigma vector:
sig <- buildSigma(xi, quiet = TRUE)

## optimize the neural net with 10 hidden neurons:
ctr <- control.hann(quiet = TRUE)
nt <- hann(xi, sig, cl, H = 10, control = ctr)

## convergence depends on the initial parameter values, so it might be
## needed to repeat the previous command a few times so that the next
## one shows only values on the diagonal (which can be reached with
## the default 100 iterations)

table(cl, predict(nt, xi, rawsignal = FALSE))

## now generate 10 new patterns...
new_cl <- rep(labs, each = 10)
new_xi <- binarize(t(sapply(new_cl, rpat)))
## ... and see how well they are predicted:
table(new_cl, predict(nt, new_xi, rawsignal = FALSE))

## visualize the optimized neural net
layout(matrix(1:6, 2, 3, TRUE))
plot(nt)
layout(1)
```

hann1 *One-layer Hopfield ANN*

Description

Optimize a one-layer Hopfield artificial neural network. The structure of the network is quite simple: a Hopfield network with N input neurons all connected to C output neurons. The number of parameters (N and C) is determined by the input data: `xi` has N columns (which is also the length of `sigma`) and the number of unique values of `classes` is equal to C.

Usage

```
hann1(xi, sigma, classes, labels = NULL,
      net = NULL, control = control.hann())

## S3 method for class 'hann1'
print(x, details = FALSE, ...)
```

Arguments

`xi`	a matrix of patterns with K rows.
`sigma`	a vector coding the Hopfield network.
`classes`	the classes of the patterns (vector of length K).
`labels`	a vector of labels used for the classes.
`net`, `x`	an object inheriting class `"hann1"`.
`control`	the control parameters.
`details`	a logical value (whether to print the parameter values of the network).
`...`	further arguments passed to `print.default`.

Details

By default, the parameters of the neural network are initialized with random values from a uniform distribution between -1 and 1 (except the biases which are initialized to zero).

If an object inheriting class `"hann1"` is given to the argument `net`, then its parameter values are used to initialize the parameters of the network.

The main control parameters are given as a list to the `control` argument. They are detailed in the page of the function `control.hann()`.

Value

an object of class `c("hann", "hann1")` with the following elements:

`parameters`	a list with one matrix, `W`, and one vector, `bias`.
`sigma`	the Hopfield network.

beta	the hyperparameter of the activation function.
labels	the labels of the classes.
call	the function call.
fitted	the raw signals of the output neurons from the input patterns.

References

Hopfield, J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, **79**, 2554–2558. doi:10.1073/pnas.79.8.2554.

Krotov, D. and Hopfield, J. J. (2016) Dense associative memory for pattern recognition. doi:10.48550/ARXIV.1606.01164.

See Also

buildSigma, hann, predict.hann1

---

hann3 *Three-layer Hopfield ANN*

Description

Optimize a three-layer Hopfield artificial neural network. The network is made of a Hopfield network with N input neurons all connected to H hidden neurons. The latter are all connected together (convolution) which is equivalent to defining two hidden layers. Each hidden neuron is connected to C output neurons. The values of the parameters N and C are determined by the input data: `xi` has N columns (which is also the length of `sigma`) and the number of unique values of `classes` is equal to C. The value of H must be given by the user (a default of half the number of input neurons is defined).

Usage

```
hann3(xi, sigma, classes,  H = 0.5 * length(sigma),
      labels = NULL, net = NULL, control = control.hann())

## S3 method for class 'hann3'
print(x, details = FALSE, ...)
```

Arguments

xi	a matrix of patterns with K rows and N columns.
sigma	a vector coding the Hopfield network (length N).
classes	the classes of the patterns (vector of length K).
H	the number of neurons in the hidden layer; by default half the number of input neurons (rounded to the lowest integer).

labels	a vector of labels used for the classes.
net, x	an object inheriting class "hann3".
control	the control parameters.
details	a logical value (whether to print the parameter values of the network).
...	further arguments passed to print.default.

Details

By default, the parameters of the neural network are initialized with random values from a uniform distribution between -1 and 1 (expect the biases which are initialized to zero).

If an object inheriting class "hann3" is given to the argument net, then its parameter values are used to initialize the parameters of the network.

The main control parameters are given as a list to the control argument. They are detaild in the page of the function control.hann().

Value

an object of class c("hann", "hann3") with the following elements:

parameters	a list with three matrices, W1, W2, and W3, and two vectors, bias1 and bias3.
sigma	the Hopfield network.
beta	the hyperparameter of the activation function.
labels	the labels of the classes.
call	the function call.
fitted	the raw signals of the output neurons from the input patterns.

References

Hopfield, J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, **79**, 2554–2558. doi:10.1073/pnas.79.8.2554.

Krotov, D. and Hopfield, J. J. (2016) Dense associative memory for pattern recognition. doi:10.48550/ARXIV.1606.01164.

See Also

buildSigma, control.hann, hann, predict.hann3

predict.hann1 *Prediction*

Description

Classification of patterns with Hopfield-based artificial neural networks.

Usage

```
## S3 method for class 'hann1'
predict(object, patterns, rawsignal = TRUE,
        useLabels = TRUE, ...)
## S3 method for class 'hann3'
predict(object, patterns, rawsignal = TRUE,
        useLabels = TRUE, ...)
```

Arguments

object	an object of class "hann1" or "hann3".
patterns	the patterns to be classified.
rawsignal	a logical value (see details).
useLabels	a logical value whether to use the labels of the classes (if any) as colnames of the predictions.
...	(unused).

Details

The patterns have to be coded in the same way than the matrix xi used to train the networks.

If rawsignal = TRUE, the raw signal of each neuron is output for each pattern. Otherwise, a classification of each pattern is done by finding the neuron with the largest signal.

Value

If rawsignal = TRUE a matrix; if rawsignal = FALSE a vector.

See Also

hann1, hann3

tune.hann *Tune Hyperparameters*

Description

Tune the two hyperparameters of the neural nets: the number of hidden neurons (H) and the slope of the activation function (beta).

Usage

```
tune.hann(xi, sigma, classes,
          ranges = list(H = seq(10, 50, by = 10),
                        beta = seq(0.2, 0.8, by = 0.1)),
          nrepeat = 10,
          control = control.hann(iterlim = 20))
```

Arguments

xi	a matrix of patterns with K rows.
sigma	a vector coding the Hopfield network.
classes	the classes of the patterns (vector of length K).
ranges	a list giving the values of the parameters to be tested.
nrepeat	the number of repeats
control	the control parameters.

Details

This function is built on the same model than functions in the package **e1071**.

The effect of the hyperparameters is usually visible with a small number of iterations. The fitting process is repeated several times for each combination of the hyperparameters.

Value

a data frame with four columns:

H	the number of hidden neurons
beta	the values of the slope of the activation function
mean	the mean of the error rate computed over the repeats
sd	the standard-deviation

See Also

hann

Examples

```
## Not run:
## simulate 200 random patterns with 30 pixels:
v <- c(-1L, 1L)
K <- 200L
N <- 30L
xi <- matrix(sample(v, K*N, TRUE), K, N)
stopifnot(nrow(unique(xi)) == K)
## build the vector sigma:
sig <- buildSigma(xi, quiet = TRUE)
## define the classes:
cl <- rep(1:2, each = K/2)
## the ranges:
ranges <- list(H = seq(10, 60, by = 10),
               beta = seq(0.1, 1, .1))
ctr <- control.hann(iterlim = 10)
res <- tune.hann(xi, sig, cl, ranges, control = ctr, nrepeat = 5)
str(res)
## visualize the results:
library(lattice)
levelplot(mean ~ beta * H, data = res, main = "Mean")
levelplot(sd ~ beta * H, data = res, main = "Standard-deviation")

## End(Not run)
```

Index