Package ‘ruta’

March 18, 2019

Title  Implementation of Unsupervised Neural Architectures

Version  1.1.0


License  GPL (>= 3) | file LICENSE

URL  https://github.com/fdavidcl/ruta

BugReports  https://github.com/fdavidcl/ruta/issues

Depends  R (>= 3.2)

Imports  graphics (>= 3.2.3), keras (>= 2.2.4), purrr (>= 0.2.4), R.utils (>= 2.7.0), stats (>= 3.2.3), utils

Suggests  knitr, magrittr (>= 1.5), rmarkdown, testthat (>= 2.0.0)

VignetteBuilder  knitr

Encoding  UTF-8

LazyData  true

RoxygenNote  6.1.1

SystemRequirements  Python (>= 2.7); keras <https://keras.io/> (>= 2.1)

NeedsCompilation  no

Author  David Charte [aut, cre] (<https://orcid.org/0000-0002-4830-9512>), Francisco Charte [aut] (<https://orcid.org/0000-0002-3083-8942>), Francisco Herrera [aut]

Maintainer  David Charte <fdavidcl@ugr.es>

Repository  CRAN

Date/Publication  2019-03-18 13:10:02 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>+.ruta_network</td>
<td>3</td>
</tr>
<tr>
<td>add_weight_decay</td>
<td>4</td>
</tr>
<tr>
<td>apply_filter.ruta_noise_zeros</td>
<td>4</td>
</tr>
<tr>
<td>as_loss</td>
<td>5</td>
</tr>
<tr>
<td>as_network</td>
<td>6</td>
</tr>
<tr>
<td>autoencode</td>
<td>6</td>
</tr>
<tr>
<td>autoencoder</td>
<td>7</td>
</tr>
<tr>
<td>autoencoder_contractive</td>
<td>8</td>
</tr>
<tr>
<td>autoencoder_denoising</td>
<td>9</td>
</tr>
<tr>
<td>autoencoder_robust</td>
<td>10</td>
</tr>
<tr>
<td>autoencoder_sparse</td>
<td>11</td>
</tr>
<tr>
<td>autoencoder_variational</td>
<td>12</td>
</tr>
<tr>
<td>contraction</td>
<td>13</td>
</tr>
<tr>
<td>conv</td>
<td>13</td>
</tr>
<tr>
<td>correntropy</td>
<td>14</td>
</tr>
<tr>
<td>decode</td>
<td>15</td>
</tr>
<tr>
<td>dense</td>
<td>15</td>
</tr>
<tr>
<td>dropout</td>
<td>16</td>
</tr>
<tr>
<td>encode</td>
<td>17</td>
</tr>
<tr>
<td>encoding_index</td>
<td>17</td>
</tr>
<tr>
<td>evaluate_mean_squared_error</td>
<td>18</td>
</tr>
<tr>
<td>evaluation_metric</td>
<td>19</td>
</tr>
<tr>
<td>generate.ruta_autoencoder_variational</td>
<td>19</td>
</tr>
<tr>
<td>input</td>
<td>20</td>
</tr>
<tr>
<td>is_contractive</td>
<td>21</td>
</tr>
<tr>
<td>is_denoising</td>
<td>21</td>
</tr>
<tr>
<td>is_robust</td>
<td>22</td>
</tr>
<tr>
<td>is_sparse</td>
<td>22</td>
</tr>
<tr>
<td>is_trained</td>
<td>23</td>
</tr>
<tr>
<td>is_variational</td>
<td>23</td>
</tr>
<tr>
<td>layer_keras</td>
<td>24</td>
</tr>
<tr>
<td>loss_variational</td>
<td>24</td>
</tr>
<tr>
<td>make_contractive</td>
<td>25</td>
</tr>
<tr>
<td>make_denoising</td>
<td>26</td>
</tr>
<tr>
<td>make_robust</td>
<td>26</td>
</tr>
<tr>
<td>make_sparse</td>
<td>26</td>
</tr>
<tr>
<td>new_autoencoder</td>
<td>27</td>
</tr>
<tr>
<td>new_layer</td>
<td>28</td>
</tr>
<tr>
<td>new_network</td>
<td>28</td>
</tr>
<tr>
<td>noise</td>
<td>29</td>
</tr>
<tr>
<td>noise_cauchy</td>
<td>30</td>
</tr>
<tr>
<td>noise_gaussian</td>
<td>30</td>
</tr>
<tr>
<td>noise_gaussian</td>
<td>31</td>
</tr>
<tr>
<td>noise_ones</td>
<td>31</td>
</tr>
<tr>
<td>noise_saltpepper</td>
<td>32</td>
</tr>
<tr>
<td>noise_zeros</td>
<td>32</td>
</tr>
<tr>
<td>output</td>
<td>33</td>
</tr>
</tbody>
</table>
+.ruta_network

plot.ruta_network ................................................................. 33
print.ruta_autoencoder .......................................................... 34
reconstruct ................................................................. 35
save_as ................................................................. 35
sparsity ................................................................. 36
to_keras ................................................................. 37
to_keras.ruta_autoencoder ..................................................... 37
to_keras.ruta_filter ............................................................ 38
to_keras.ruta_layer_input ...................................................... 39
to_keras.ruta_layer_variational .............................................. 39
to_keras.ruta_loss_contraction .............................................. 40
to_keras.ruta_network .......................................................... 41
to_keras.ruta_sparsity .......................................................... 42
to_keras.ruta_weight_decay ................................................... 42
train.ruta_autoencoder ......................................................... 43
variational_block .............................................................. 44
weight_decay ................................................................. 45
[.ruta_network ................................................................. 46

Index 47

+.ruta_network  Add layers to a network/Join networks

Description

Add layers to a network/Join networks

Usage

## S3 method for class 'ruta_network'
e1 + e2

## S3 method for class 'ruta_network'
c(...)

Arguments

e1  First network
e2  Second network
...
networks or layers to be concatenated

Value

Network combination
Example

network <- input() + dense(30) + output("sigmoid")
another <- c(input(), dense(30), dense(3), dense(30), output())

add_weight_decay 
Add weight decay to any autoencoder

Description

Adds a weight decay regularization to the encoding layer of a given autoencoder

Usage

add_weight_decay(learner, decay = 0.02)

Arguments

learner The "ruta_autoencoder" object
decay Numeric value indicating the amount of decay

Value

An autoencoder object which contains the weight decay

apply_filter.ruta_noise_zeros
Apply filters

Description

Apply a filter to input data, generally a noise filter in order to train a denoising autoencoder. Users won't generally need to use these functions

Usage

## S3 method for class 'ruta_noise_zeros'
apply_filter(filter, data, ...)

## S3 method for class 'ruta_noise_ones'
apply_filter(filter, data, ...)

## S3 method for class 'ruta_noise_saltpepper'
apply_filter(filter, data, ...)

## S3 method for class 'ruta_noise_gaussian'
as_loss

apply_filter(filter, data, ...)

## S3 method for class 'ruta_noise_cauchy'
apply_filter(filter, data, ...)

Arguments

filter Filter object to be applied
data Input data to be filtered...

Other parameters

See Also

autoencoder_denoising

---

as_loss Coercion to ruta_loss

Description

Generic function to coerce objects into loss objects.

Usage

as_loss(x)

## S3 method for class 'character'
as_loss(x)

## S3 method for class 'ruta_loss'
as_loss(x)

Arguments

x Object to be converted into a loss

Value

A "ruta_loss" construct
as_network

Coercion to ruta_network

Description

Generic function to coerce objects into networks.

Usage

as_network(x)

## S3 method for class 'ruta_layer'
as_network(x)

## S3 method for class 'ruta_network'
as_network(x)

## S3 method for class 'numeric'
as_network(x)

## S3 method for class 'integer'
as_network(x)

Arguments

x

Object to be converted into a network

Value

A "ruta_network" construct

Examples

net <- as_network(c(784, 1000, 32))

autoencode

Automatically compute an encoding of a data matrix

Description

Trains an autoencoder adapted to the data and extracts its encoding for the same data matrix.

Usage

autoencode(data, dim, type = "basic", activation = "linear", epochs = 20)
Arguments

- **data**: Numeric matrix to be encoded
- **dim**: Number of variables to be used in the encoding
- **type**: Type of autoencoder to use: "basic", "sparse", "contractive", "denoising", "robust" or "variational"
- **activation**: Activation type to be used in the encoding layer. Some available activations are "tanh", "sigmoid", "relu", "elu" and "selu"
- **epochs**: Number of times the data will traverse the autoencoder to update its weights

Value

- Matrix containing the encodings

See Also

- `autoencoder`

Examples

```r
inputs <- as.matrix(iris[, 1:4])

# Train a basic autoencoder and generate a 2-variable encoding
encoded <- autoencode(inputs, 2)

# Train a contractive autoencoder with tanh activation
encoded <- autoencode(inputs, 2, type = "contractive", activation = "tanh")
```

autoencoder

Create an autoencoder learner

Description

Represents a generic autoencoder network.

Usage

```r
autoencoder(network, loss = "mean_squared_error")
```

Arguments

- **network**: Layer construct of class "ruta_network" or coercible
- **loss**: A "ruta_loss" object or a character string specifying a loss function
Value

A construct of class "ruta_autoencoder"

References

- A practical tutorial on autoencoders for nonlinear feature fusion

See Also

- train.ruta_autoencoder
  
Other autoencoder variants: autoencoder_contractive, autoencoder_denoising, autoencoder_robust, autoencoder_sparse, autoencoder_variational

Examples

```r
# Basic autoencoder with a network of [input]-256-36-256-[input] and
# no nonlinearities
autoencoder(c(256, 36), loss = "binary_crossentropy")

# Customizing the activation functions in the same network
network <-
  input() +
  dense(256, "relu") +
  dense(36, "tanh") +
  dense(256, "relu") +
  output("sigmoid")

learner <- autoencoder(
  network,
  loss = "binary_crossentropy"
)
```

autoencoder_contractive

Create a contractive autoencoder

Description

A contractive autoencoder adds a penalty term to the loss function of a basic autoencoder which attempts to induce a contraction of data in the latent space.

Usage

```r
autoencoder_contractive(network, loss = "mean_squared_error",
  weight = 2e-04)
```
autoencoder_denoising

Arguments

- **network**: Layer construct of class "ruta_network"
- **loss**: Character string specifying the reconstruction error part of the loss function
- **weight**: Weight assigned to the contractive loss

Value

A construct of class "ruta_autoencoder"

References

- A practical tutorial on autoencoders for nonlinear feature fusion

See Also

Other autoencoder variants: autoencoder_denoising, autoencoder_robust, autoencoder_sparse, autoencoder_variational, autoencoder

autoencoder_denoising  Create a denoising autoencoder

Description

A denoising autoencoder trains with noisy data in order to create a model able to reduce noise in reconstructions from input data

Usage

autoencoder_denoising(network, loss = "mean_squared_error",
                      noise_type = "zeros", ...)

Arguments

- **network**: Layer construct of class "ruta_network"
- **loss**: Loss function to be optimized
- **noise_type**: Type of data corruption which will be used to train the autoencoder, as a character string. Available types:
  - "zeros" Randomly set components to zero (noise_zeros)
  - "ones" Randomly set components to one (noise_ones)
  - "saltpepper" Randomly set components to zero or one (noise_saltpepper)
  - "gaussian" Randomly offset each component of an input as drawn from Gaussian distributions with the same variance (additive Gaussian noise, noise_gaussian)
  - "cauchy" Randomly offset each component of an input as drawn from Cauchy distributions with the same scale (additive Cauchy noise, noise_cauchy)
Extra parameters to customize the noisy filter:

- \( p \) The probability that each instance in the input data which will be altered by random noise (for "zeros", "ones" and "saltpepper")
- \( \text{var or sd} \) The variance or standard deviation of the Gaussian distribution from which additive noise will be drawn (for "gaussian", only one of those parameters is necessary)
- \( \text{scale} \) For the Cauchy distribution

Value

A construct of class "ruta_autoencoder"

References

- Extracting and composing robust features with denoising autoencoders

See Also

Other autoencoder variants: autoencoder_contractive, autoencoder_robust, autoencoder_sparse, autoencoder_variational, autoencoder

```
autoencoder_robust(network, sigma = 0.2)
```

Arguments

- \( \text{network} \) Layer construct of class "ruta_network"
- \( \text{sigma} \) Sigma parameter in the kernel used for correntropy

Value

A construct of class "ruta_autoencoder"

References

- Robust feature learning by stacked autoencoder with maximum correntropy criterion
autoencoder_sparse

See Also

Other autoencoder variants: autoencoder_contractive, autoencoder_denoising, autoencoder_sparse, autoencoder_variational, autoencoder

---

autoencoder_sparse  Sparse autoencoder

Description

Creates a representation of a sparse autoencoder.

Usage

autoencoder_sparse(network, loss = "mean_squared_error", high_probability = 0.1, weight = 0.2)

Arguments

network  Layer construct of class "ruta_network"
loss  Character string specifying a loss function
high_probability  Expected probability of the high value of the encoding layer. Set this to a value near zero in order to minimize activations in that layer.
weight  The weight of the sparsity regularization

Value

A construct of class "ruta_autoencoder"

References

• Sparse deep belief net model for visual area V2
• Andrew Ng. Sparse Autoencoder. CS294A Lecture Notes

See Also

sparsity, make_sparse, is_sparse

Other autoencoder variants: autoencoder_contractive, autoencoder_denoising, autoencoder_robust, autoencoder_variational, autoencoder
autoencoder_variational

Build a variational autoencoder

Description

A variational autoencoder assumes that a latent, unobserved random variable produces the observed data and attempts to approximate its distribution. This function constructs a wrapper for a variational autoencoder using a Gaussian distribution as the prior of the latent space.

Usage

autoencoder_variational(network, loss = "binary_crossentropy", auto_transform_network = TRUE)

Arguments

- **network**: Network architecture as a "ruta_network" object (or coercible)
- **loss**: Reconstruction error to be combined with KL divergence in order to compute the variational loss
- **auto_transform_network**: Boolean: convert the encoding layer into a variational block if none is found?

Value

A construct of class "ruta_autoencoder"

References

- Auto-Encoding Variational Bayes
- Under the Hood of the Variational Autoencoder (in Prose and Code)
- Keras example: Variational autoencoder

See Also

Other autoencoder variants: autoencoder_contractive, autoencoder_denoising, autoencoder_robust, autoencoder_sparse, autoencoder

Examples

```r
network <-
  input() +
  dense(256, "elu") +
  variational_block(3) +
  dense(256, "elu") +
  output("sigmoid")

learner <- autoencoder_variational(network, loss = "binary_crossentropy")
```
contraction

<table>
<thead>
<tr>
<th>contraction</th>
<th>Contractive loss</th>
</tr>
</thead>
</table>

**Description**

This is a wrapper for a loss which induces a contraction in the latent space.

**Usage**

```python
contraction(reconstruction_loss = "mean_squared_error", weight = 2e-04)
```

**Arguments**

- **reconstruction_loss**
  
  Original reconstruction error to be combined with the contractive loss (e.g. "binary_crossentropy")

- **weight**
  
  Weight assigned to the contractive loss

**Value**

A loss object which can be converted into a Keras loss

**See Also**

```python
autoencoder_contractive
```

Other loss functions: correntropy, loss_variational

conv

<table>
<thead>
<tr>
<th>conv</th>
<th>Create a convolutional layer</th>
</tr>
</thead>
</table>

**Description**

Wrapper for a convolutional layer. The dimensions of the convolution operation are inferred from the shape of the input data. This shape must follow the pattern (batch_shape, x, [y, [z, ]], channel) where dimensions y and z are optional, and channel will be either 1 for grayscale images or generally 3 for colored ones.

**Usage**

```python
conv(filters, kernel_size, padding = "same", max_pooling = NULL, average_pooling = NULL, upsampling = NULL, activation = "linear")
```
**Arguments**

- **filters**: Number of filters learned by the layer
- **kernel_size**: Integer or list of integers indicating the size of the weight matrices to be convolved with the image
- **padding**: One of "valid" or "same" (case-insensitive). See `layer_conv_2d` for more details
- **max_pooling**: NULL or an integer indicating the reduction ratio for a max pooling operation after the convolution
- **average_pooling**: NULL or an integer indicating the reduction ratio for an average pooling operation after the convolution
- **upsampling**: NULL or an integer indicating the augmentation ratio for an upsampling operation after the convolution
- **activation**: Optional, string indicating activation function (linear by default)

**Value**

A construct with class "ruta_network"

**See Also**

Other neural layers: `dense`, `dropout`, `input`, `layer_keras`, `output`, `variational_block`

**Examples**

```r
# Sample convolutional autoencoder
net <- input() +
  conv(16, 3, max_pooling = 2, activation = "relu") +
  conv(8, 3, max_pooling = 2, activation = "relu") +
  conv(8, 3, upsampling = 2, activation = "relu") +
  conv(16, 3, upsampling = 2, activation = "relu") +
  conv(1, 3, activation = "sigmoid")
```

---

**correntropy**  
**Correntropy loss**

**Description**

A wrapper for the correntropy loss function

**Usage**

```r
correntropy(sigma = 0.2)
```

**Arguments**

- **sigma**: Sigma parameter in the kernel
decode

Value
A "ruta_loss" object

See Also
autoencoder_robust

Other loss functions: contraction, loss_variational

decode  Retrieve decoding of encoded data

Description
Extracts the decodification calculated by a trained autoencoder for the specified data.

Usage
decode(learner, data)

Arguments
learner  Trained autoencoder model
data      data.frame to be decoded

Value
Matrix containing the decodifications

See Also
encode, reconstruct

dense  Create a fully-connected neural layer

Description
Wrapper for a dense/fully-connected layer.

Usage
dense(units, activation = "linear")
**Arguments**

- **units**: Number of units
- **activation**: Optional, string indicating activation function (linear by default)

**Value**

A construct with class "ruta_network"

**See Also**

Other neural layers: *conv, dropout, input, layer_keras, output, variational_block*

**Examples**

```r
dense(30, "tanh")
```

---

**Dropout layer**

**Description**

Randomly sets a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

**Usage**

```r
dropout(rate = 0.5)
```

**Arguments**

- **rate**: The fraction of affected units

**Value**

A construct of class "ruta_network"

**See Also**

Other neural layers: *conv, dense, input, layer_keras, output, variational_block*
**encode**

**Description**

Extracts the encoding calculated by a trained autoencoder for the specified data.

**Usage**

\[\text{encode(learner, data)}\]

**Arguments**

- **learner**: Trained autoencoder model
- **data**: data.frame to be encoded

**Value**

Matrix containing the encodings

**See Also**

- decode
- reconstruct

**encoding_index**

**Description**

Calculates the index of the middle layer of an encoder-decoder network.

**Usage**

\[\text{encoding_index(net)}\]

**Arguments**

- **net**: A network of class "ruta_network"

**Value**

Index of the middle layer
**Description**

Performance evaluation metrics for autoencoders

**Usage**

```r
evaluate_mean_squared_error(learner, data, ...)
```

```r
evaluate_mean_absolute_error(learner, data, ...)
```

```r
evaluate_binary_crossentropy(learner, data, ...)
```

```r
evaluate_binary_accuracy(learner, data, ...)
```

```r
evaluate_kullback_leibler_divergence(learner, data, ...)
```

**Arguments**

- `learner` A trained learner object
- `data` Test data for evaluation
- `...` Additional parameters passed to `keras::evaluate`

**Value**

A named list with the autoencoder training loss and evaluation metric for the given data

**See Also**

`evaluation_metric`

**Examples**

```r
library(purrr)

x <- as.matrix(sample(iris[, 1:4]))
x_train <- x[1:100,]
x_test <- x[101:150,]

autoencoder(2) %>%
  train(x_train) %>%
  evaluate_mean_squared_error(x_test)
```
**evaluation_metric**

---

**Custom evaluation metrics**

**Description**

Create a different evaluation metric from a valid Keras metric

**Usage**

```r
evaluation_metric(evaluate_f)
```

**Arguments**

- `evaluate_f`: Must be either a metric function defined by Keras (e.g. `keras::metric_binary_crossentropy`) or a valid function for Keras to create a performance metric (see `metric_binary_accuracy` for details).

**Value**

A function which can be called with parameters `learner` and `data` just like the ones defined in `evaluate`.

**See Also**

- `evaluate`

---

**generate.ruta_autoencoder_variational**

---

*Generate samples from a generative model*

**Description**

Generate samples from a generative model

**Usage**

```r
## S3 method for class 'ruta_autoencoder_variational'
generate(learner,
  dimensions = c(1, 2), from = 0.05, to = 0.95, side = 10,
  fixed_values = 0.5, ...)

generate(learner, ...)
```
Arguments

learner  Trained learner object
dimensions  Indices of the dimensions over which the model will be sampled
from  Lower limit on the values which will be passed to the inverse CDF of the prior
to  Upper limit on the values which will be passed to the inverse CDF of the prior
side  Number of steps to take in each traversed dimension
fixed_values  Value used as parameter for the inverse CDF of all non-traversed dimensions
...  Unused

See Also

autoencoder_variational

input

Create an input layer

Description

This layer acts as a placeholder for input data. The number of units is not needed as it is deduced from the data during training.

Usage

input()

Value

A construct with class "ruta_network"

See Also

Other neural layers: conv, dense, dropout, layer_keras, output, variational_block
is_contractive

Detect whether an autoencoder is contractive

Description
Detect whether an autoencoder is contractive

Usage
is_contractive(learner)

Arguments
learner A "ruta_autoencoder" object

Value
Logical value indicating if a contractive loss was found

See Also
contraction, autoencoder_contractive, make_contractive

is_denoising

Detect whether an autoencoder is denoising

Description
Detect whether an autoencoder is denoising

Usage
is_denoising(learner)

Arguments
learner A "ruta_autoencoder" object

Value
Logical value indicating if a noise generator was found

See Also
noise, autoencoder_denoising, make_denoising
is_robust

Detect whether an autoencoder is robust

Description
Detect whether an autoencoder is robust

Usage
is_robust(learner)

Arguments
learner A "ruta_autoencoder" object

Value
Logical value indicating if a correntropy loss was found

See Also
correntropy, autoencoder_robust, make_robust

is_sparse

Detect whether an autoencoder is sparse

Description
Detect whether an autoencoder is sparse

Usage
is_sparse(learner)

Arguments
learner A "ruta_autoencoder" object

Value
Logical value indicating if a sparsity regularization in the encoding layer was found

See Also
sparsity, autoencoder_sparse, make_sparse
is_trained  

Detect trained models

Description

Inspects a learner and figures out whether it has been trained

Usage

is_trained(learner)

Arguments

learner  
Learner object

Value

A boolean

See Also

train

is_variational  

Detect whether an autoencoder is variational

Description

Detect whether an autoencoder is variational

Usage

is_variational(learner)

Arguments

learner  
A "ruta_autoencoder" object

Value

Logical value indicating if a variational loss was found

See Also

autoencoder_variational
**layer_keras**  
*Custom layer from Keras*

**Description**
Gets any layer available in Keras with the specified parameters.

**Usage**
```
layer_keras(type, ...)  
```

**Arguments**
- `type`  
  The name of the layer, e.g. "activity_regularization" for a `keras::layer_activity_regularization` object
- `...`  
  Named parameters for the Keras layer constructor

**Value**
A wrapper for the specified layer, which can be combined with other Ruta layers.

**See Also**
Other neural layers: `conv`, `dense`, `dropout`, `input`, `output`, `variational_block`

---

**loss_variational**  
*Variational loss*

**Description**
Specifies an evaluation function adapted to the variational autoencoder. It combines a base reconstruction error and the Kullback-Leibler divergence between the learned distribution and the true latent posterior.

**Usage**
```
loss_variational(reconstruction_loss)  
```

**Arguments**
- `reconstruction_loss`  
  Another loss to be used as reconstruction error (e.g. "binary_crossentropy")

**Value**
A "ruta_loss" object
**make_contractive**

**References**
- Auto-Encoding Variational Bayes
- Under the Hood of the Variational Autoencoder (in Prose and Code)
- Keras example: Variational autoencoder

**See Also**
- autoencoder_variational

Other loss functions: *contraction*, *correntropy*

```r
make_contractive(learner, weight = 2e-04)
```

**Description**
Converts an autoencoder into a contractive one by assigning a contractive loss to it

**Usage**

```r
make_contractive(learner, weight = 2e-04)
```

**Arguments**
- `learner`  The "ruta_autoencoder" object
- `weight`   Weight assigned to the contractive loss

**Value**
An autoencoder object which contains the contractive loss

**See Also**
- autoencoder_contractive
make_denoising

Add denoising behavior to any autoencoder

Description

Converts an autoencoder into a denoising one by adding a filter for the input data

Usage

make_denoising(learner, noise_type = "zeros", ...)

Arguments

learner
The "ruta_autoencoder" object

noise_type
Type of data corruption which will be used to train the autoencoder, as a character string. See autoencoder_denoising for details

... Extra parameters to customize the noisy filter. See autoencoder_denoising for details

Value

An autoencoder object which contains the noisy filter

See Also

autoencoder_denoising

make_robust

Add robust behavior to any autoencoder

Description

Converts an autoencoder into a robust one by assigning a correntropy loss to it. Notice that this will replace the previous loss function

Usage

make_robust(learner, sigma = 0.2)

Arguments

learner
The "ruta_autoencoder" object

sigma
Sigma parameter in the kernel used for correntropy
**make_sparse**

**Value**

An autoencoder object which contains the correntropy loss

**See Also**

`autoencoder_robust`

---

**make_sparse**

Add sparsity regularization to an autoencoder

**Description**

Add sparsity regularization to an autoencoder

**Usage**

`make_sparse(learner, high_probability = 0.1, weight = 0.2)`

**Arguments**

- **learner**: A "ruta_autoencoder" object
- **high_probability**: Expected probability of the high value of the encoding layer. Set this to a value near zero in order to minimize activations in that layer.
- **weight**: The weight of the sparsity regularization

**Value**

The same autoencoder with the sparsity regularization applied

**See Also**

`sparsity, autoencoder_sparse, is_sparse`
new_autoencoder  Create an autoencoder learner

Description

Internal function to create autoencoder objects. Instead, consider using autoencoder.

Usage

new_autoencoder(network, loss, extra_class = NULL)

Arguments

network Layer construct of class "ruta_network" or coercible
loss A "ruta_loss" object or a character string specifying a loss function
extra_class Vector of classes in case this autoencoder needs to support custom methods (for to_keras, train, generate or others)

Value

A construct of class "ruta_autoencoder"

new_layer  Layer wrapper constructor

Description

Constructor function for layers. You shouldn’t generally need to use this. Instead, consider using individual functions such as dense.

Usage

new_layer(cl, ...)

Arguments

cl Character string specifying class of layer (e.g. "ruta_layer_dense"), which will be used to call the corresponding methods
... Other parameters (usually units, activation)

Value

A construct with class "ruta_layer"
new_network

Examples

```r
my_layer <- new_layer("dense", 30, "tanh")

# Equivalent:
my_layer <- dense(30, "tanh")[[1]]
```

---

**new_network**  
*Sequential network constructor*

**Description**

Constructor function for networks composed of several sequentially placed layers. You shouldn’t generally need to use this. Instead, consider concatenating several layers with `+.ruta_network`.

**Usage**

```r
new_network(...)```

**Arguments**

...  
Zero or more objects of class "ruta_layer"

**Value**

A construct with class "ruta_network"

**Examples**

```r
my_network <- new_network(
  new_layer("input", 784, "linear"),
  new_layer("dense", 32, "tanh"),
  new_layer("dense", 784, "sigmoid")
)

# Instead, consider using
my_network <- input() + dense(32, "tanh") + output("sigmoid")
```
noise | Noise generator

Description
Delegates on noise classes to generate noise of some type

Usage
noise(type, ...)

Arguments

<table>
<thead>
<tr>
<th>type</th>
<th>Type of noise, as a character string</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Parameters for each noise class</td>
</tr>
</tbody>
</table>

noise_cauchy | Additive Cauchy noise

Description
A data filter which adds noise from a Cauchy distribution to instances

Usage
noise_cauchy(scale = 0.005)

Arguments

| scale | Scale for the Cauchy distribution |

Value
Object which can be applied to data with apply_filter

See Also
Other noise generators: noise_gaussian, noise Ones, noise_saltpepper, noise_zeros
### noise_gaussian

**Additive Gaussian noise**

**Description**
A data filter which adds Gaussian noise to instances

**Usage**

```r
noise_gaussian(sd = NULL, var = NULL)
```

**Arguments**
- `sd`: Standard deviation for the Gaussian distribution
- `var`: Variance of the Gaussian distribution (optional, only used if `sd` is not provided)

**Value**
Object which can be applied to data with `apply_filter`

**See Also**
Other noise generators: `noise_cauchy`, `noise_ones`, `noise_saltpepper`, `noise_zeros`

### noise_ones

**Filter to add ones noise**

**Description**
A data filter which replaces some values with ones

**Usage**

```r
noise_ones(p = 0.05)
```

**Arguments**
- `p`: Probability that a feature in an instance is set to one

**Value**
Object which can be applied to data with `apply_filter`

**See Also**
Other noise generators: `noise_cauchy`, `noise_gaussian`, `noise_saltpepper`, `noise_zeros`
noise_saltpepper  
*Filter to add salt-and-pepper noise*

**Description**
A data filter which replaces some values with zeros or ones

**Usage**
```python
noise_saltpepper(p = 0.05)
```

**Arguments**
- `p`  
  Probability that a feature in an instance is set to zero or one

**Value**
Object which can be applied to data with `apply_filter`

**See Also**
Other noise generators: `noise_cauchy`, `noise_gaussian`, `noise_ones`, `noise_zeros`

---

noise_zeros  
*Filter to add zero noise*

**Description**
A data filter which replaces some values with zeros

**Usage**
```python
noise_zeros(p = 0.05)
```

**Arguments**
- `p`  
  Probability that a feature in an instance is set to zero

**Value**
Object which can be applied to data with `apply_filter`

**See Also**
Other noise generators: `noise_cauchy`, `noise_gaussian`, `noise_ones`, `noise_saltpepper`
Create an output layer

Description
This layer acts as a placeholder for the output layer in an autoencoder. The number of units is not needed as it is deduced from the data during training.

Usage
output(activation = "linear")

Arguments
activation Optional, string indicating activation function (linear by default)

Value
A construct with class "ruta_network"

See Also
Other neural layers: conv, dense, dropout, input, layer_keras, variational_block

Draw a neural network

Description
Draw a neural network

Usage
## S3 method for class 'ruta_network'
plot(x, ...)

Arguments
x A "ruta_network" object
... Additional parameters for style. Available parameters:
• bg: Color for the text over layers
• fg: Color for the background of layers
• log: Use logarithmic scale
print.ruta_autoencoder

Examples

```r
net <-
  input() +
  dense(1000, "relu") + dropout() +
  dense(100, "tanh") +
  dense(1000, "relu") + dropout() +
  output("sigmoid")
plot(net, log = TRUE, fg = "#30707a", bg = "#e0e6ea")
```

print.ruta_autoencoder

Inspect Ruta objects

Description

Inspect Ruta objects

Usage

```r
## S3 method for class 'ruta_autoencoder'
print(x, ...)

## S3 method for class 'ruta_loss_named'
print(x, ...)

## S3 method for class 'ruta_loss'
print(x, ...)

## S3 method for class 'ruta_network'
print(x, ...)
```

Arguments

- `x` An object
- `...` Unused

Value

Invisibly returns the same object passed as parameter

Examples

```r
print(autoencoder(c(256, 10), loss = correntropy()))
```
reconstruct

Retrieve reconstructions for input data

Description

Extracts the reconstructions calculated by a trained autoencoder for the specified input data after encoding and decoding. `predict` is an alias for `reconstruct`.

Usage

```r
reconstruct(learner, data)

## S3 method for class 'rura_autoencoder'
predict(object, ...)
```

Arguments

- **learner**: Trained autoencoder model
- **data**: data.frame to be passed through the network
- **object**: Trained autoencoder model
- **...**: Rest of parameters, unused

Value

Matrix containing the reconstructions

See Also

`encode`, `decode`

save_as

Save and load Ruta models

Description

Functions to save a trained or untrained Ruta learner into a file and load it

Usage

```r
save_as(learner, file = paste0(substitute(learner), ",.tar.gz"), dir, compression = "gzip")

load_from(file)
```
Arguments

learner  The "ruta_autoencoder" object to be saved
file     In save, filename with extension (usually .tar.gz) where the object will be saved. In load, path to the saved model
dir      Directory where to save the file. Use "." to save in the current working directory or tempdir() to use a temporary one
compression Type of compression to be used, for R function tar

Value

save_as returns the filename where the model has been saved, load_from returns the loaded model as a "ruta_autoencoder" object

Examples

library(purrr)

x <- as.matrix(iris[, 1:4])

# Save a trained model
saved_file <-
  autoencoder(2) %>%
  train(x) %>%
  save_as("my_model.tar.gz", dir = tempdir())

# Load and use the model
encoded <- load_from(saved_file) %>% encode(x)

sparsity | Sparsity regularization

Description

Sparsity regularization

Usage

sparsity(high_probability, weight)

Arguments

high_probability Expected probability of the high value of the encoding layer. Set this to a value near zero in order to minimize activations in that layer.

weight The weight of the sparsity regularization
Value
A Ruta regularizer object for the sparsity, to be inserted in the encoding layer.

References
- Sparse deep belief net model for visual area V2
- Andrew Ng, Sparse Autoencoder. CS294A Lecture Notes

See Also
autoencoder_sparse, make_sparse, is_spare

---

### to_keras
Convert a Ruta object onto Keras objects and functions

**Description**
Generic function which uses the Keras API to build objects out of Ruta wrappers

**Usage**
```
to_keras(x, ...)
```

**Arguments**
- `x` Object to be converted
- `...` Remaining parameters depending on the method

---

### to_keras.ruta_autoencoder
Extract Keras models from an autoencoder wrapper

**Description**
Extract Keras models from an autoencoder wrapper

**Usage**
```
## S3 method for class 'ruta_autoencoder'
to_keras(learner, encoder_end = "encoding",
         decoder_start = "encoding", weights_file = NULL)

## S3 method for class 'ruta_autoencoder_variational'
to_keras(learner, ...)
```
to_keras.ruta_filter

Arguments

learner Object of class "ruta_autoencoder". Needs to have a member input_shape indicating the number of attributes of the input data

encoder_end Name of the Keras layer where the encoder ends

decoder_start Name of the Keras layer where the decoder starts

weights_file The name of a hdf5 weights file in order to load from a trained model

... Additional parameters for to_keras.ruta_autoencoder

Value

A list with several Keras models:

- autoencoder: model from the input layer to the output layer
- encoder: model from the input layer to the encoding layer
- decoder: model from the encoding layer to the output layer

See Also

autoencoder

to_keras.ruta_filter Get a Keras generator from a data filter

Description

Noise filters can be applied during training (in denoising autoencoders), for this a generator is used to get data batches.

Usage

```r
## S3 method for class 'ruta_filter'
to_keras(x, data, batch_size, ...)
```

Arguments

x Filter object
data Matrix where the filter will be applied
batch_size Size of the sample (for the training stage)
... Additional parameters, currently unused
to_keras.ruta_layer_input

Convert Ruta layers onto Keras layers

Description
Convert Ruta layers onto Keras layers

Usage

```r
## S3 method for class 'ruta_layer_input'
to_keras(x, input_shape, ...)

## S3 method for class 'ruta_layer_dense'
to_keras(x, input_shape,
  model = keras::keras_model_sequential(), ...)

## S3 method for class 'ruta_layer_conv'
to_keras(x, input_shape,
  model = keras::keras_model_sequential(), ...)

## S3 method for class 'ruta_layer_custom'
to_keras(x, input_shape,
  model = keras::keras_model_sequential(), ...)
```

Arguments

- `x` The layer object
- `input_shape` Number of features in training data
- `...` Unused
- `model` Keras model where the layer will be added

Value
A Layer object from Keras

to_keras.ruta_layer_variational

Obtain a Keras block of layers for the variational autoencoder

Description
This block contains two dense layers representing the mean and log var of a Gaussian distribution and a sampling layer.
Usage

```r
## S3 method for class 'ruta_layer_variational'
to_keras(x, input_shape,
    model = keras::keras_model_sequential(), ...)
```

Arguments

- `x` The layer object
- `input_shape` Number of features in training data
- `model` Keras model where the layers will be added
- `...` Unused

Value

A Layer object from Keras

References

- Auto-Encoding Variational Bayes
- Under the Hood of the Variational Autoencoder (in Prose and Code)
- Keras example: Variational autoencoder

---

to_keras.ruta_loss_contraction

*Obtain a Keras loss*

Description

Builds the Keras loss function corresponding to a name

Usage

```r
## S3 method for class 'ruta_loss_contraction'
to_keras(x, learner, ...)

## S3 method for class 'ruta_loss_correntropy'
to_keras(x, ...)

## S3 method for class 'ruta_loss_variational'
to_keras(x, learner, ...)

## S3 method for class 'ruta_loss_named'
to_keras(x, ...)
```
Arguments

- **x**: A "ruta_loss_named" object
- **learner**: The learner object including the keras model which will use the loss function
- ... Rest of parameters, ignored

Value

A function which returns the corresponding loss for given true and predicted values

References

- Contractive loss: Deriving Contractive Autoencoder and Implementing it in Keras
- Correntropy loss: Robust feature learning by stacked autoencoder with maximum correntropy criterion
- Variational loss:
  - Auto-Encoding Variational Bayes
  - Under the Hood of the Variational Autoencoder (in Prose and Code)
  - Keras example: Variational autoencoder

---

**to_keras.ruta_network  Build a Keras network**

Description

Build a Keras network

Usage

```
## S3 method for class 'ruta_network'
to_keras(x, input_shape)
```

Arguments

- **x**: A "ruta_network" object
- **input_shape**: The length of each input vector (number of input attributes)

Value

A list of Keras Tensor objects with an attribute "encoding" indicating the index of the encoding layer
to_keras.ruta_sparsity

*Translate sparsity regularization to Keras regularizer*

**Description**

Translate sparsity regularization to Keras regularizer

**Usage**

```r
## S3 method for class 'ruta_sparsity'
to_keras(x, activation)
```

**Arguments**

- `x` : Sparsity object
- `activation` : Name of the activation function used in the encoding layer

**Value**

Function which can be used as activity regularizer in a Keras layer

**References**

- Sparse deep belief net model for visual area V2
- Andrew Ng, Sparse Autoencoder. CS294A Lecture Notes (2011)

---

**to_keras.ruta_weight_decay**

*Obtain a Keras weight decay*

**Description**

Builds the Keras regularizer corresponding to the weight decay

**Usage**

```r
## S3 method for class 'ruta_weight_decay'
con Hispanidad al lado de endeble
```

**Arguments**

- `x` : A "ruta_weight_decay" object
- `...` : Rest of parameters, ignored
train.ruta_autoencoder

Train a learner object with data

Description

This function compiles the neural network described by the learner object and trains it with the input data.

Usage

```r
# S3 method for class 'ruta_autoencoder'
train(learner, data, validation_data = NULL,
     metrics = NULL, epochs = 20,
     optimizer = keras::optimizer_rmsprop(), ...)
```

Arguments

- **learner**: A "ruta_autoencoder" object
- **data**: Training data: columns are attributes and rows are instances
- **validation_data**: Additional numeric data matrix which will not be used for training but the loss measure and any metrics will be computed against it
- **metrics**: Optional list of metrics which will evaluate the model but won’t be optimized. See `keras::compile`
- **epochs**: The number of times data will pass through the network
- **optimizer**: The optimizer to be used in order to train the model, can be any optimizer object defined by Keras (e.g. `keras::optimizer_adam()`)
- **...**: Additional parameters for `keras::fit`. Some useful parameters:
  - `batch_size` The number of examples to be grouped for each gradient update. Use a smaller batch size for more frequent weight updates or a larger one for faster optimization.
  - `shuffle` Whether to shuffle the training data before each epoch, defaults to TRUE

Value

Same autoencoder passed as parameter, with trained internal models

See Also

- `autoencoder`
Examples

# Minimal example

iris_model <- train(autoencoder(2), as.matrix(iris[, 1:4]))

# Simple example with MNIST

library(keras)

# Load and normalize MNIST

mnist <- dataset_mnist()
x_train <- array_reshape(
  mnist$train$x, c(dim(mnist$train$x)[1], 784)
)
x_train <- x_train / 255.0

x_test <- array_reshape(
  mnist$test$x, c(dim(mnist$test$x)[1], 784)
)
x_test <- x_test / 255.0

# Autoencoder with layers: 784-256-36-256-784

learner <- autoencoder(c(256, 36), "binary_crossentropy")

train(
  learner,
  x_train,
  epochs = 1,
  optimizer = "rmsprop",
  batch_size = 64,
  validation_data = x_test,
  metrics = list("binary_accuracy")
)

---

**variational_block**  
*Create a variational block of layers*

**Description**

This variational block consists in two dense layers which take as input the previous layer and a sampling layer. More specifically, these layers aim to represent the mean and the log variance of the learned distribution in a variational autoencoder.

**Usage**

`variational_block(units, epsilon_std = 1, seed = NULL)`
weight_decay

Arguments

units  Number of units
epsilon_std  Standard deviation for the normal distribution used for sampling
seed  A seed for the random number generator. Setting a seed is required if you want to save the model and be able to load it correctly

Value

A construct with class "ruta_layer"

See Also

autoencoder_variational

Other neural layers: conv, dense, dropout, input, layer_keras, output

Examples

variational_block(3)

weight_decay  Weight decay

Description

A wrapper that describes a weight decay regularization of the encoding layer

Usage

weight_decay(decay = 0.02)

Arguments

decay  Numeric value indicating the amount of decay

Value

A regularizer object containing the set parameters
Description

Access subnetworks of a network

Usage

```r
## S3 method for class 'ruta_network'
net[index]
```

Arguments

- `net` A "ruta_network" object
- `index` An integer vector of indices of layers to be extracted

Value

A "ruta_network" object containing the specified layers.

Examples

```r
(input() + dense(30))[2]
long <- input() + dense(1000) + dense(100) + dense(1000) + output()
short <- long[c(1, 3, 5)]
```
Index

+ .ruta_network, 3, 29
[ .ruta_network, 46

add_weight_decay, 4
apply_filter, 30–32
apply_filter
   (apply_filter .ruta_noise_zeros), 4
apply_filter .ruta_noise_zeros, 4
as_loss, 5
as_network, 6
autoencode, 6
autoencoder, 7, 7, 9–12, 28, 38, 43
autoencoder_contractive, 8, 8, 10–13, 21, 25
autoencoder_denoising, 5, 8, 9, 9, 11, 12, 21, 26
autoencoder_robust, 8–10, 10, 11, 12, 15, 22, 27
autoencoder_sparse, 8–11, 11, 12, 22, 27, 37
autoencoder_variational, 8–11, 12, 20, 23, 25, 45

c .ruta_network (+ .ruta_network), 3
compile, 43
contraction, 13, 15, 21, 25
conv, 13, 16, 20, 24, 33, 45
correntropy, 13, 14, 22, 25

decode, 15, 17, 35
dense, 14, 15, 16, 20, 24, 28, 33, 45
dropout, 14, 16, 16, 20, 24, 33, 45
evaluate_binary_accuracy
   (evaluate_mean_squared_error), 18
evaluate_binary_crossentropy
   (evaluate_mean_squared_error), 18
evaluate_kullback_leibler_divergence
   (evaluate_mean_squared_error), 18
evaluate_mean_absolute_error
   (evaluate_mean_squared_error), 18
evaluate_mean_squared_error, 18
evaluation_metric, 18, 19
fit, 43

generate
   (generate .ruta_autoencoder_variational), 19
generate .ruta_autoencoder_variational, 19

input, 14, 16, 20, 24, 33, 45
is_contractive, 21
is_denoising, 21
is_robust, 22
is_sparse, 11, 22, 27, 37
is_trained, 23
is_variational, 23

layer_conv_2d, 14
layer_keras, 14, 16, 20, 24, 33, 45
load_from (save_as), 35
loss_variational, 13, 15, 24

make_contractive, 21, 25
make_denoising, 21, 26
make_robust, 22, 26
make_sparse, 11, 22, 27, 37
metric_binary_accuracy, 19

new_autoencoder, 28
new_layer, 28
INDEX

new_network, 29
noise, 21
noise_cauchy, 9, 30, 31
noise_gaussian, 9, 30, 31, 32
noise_ones, 9, 30, 31
noise_saltpepper, 9, 30–32
noise_zeros, 9, 30–32
output, 14, 16, 20, 24, 27, 36
plotNruta_network, 33
predictNruta_autoencoder, 34
printNruta_autoencoder, 34
printNruta_autoencoder_variational, 34
printNruta_loss, 34
printNruta_loss_contraction, 34
printNruta_network, 34
reconstruct, 15, 17, 35
save_as, 35
sparsity, 11, 22, 27, 36
to_kerasNruta_autoencoder, 37
to_kerasNruta_autoencoder_variational, 37
to_kerasNruta_filter, 38
to_kerasNruta_layer_conv, 38
to_kerasNruta_layer_custom, 38
to_kerasNruta_layer_input, 39
to_kerasNruta_layer_layer, 39
to_kerasNruta_layer_layer, 39
variational, 39
weight_decay, 45
train, 23
trainNruta_autoencoder, 8, 43
trainNruta_network, 43
weight_decay, 45