Add weight decay to any autoencoder

Source: `R/autoencoder_weight_decay.R`

`add_weight_decay.Rd`

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

```
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 filters

Source: `R/filter.R`, `R/generics.R`

`apply_filter.Rd`

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

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

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

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

# S3 method for ruta_noise_gaussian
apply_filter(filter, data, ...)

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

apply_filter(filter, data, ...)
```

Arguments

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

See also

`autoencoder_denoising`

Coercion to ruta_loss

Source: `R/generics.R`, `R/loss.R`

`as_loss.Rd`

Generic function to coerce objects into loss objects.

```
as_loss(x)

# S3 method for character
as_loss(x)

# S3 method for ruta_loss
as_loss(x)
```

Arguments

x	Object to be converted into a loss

Value

A `"ruta_loss"` construct

Coercion to ruta_network

Source: `R/generics.R`, `R/layers.R`, `R/network.R`

`as_network.Rd`

Generic function to coerce objects into networks.

```
as_network(x)

# S3 method for ruta_layer
as_network(x)

# S3 method for ruta_network
as_network(x)

# S3 method for numeric
as_network(x)

# S3 method for 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))
```

Automatically compute an encoding of a data matrix

Source: `R/autoencoder.R`

`autoencode.Rd`

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

```
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 `"v`
activation	Activation type to be used in the encoding layer. Some available activations are `"tanh"`, `"sigmoi`
epochs	Number of times the data will traverse the autoencoder to update its weights

Value

Matrix containing the encodings

See also

`autoencoder`

Examples

```
inputs <- as.matrix(iris[, 1:4])# Train a basic autoencoder and generate a 2-variable encodi
encoded <- autoencode(inputs, 2)

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

Create an autoencoder learner

Source: `R/autoencoder.R`

`autoencoder.Rd`

Represents a generic autoencoder network.

```
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

```
# Basic autoencoder with a network of [input]-256-36-256-[input] and
# no nonlinearities
autoencoder(c(256, 36), loss = "binary_crossentropy")#> Autoencoder learner
#> ----------------------------------------
#> Type: basic
#>
#> Network structure:
#>  input
#>  dense(256 units) - linear
#>  dense(36 units) - linear
#>  dense(256 units) - linear
#>  dense - linear
#>
#> 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"
)
```

Create a contractive autoencoder

Source: `R/autoencoder_contractive.R`

`autoencoder_contractive.Rd`

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.

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

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`

Create a denoising autoencoder

Source: `R/autoencoder_denoising.R`

`autoencoder_denoising.Rd`

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

```
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"`) - `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) - `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`

Create a robust autoencoder

Source: `R/autoencoder_robust.R`

`autoencoder_robust.Rd`

A robust autoencoder uses a special objective function, correntropy, a localized similarity measure which makes it less sensitive to noise in data. Correntropy specifically measures the probability density that two events are equal, and is less affected by outliers than the mean squared error.

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

Arguments

network	Layer construct of class `"ruta_network"`
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

See also

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

Sparse autoencoder

Source: `R/autoencoder_sparse.R`

`autoencoder_sparse.Rd`

Creates a representation of a sparse autoencoder.

```
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
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`

Build a variational autoencoder

Source: `R/autoencoder_variational.R`

`autoencoder_variational.Rd`

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.

```
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

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

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

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

Contractive loss

Source: `R/autoencoder_contractive.R`

`contraction.Rd`

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

```
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_cr`
weight	Weight assigned to the contractive loss

Value

A loss object which can be converted into a Keras loss

See also

`autoencoder_contractive`

Other loss functions: `correntropy`, `loss_variational`

Correntropy loss

Source: `R/autoencoder_robust.R`

`correntropy.Rd`

A wrapper for the correntropy loss function

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

Arguments

sigma	Sigma parameter in the kernel

Value

A `"ruta_loss"` object

See also

`autoencoder_robust`

Other loss functions: `contraction`, `loss_variational`

Retrieve decoding of encoded data

Source: `R/autoencoder.R`

`decode.Rd`

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

```
decode(learner, data)
```

Arguments

learner	Trained autoencoder model
data	data.frame to be decoded

Value

Matrix containing the decodifications

See also

`encode`, `reconstruct`

Create a fully-connected neural layer

Source: `R/layers.R`

`dense.Rd`

Wrapper for a dense/fully-connected layer.

```
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: dropout, input, layer_keras, output, variational_block

Examples

```
dense(30, "tanh")#> Network structure:
#>  dense(30 units) - tanh
```

Dropout layer

Source: R/layers.R

dropout.Rd

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

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

Arguments

rate	The fraction of affected units

Value

A construct of class "ruta_network"

See also

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

Retrieve encoding of data

Source: R/autoencoder.R

encode.Rd

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

```
encode(learner, data)
```

Arguments

learner	Trained autoencoder model
data	data.frame to be encoded

Value

Matrix containing the encodings

See also

decode, reconstruct

Get the index of the encoding

Source: R/network.R

encoding_index.Rd

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

```
encoding_index(net)
```

Arguments

net	A network of class "ruta_network"

Value

Index of the middle layer

Evaluation metrics

Source: `R/evaluate.R`

`evaluate.Rd`

Performance evaluation metrics for autoencoders

```
evaluate_mean_squared_error(learner, data)

evaluate_mean_absolute_error(learner, data)

evaluate_binary_crossentropy(learner, data)

evaluate_binary_accuracy(learner, data)

evaluate_kullback_leibler_divergence(learner, data)
```

Arguments

learner	A trained learner object
data	Test data for evaluation

Value

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

See also

`evaluation_metric`

Examples

```
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)#> $loss
#> [1] 23.26905
#>
```

```
#> $mean_squared_error
#> [1] 23.26905
#>
```

Custom evaluation metrics

Source: `R/evaluate.R`

`evaluation_metric.Rd`

Create a different evaluation metric from a valid Keras metric

```
evaluation_metric(evaluate_f)
```

Arguments

evaluate_f	Must be either a metric function defined by Keras (e.g. `keras::metric_binary_crossentropy`)

Value

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

See also

`evaluate`

Generate samples from a generative model

Source: `R/autoencoder_variational.R`, `R/generics.R`

`generate.Rd`

Generate samples from a generative model

```
# S3 method for 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`

Create an input layer

Source: `R/layers.R`

`input.Rd`

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.

```
input()
```

Value

A construct with class `"ruta_network"`

See also

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

Detect whether an autoencoder is contractive

Source: `R/autoencoder_contractive.R`

`is_contractive.Rd`

Detect whether an autoencoder is contractive

```
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`

Detect whether an autoencoder is denoising

Source: `R/autoencoder_denoising.R`

`is_denoising.Rd`

Detect whether an autoencoder is denoising

```
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`

Detect whether an autoencoder is robust

Source: `R/autoencoder_robust.R`

`is_robust.Rd`

Detect whether an autoencoder is robust

`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`

Detect whether an autoencoder is sparse

Source: `R/autoencoder_sparse.R`

`is_sparse.Rd`

Detect whether an autoencoder is sparse

`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`

Detect trained models

Source: `R/autoencoder.R`

`is_trained.Rd`

Inspects a learner and figures out whether it has been trained

```
is_trained(learner)
```

Arguments

learner	Learner object

Value

A boolean

See also

`train`

Detect whether an autoencoder is variational

Source: `R/autoencoder_variational.R`

`is_variational.Rd`

Detect whether an autoencoder is variational

```
is_variational(learner)
```

Arguments

learner	A `"ruta_autoencoder"` object

Value

Logical value indicating if a variational loss was found

See also

`autoencoder_variational`

Add layers to a network/Join networks

Source: `R/network.R`

`join-networks.Rd`

Add layers to a network/Join networks

```
# S3 method for ruta_network
+(e1, e2)

# S3 method for ruta_network
c(...)
```

Arguments

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

Value

Network combination

Examples

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

Custom layer from Keras

Source: `R/layers.R`

`layer_keras.Rd`

Gets any layer available in Keras with the specified parameters

`layer_keras(name, ...)`

Arguments

name	The name of the layer, e.g. `"activity_regularization"` for a `keras::layer_activity_regulariza`
...	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: `dense`, `dropout`, `input`, `output`, `variational_block`

Variational loss

Source: `R/autoencoder_variational.R`

`loss_variational.Rd`

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.

```
loss_variational(reconstruction_loss)
```

Arguments

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

Value

A `"ruta_loss"` object

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`

Add contractive behavior to any autoencoder

Source: `R/autoencoder_contractive.R`

`make_contractive.Rd`

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

```
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`

Add denoising behavior to any autoencoder

Source: `R/autoencoder_denoising.R`

`make_denoising.Rd`

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

```
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
...	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`

Add robust behavior to any autoencoder

Source: `R/autoencoder_robust.R`

`make_robust.Rd`

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

```
make_robust(learner, sigma = 0.2)
```

Arguments

learner	The `"ruta_autoencoder"` object
sigma	Sigma parameter in the kernel used for correntropy

Value

An autoencoder object which contains the correntropy loss

See also

`autoencoder_robust`

Add sparsity regularization to an autoencoder

Source: `R/autoencoder_sparse.R`

`make_sparse.Rd`

Add sparsity regularization to an autoencoder

```
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
weight	The weight of the sparsity regularization

Value

The same autoencoder with the sparsity regularization applied

See also

`sparsity`, `autoencoder_sparse`, `is_sparse`

Create an autoencoder learner

Source: `R/autoencoder.R`

`new_autoencoder.Rd`

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

```
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`, `tra`

Value

A construct of class `"ruta_autoencoder"`

Layer wrapper constructor

Source: `R/layers.R`

`new_layer.Rd`

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

```
new_layer(cl, ...)
```

Arguments

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

Value

A construct with class `"ruta_layer"`

Examples

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

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

Sequential network constructor

Source: `R/network.R`

`new_network.Rd`

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`.

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

Arguments

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

Value

A construct with class `"ruta_network"`

Examples

```
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 generator

Source: `R/filter.R`

`noise.Rd`

Delegates on noise classes to generate noise of some type

```
noise(type, ...)
```

Arguments

type	Type of noise, as a character string
...	Parameters for each noise class

Additive Cauchy noise

Source: `R/filter.R`

`noise_cauchy.Rd`

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

```
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`

Additive Gaussian noise

Source: `R/filter.R`

`noise_gaussian.Rd`

A data filter which adds Gaussian noise to instances

```
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`

Filter to add ones noise

Source: `R/filter.R`

`noise_ones.Rd`

A data filter which replaces some values with ones

```
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`

Filter to add salt-and-pepper noise

Source: `R/filter.R`

`noise_saltpepper.Rd`

A data filter which replaces some values with zeros or ones

```
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`

Filter to add zero noise

Source: `R/filter.R`

`noise_zeros.Rd`

A data filter which replaces some values with zeros

```
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

Source: `R/layers.R`

`output.Rd`

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.

```
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: `dense`, `dropout`, `input`, `layer_keras`, `variational_block`

Draw a neural network

Source: `R/network_plot.R`

`plot.ruta_network.Rd`

Draw a neural network

```
# S3 method for 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

Examples

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

Inspect Ruta objects

Source: `R/autoencoder.R`, `R/loss.R`, `R/network.R`

`print-methods.Rd`

Inspect Ruta objects

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

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

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

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

Arguments

x	An object
...	Unused

Value

Invisibly returns the same object passed as parameter

Examples

```
print(autoencoder(c(256, 10), loss = correntropy()))#> Autoencoder learner
#> ---------------------------------------
#> Type: robust
#>
#> Network structure:
#>  input
#>  dense(256 units) - linear
#>  dense(10 units) - linear
#>  dense(256 units) - linear
#>  dense - linear
#>
#> Loss: correntropy
#> ---------------------------------------
```

Retrieve reconstructions for input data

Source: `R/autoencoder.R`

`reconstruct.Rd`

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

```
reconstruct(learner, data)

# S3 method for ruta_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`

Sparsity regularization

Source: `R/autoencoder_sparse.R`

`sparsity.Rd`

Sparsity regularization

```
sparsity(high_probability, weight)
```

Arguments

high_probability	Expected probability of the high value of the encoding layer. Set this to a value near zero

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_sparse`

Access subnetworks of a network

Source: `R/network.R`

`sub-.ruta_network.Rd`

Access subnetworks of a network

```
# S3 method for 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

```
(input() + dense(30))[2]#> Network structure:
#>  dense(30 units) - linearlong <- input() + dense(1000) + dense(100) + dense(1000) + outpu
```

```
short <- long[c(1, 3, 5)]
```

Convert a Ruta object onto Keras objects and functions

Source: `R/generics.R`

`to_keras.Rd`

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

```
to_keras(x, ...)
```

Arguments

x	Object to be converted
...	Remaining parameters depending on the method

Extract Keras models from an autoencoder wrapper

Source: `R/autoencoder.R`, `R/autoencoder_variational.R`

`to_keras.ruta_autoencoder.Rd`

Extract Keras models from an autoencoder wrapper

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

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

Arguments

learner	Object of class `"ruta_autoencoder"`. Needs to have a member `input_shape` indicating the i
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`

Convert Ruta layers onto Keras layers

Source: `R/layers.R`

`to_keras.ruta_layer_input.Rd`

Convert Ruta layers onto Keras layers

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

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

# S3 method for 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

Obtain a Keras block of layers for the variational autoencoder

Source: `R/autoencoder_variational.R`

`to_keras.ruta_layer_variational.Rd`

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

```
# S3 method for 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

Obtain a Keras loss

Source: `R/autoencoder_contractive.R`, `R/autoencoder_robust.R`, `R/autoencoder_variational.R`, and 2 more

`to_keras.ruta_loss_named.Rd`

Builds the Keras loss function corresponding to a name

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

```
# S3 method for ruta_loss_correntropy
to_keras(x, ...)

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

# S3 method for 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

Build a Keras network

Source: `R/network.R`

`to_keras.ruta_network.Rd`

Build a Keras network

```
# S3 method for 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

Translate sparsity regularization to Keras regularizer

Source: `R/autoencoder_sparse.R`

`to_keras.ruta_sparsity.Rd`

Translate sparsity regularization to Keras regularizer

```
# S3 method for 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)

Obtain a Keras weight decay

Source: `R/autoencoder_weight_decay.R`

`to_keras.ruta_weight_decay.Rd`

Builds the Keras regularizer corresponding to the weight decay

```
# S3 method for ruta_weight_decay
to_keras(x, ...)
```

Arguments

x	A `"ruta_weight_decay"` object
...	Rest of parameters, ignored

Train a learner object with data

Source: `R/autoencoder.R`, `R/generics.R`

`train.ruta_autoencoder.Rd`

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

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

train(learner, ...)
```

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")
)
```

Create a variational block of layers

Source: `R/autoencoder_variational.R`

`variational_block.Rd`

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.

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

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**

Value

A construct with class `"ruta_layer"`

See also

`autoencoder_variational`

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

Examples

```
variational_block(3)#> Network structure:
#>  variational(3 units)
```

Weight decay

Source: `R/autoencoder_weight_decay.R`

`weight_decay.Rd`

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

```
weight_decay(decay = 0.02)
```

Arguments

decay	Numeric value indicating the amount of decay

Value

A regularizer object containing the set parameters

Save and load Ruta models

Source: `R/save.R`

`save_as.Rd`

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

```
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`, pa
dir	Directory where to save the file. Use `"."` to save in the current working directory or `tempdir(`
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)#> Loading weights from /tmp/RtmphVC8m6/ruta/w
```