# Package ‘autoencoder’

July 2, 2015

**Type** Package

**Title** Sparse Autoencoder for Automatic Learning of Representative Features from Unlabeled Data

**Version** 1.1

**Date** 2015-06-30

**Author** Eugene Dubossarsky (project leader, chief designer), Yuriy Tyshetskiy (design, implementation, testing)

**Maintainer** Yuriy Tyshetskiy `<yuriy.tyshetskiy@nicta.com.au>`

**Description** Implementation of the sparse autoencoder in R environment, following the notes of Andrew Ng (http://www.stanford.edu/class/archive/cs/cs294a/cs294a.1104/sparseAutoencoder.pdf). The features learned by the hidden layer of the autoencoder (through unsupervised learning of unlabeled data) can be used in constructing deep belief neural networks.

**License** GPL-2

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2015-07-02 09:09:12

## R topics documented:

## autoencoder-package *Implementation of sparse autoencoder for automatic learning of representative features from unlabeled data.*

### Description

The package implements a sparse autoencoder, descibed in Andrew Ng's notes (see the reference below), that can be used to automatically learn features from unlabeled data. These features can then be used, e.g., for weight initialization in hidden layers of deep-belief neural networks.

### Details

| | |
|---|---|
| Package: | autoencoder |
| Type: | Package |
| Version: | 1.0 |
| Date: | 2014-03-05 |
| License: | GPL-2 |

The current version of the package consists of two main functions: autoencode() and predict(). See help for autoencode() and predict.autoencoder() for more details on how to use them.

### Author(s)

Eugene Dubossarsky (project leader, chief designer), Yuriy Tyshetskiy (design, implementation, testing)

Maintainer: Yuriy Tyshetskiy <y.tyshetskiy@usask.ca>

### References

Andrew Ng, Sparse autoencoder (Lecture notes) `http://www.stanford.edu/class/archive/cs/cs294a/cs294a.1104/sparseAutoencoder.pdf`

### See Also

`autoencode`, `predict.autoencoder`

### Examples

```
## Train the autoencoder on unlabeled set of 5000 image patches of
## size Nx.patch by Ny.patch, randomly cropped from 10 nature photos:
## Load a training matrix with rows corresponding to training examples,
## and columns corresponding to input channels (e.g., pixels in images):
data('training_matrix_N=5e3_Ninput=100')  ## the matrix contains 5e3 image
                                           ## patches of 10 by 10 pixels

## Set up the autoencoder architecture:
```

```
nl=3                          ## number of layers (default is 3: input, hidden, output)
unit.type = "logistic"        ## specify the network unit type, i.e., the unit's
                              ## activation function ("logistic" or "tanh")
Nx.patch=10                   ## width of training image patches, in pixels
Ny.patch=10                   ## height of training image patches, in pixels
N.input = Nx.patch*Ny.patch  ## number of units (neurons) in the input layer (one unit per pixel)
N.hidden = 5*5                ## number of units in the hidden layer
lambda = 0.0002               ## weight decay parameter
beta = 6                      ## weight of sparsity penalty term
rho = 0.01                    ## desired sparsity parameter
epsilon <- 0.001              ## a small parameter for initialization of weights
                          ## as small gaussian random numbers sampled from N(0,epsilon^2)
max.iterations = 2000         ## number of iterations in optimizer

## Train the autoencoder on training.matrix using BFGS optimization method
## (see help('optim') for details):
## WARNING: the training can take as long as 20 minutes for this dataset!

## Not run:
autoencoder.object <- autoencode(X.train=training.matrix,nl=nl,N.hidden=N.hidden,
          unit.type=unit.type,lambda=lambda,beta=beta,rho=rho,epsilon=epsilon,
          optim.method="BFGS",max.iterations=max.iterations,
          rescale.flag=TRUE,rescaling.offset=0.001)

## End(Not run)
## N.B.: Training this autoencoder takes a long time, so in this example we do not run the above
## autoencode function, but instead load the corresponding pre-trained autoencoder.object.


## Report mean squared error for training and test sets:
cat("autoencode(): mean squared error for training set: ",
round(autoencoder.object$mean.error.training.set,3),"\n")

## Visualize hidden units' learned features:
visualize.hidden.units(autoencoder.object,Nx.patch,Ny.patch)

## Compare the output and input images (the autoencoder learns to approximate
## inputs in outputs using features learned by the hidden layer):
## Evaluate the output matrix corresponding to the training matrix
## (rows are examples, columns are input channels, i.e., pixels)
X.output <- predict(autoencoder.object, X.input=training.matrix, hidden.output=FALSE)$X.output

## Compare outputs and inputs for 3 image patches (patches 7,26,16 from
## the training set) - outputs should be similar to inputs:
op <- par(no.readonly = TRUE)  ## save the whole list of settable par's.
par(mfrow=c(3,2),mar=c(2,2,2,2))
for (n in c(7,26,16)){
## input image:
  image(matrix(training.matrix[n,],nrow=Ny.patch,ncol=Nx.patch),axes=FALSE,main="Input image",
   col=gray((0:32)/32))
## output image:
  image(matrix(X.output[n,],nrow=Ny.patch,ncol=Nx.patch),axes=FALSE,main="Output image",
   col=gray((0:32)/32))
}
par(op)  ## restore plotting par's
```

---

autoencode *Train a sparse autoencoder using unlabeled data*

---

## Description

autoencode implements the sparse autoencoder (described in Andrew Ng's lecture notes http://www.stanford.edu/class/archive/cs/cs294a/cs294a.1104/sparseAutoencoder.pdf). The features learned by the autoencoder trained on unlabeled data are available through weights of the trained autoencoder object. These automatically learned features are useful, e.g., in constructing deep belief networks.

## Usage

```
autoencode(X.train, X.test = NULL, nl = 3, N.hidden, 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 = c(F, T), rescaling.offset = 0.001)
```

## Arguments

| | |
|---|---|
| `X.train` | a matrix of training data, with rows corresponding to training examples, and columns corresponding to input channels. For example, if training data consists of 10x10-pixel images, then X.train has 100 columns corresponding to each pixel. |
| `X.test` | an optional matrix of test data in the same format as `X.train`, used for testing of the trained autoencoder by evaluating the squared error for the test data set. |
| `nl` | number of layers in the autoencoder (default is 3 layers: input, hidden, output). |
| `N.hidden` | a vector of numbers of units (neurons) in each of the hidden layers. For `nl=3` (default architecture) this is just the number of units in the single hidden layer of the autoencoder. |
| `unit.type` | type of units used in the autoencoder, defined by the activation function of the units ('logistic' or 'tanh'). |
| `lambda` | weight decay parameter controlling the relative importance of the regularization term in the autoencoder's cost function. |
| `beta` | weight of sparsity penalty term. |
| `rho` | sparsity parameter, constrains the average (over training examples) activation of hidden units. Typically should be a small value close to zero (hence sparse autoencoder). |
| `epsilon` | a small parameter for initialization of autoencoder weights as small gaussian random numbers sampled from the normal distribution N(0,epsilon^2). |
| `optim.method` | the optimization method to be used for searching the minimum of the cost function. See `method` in `help('optim')` for details. |

| | |
|---|---|
| `rel.tol` | relative convergence tolerance determining the convergence of `optim()` optimizer. The algorithm stops if it is unable to reduce the value by a factor of `reltol * (abs(value) + reltol)` at a step. Defaults to `sqrt(.Machine$double.eps)`, typically about `1e-8`. |
| `max.iterations` | maximum number of iterations in searching for cost function minimum. Defaults to 2000. |
| `rescale.flag` | a logical flag indicating whether to uniformly rescale the training matrix to make sure the values of all input channels are within the range of unit outputs (the range is [0,1] for `'logistic'` units, and [-1,1] for `'tanh'` units). |
| `rescaling.offset` | a small non-negative value used in rescaling to `[offset,1-offset]` for `'logistic'` units, and to `[-1+offset,1-offset]` for `'tanh'` units. Defaults to 0.001. |

## Details

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation to adjust its weights, attempting to learn to make its target values (outputs) to be equal to its inputs. In other words, it is trying to learn an approximation to the identity function, so as its output is similar to its input, for all training examples. With the sparsity constraint enforced (requiring that the average, over training set, activation of hidden units be small), such autoencoder automatically learns useful features of the unlabeled training data, which can be used for, e.g., data compression (with losses), or as features in deep belief networks.

The training is performed by optimizing the autoencoder's cost function `J(W,b)` that depends on the autoencoder's weights W and biases b. The optimization (searching for a local minimum) is performed with the `optim` function using one of the three methods: 'BFGS', 'L-BFGS-B', or 'CG' (see details in `help(optim)`).

After the optimization converges, the mean squared error between the output and input matrix (either the training matrix, or a test matrix) is evaluated as a measure of goodness of fit of the autoencoder.

For the autoencoder to work well, one must rescale, if necessary, the training matrix to make sure all the input channels (and hence all the output channels) have values within the range of unit activation function values: [0,1] for 'logistic' units, [-1,1] for 'tanh' units. If rescaling flag is true (`rescale.flag=TRUE`), the input matrix is rescaled uniformly using its minimum and maximum elements `min(X.train)` and `max(X.train)` as

`X.train.rescaled=(X.train-min(X.train))/(max(X.train)-min(X.train))` for 'logistic' units, and `X.train.rescaled=2*(X.train-min(X.train))/(max(X.train)-min(X.train))-1` for 'tanh' units.

The minimum and maximum elements of the training matrix are then passed to the object returned by the function, to be used for rescaling input data in `predict.autoencode` function, for compatibility with the rescaling of the data used for training the autoencoder.

## Value

An object of class `autoencoder`, containing a list with the following components:

| | |
|---|---|
| `W` | a list of weight matrices in the format `W[[l]][i,j]`, where `l` is the number of the layer, `i` and `j` are the row and column indices. An element `W[[l]][i,j]` is |

the weight associated with a connection between unit `j` in layer `l` and unit `i` in layer `l+1`.

| | |
|---|---|
| `b` | a list of biases; `b[[l]][i]` is the bias associated with unit `i` in layer `l+1`. |
| `unit.type` | type of units used in the autoencoder, the value if the same as `unit.type` argument. |
| `rescaling` | a list with elements `rescale.flag` (a logical flag indicating whether rescaling was applied to the training matrix before training the autoencoder), `rescaling.min`, and `rescaling.max` (the minimum and maximum elements of the training matrix used for rescaling - see 'Details'). |
| `mean.error.training.set` | average, over all training matrix rows (training examples), sum of `(X.output-X.train)^2`, where `X.output` is the matrix of autoencoder outputs corresponding to the training matrix `X.train` (i.e., each row in `X.output` is a result of the feed-forward pass of the corresponding training example through the trained autoencoder). |
| `mean.error.test.set` | average, over all test matrix rows (test examples), sum of `(X.output-X.test)^2`, where `X.output` is the matrix of autoencoder outputs corresponding to the test matrix `X.test`(i.e., each row in `X.output` is a result of the feed-forward pass of the corresponding test example through the trained autoencoder). |

## Author(s)

Eugene Dubossarsky (project leader, chief designer), Yuriy Tyshetskiy (design, implementation, testing)

## References

See Andrew Ng's lecture notes at http://www.stanford.edu/class/archive/cs/cs294a/cs294a.1104/sparseAutoencoder.pdf

## See Also

`predict.autoencoder`

## Examples

```
## Train the autoencoder on unlabeled set of 5000 image patches of
## size Nx.patch by Ny.patch, randomly cropped from 10 nature photos:
## Load a training matrix with rows corresponding to training examples,
## and columns corresponding to input channels (e.g., pixels in images):
data('training_matrix_N=5e3_Ninput=100')  ## the matrix contains 5e3 image
                                          ## patches of 10 by 10 pixels

## Set up the autoencoder architecture:
nl=3                          ## number of layers (default is 3: input, hidden, output)
unit.type = "logistic"        ## specify the network unit type, i.e., the unit's
                              ## activation function ("logistic" or "tanh")
Nx.patch=10                   ## width of training image patches, in pixels
Ny.patch=10                   ## height of training image patches, in pixels
```

```
N.input = Nx.patch*Ny.patch   ## number of units (neurons) in the input layer (one unit per pixel)
N.hidden = 10*10                ## number of units in the hidden layer
lambda = 0.0002               ## weight decay parameter
beta = 6                      ## weight of sparsity penalty term
rho = 0.01                    ## desired sparsity parameter
epsilon <- 0.001              ## a small parameter for initialization of weights
                          ## as small gaussian random numbers sampled from N(0,epsilon^2)
max.iterations = 2000         ## number of iterations in optimizer

## Train the autoencoder on training.matrix using BFGS optimization method
## (see help('optim') for details):
## WARNING: the training can take a long time (~1 hour) for this dataset!

## Not run:
autoencoder.object <- autoencode(X.train=training.matrix,nl=nl,N.hidden=N.hidden,
          unit.type=unit.type,lambda=lambda,beta=beta,rho=rho,epsilon=epsilon,
          optim.method="BFGS",max.iterations=max.iterations,
          rescale.flag=TRUE,rescaling.offset=0.001)

## End(Not run)
## N.B.: Training this autoencoder takes a long time, so in this example we do not run the above
## autoencode function, but instead load the corresponding pre-trained autoencoder.object.


## Report mean squared error for training and test sets:
cat("autoencode(): mean squared error for training set: ",
round(autoencoder.object$mean.error.training.set,3),"\n")

## Extract weights W and biases b from autoencoder.object:
W <- autoencoder.object$W
b <- autoencoder.object$b
## Visualize hidden units' learned features:
visualize.hidden.units(autoencoder.object,Nx.patch,Ny.patch)
```

## autoencoder_Ninput=100_Nhidden=100_rho=1e-2

*A trained autoencoder example with 100 hidden units*

### Description

`autoencoder.object` is an example object of class `autoencoder` containing the weights, biases and other parameter of a sparse autoencoder with `N.input=100`, `N.hidden=100`, with sparsity parameter `rho=0.01`, trained on a dataset of 5000 image patches of 10 by 10 pixels, randomly cropped from decoloured nature photos.

### Usage

```
data('autoencoder_Ninput=100_Nhidden=100_rho=1e-2')
```

## Format

The format is: chr "autoencoder_Ninput=100_Nhidden=100_rho=1e-2"

## Examples

```
## Load a pre-trained autoencoder object with N.input=100 and N.hidden=10*10,
## trained on unlabeled set of 5000 image patches of size Nx.patch by Ny.patch,
## randomly picked from 10 nature photos, and visualize the features
## learned by its hidden units:

data('autoencoder_Ninput=100_Nhidden=100_rho=1e-2')

## Visualize hidden units' learned features:
visualize.hidden.units(autoencoder.object,Nx.patch=10,Ny.patch=10)
```

---

## autoencoder_Ninput=100_Nhidden=25_rho=1e-2
*A trained autoencoder example with 25 hidden units*

---

## Description

autoencoder.object is an example object of class autoencoder containing the weights, biases and other parameter of a sparse autoencoder with N.input=100, N.hidden=25, with sparsity parameter rho=0.01, trained on a dataset of 5000 image patches of 10 by 10 pixels, randomly cropped from decoloured nature photos.

## Usage

```
data('autoencoder_Ninput=100_Nhidden=25_rho=1e-2')
```

## Format

The format is: chr "autoencoder_Ninput=100_Nhidden=25_rho=1e-2"

## Examples

```
## Load a pre-trained autoencoder object with N.input=100 and N.hidden=5*5,
## trained on unlabeled set of 5000 image patches of size Nx.patch by Ny.patch,
## randomly picked from 10 nature photos, and visualize the features
## learned by its hidden units:

data('autoencoder_Ninput=100_Nhidden=25_rho=1e-2')

## Visualize hidden units' learned features:
visualize.hidden.units(autoencoder.object,Nx.patch=10,Ny.patch=10)
```

## `predict` *Predict outputs of a sparse autoencoder*

### Description

Predict outputs for a given set of inputs, and estimate mean squared error between inputs and outputs of a sparse autoencoder network trained using `autoencode` function. Optionally, predict outputs of units in the hidden layer of the autoencoder instead of outputs of units in the output layer.

### Usage

```
## S3 method for class 'autoencoder'
predict(object, X.input=NULL, hidden.output = c(F, T), ...)
```

### Arguments

| | |
|---|---|
| `object` | an object of class `autoencoder` produced by the `autoencode` function. |
| `X.input` | a matrix of inputs, with columns corresponding to the columns of the training matrix used in training the autoencoder `object`, and an arbitrary number of rows corresponding to the number of inputs. |
| `hidden.output` | a logical switch telling whether to produce outputs of units in the output layer (for `hidden.output=FALSE`) or of units in the hidden layer (for `hidden.output=TRUE`). The latter can be used for stacked autoencoders. |
| `...` | not used. |

### Details

All the information about the autoencoder (weights and biases, unit type, rescaling data) is contained in `object` of class `autoencoder` produced by `autoencode` function. See `autoencode` for details about the `autoencoder` class object.

### Value

A list with elements:

| | |
|---|---|
| `X.output` | output matrix with rows corresponding to outputs corresponding to examples (rows) in `X.input`. Depending on the value of `hidden.output`, these are outputs of units in the hidden layer (for `hidden.output=TRUE`) or in the output layer (for `hidden.output=FALSE`). |
| `hidden.output` | (same as `hidden.output` in the arguments) logical flag telling whether the outputs are generated by units in the hidden layer (for `hidden.output=TRUE`) or in the output layer (for `hidden.output=FALSE`) of the autoencoder. |
| `mean.error` | average, over rows, sum of `(X.output - X.input)^2`. |

### Author(s)

Eugene Dubossarsky (project leader, chief designer), Yuriy Tyshetskiy (design, implementation, testing)

### Examples

```
## Train the autoencoder on unlabeled set of 5000 image patches of
## size Nx.patch by Ny.patch, randomly cropped from 10 nature photos:

## Load a training matrix with rows corresponding to training examples,
## and columns corresponding to input channels (e.g., pixels in images):
data('training_matrix_N=5e3_Ninput=100')  ## the matrix contains 5e3 image
                                           ## patches of 10 by 10 pixels

## Set up the autoencoder architecture:
nl=3                         ## number of layers (default is 3: input, hidden, output)
unit.type = "logistic"       ## specify the network unit type, i.e., the unit's
                             ## activation function ("logistic" or "tanh")
Nx.patch=10                  ## width of training image patches, in pixels
Ny.patch=10                  ## height of training image patches, in pixels
N.input = Nx.patch*Ny.patch  ## number of units (neurons) in the input layer (one unit per pixel)
N.hidden = 10*10             ## number of units in the hidden layer
lambda = 0.0002              ## weight decay parameter
beta = 6                     ## weight of sparsity penalty term
rho = 0.01                   ## desired sparsity parameter
epsilon <- 0.001             ## a small parameter for initialization of weights
                          ## as small gaussian random numbers sampled from N(0,epsilon^2)
max.iterations = 2000        ## number of iterations in optimizer

## Train the autoencoder on training.matrix using BFGS optimization method
## (see help('optim') for details):
## Not run:
autoencoder.object <- autoencode(X.train=training.matrix,nl=nl,N.hidden=N.hidden,
          unit.type=unit.type,lambda=lambda,beta=beta,rho=rho,epsilon=epsilon,
          optim.method="BFGS",max.iterations=max.iterations,
          rescale.flag=TRUE,rescaling.offset=0.001)

## End(Not run)
## N.B.: Training this autoencoder takes a long time, so in this example we do not run the above
## autoencode function, but instead load the corresponding pre-trained autoencoder.object.

## Report mean squared error for training and test sets:
cat("autoencode(): mean squared error for training set: ",
round(autoencoder.object$mean.error.training.set,3),"\n")

## Visualize hidden units' learned features:
visualize.hidden.units(autoencoder.object,Nx.patch,Ny.patch)

## Compare the output and input images (the autoencoder learns to approximate
## inputs in its outputs using features learned by the hidden layer):

## Predict the output matrix corresponding to the training matrix
## (rows are examples, columns are input channels, i.e., pixels)
X.output <- predict(autoencoder.object, X.input=training.matrix, hidden.output=FALSE)$X.output

## Compare outputs and inputs for 3 image patches (patches 7,26,16 from
## the training set) - outputs should be similar to inputs:
op <- par(no.readonly = TRUE)   ## save the whole list of settable par's.
par(mfrow=c(3,2),mar=c(2,2,2,2))
for (n in c(7,26,16)){
## input image:
  image(matrix(training.matrix[n,],nrow=Ny.patch,ncol=Nx.patch),axes=FALSE,main="Input image",
  col=gray((0:32)/32))
## output image:
  image(matrix(X.output[n,],nrow=Ny.patch,ncol=Nx.patch),axes=FALSE,main="Output image",
  col=gray((0:32)/32))
}
par(op)  ## restore plotting par's
```

---

## `training_matrix_N=5e3_Ninput=100`

*An example training set of images for training sparse autoencoder*

---

### Description

This is an example set `training.matrix` of 5000 image patches of 10 by 10 pixels, randomly cropped from a set of 10 decoloured nature photos. The rows of `training.matrix` correspond to the training examples, the columns correspond to pixels of the image patches.

### Usage

```
data('training_matrix_N=5e3_Ninput=100')
```

### Format

The format is: chr "training_matrix_N=5e3_Ninput=100"

### Examples

```
data('training_matrix_N=5e3_Ninput=100') ## load the example training.matrix

## Set up the autoencoder architecture:
nl=3                          ## number of layers (default is 3: input, hidden, output)
unit.type = "logistic"        ## specify the network unit type, i.e., the unit's
                              ## activation function ("logistic" or "tanh")
Nx.patch=10                   ## width of training image patches, in pixels
Ny.patch=10                   ## height of training image patches, in pixels
N.input = Nx.patch*Ny.patch  ## number of units (neurons) in the input layer (one unit per pixel)
N.hidden = 10*10              ## number of units in the hidden layer
lambda = 0.0002               ## weight decay parameter
beta = 6                      ## weight of sparsity penalty term
rho = 0.01                    ## desired sparsity parameter
```

```
epsilon <- 0.001              ## a small parameter for initialization of weights
                          ## as small gaussian random numbers sampled from N(0,epsilon^2)
max.iterations = 2000         ## number of iterations in optimizer

## Train the autoencoder on training.matrix using BFGS optimization method
## (see help('optim') for details):

## Not run:
autoencoder.object <- autoencode(X.train=training.matrix,nl=nl,N.hidden=N.hidden,
          unit.type=unit.type,lambda=lambda,beta=beta,rho=rho,epsilon=epsilon,
          optim.method="BFGS",max.iterations=max.iterations,
          rescale.flag=TRUE,rescaling.offset=0.001)

## End(Not run)


## Extract weights W and biases b from autoencoder.object:
W <- autoencoder.object$W
b <- autoencoder.object$b
## Visualize learned features of the autoencoder:
visualize.hidden.units(autoencoder.object,Nx.patch,Ny.patch)
```

---

## visualize.hidden.units *Visualize features learned by a sparse autoencoder*

---

### Description

Visualizes features learned by a sparse autoencoder, by plotting (norm bounded) input images that maximally activate each of the hidden units of the trained autoencoder. Here it is assumed that the autoencoder is trained on a set of images of size Nx.patch by Ny.patch (in pixels).

### Usage

```
visualize.hidden.units(object, Nx.patch, Ny.patch)
```

### Arguments

| | |
|---|---|
| `object` | an object of class `autoencoder` produced by the `autoencode` function, and containing information (architecture, weights, biases, unit type, etc.) about the autoencoder network. |
| `Nx.patch` | width (in pixels) of images in data set used for training the autoencoder. See 'Examples'. |
| `Ny.patch` | height (in pixels) of images in data set used for training the autoencoder. See 'Examples'. |

## Value

A figure in which each square shows the (norm bounded) input image of size Nx.patch by Ny.patch (in pixels) that maximally activates each of the hidden units. These squares represent the features learned by the autoencoder from the unlabeled data used for its training.

## Author(s)

Yuriy Tyshetskiy

## Examples

```
## Load a pre-trained autoencoder object with N.input=100 and N.hidden=10*10,
## trained on unlabeled set of 5000 image patches of size
## Nx.patch=10 by Ny.patch=10 pixels,
## randomly cropped from 10 nature photos, and visualize the features
## learned by its hidden units:

data('autoencoder_Ninput=100_Nhidden=100_rho=1e-2')

## Visualize hidden units' learned features:
visualize.hidden.units(autoencoder.object,Nx.patch=10,Ny.patch=10)
```

# Index