Package 'RGAN'

January 20, 2025

Title Generative Adversarial Nets (GAN) in R

Version 0.1.1

Description An easy way to get started with Generative Adversarial Nets (GAN) in R. The GAN algorithm was initially
described by Goodfellow et al. 2014 <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>. A GAN can be used to learn the joint distribution of complex data by
comparison. A GAN consists of two neural networks a Generator and a Discriminator, where the two
neural networks play an adversarial minimax game.
Built-in GAN models make the training of GANs in R possible in one line and make it easy to
experiment with different design choices (e.g. different network architectures, value functions, optimizers).
The built-in GAN models work with tabular data (e.g. to produce synthetic data) and image data. Methods to post-
process the output of GAN models to enhance the quality of samples are available.

License MIT + file LICENSE

URL https://github.com/mneunhoe/RGAN

BugReports https://github.com/mneunhoe/RGAN/issues

Encoding UTF-8

RoxygenNote 7.1.2

Imports cli, torch, viridis

NeedsCompilation no

Author Marcel Neunhoeffer [aut, cre] (<https://orcid.org/0000-0002-9137-5785>)

Maintainer Marcel Neunhoeffer <marcel.neunhoeffer@gmail.com>

Repository CRAN

Date/Publication 2022-03-29 18:30:06 UTC

Contents

`data_transformer` *Data Transformer*

Description

Provides a class to transform data for RGAN. Method `$new()` initializes a new transformer, method `$fit(data)` learns the parameters for the transformation from data (e.g. means and sds). Methods `$transform()` and `$inverse_transform()` can be used to transform and back transform a data set based on the learned parameters. Currently, DataTransformer supports z-transformation (a.k.a. normalization) for numerical features/variables and one hot encoding for categorical features/variables. In your call to fit you just need to indicate which columns contain discrete features.

Value

A class to transform (normalize or one hot encode) tabular data for RGAN

Methods

Public methods:

- `data_transformer$new()`
- `data_transformer$fit_continuous()`
- `data_transformer$fit_discrete()`
- `data_transformer$fit()`
- `data_transformer$transform_continuous()`
- `data_transformer$transform_discrete()`

- `data_transformer$transform()`
- `data_transformer$inverse_transform_continuous()`
- `data_transformer$inverse_transform_discrete()`
- `data_transformer$inverse_transform()`
- `data_transformer$clone()`

Method `new()`**:** Create a new data_transformer object

Usage:

```
data_transformer$new()
```

Method `fit_continuous()`**:**

Usage:

```
data_transformer$fit_continuous(column = NULL, data = NULL)
```

Method `fit_discrete()`**:**

Usage:

```
data_transformer$fit_discrete(column = NULL, data = NULL)
```

Method `fit()`**:** Fit a data_transformer to data.

Usage:

```
data_transformer$fit(data, discrete_columns = NULL)
```

Arguments:

`data` The data set to transform

`discrete_columns` Column ids for columns with discrete/nominal values to be one hot encoded.

Examples:

```
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
```

Method `transform_continuous()`**:**

Usage:

```
data_transformer$transform_continuous(column_meta, data)
```

Method `transform_discrete()`**:**

Usage:

```
data_transformer$transform_discrete(column_meta, data)
```

Method `transform()`**:** Transform data using a fitted data_transformer. (From original format to transformed format.)

Usage:

```
data_transformer$transform(data)
```

Arguments:

`data` The data set to transform

Examples:

```
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
```

Method `inverse_transform_continuous()`**:**

Usage:

```
data_transformer$inverse_transform_continuous(meta, data)
```

Method `inverse_transform_discrete()`**:**

Usage:

```
data_transformer$inverse_transform_discrete(meta, data)
```

Method `inverse_transform()`**:** Inverse Transform data using a fitted data_transformer. (From transformed format to original format.)

Usage:

```
data_transformer$inverse_transform(data)
```

Arguments:

`data` The data set to transform

Examples:

```
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
reconstructed_data <- transformer$inverse_transform(transformed_data)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
data_transformer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# Before running the first time the torch backend needs to be installed
torch::install_torch()
# Load data
data <- sample_toydata()
# Build new transformer
transformer <- data_transformer$new()
# Fit transformer to data
transformer$fit(data)
# Transform data and store as new object
transformed_data <-  transformer$transform(data)
# Train the default GAN
```

```
trained_gan <- gan_trainer(transformed_data)
# Sample synthetic data from the trained GAN
synthetic_data <- sample_synthetic_data(trained_gan, transformer)
# Plot the results
GAN_update_plot(data = data,
synth_data = synthetic_data,
main = "Real and Synthetic Data after Training")

## End(Not run)

## ------------------------------------------------
## Method `data_transformer$fit`
## ------------------------------------------------

data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)

## ------------------------------------------------
## Method `data_transformer$transform`
## ------------------------------------------------

data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)

## ------------------------------------------------
## Method `data_transformer$inverse_transform`
## ------------------------------------------------

data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
reconstructed_data <- transformer$inverse_transform(transformed_data)
```

DCGAN_Discriminator *DCGAN Discriminator*

Description

Provides a torch::nn_module with a simple deep convolutional neural net architecture, for use as the default architecture for image data in RGAN. Architecture inspired by: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

Usage

```
DCGAN_Discriminator(
  number_channels = 3,
  ndf = 64,
  dropout_rate = 0.5,
  sigmoid = FALSE
)
```

Arguments

`number_channels` The number of channels in the image (RGB is 3 channels)

`ndf` The number of feature maps in discriminator

`dropout_rate` The dropout rate for each hidden layer

`sigmoid` Switch between a sigmoid and linear output layer (the sigmoid is needed for the original GAN value function)

Value

A torch::nn_module for the DCGAN Discriminator

`DCGAN_Generator` *DCGAN Generator*

Description

Provides a torch::nn_module with a simple deep convolutional neural net architecture, for use as the default architecture for image data in RGAN. Architecture inspired by: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

Usage

```
DCGAN_Generator(
  noise_dim = 100,
  number_channels = 3,
  ngf = 64,
  dropout_rate = 0.5
)
```

Arguments

`noise_dim` The length of the noise vector per example

`number_channels` The number of channels in the image (RGB is 3 channels)

`ngf` The number of feature maps in generator

`dropout_rate` The dropout rate for each hidden layer

Value

A torch::nn_module for the DCGAN Generator

`Discriminator` *Discriminator*

Description

Provides a torch::nn_module with a simple fully connected neural net, for use as the default architecture for tabular data in RGAN.

Usage

```
Discriminator(
  data_dim,
  hidden_units = list(128, 128),
  dropout_rate = 0.5,
  sigmoid = FALSE
)
```

Arguments

`data_dim`	The number of columns in the data set
`hidden_units`	A list of the number of neurons per layer, the length of the list determines the number of hidden layers
`dropout_rate`	The dropout rate for each hidden layer
`sigmoid`	Switch between a sigmoid and linear output layer (the sigmoid is needed for the original GAN value function)

Value

A torch::nn_module for the Discriminator

`expert_sample_synthetic_data` *Sample Synthetic Data with explicit noise input*

Description

Provides a function that makes it easy to sample synthetic data from a Generator

Usage

```
expert_sample_synthetic_data(g_net, z, device, eval_dropout = FALSE)
```

Arguments

`g_net`	A torch::nn_module with a Generator
`z`	A noise vector
`device`	The device on which synthetic data should be sampled (cpu or cuda)
`eval_dropout`	Should dropout be applied during inference

Value

Synthetic data

`gan_trainer` *gan_trainer*

Description

Provides a function to quickly train a GAN model.

Usage

```
gan_trainer(
  data,
  noise_dim = 2,
  noise_distribution = "normal",
  value_function = "original",
  data_type = "tabular",
  generator = NULL,
  generator_optimizer = NULL,
  discriminator = NULL,
  discriminator_optimizer = NULL,
  base_lr = 1e-04,
  ttur_factor = 4,
  weight_clipper = NULL,
  batch_size = 50,
  epochs = 150,
  plot_progress = FALSE,
  plot_interval = "epoch",
  eval_dropout = FALSE,
  synthetic_examples = 500,
  plot_dimensions = c(1, 2),
  device = "cpu"
)
```

Arguments

`data` Input a data set. Needs to be a matrix, array, torch::torch_tensor or torch::dataset.

`noise_dim` The dimensions of the GAN noise vector z. Defaults to 2.

`noise_distribution`
The noise distribution. Expects a function that samples from a distribution and returns a torch_tensor. For convenience "normal" and "uniform" will automatically set a function. Defaults to "normal".

`value_function` The value function for GAN training. Expects a function that takes discriminator scores of real and fake data as input and returns a list with the discriminator loss and generator loss. For reference see: . For convenience three loss functions "original", "wasserstein" and "f-wgan" are already implemented. Defaults to "original".

`data_type` "tabular" or "image", controls the data type, defaults to "tabular".

`generator` The generator network. Expects a neural network provided as torch::nn_module. Default is NULL which will create a simple fully connected neural network.

`generator_optimizer`
The optimizer for the generator network. Expects a torch::optim_xxx function, e.g. torch::optim_adam(). Default is NULL which will setup `torch::optim_adam(g_net$parameters, lr = base_lr)`.

`discriminator` The discriminator network. Expects a neural network provided as torch::nn_module. Default is NULL which will create a simple fully connected neural network.

`discriminator_optimizer`
The optimizer for the generator network. Expects a torch::optim_xxx function, e.g. torch::optim_adam(). Default is NULL which will setup `torch::optim_adam(g_net$parameters, lr = base_lr * ttur_factor)`.

`base_lr` The base learning rate for the optimizers. Default is 0.0001. Only used if no optimizer is explicitly passed to the trainer.

`ttur_factor` A multiplier for the learning rate of the discriminator, to implement the two time scale update rule.

`weight_clipper` The wasserstein GAN puts some constraints on the weights of the discriminator, therefore weights are clipped during training.

`batch_size` The number of training samples selected into the mini batch for training. Defaults to 50.

`epochs` The number of training epochs. Defaults to 150.

`plot_progress` Monitor training progress with plots. Defaults to FALSE.

`plot_interval` Number of training steps between plots. Input number of steps or "epoch". Defaults to "epoch".

`eval_dropout` Should dropout be applied during the sampling of synthetic data? Defaults to FALSE.

`synthetic_examples`
Number of synthetic examples that should be generated. Defaults to 500. For image data e.g. 16 would be more reasonable.

`plot_dimensions`
: If you monitor training progress with a plot which dimensions of the data do you want to look at? Defaults to c(1, 2), i.e. the first two columns of the tabular data.

`device`
: Input on which device (e.g. "cpu" or "cuda") training should be done. Defaults to "cpu".

Value

gan_trainer trains the neural networks and returns an object of class trained_RGAN that contains the last generator, discriminator and the respective optimizers, as well as the settings.

Examples

```
## Not run:
# Before running the first time the torch backend needs to be installed
torch::install_torch()
# Load data
data <- sample_toydata()
# Build new transformer
transformer <- data_transformer$new()
# Fit transformer to data
transformer$fit(data)
# Transform data and store as new object
transformed_data <-  transformer$transform(data)
# Train the default GAN
trained_gan <- gan_trainer(transformed_data)
# Sample synthetic data from the trained GAN
synthetic_data <- sample_synthetic_data(trained_gan, transformer)
# Plot the results
GAN_update_plot(data = data,
synth_data = synthetic_data,
main = "Real and Synthetic Data after Training")

## End(Not run)
```

`GAN_update_plot` *GAN_update_plot*

Description

Provides a function to send the output of a DataTransformer to a torch tensor, so that it can be accessed during GAN training.

Usage

```
GAN_update_plot(data, dimensions = c(1, 2), synth_data, epoch, main = NULL)
```

Arguments

`data`	Real data to be plotted
`dimensions`	Which columns of the data should be plotted
`synth_data`	The synthetic data to be plotted
`epoch`	The epoch during training for the plot title
`main`	An optional plot title

Value

A function

Examples

```
## Not run:
# Before running the first time the torch backend needs to be installed
torch::install_torch()
# Load data
data <- sample_toydata()
# Build new transformer
transformer <- data_transformer$new()
# Fit transformer to data
transformer$fit(data)
# Transform data and store as new object
transformed_data <-  transformer$transform(data)
# Train the default GAN
trained_gan <- gan_trainer(transformed_data)
# Sample synthetic data from the trained GAN
synthetic_data <- sample_synthetic_data(trained_gan, transformer)
# Plot the results
GAN_update_plot(data = data,
synth_data = synthetic_data,
main = "Real and Synthetic Data after Training")

## End(Not run)
```

`GAN_update_plot_image` *GAN_update_plot_image*

Description

Provides a function to send the output of a DataTransformer to a torch tensor, so that it can be accessed during GAN training.

Usage

```
GAN_update_plot_image(mfrow = c(4, 4), synth_data)
```

Arguments

mfrow	The dimensions of the grid of images to be plotted
synth_data	The synthetic data (images) to be plotted

Value

A function

gan_update_step *gan_update_step*

Description

Provides a function to send the output of a DataTransformer to a torch tensor, so that it can be accessed during GAN training.

Usage

```
gan_update_step(
  data,
  batch_size,
  noise_dim,
  sample_noise,
  device = "cpu",
  g_net,
  d_net,
  g_optim,
  d_optim,
  value_function,
  weight_clipper
)
```

Arguments

data	Input a data set. Needs to be a matrix, array, torch::torch_tensor or torch::dataset.
batch_size	The number of training samples selected into the mini batch for training. Defaults to 50.
noise_dim	The dimensions of the GAN noise vector z. Defaults to 2.
sample_noise	A function to sample noise to a torch::tensor
device	Input on which device (e.g. "cpu" or "cuda") training should be done. Defaults to "cpu".
g_net	The generator network. Expects a neural network provided as torch::nn_module. Default is NULL which will create a simple fully connected neural network.
d_net	The discriminator network. Expects a neural network provided as torch::nn_module. Default is NULL which will create a simple fully connected neural network.

`g_optim` The optimizer for the generator network. Expects a torch::optim_xxx function, e.g. torch::optim_adam(). Default is NULL which will setup `torch::optim_adam(g_net$parameters, lr = base_lr)`.

`d_optim` The optimizer for the generator network. Expects a torch::optim_xxx function, e.g. torch::optim_adam(). Default is NULL which will setup `torch::optim_adam(g_net$parameters, lr = base_lr * ttur_factor)`.

`value_function` The value function for GAN training. Expects a function that takes discriminator scores of real and fake data as input and returns a list with the discriminator loss and generator loss. For reference see: . For convenience three loss functions "original", "wasserstein" and "f-wgan" are already implemented. Defaults to "original".

`weight_clipper` The wasserstein GAN puts some constraints on the weights of the discriminator, therefore weights are clipped during training.

Value

A function

`GAN_value_fct` *GAN Value Function*

Description

Implements the original GAN value function as a function to be called in gan_trainer. The function can serve as a template to implement new value functions in RGAN.

Usage

```
GAN_value_fct(real_scores, fake_scores)
```

Arguments

`real_scores` The discriminator scores on real examples ($D(x)$)

`fake_scores` The discriminator scores on fake examples ($D(G(z))$)

Value

The function returns a named list with the entries d_loss and g_loss

`Generator` *Generator*

Description

Provides a torch::nn_module with a simple fully connected neural net, for use as the default architecture for tabular data in RGAN.

Usage

```
Generator(
  noise_dim,
  data_dim,
  hidden_units = list(128, 128),
  dropout_rate = 0.5
)
```

Arguments

`noise_dim` The length of the noise vector per example

`data_dim` The number of columns in the data set

`hidden_units` A list of the number of neurons per layer, the length of the list determines the number of hidden layers

`dropout_rate` The dropout rate for each hidden layer

Value

A torch::nn_module for the Generator

`KLWGAN_value_fct` *KLWGAN Value Function*

Description

Provides a function to send the output of a DataTransformer to a torch tensor, so that it can be accessed during GAN training.

Usage

```
KLWGAN_value_fct(real_scores, fake_scores)
```

Arguments

`real_scores` The discriminator scores on real examples ($D(x)$)

`fake_scores` The discriminator scores on fake examples ($D(G(z))$)

Value

The function returns a named list with the entries d_loss and g_loss

---

`kl_fake` *KL WGAN loss on fake examples*

Description

Utility function for the kl WGAN loss for fake examples as described in https://arxiv.org/abs/1910.09779 and implemented in python in https://github.com/ermongroup/f-wgan.

Usage

```
kl_fake(dis_fake)
```

Arguments

`dis_fake`	Discriminator scores on fake examples ($D(G(z))$)

Value

The loss

`kl_gen` *KL WGAN loss for Generator training*

Description

Utility function for the kl WGAN loss for Generator training as described in https://arxiv.org/abs/1910.09779 and implemented in python in https://github.com/ermongroup/f-wgan.

Usage

```
kl_gen(dis_fake)
```

Arguments

`dis_fake`	Discriminator scores on fake examples ($D(G(z))$)

Value

The loss

kl_real *KL WGAN loss on real examples*

Description

Utility function for the kl WGAN loss for real examples as described in https://arxiv.org/abs/1910.09779 and implemented in python in https://github.com/ermongroup/f-wgan.

Usage

```
kl_real(dis_real)
```

Arguments

dis_real	Discriminator scores on real examples ($D(x)$)

Value

The loss

sample_synthetic_data *Sample Synthetic Data from a trained RGAN*

Description

Provides a function that makes it easy to sample synthetic data from a Generator

Usage

```
sample_synthetic_data(trained_gan, transformer = NULL)
```

Arguments

trained_gan	A trained RGAN object of class "trained_RGAN"
transformer	The transformer object used to pre-process the data

Value

Function to sample from a

Examples

```
## Not run:
# Before running the first time the torch backend needs to be installed
torch::install_torch()
# Load data
data <- sample_toydata()
# Build new transformer
transformer <- data_transformer$new()
# Fit transformer to data
transformer$fit(data)
# Transform data and store as new object
transformed_data <-  transformer$transform(data)
# Train the default GAN
trained_gan <- gan_trainer(transformed_data)
# Sample synthetic data from the trained GAN
synthetic_data <- sample_synthetic_data(trained_gan, transformer)
# Plot the results
GAN_update_plot(data = data,
synth_data = synthetic_data,
main = "Real and Synthetic Data after Training")

## End(Not run)
```

`sample_toydata` *Sample Toydata*

Description

Sample Toydata to reproduce the examples in the paper.

Usage

```
sample_toydata(n = 1000, sd = 0.3, seed = 20211111)
```

Arguments

`n`	Number of observations to generate
`sd`	Standard deviation of the normal distribution to generate y
`seed`	A seed for the pseudo random number generator

Value

A matrix with two columns x and y

Examples

```
## Not run:
# Before running the first time the torch backend needs to be installed
torch::install_torch()
# Load data
data <- sample_toydata()
# Build new transformer
transformer <- data_transformer$new()
# Fit transformer to data
transformer$fit(data)
# Transform data and store as new object
transformed_data <-  transformer$transform(data)
# Train the default GAN
trained_gan <- gan_trainer(transformed_data)
# Sample synthetic data from the trained GAN
synthetic_data <- sample_synthetic_data(trained_gan, transformer)
# Plot the results
GAN_update_plot(data = data,
synth_data = synthetic_data,
main = ”Real and Synthetic Data after Training”)

## End(Not run)
```

---

torch_rand_ab *Uniform Random numbers between values a and b*

Description

Provides a function to sample torch tensors from an arbitrary uniform distribution.

Usage

```
torch_rand_ab(shape, a = -1, b = 1, ...)
```

Arguments

shape	Vector of dimensions of resulting tensor
a	Lower bound of uniform distribution to sample from
b	Upper bound of uniform distribution to sample from
...	Potential additional arguments

Value

A sample from the specified uniform distribution in a tensor with the specified shape

`WGAN_value_fct` *WGAN Value Function*

Description

Implements the Wasserstein GAN (WGAN) value function as a function to be called in gan_trainer. Note that for this to work properly you also need to implement a weight clipper (or other procedure) to constrain the Discriminator weights.

Usage

```
WGAN_value_fct(real_scores, fake_scores)
```

Arguments

`real_scores`	The discriminator scores on real examples ($D(x)$)
`fake_scores`	The discriminator scores on fake examples ($D(G(z))$)

Value

The function returns a named list with the entries d_loss and g_loss

`WGAN_weight_clipper` *WGAN Weight Clipper*

Description

A function that clips the weights of a Discriminator (for WGAN training).

Usage

```
WGAN_weight_clipper(d_net, clip_values = c(-0.01, 0.01))
```

Arguments

`d_net`	A torch::nn_module (typically a discriminator/critic) for which the weights should be clipped
`clip_values`	A vector with the lower and upper bound for weight values. Any value outside this range will be set to the closer value.

Value

The function modifies the torch::nn_module weights in place

Index