

Package: RGAN (via r-universe)

October 15, 2026

Title Generative Adversarial Nets (GAN) in R

Version 0.2.2

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

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports cli, R6, torch, viridis

Suggests opendp, rmarkdown, knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Repository <https://mneunhoe.r-universe.dev>

Date/Publication 2026-09-15 13:27:53 UTC

RemoteUrl <https://github.com/mneunhoe/rgan>

RemoteRef HEAD

RemoteSha fe64f21fd23b52de03b718277de8136adcaf2616

Contents

apply_post_gan_boosting	3
compute_discriminator_scores	4
data_transformer	5
DCGAN_Discriminator	14
DCGAN_Generator	14
Discriminator	15
dp_gan_trainer	16
expert_sample_synthetic_data	19
gan_trainer	20
GAN_update_plot	24
GAN_update_plot_image	25
gan_update_step	25
GAN_value_fct	26
Generator	27
gradient_penalty	28
gumbel_softmax	28
kl_fake	29
kl_gen	29
kl_real	30
KLWGAN_value_fct	30
load_gan	31
plot_losses	32
post_gan_boosting	32
print.trained_RGAN	34
sample_synthetic_data	35
sample_toydata	36
save_gan	37
TabularGenerator	38
torch_rand_ab	40
WGAN_value_fct	41
WGAN_weight_clipper	41
Index	42

 apply_post_gan_boosting

Apply Post-GAN Boosting to a Trained GAN

Description

High-level wrapper that orchestrates the full post-GAN boosting workflow:

1. Generates candidate samples from checkpointed generators
2. Computes discriminator scores using checkpointed discriminators
3. Applies the post-GAN boosting algorithm to select high-quality samples

Usage

```

apply_post_gan_boosting(
    trained_gan,
    real_data,
    transformer = NULL,
    n_candidates = 1000,
    steps = 400,
    dp = FALSE,
    MW_epsilon = 0.1,
    weighted_average = FALSE,
    averaging_window = NULL,
    score_transform = c("none", "sigmoid"),
    device = NULL,
    seed = NULL
)

```

Arguments

trained_gan	A trained GAN object of class "trained_RGAN" with checkpoints
real_data	The original training data (matrix)
transformer	Optional data_transformer for inverse transformation
n_candidates	Number of candidate samples to generate per generator. Defaults to 1000.
steps	Number of boosting steps. Defaults to 400.
dp	Use differential privacy for discriminator selection. Defaults to FALSE. This privatises the <i>selection</i> of discriminators only: if trained_gan came from gan_trainer() rather than dp_gan_trainer(), the checkpoints already encode the training data and the boosted samples are not differentially private end to end. Total budget is the training epsilon plus MW_epsilon.
MW_epsilon	Privacy budget for multiplicative weights (only used if dp=TRUE). Defaults to 0.1.
weighted_average	Use weighted averaging for final distribution. Defaults to FALSE.

averaging_window Window size for averaging phi distributions. Defaults to NULL (use all steps).

score_transform Either "none" (default) or "sigmoid"; see compute_discriminator_scores(). Use "sigmoid" with dp = TRUE when the GAN was trained with a value function other than "original", whose discriminator has no sigmoid output layer.

device Device for computation. Defaults to trained_gan's device.

seed Optional seed for reproducibility.

Value

A list with:

- samples: Matrix of selected high-quality synthetic samples
- scores: Discriminator scores for selected samples
- n_unique: Number of unique samples selected

Examples

```
## Not run:
# Train a GAN with checkpoints
trained_gan <- gan_trainer(
  transformed_data,
  epochs = 100,
  checkpoint_epochs = 10 # Save every 10 epochs
)

# Apply post-GAN boosting
boosted <- apply_post_gan_boosting(
  trained_gan,
  real_data = transformed_data,
  n_candidates = 5000,
  steps = 200
)

# Use the boosted samples
high_quality_samples <- boosted$samples

## End(Not run)
```

compute_discriminator_scores

Compute Discriminator Scores from Checkpointed Discriminators

Description

Evaluates generated samples using multiple discriminator checkpoints to create the discriminator score matrix required for post-GAN boosting.

Usage

```
compute_discriminator_scores(
  trained_gan,
  generated_samples,
  real_data,
  batch_size = 1000,
  device = NULL,
  score_transform = c("none", "sigmoid")
)
```

Arguments

trained_gan	A trained GAN object of class "trained_RGAN" with checkpoints
generated_samples	A matrix of generated samples (N_samples x data_dim)
real_data	A matrix of real data for computing real scores
batch_size	Batch size for scoring (to manage memory). Defaults to 1000.
device	Device for computation ("cpu", "cuda", "mps"). Defaults to trained_gan's device.
score_transform	Either "none" (default) to return the discriminator outputs as they are, or "sigmoid" to squash them into $[0, 1]$. The squash is monotone, so it preserves the ranking of samples, and it is what makes the scores of an unbounded critic (WGAN, KLWGAN) usable with <code>dp = TRUE</code> .

Value

A list with:

- `d_score_fake`: Matrix of discriminator scores (N_discriminators x N_samples)
- `d_score_real`: Vector of mean discriminator scores on real data (length N_discriminators)
- `n_real_units`: Number of independently scored units averaged into each entry of `d_score_real`. Equals the number of real records unless `pac > 1`, in which case it is `n / pac`. Pass this as `real_N` to `post_gan_boosting()`.
- `epochs`: Vector of epoch numbers corresponding to each discriminator

data_transformer	<i>Data Transformer</i>
------------------	-------------------------

Description

An R6 class for preprocessing tabular data before GAN training. The transformer learns normalization parameters from data and provides reversible transformations to convert between original and GAN-ready formats.

Details

Overview:

GANs work best when input data is normalized to a consistent scale. The `data_transformer` class handles this preprocessing automatically:

1. **Fit:** Learn transformation parameters from your data
2. **Transform:** Convert data to normalized format for GAN training
3. **Inverse Transform:** Convert GAN output back to original scale

Normalization Methods:

Standard Normalization (default):

Applies z-score standardization to continuous columns:

$$z = \frac{x - \mu}{\sigma}$$

where μ is the column mean and σ is the standard deviation. This maps data to approximately zero mean and unit variance.

Best for: Data with roughly Gaussian distributions or when simplicity is preferred.

Mode-Specific Normalization (CTGAN-style):

For columns with multi-modal, skewed, or complex distributions, mode-specific normalization fits a Gaussian Mixture Model (GMM) and normalizes each value within its assigned mode. This approach is used by CTGAN (Xu et al., 2019).

How it works:

1. Fit a GMM with `n_modes` components using the EM algorithm
2. For each value, assign it to the most likely mode
3. Normalize the value within that mode's distribution
4. Output includes: one-hot encoded mode indicator + normalized value

Output dimensions: For a column with `k` modes, the transformed output has `k + 1` columns: `k` columns for the mode indicator (one-hot) and 1 column for the normalized value (clipped to [-1, 1]).

Best for: Columns with multiple peaks, heavy tails, or skewed distributions. Significantly improves GAN performance on real-world tabular data.

Categorical Columns:

Categorical (discrete) columns are one-hot encoded. Each category becomes a separate binary column. The inverse transform selects the category with the highest value (argmax).

Fields:

After fitting, the transformer stores:

meta List of metadata for each column (means, stds, levels, etc.)

output_info List describing the output structure for each column

output_dimensions Total number of columns in transformed data

mode_specific Whether mode-specific normalization was used

n_modes Number of GMM modes (if `mode_specific = TRUE`)

Integration with RGAN:

The transformer integrates seamlessly with RGAN's training and sampling functions:

```
# 1. Create and fit transformer
transformer <- data_transformer$new()
transformer$fit(data, discrete_columns = c("category_col"))

# 2. Transform data for training
transformed_data <- transformer$transform(data)

# 3. Train GAN
trained_gan <- gan_trainer(transformed_data)

# 4. Sample and inverse transform
synthetic_data <- sample_synthetic_data(trained_gan, transformer)
```

For mode-specific normalization with categorical columns, use TabularGenerator with Gumbel-Softmax for differentiable sampling (see gan_trainer with output_info).

Value

An R6 class object for transforming tabular data

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()`: Fit parameters for a continuous column (internal method)

Usage:

```
data_transformer$fit_continuous(
  column = NULL,
  data = NULL,
  mode_specific = FALSE,
  n_modes = 10
)
```

Arguments:

column Column name or index
 data Column data as a single-column matrix
 mode_specific Whether to use GMM-based normalization
 n_modes Number of GMM components

Method `fit_discrete()`: Fit parameters for a discrete/categorical column (internal method)

Usage:

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

Arguments:

column Column name or index
 data Column data as a single-column matrix

Method `fit()`: Fit the transformer to learn normalization parameters from data.

This method analyzes each column in the data and stores the parameters needed for transformation (means, standard deviations, category levels, etc.). Must be called before `transform()` or `inverse_transform()`.

Usage:

```
data_transformer$fit(
  data,
  discrete_columns = NULL,
  mode_specific = FALSE,
  n_modes = 10
)
```

Arguments:

data A data.frame, matrix, or array containing the training data. Column names are preserved and used for tracking transformations.

discrete_columns Character or integer vector specifying which columns contain categorical/discrete values. These columns will be one-hot encoded. Can be column names (character) or column indices (integer). Columns not listed here are treated as continuous. Defaults to NULL (all continuous).

mode_specific Logical. If TRUE, use mode-specific normalization (GMM) for continuous columns. This fits a Gaussian Mixture Model to each continuous column and normalizes values within their assigned mode. Recommended for columns with multi-modal or skewed distributions. Defaults to FALSE (standard z-score normalization).

n_modes Integer. Maximum number of Gaussian components for GMM fitting. The actual number may be lower if modes with weight < 0.01 are pruned. Only used when `mode_specific = TRUE`. Defaults to 10.

Returns: The transformer object (invisibly), allowing method chaining.

Examples:

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

```
# Mode-specific normalization
transformer$fit(data, mode_specific = TRUE, n_modes = 10)

# With categorical columns
transformer$fit(data, discrete_columns = c("category"))
```

Method transform_continuous(): Transform a continuous column (internal method)

Usage:

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

Arguments:

column_meta Metadata for this column from fit_continuous
data Vector of values to transform

Method transform_discrete(): Transform a discrete column to one-hot encoding (internal method)

Usage:

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

Arguments:

column_meta Metadata for this column from fit_discrete
data Vector of values to transform

Method transform(): Transform data from original format to normalized format for GAN training.

Applies the learned transformations to convert data into a format suitable for neural network training:

- Continuous columns: z-score normalization or mode-specific normalization
- Categorical columns: one-hot encoding

The transformer must be fitted before calling this method.

Usage:

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

Arguments:

data A data.frame, matrix, or array with the same columns as the data used for fitting. Column order and names must match.

Returns: A numeric matrix with transformed data. The number of columns depends on the transformation:

- Standard normalization: same number of columns as input
- Mode-specific: (n_modes + 1) columns per continuous column
- Categorical: one column per category level

Use transformer\$output_dimensions to check the total output columns.

Examples:

```
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
cat("Output dimensions:", dim(transformed_data))
```

Method `inverse_transform_continuous()`: Inverse transform a continuous column (internal method)

Usage:

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

Arguments:

meta Metadata for this column

data Transformed data to inverse transform

Method `inverse_transform_discrete()`: Inverse transform a discrete column from one-hot (internal method)

Usage:

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

Arguments:

meta Metadata for this column

data One-hot encoded data to inverse transform

Method `inverse_transform()`: Inverse transform data from normalized format back to original scale.

Reverses the transformations applied by `transform()`:

- Continuous columns: denormalized using stored means/stds
- Mode-specific: selects mode with highest probability, then denormalizes
- Categorical columns: selects category with highest value (argmax)

This is typically used to convert GAN-generated samples back to the original data format for analysis and use.

Usage:

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

Arguments:

data A numeric matrix in the transformed format. Must have the same number of columns as `transformer$output_dimensions`.

Returns: A data.frame with columns in the original format:

- Continuous columns as numeric
- Categorical columns as character (or numeric if original levels were numeric)

Examples:

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

```
# Round-trip transformation
transformed_data <- transformer$transform(data)
reconstructed_data <- transformer$inverse_transform(transformed_data)

# Use with GAN output
# synthetic_raw <- trained_gan$generator(noise)
# synthetic_data <- transformer$inverse_transform(as_array(synthetic_raw))
```

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.

References

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional GAN. *Advances in Neural Information Processing Systems*, 32.

Examples

```
## Not run:
# =====
# Example 1: Basic usage with standard normalization
# =====

# Load sample data
data <- sample_toydata()

# Create and fit transformer
transformer <- data_transformer$new()
transformer$fit(data)

# Transform data
transformed_data <- transformer$transform(data)
cat("Original dimensions:", dim(data), "\n")
cat("Transformed dimensions:", dim(transformed_data), "\n")

# Train GAN and generate synthetic data
trained_gan <- gan_trainer(transformed_data, epochs = 50)
synthetic_data <- sample_synthetic_data(trained_gan, transformer)

# Compare distributions
par(mfrow = c(1, 2))
plot(data, main = "Original Data")
plot(synthetic_data, main = "Synthetic Data")

# =====
# Example 2: Mode-specific normalization for complex distributions
# =====
```

```

# Create data with multiple modes
set.seed(42)
bimodal_data <- data.frame(
  x = c(rnorm(500, mean = -3), rnorm(500, mean = 3)),
  y = c(rnorm(500, mean = 0), rnorm(500, mean = 5))
)

# Fit with mode-specific normalization
transformer_gmm <- data_transformer$new()
transformer_gmm$fit(bimodal_data, mode_specific = TRUE, n_modes = 5)

# Check output dimensions (more columns due to mode indicators)
transformed_gmm <- transformer_gmm$transform(bimodal_data)
cat("Original columns:", ncol(bimodal_data), "\n")
cat("Transformed columns:", ncol(transformed_gmm), "\n")

# =====
# Example 3: Mixed continuous and categorical columns
# =====

# Create mixed data
mixed_data <- data.frame(
  age = rnorm(1000, mean = 40, sd = 15),
  income = rexp(1000, rate = 0.00002),
  gender = sample(c("M", "F"), 1000, replace = TRUE),
  education = sample(c("HS", "BA", "MA", "PhD"), 1000, replace = TRUE)
)

# Fit transformer specifying categorical columns
transformer_mixed <- data_transformer$new()
transformer_mixed$fit(
  mixed_data,
  discrete_columns = c("gender", "education"),
  mode_specific = TRUE, # GMM for continuous columns
  n_modes = 5
)

# Transform
transformed_mixed <- transformer_mixed$transform(mixed_data)
cat("Output dimensions:", transformer_mixed$output_dimensions, "\n")

# Inverse transform preserves types
reconstructed <- transformer_mixed$inverse_transform(transformed_mixed)
str(reconstructed)

# =====
# Example 4: Verifying inverse transform accuracy
# =====

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

```

```

# Round-trip transformation
transformed <- transformer$transform(data)
reconstructed <- transformer$inverse_transform(transformed)

# Check reconstruction error (should be very small)
max_error <- max(abs(as.matrix(data) - as.matrix(reconstructed)))
cat("Maximum reconstruction error:", max_error, "\n")

## End(Not run)

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

# Standard normalization
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)

# Mode-specific normalization
transformer$fit(data, mode_specific = TRUE, n_modes = 10)

# With categorical columns
transformer$fit(data, discrete_columns = c("category"))

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

data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
cat("Output dimensions:", dim(transformed_data))

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

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

# Round-trip transformation
transformed_data <- transformer$transform(data)
reconstructed_data <- transformer$inverse_transform(transformed_data)

# Use with GAN output
# synthetic_raw <- trained_gan$generator(noise)
# synthetic_data <- transformer$inverse_transform(as_array(synthetic_raw))

```

DCGAN_Discriminator	<i>DCGAN Discriminator</i>
---------------------	----------------------------

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	<i>DCGAN Generator</i>
-----------------	------------------------

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	<i>Discriminator</i>
---------------	----------------------

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

dp_gan_trainer	<i>dp_gan_trainer</i>
----------------	-----------------------

Description

Provides a function to train a GAN model with differential privacy guarantees using DP-SGD (Differentially Private Stochastic Gradient Descent). Uses OpenDP for cryptographically secure noise generation and Poisson subsampling.

Usage

```
dp_gan_trainer(
  data,
  noise_dim = 2,
  noise_distribution = "normal",
  data_type = "tabular",
  generator = NULL,
  discriminator = NULL,
  base_lr = 1e-04,
  target_epsilon = 1,
  target_delta = 1e-05,
  max_grad_norm = 1,
  noise_multiplier = NULL,
  sampling_rate = NULL,
  batch_size = 50,
  epochs = 50,
  plot_progress = FALSE,
  plot_interval = "epoch",
  eval_dropout = FALSE,
  synthetic_examples = 500,
  plot_dimensions = c(1, 2),
  track_loss = FALSE,
  device = "cpu",
  seed = NULL,
  verbose = TRUE,
  secure_rng = TRUE,
  checkpoint_epochs = NULL,
  checkpoint_path = NULL
)
```

Arguments

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

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 <code>torch_tensor</code> . For convenience "normal" and "uniform" will automatically set a function. Defaults to "normal".
data_type	"tabular" or "image", controls the data type, defaults to "tabular".
generator	The generator network. Expects a neural network provided as <code>torch::nn_module</code> . Default is NULL which will create a simple fully connected network.
discriminator	The discriminator network. Expects a neural network provided as <code>torch::nn_module</code> . Default is NULL which will create a simple fully connected network.
base_lr	The base learning rate for the optimizers. Default is 0.0001.
target_epsilon	Target epsilon for differential privacy. Training will stop if this budget is exhausted. Defaults to 1.0.
target_delta	Target delta for differential privacy. Defaults to 1e-5.
max_grad_norm	Maximum gradient norm for per-sample gradient clipping. Bounds the sensitivity of individual gradients. Defaults to 1.0.
noise_multiplier	Multiplier for Gaussian noise added to gradients. If NULL (default), it will be calibrated to achieve <code>target_epsilon</code> over the specified epochs. Higher values provide more privacy but reduce utility.
sampling_rate	Expected sampling rate for Poisson subsampling. If NULL (default), calculated as <code>batch_size / nrow(data)</code> . Must be between 0 and 1.
batch_size	Target batch size for training. With Poisson subsampling, actual batch sizes will vary. Defaults to 50.
epochs	The number of training epochs. Defaults to 50.
plot_progress	Monitor training progress with plots. Defaults to FALSE.
plot_interval	Number of training steps between plots. Defaults to "epoch".
eval_dropout	Should dropout be applied during sampling? Defaults to FALSE.
synthetic_examples	Number of synthetic examples to generate. Defaults to 500.
plot_dimensions	Which dimensions to plot. Defaults to <code>c(1, 2)</code> .
track_loss	Store training losses as output. Defaults to FALSE.
device	Device for computation ("cpu", "cuda", "mps"). Defaults to "cpu".
seed	Optional seed for reproducibility. Defaults to NULL.
verbose	Print privacy accounting information during training. Defaults to TRUE.
secure_rng	Use cryptographically secure RNG from OpenDP for noise generation. Defaults to TRUE. Set to FALSE for faster training during development/testing (uses <code>torch</code> 's standard RNG which is not cryptographically secure).
checkpoint_epochs	Interval for saving model checkpoints (in epochs). If NULL (default), no checkpoints are saved. Checkpoints are required for post-GAN boosting.
checkpoint_path	Optional path for disk-based checkpoint persistence. If NULL (default), checkpoints are stored in memory only.

Details

This function implements DP-SGD (Abadi et al., 2016) for training GANs with formal differential privacy guarantees. Key privacy mechanisms include:

Poisson Subsampling: Each training example is included in a batch independently with probability $q = \text{sampling_rate}$, providing privacy amplification. Uses OpenDP's cryptographically secure random number generation.

Per-Sample Gradient Clipping: Each sample's gradient is computed individually and clipped to bound sensitivity.

Gaussian Noise: Calibrated Gaussian noise is added to clipped gradients using OpenDP's secure Gaussian mechanism.

RDP Accounting: Uses Renyi Differential Privacy for tight composition of privacy loss across training steps.

The discriminator is trained with DP-SGD while the generator is trained normally (since it only sees synthetic data from the discriminator's gradients).

Value

A list of class "trained_RGAN" containing:

- generator: The trained generator network
- discriminator: The trained discriminator network
- losses: Training losses if track_loss is TRUE
- privacy: Privacy accounting information including final epsilon
- settings: Training settings used

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (pp. 308-318).

Mironov, I. (2017). Renyi differential privacy. In 2017 IEEE 30th computer security foundations symposium (CSF) (pp. 263-275).

Examples

```
## Not run:
# Before running, install OpenDP: install.packages("opendp")
# and torch: torch::install_torch()

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

# Train with differential privacy (epsilon = 1)
trained_gan <- dp_gan_trainer(
```

```

    transformed_data,
    target_epsilon = 1.0,
    target_delta = 1e-5,
    max_grad_norm = 1.0,
    epochs = 50
)

# Check final privacy budget
print(trained_gan$privacy$final_epsilon)

# Sample synthetic data
synthetic_data <- sample_synthetic_data(trained_gan, transformer)

## End(Not run)

```

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	<i>gan_trainer</i>
-------------	--------------------

Description

Provides a function to quickly train a GAN model.

Usage

```
gan_trainer(  
  data,  
  noise_dim = 2,  
  noise_distribution = "normal",  
  value_function = "original",  
  gp_lambda = 10,  
  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),  
  track_loss = FALSE,  
  plot_loss = FALSE,  
  device = "cpu",  
  seed = NULL,  
  validation_data = NULL,  
  early_stopping = FALSE,  
  patience = 10,  
  lr_schedule = "constant",  
  lr_decay_factor = 0.1,  
  lr_decay_steps = 50,  
  pac = 1,  
  output_info = NULL,  
  gumbel_tau = 0.2,  
  generator_hidden_units = list(256, 256),  
  generator_normalization = "batch",  
  generator_activation = "relu",  
  generator_init = "xavier_uniform",
```

```

    generator_residual = TRUE,
    checkpoint_epochs = NULL,
    checkpoint_path = NULL
)

```

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 convenience four loss functions "original", "wasserstein", "wgan-gp", and "f-wgan" are already implemented. Defaults to "original".
gp_lambda	The gradient penalty coefficient for WGAN-GP training. Only used when value_function is "wgan-gp". Defaults to 10.
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.
track_loss	Store the training losses as additional output. Defaults to FALSE.
plot_loss	Monitor the losses during training with plots. Defaults to FALSE.
device	Input on which device (e.g. "cpu", "cuda", or "mps") training should be done. Defaults to "cpu".
seed	Optional seed for reproducibility. Sets both R's random seed and torch's random seed. Defaults to NULL (no seed).
validation_data	Optional validation data for monitoring training. Should be in the same format as training data.
early_stopping	Enable early stopping based on validation metrics. Defaults to FALSE.
patience	Number of epochs without improvement before stopping. Only used if early_stopping is TRUE. Defaults to 10.
lr_schedule	Learning rate schedule type. One of "constant" (default), "step", "exponential", or "cosine". "step" reduces LR by lr_decay_factor every lr_decay_steps epochs. "exponential" applies lr_decay_factor decay each epoch. "cosine" uses cosine annealing from base_lr to 0 over all epochs.
lr_decay_factor	Multiplicative factor for learning rate decay. Used with "step" and "exponential" schedules. Defaults to 0.1.
lr_decay_steps	Number of epochs between learning rate reductions for "step" schedule. Defaults to 50.
pac	Number of samples to pack together for PacGAN (reduces mode collapse). The discriminator sees pac samples concatenated together, helping it detect lack of diversity. Must divide batch_size evenly. Defaults to 1 (standard GAN, no packing). Common values are 8 or 10.
output_info	Optional output structure from data_transformer\$output_info. When provided, enables Gumbel-Softmax for categorical columns, improving gradient flow for discrete variables. Each element should be a list with (dimension, type) where type is "linear", "mode_specific", or "softmax".
gumbel_tau	Temperature for Gumbel-Softmax. Lower values (e.g., 0.2) produce more discrete outputs. Only used when output_info is provided. Defaults to 0.2.
generator_hidden_units	List of hidden layer sizes for TabularGenerator. Defaults to list(256, 256) as used in CTGAN. Only used when output_info is provided.

generator_normalization	Normalization type for TabularGenerator: "batch" (default, CTGAN-style), "layer", or "none". Only used when output_info is provided.
generator_activation	Activation function for TabularGenerator: "relu" (default), "leaky_relu", "gelu", or "silu". Only used when output_info is provided.
generator_init	Weight initialization for TabularGenerator: "xavier_uniform" (default), "xavier_normal", "kaiming_uniform", or "kaiming_normal". Only used when output_info is provided.
generator_residual	Enable residual connections in TabularGenerator. Defaults to TRUE. Only used when output_info is provided.
checkpoint_epochs	Interval for saving model checkpoints (in epochs). If NULL (default), no checkpoints are saved. For example, checkpoint_epochs = 10 saves checkpoints at epochs 10, 20, 30, etc. Checkpoints enable post-GAN boosting for improved sample quality.
checkpoint_path	Optional path for disk-based checkpoint persistence. If NULL (default), checkpoints are stored in memory only. If provided, checkpoints are saved to disk, enabling post-GAN boosting for large training runs with many checkpoints.

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	<i>GAN_update_plot</i>
-----------------	------------------------

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 perform a single GAN training update step, including discriminator and generator updates.

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,
  gp_lambda = 0,
  track_loss = FALSE,
  pac = 1
)
```

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 convenience four loss functions "original", "wasserstein", "wgan-gp", 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.
gp_lambda	The gradient penalty coefficient for WGAN-GP. Set to 0 to disable. Defaults to 0.
track_loss	Store the training losses as additional output. Defaults to FALSE.
pac	Number of samples to pack together for PacGAN. Defaults to 1 (no packing).

Value

A list with generator and discriminator losses if track_loss is TRUE, otherwise NULL

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, epsilon = 1e-07)
```

Arguments

- real_scores The discriminator scores on real examples ($D(x)$)
- fake_scores The discriminator scores on fake examples ($D(G(z))$)
- epsilon Small constant for numerical stability to avoid $\log(0)$. Defaults to $1e-7$.

Value

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

Generator	<i>Generator</i>
-----------	------------------

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

gradient_penalty	<i>Gradient Penalty for WGAN-GP</i>
------------------	-------------------------------------

Description

Computes the gradient penalty for WGAN-GP training as described in Gulrajani et al. (2017) "Improved Training of Wasserstein GANs". The gradient penalty enforces the Lipschitz constraint on the discriminator by penalizing gradients that deviate from norm 1 on interpolated samples.

Usage

```
gradient_penalty(d_net, real_data, fake_data, device = "cpu")
```

Arguments

d_net	The discriminator network (torch::nn_module)
real_data	Real data samples (torch_tensor)
fake_data	Generated fake data samples (torch_tensor)
device	The device to use ("cpu", "cuda", or "mps")

Value

The gradient penalty loss (torch_tensor)

gumbel_softmax	<i>Gumbel-Softmax Sampling</i>
----------------	--------------------------------

Description

Implements the Gumbel-Softmax (Concrete) distribution for differentiable sampling from categorical distributions. During training, returns soft samples that allow gradients to flow. During inference, can return hard one-hot samples.

Usage

```
gumbel_softmax(logits, tau = 1, hard = FALSE, dim = -1)
```

Arguments

logits	A torch tensor of unnormalized log probabilities
tau	Temperature parameter. Lower values make the distribution more discrete. Defaults to 1.0.
hard	If TRUE, returns hard one-hot samples but gradients are computed as if soft samples were used (straight-through estimator). Defaults to FALSE.
dim	The dimension along which to apply softmax. Defaults to -1 (last dimension).

Value

A torch tensor of the same shape as logits, containing either soft or hard samples

Examples

```
## Not run:
logits <- torch::torch_randn(c(10, 5)) # 10 samples, 5 categories
soft_samples <- gumbel_softmax(logits, tau = 0.5)
hard_samples <- gumbel_softmax(logits, tau = 0.5, hard = TRUE)

## End(Not run)
```

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 ($\mathbb{D}(G(z))$)

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 ($\mathbb{D}(G(z))$)

Value

The loss

kl_real	<i>KL WGAN loss on real examples</i>
---------	--------------------------------------

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

KLWGAN_value_fct	<i>KLWGAN Value Function</i>
------------------	------------------------------

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

load_gan	<i>Load a Trained GAN</i>
----------	---------------------------

Description

Loads a trained GAN model that was previously saved using [save_gan](#). The loaded model can be used for sampling synthetic data or continued training.

Usage

```
load_gan(path, device = "cpu")
```

Arguments

path	The base file path to the saved model (without extension, same as used in save_gan)
device	The device to load the model onto ("cpu", "cuda", or "mps"). Defaults to "cpu". Use this to move a model trained on GPU to CPU or vice versa.

Value

A trained GAN object of class "trained_RGAN" that can be used with [sample_synthetic_data](#) or passed to [gan_trainer](#) for continued training.

Examples

```
## Not run:
# Load a previously saved GAN
loaded_gan <- load_gan("my_gan_model")

# Use it to generate synthetic data
transformer <- data_transformer$new()
# (fit transformer to original data or load it separately)
synthetic_data <- sample_synthetic_data(loaded_gan, transformer, n_samples = 100)

# Or continue training
continued_gan <- gan_trainer(
  transformed_data,
  generator = loaded_gan$generator,
  discriminator = loaded_gan$discriminator,
  generator_optimizer = loaded_gan$generator_optimizer,
  discriminator_optimizer = loaded_gan$discriminator_optimizer,
  epochs = 50
)

## End(Not run)
```

plot_losses	<i>Plot GAN Training Losses</i>
-------------	---------------------------------

Description

Plots the generator and discriminator loss curves from GAN training. Requires the GAN to have been trained with `track_loss = TRUE`.

Usage

```
plot_losses(trained_gan, smooth = 0, ...)
```

Arguments

<code>trained_gan</code>	A trained GAN object of class "trained_RGAN" with tracked losses
<code>smooth</code>	Smoothing factor for the loss curves (0 = no smoothing, higher = more smoothing). Uses exponential moving average. Defaults to 0.
<code>...</code>	Additional arguments passed to <code>plot()</code>

Value

Invisibly returns NULL. Called for side effect of producing a plot.

Examples

```
## Not run:
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
trained_gan <- gan_trainer(transformed_data, epochs = 50, track_loss = TRUE)
plot_losses(trained_gan)
plot_losses(trained_gan, smooth = 0.9) # With smoothing

## End(Not run)
```

post_gan_boosting	<i>Post-GAN Boosting</i>
-------------------	--------------------------

Description

Implements the Post-GAN Boosting algorithm from Neunhoeffter et al. (2021) "Private Post-GAN Boosting" (ICLR 2021). This algorithm improves the quality of GAN samples by learning a distribution over candidate samples that fools an ensemble of discriminators using multiplicative weights.

Usage

```

post_gan_boosting(
  d_score_fake,
  d_score_real,
  B,
  real_N,
  steps = 400,
  N_generators = 200,
  uniform_init = TRUE,
  dp = FALSE,
  MW_epsilon = 0.1,
  weighted_average = FALSE,
  averaging_window = NULL
)

```

Arguments

d_score_fake	Matrix of discriminator scores on fake samples (N_discriminators x N_samples). Each row contains scores from one discriminator checkpoint for all candidate samples.
d_score_real	Vector of mean discriminator scores on real data (length N_discriminators).
B	Matrix of candidate synthetic samples (N_samples x data_dim).
real_N	Number of <i>independently scored</i> records behind d_score_real. The exponential mechanism is calibrated to a sensitivity of $1 / \text{real_N}$, so this must be the number of scores that were averaged, not the number of rows in the training data. With PacGAN packing ($\text{pac} > 1$) each score covers pac records and the correct value is n / pac ; <code>compute_discriminator_scores()</code> reports it as <code>n_real_units</code> .
steps	Number of boosting iterations. Defaults to 400.
N_generators	Number of discriminator checkpoints used. Must equal <code>nrow(d_score_fake)</code> . Defaults to 200.
uniform_init	Initialize phi with uniform distribution. Defaults to TRUE.
dp	Use differential privacy for discriminator selection via exponential mechanism. Defaults to FALSE. Requires all discriminator scores to lie in $[0, 1]$; unbounded critics (WGAN, KLWGAN) are rejected with an error, since the $1 / \text{real_N}$ sensitivity the mechanism is calibrated to does not hold for them. Rescore with <code>score_transform = "sigmoid"</code> to use them.
MW_epsilon	Total privacy budget for multiplicative weights (only if <code>dp=TRUE</code>). Split as $\text{MW_epsilon} / \text{steps}$ per round, so basic composition over the steps rounds spends MW_epsilon in total. Defaults to 0.1.
weighted_average	Use weighted averaging (weights proportional to $\sqrt{\text{step}}$). Defaults to FALSE.
averaging_window	Number of final steps to average over. Defaults to NULL (all steps).

Value

A list with:

- PGB_sample: Matrix of selected high-quality samples
- d_score_PGB: Discriminator scores for selected samples
- D_bar: The mixed discriminator strategy, i.e. the fraction of the steps rounds in which each discriminator was selected (length N_generators).

References

Neunhoeffer, M., Wu, Z. S., & Dwork, C. (2021). Private Post-GAN Boosting. International Conference on Learning Representations (ICLR).

Examples

```
## Not run:
# Typically called via apply_post_gan_boosting(), but can be used directly:
result <- post_gan_boosting(
  d_score_fake = discriminator_scores_matrix,
  d_score_real = real_scores_vector,
  B = candidate_samples,
  real_N = 10000,
  steps = 200
)
boosted_samples <- result$PGB_sample

## End(Not run)
```

print.trained_RGAN	<i>Print Method for Trained RGAN Objects</i>
--------------------	--

Description

Displays a summary of a trained GAN model, including network architecture, training settings, and final losses.

Usage

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

Arguments

x	A trained GAN object of class "trained_RGAN"
...	Additional arguments (currently unused)

Value

Invisibly returns the input object

Examples

```
## Not run:
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
trained_gan <- gan_trainer(transformed_data, epochs = 10, track_loss = TRUE)
print(trained_gan)

## End(Not run)
```

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, n_samples = NULL)
```

Arguments

trained_gan	A trained RGAN object of class "trained_RGAN"
transformer	The transformer object used to pre-process the data
n_samples	The number of synthetic samples to generate. Defaults to NULL, which will use trained_gan\$settings\$synthetic_examples.

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	<i>Sample Toydata</i>
----------------	-----------------------

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)
```

save_gan

*Save a Trained GAN***Description**

Saves a trained GAN model to disk, including the generator, discriminator, optimizers, and all training settings. The model can be restored later using [load_gan](#).

The function creates multiple files with the given path as base name:

- path_generator.pt - Generator network weights
- path_discriminator.pt - Discriminator network weights
- path_metadata.rds - Settings, losses, and metadata
- path_g_optim.pt - Generator optimizer state (if include_optimizers=TRUE)
- path_d_optim.pt - Discriminator optimizer state (if include_optimizers=TRUE)

Usage

```
save_gan(trained_gan, path, include_optimizers = TRUE)
```

Arguments

trained_gan	A trained GAN object of class "trained_RGAN" returned by gan_trainer
path	The base file path for saving (without extension). Files will be created with suffixes.
include_optimizers	Whether to include optimizer states for resuming training. Defaults to TRUE.

Value

Invisibly returns the base path where the model was saved

Examples

```
## Not run:
# Train a GAN
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
trained_gan <- gan_trainer(transformed_data, epochs = 10)

# Save the trained GAN
save_gan(trained_gan, "my_gan_model")

# Load it back later
loaded_gan <- load_gan("my_gan_model")

## End(Not run)
```

TabularGenerator

Tabular Generator with Gumbel-Softmax

Description

Provides a `torch::nn_module` Generator for tabular data that applies Gumbel-Softmax to categorical outputs for differentiable sampling. This improves gradient flow for discrete variables compared to standard softmax.

Supports state-of-the-art architectural choices from CTGAN and other modern tabular GAN architectures:

- **Residual connections:** Skip connections that improve gradient flow in deeper networks (enabled by default when consecutive layers have same width)
- **Batch Normalization:** Stabilizes training (CTGAN default)
- **Layer Normalization:** Alternative that works better with small batches
- **Multiple activation functions:** ReLU, LeakyReLU, GELU, SiLU
- **Weight initialization:** Xavier or Kaiming initialization
- **Self-Attention:** Captures relationships between features
- **Progressive Training:** Gradually increase network capacity

Usage

```
TabularGenerator(
  noise_dim,
  output_info,
  hidden_units = list(256, 256),
  dropout_rate = 0,
  tau = 0.2,
  normalization = "batch",
```

```

        activation = "relu",
        init_method = "xavier_uniform",
        residual = True,
        attention = False,
        attention_heads = 4,
        attention_dropout = 0.1
    )

```

Arguments

noise_dim	The length of the noise vector per example
output_info	A list describing the output structure from data_transformer\$output_info. Each element is a list with (dimension, type) where type is "linear", "mode_specific", or "softmax".
hidden_units	A list of the number of neurons per layer. Defaults to list(256, 256) as used in CTGAN.
dropout_rate	The dropout rate for each hidden layer. Only used when normalization is "none". Defaults to 0.0.
tau	Temperature for Gumbel-Softmax. Lower values produce more discrete outputs. Defaults to 0.2.
normalization	Type of normalization to use: "batch" (default, as in CTGAN), "layer", or "none". Batch normalization is generally preferred for GANs.
activation	Activation function: "relu" (default, as in CTGAN), "leaky_relu", "gelu", or "silu". GELU and SiLU are modern alternatives that can improve performance.
init_method	Weight initialization method: "xavier_uniform" (default), "xavier_normal", "kaiming_uniform", or "kaiming_normal". Xavier is generally preferred for networks with tanh/sigmoid outputs.
residual	Enable residual connections between layers of the same width. Defaults to True.
attention	Enable self-attention layers after residual blocks. Can be True (add attention after each block), False (no attention), or a vector of layer indices where attention should be added (e.g., c(2, 4) adds attention after blocks 2 and 4). Defaults to False.
attention_heads	Number of attention heads. Must divide hidden layer size evenly. Defaults to 4.
attention_dropout	Dropout rate for attention weights. Defaults to 0.1.

Value

A torch::nn_module for the Tabular Generator

Examples

```

## Not run:
# Basic usage with CTGAN-style defaults

```

```

output_info <- list(list(1, "linear"), list(3, "softmax"))
gen <- TabularGenerator(noise_dim = 128, output_info = output_info)

# With self-attention for capturing feature relationships
gen <- TabularGenerator(
  noise_dim = 128,
  output_info = output_info,
  hidden_units = list(256, 256, 256),
  attention = TRUE,
  attention_heads = 8
)

# Custom architecture with layer normalization and GELU
gen <- TabularGenerator(
  noise_dim = 128,
  output_info = output_info,
  hidden_units = list(256, 256, 256),
  normalization = "layer",
  activation = "gelu",
  init_method = "kaiming_uniform"
)

## 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	<i>WGAN Value Function</i>
----------------	----------------------------

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

<code>real_scores</code>	The discriminator scores on real examples ($D(x)$)
<code>fake_scores</code>	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	<i>WGAN Weight Clipper</i>
---------------------	----------------------------

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

<code>d_net</code>	A <code>torch::nn_module</code> (typically a discriminator/critic) for which the weights should be clipped
<code>clip_values</code>	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

`apply_post_gan_boosting`, [3](#)

`compute_discriminator_scores`, [4](#)

`data_transformer`, [5](#)

`DCGAN_Discriminator`, [14](#)

`DCGAN_Generator`, [14](#)

`Discriminator`, [15](#)

`dp_gan_trainer`, [16](#)

`expert_sample_synthetic_data`, [19](#)

`gan_trainer`, [20](#), [31](#), [37](#)

`GAN_update_plot`, [24](#)

`GAN_update_plot_image`, [25](#)

`gan_update_step`, [25](#)

`GAN_value_fct`, [26](#)

`Generator`, [27](#)

`gradient_penalty`, [28](#)

`gumbel_softmax`, [28](#)

`kl_fake`, [29](#)

`kl_gen`, [29](#)

`kl_real`, [30](#)

`KLWGAN_value_fct`, [30](#)

`load_gan`, [31](#), [37](#)

`plot_losses`, [32](#)

`post_gan_boosting`, [32](#)

`print.trained_RGAN`, [34](#)

`sample_synthetic_data`, [31](#), [35](#)

`sample_toydata`, [36](#)

`save_gan`, [31](#), [37](#)

`TabularGenerator`, [38](#)

`torch_rand_ab`, [40](#)

`WGAN_value_fct`, [41](#)

`WGAN_weight_clipper`, [41](#)