Package 'SAENET'

June 4, 2015

Type Package

Title A Stacked Autoencoder Implementation with Interface to 'neuralnet'

Version 1.1

Date 2015-06-04

Description

An implementation of a stacked sparse autoencoder for dimension reduction of features and pre-training of feed-forward neural networks
with the 'neuralnet' package is contained within this package. The package also includes a predict function for the stacked autoencoder object to generate the compressed
representation of new data if required. For the purposes of this package, 'stacked' is defined in line with http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders .
The underlying sparse autoencoder is defined in the documentation of 'autoencoder'.

License GPL-3

URL http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders ,
http://cran.r-project.org/package=autoencoder

Imports autoencoder, neuralnet

NeedsCompilation no

Repository CRAN

Author Stephen Hogg [aut, cre],
Eugene Dubossarsky [aut]

Maintainer Stephen Hogg <saenet.r@gmail.com>

Date/Publication 2015-06-04 11:50:48

R topics documented:

SAENET.nnet *Use a stacked autoencoder to pre-train a feed-forward neural network.*

Description

Use a stacked autoencoder to pre-train a feed-forward neural network.

Usage

```
SAENET.nnet(h, X.train, target, nn.control = NULL)
```

Arguments

h	The object returned from SAENET.train()
X.train	A matrix of training data.
target	A vector of target values. If given as a factor, classification will be performed, otherwise if an integer or numeric vector is given neuralnet() will default to regression
nn.control	A named list with elements to be passed as control parameters to neuralnet(). Package defaults used if no values entered.

Value

An object of class nn which can be used with the neuralnet package as normal

SAENET.predict *Obtain the compressed representation of new data for specified layers from a stacked autoencoder.*

Description

Obtain the compressed representation of new data for specified layers from a stacked autoencoder.

Usage

```
SAENET.predict(h, new.data, layers = c(1), all.layers = FALSE)
```

Arguments

h	The object returned from SAENET.train()
new.data	A matrix of training data.
layers	A numeric vector indicating which layers of the stacked autoencoder to return output for
all.layers	A boolean value indicating whether to override layers and return the encoded output for all layers. Defaults to FALSE

Value

A list, for which each element corresponds to the output of `predict.autoencoder()` from package `autoencoder` for the specified layers of the stacked autoencoder.

Examples

```
library(autoencoder)
data(iris)
#### Train a stacked sparse autoencoder with a (5,3) architecture and
#### a relatively minor sparsity penalty. Try experimenting with the
#### lambda and beta parameters if you haven't worked with sparse
#### autoencoders before - it's worth inspecting the final layer
#### to ensure that output activations haven't simply converged to the value of
#### rho that you gave (which is the desired activation level on average).
#### If the lambda/beta parameters are set high, this is likely to happen.


output <- SAENET.train(as.matrix(iris[1:100,1:4]), n.nodes = c(5,3),
                       lambda = 1e-5, beta = 1e-5, rho = 0.01, epsilon = 0.01)


predict.out <- SAENET.predict(output, as.matrix(iris[101:150,1:4]), layers = c(2))
```

`SAENET.train` *Build a stacked Autoencoder.*

Description

Build a stacked Autoencoder.

Usage

```
SAENET.train(X.train, n.nodes = c(4, 3, 2), unit.type = c("logistic",
  "tanh"), lambda, beta, rho, epsilon, optim.method = c("BFGS", "L-BFGS-B",
  "CG"), rel.tol = sqrt(.Machine$double.eps), max.iterations = 2000,
  rescale.flag = F, rescaling.offset = 0.001)
```

Arguments

`X.train`	A matrix of training data.
`n.nodes`	A vector of numbers containing the number of units at each hidden layer.
`unit.type`	hidden unit activation type as per `autoencode()` params.
`lambda`	Vector of scalars indicating weight decay per layer as per `autoencode()`.
`beta`	Vector of scalars indicating sparsity penalty per layer as per `autoencode()`.
`rho`	Vector of scalars indicating sparsity parameter per layer as per `autoencode()`.
`epsilon`	Vector of scalars indicating initialisation parameter for weights per layer as per `autoencode()`.

optim.method	Optimization method as per optim().
rel.tol	Relative convergence tolerance as per optim()
max.iterations	Maximum iterations for optim().
rescale.flag	A logical flag indicating whether input data should be rescaled.
rescaling.offset	A small non-negative value used for rescaling. Further description available in the documentation of autoencoder.

Value

An object of class SAENET containing the following elements for each layer of the stacked autoencoder:

ae.out	An object of class autoencoder containing the autoencoder created in that layer of the stacked autoencoder.
X.output	In layers subsequent to the first, a matrix containing the activations of the hidden neurons.

Examples

```
library(autoencoder)
data(iris)
#### Train a stacked sparse autoencoder with a (5,3) architecture and
#### a relatively minor sparsity penalty. Try experimenting with the
#### lambda and beta parameters if you haven't worked with sparse
#### autoencoders before - it's worth inspecting the final layer
#### to ensure that output activations haven't simply converged to the value of
#### rho that you gave (which is the desired activation level on average).
#### If the lambda/beta parameters are set high, this is likely to happen.


output <- SAENET.train(as.matrix(iris[,1:4]), n.nodes = c(5,3),
                       lambda = 1e-5, beta = 1e-5, rho = 0.01, epsilon = 0.01)
```

Index