# Package 'soilVAE'

November 8, 2026

**Type** Package

**Title** Supervised Variational Autoencoder Regression via 'reticulate'

**Version** 0.2.0

**Description** Supervised latent-variable regression for high-dimensional predictors such as soil reflectance spectra. The model uses an encoder-decoder neural network with a stochastic Gaussian latent representation regularized by a Kullback-Leibler term, and a supervised prediction head trained jointly with the reconstruction objective (Kingma and Welling (2014) <doi:10.48550/arXiv.1312.6114>). Predictors and response are standardised inside the model, so new data are scaled like the training data. The implementation interfaces R with a 'Python' deep-learning backend ('Keras' and 'TensorFlow') and provides utilities for installation, training with early stopping, tuning, prediction, latent embeddings, and saving models.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Imports** reticulate (>= 1.41), stats, utils

**Suggests** knitr, rmarkdown, prospectr, pls, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**SystemRequirements** Python (>= 3.9); TensorFlow (>= 2.16); Keras (>= 3)

**Config/roxygen2/version** 8.1.1

**NeedsCompilation** no

**Author** Hugo Rodrigues [aut, cre]

**Maintainer** Hugo Rodrigues <rodrigues.machado.hugo@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-11-08 21:50:02 UTC

## Contents

select_best_from_grid . . . . . 2

tune_vae_train_val . . . . . 3
vae_available . . . . . 4
vae_build . . . . . 4
vae_configure . . . . . 6
vae_encode . . . . . 6
vae_fit . . . . . 7
vae_history . . . . . 9
vae_install . . . . . 9
vae_predict . . . . . 10
vae_save . . . . . 11

**Index** **13**

---

`select_best_from_grid` *Select the best configuration from a tuning table*

---

## Description

Select the best configuration from a tuning table

## Usage

```
select_best_from_grid(
  tuning_df,
  selection_metric = c("euclid", "rmse", "r2", "rpiq")
)
```

## Arguments

| | |
|---|---|
| `tuning_df` | Data frame containing `RMSE_val`, `R2_val`, and `RPIQ_val`. |
| `selection_metric` | One of: `"euclid"`, `"rmse"`, `"r2"`, `"rpiq"`. |

## Value

List with `best` (one-row data frame) and `best_score`.

## Examples

```
tuning_df <- data.frame(
  cfg_id   = 1:3,
  RMSE_val = c(0.5, 0.3, 0.4),
  R2_val   = c(0.8, 0.9, 0.85),
  RPIQ_val = c(2.0, 3.0, 2.5)
)
best <- select_best_from_grid(tuning_df, selection_metric = "rmse")
best$best
```

tune_vae_train_val *Tune VAEReg on a train/validation split*

## Description

Tune VAEReg on a train/validation split

## Usage

```
tune_vae_train_val(X_tr, y_tr, X_va, y_va, seed = 123, grid_vae)
```

## Arguments

| | |
|---|---|
| X_tr | Train predictors matrix. |
| y_tr | Train response numeric. |
| X_va | Validation predictors matrix. |
| y_va | Validation response numeric. |
| seed | Integer seed. Configuration i is built with seed seed + i, so a tuning run is reproducible. |
| grid_vae | Data frame with required columns: hidden_enc (list), hidden_dec (list), latent_dim, dropout, lr, beta_kl, alpha_y, epochs, batch_size, patience; optionally l2 (default 1e-4). |

## Value

A list with grid (input grid) and tuning_df (metrics per config).

## Examples

```
if (vae_available()) {
  X <- matrix(rnorm(1500), nrow = 150, ncol = 10)
  y <- X[, 1] - X[, 2] + rnorm(150, sd = 0.1)
  grid <- data.frame(latent_dim = c(2L, 4L), dropout = 0.1, lr = 5e-4,
                     beta_kl = 1e-3, alpha_y = 1, epochs = 5L,
                     batch_size = 50L, patience = 3L)
  grid$hidden_enc <- list(16L, 16L)
  grid$hidden_dec <- list(16L, 16L)
  res <- tune_vae_train_val(X[1:100, ], y[1:100], X[101:150, ], y[101:150],
                            seed = 1, grid_vae = grid)
  res$tuning_df[, c("latent_dim", "R2_val", "RMSE_val")]
}
```

## vae_available *Is the Python backend available?*

### Description

Checks that 'Python' with 'TensorFlow' and 'Keras' (version 3 or later, running on the 'TensorFlow' backend) can be found by 'reticulate'. Use it to guard code that builds or fits a model.

### Usage

```
vae_available()
```

### Details

The check never installs or downloads anything. If 'Python' is not yet running in this R session, it first looks for an environment that has 'TensorFlow' (the `"r-soilvae"` environment made by `vae_install()`, one chosen with `vae_configure()`, or the other places 'reticulate' searches) and returns `FALSE` without starting 'Python' if there is none.

### Value

`TRUE` or `FALSE`.

### See Also

`vae_install()` to install the backend, `vae_configure()` to select a 'Python' environment.

### Examples

```
vae_available()
```

## vae_build *Build a supervised VAE regression model (VAEReg)*

### Description

The model encodes `x` into a Gaussian latent space, reconstructs `x` from a latent sample, and predicts `y` from the latent mean. It minimises

$$\mathrm{MSE}(x, \hat{x}) + \beta\,\mathrm{KL} + \alpha\,\mathrm{MSE}(y_s, \hat{y}_s)$$

where the reconstruction error is averaged over features and $y_s$ is `y` standardised by `vae_fit()`, so the three terms are on comparable scales whatever the number of predictors or the units of `y`.

## Usage

```
vae_build(
  input_dim,
  hidden_enc = c(512L, 256L),
  hidden_dec = c(256L, 512L),
  latent_dim = 32L,
  dropout = 0.1,
  lr = 5e-04,
  beta_kl = 0.001,
  alpha_y = 1,
  l2 = 1e-04,
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| input_dim | Number of predictors (columns of X). |
| hidden_enc | Integer vector of encoder layer sizes (one or more). |
| hidden_dec | Integer vector of decoder layer sizes (one or more). |
| latent_dim | Latent dimension. |
| dropout | Dropout rate after each hidden layer, in [0, 1) (0 for none). |
| lr | Adam learning rate (default 5e-4). |
| beta_kl | Weight of the KL term (default 1e-3; 0 turns the model into a plain supervised autoencoder). |
| alpha_y | Weight of the regression term (default 1). |
| l2 | L2 penalty on the weights of the hidden layers (default 1e-4; 0 for none). |
| seed | Optional integer seed for weight initialisation and latent sampling (seeds Python, NumPy and TensorFlow). Use it for reproducible fits. |

## Details

By default vae_fit() also centres and scales the predictors and stores the scaling in the model, so raw (but preprocessed, e.g. SNV) spectra can be passed to both vae_fit() and vae_predict().

## Value

A compiled 'Python' 'Keras' model object (VAEReg), not yet fitted.

## References

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. doi:10.48550/arXiv.1312.6114

## Examples

```
if (vae_available()) {
  model <- vae_build(input_dim = 100L, seed = 1L)
  model$count_params()
}
```

## vae_configure *Select the Python environment for soilVAE*

### Description

Call it at the start of an R session, before any other soilVAE function: once 'Python' has started, 'reticulate' cannot switch to another one. It is needed only for an environment other than `"r-soilvae"` (the one `vae_install()` creates and soilVAE uses by default).

### Usage

```
vae_configure(python = NULL, venv = NULL, conda = NULL)
```

### Arguments

| | |
|---|---|
| `python` | Path to a 'Python' executable. |
| `venv` | Name or path of a virtual environment. |
| `conda` | Name of a conda environment. |

### Value

`TRUE`, invisibly.

### Examples

```
## Not run:
  vae_configure(venv = "my-tf-env")
  vae_available()

## End(Not run)
```

## vae_encode *Extract latent embeddings from VAEReg*

### Description

Returns the latent mean of each sample (deterministic).

### Usage

```
vae_encode(model, X)
```

### Arguments

| | |
|---|---|
| `model` | Fitted Python VAEReg. |
| `X` | Numeric matrix or data frame with the same columns as the training data, or a numeric vector for a single sample. |

## Value

Matrix of latent embeddings (samples x latent_dim).

## Examples

```
if (vae_available()) {
  X <- matrix(rnorm(2000), nrow = 200, ncol = 10)
  model <- vae_build(input_dim = ncol(X), hidden_enc = 32L, hidden_dec = 32L,
                     latent_dim = 4L, seed = 1L)
  vae_fit(model, X, X[, 1], epochs = 20L)
  dim(vae_encode(model, X))
}
```

---

# vae_fit *Fit VAEReg*

---

## Description

Standardises `y` and, by default, `X` with training statistics. Both are stored in the model, so `vae_predict()` scales new data the same way and returns the original units of `y`.

## Usage

```
vae_fit(
  model,
  X,
  y,
  X_val = NULL,
  y_val = NULL,
  epochs = 80L,
  batch_size = 64L,
  patience = 10L,
  verbose = 0L,
  reduce_lr = TRUE,
  validation_split = 0,
  scale_x = c("global", "band", "none")
)
```

## Arguments

| | |
|---|---|
| `model` | Python VAEReg object from `vae_build()`. |
| `X` | Numeric predictor matrix or data frame (n x p) of finite values. Its column names, if any, are stored and checked by `vae_predict()`. |
| `y` | Numeric response (length n), finite values only. |
| `X_val, y_val` | Optional validation predictors (same columns as `X`) and response. Give both or neither. |

epochs — Maximum number of epochs.

batch_size — Mini-batch size.

patience — Early-stopping patience in epochs (only with validation data).

verbose — 0, 1 or 2, passed to Keras.

reduce_lr — With validation data, halve the learning rate when the validation prediction loss stops improving (default TRUE).

validation_split — Share of the training rows held out for validation when X_val is not given (default 0: no validation, no early stopping). Rows are drawn with R's random number generator, so use set.seed() for a reproducible split.

scale_x — How to scale the predictors (stored in the model and applied to new data automatically):

- "global" **(default, or** TRUE**)** centre each predictor and divide all of them by one overall SD. This keeps the relative size of the predictors, which suits spectra: noisy bands with little variance are not inflated.
- "band" centre and scale each predictor by its own SD, for predictors on unrelated scales.
- "none" **(or** FALSE**)** no scaling, if you scale X yourself.

## Details

With validation data (X_val/y_val, or a share of the training rows via validation_split), training stops when the validation *prediction* loss stops improving and the best weights are restored, and the learning rate is halved when that loss stalls. Neither happens during the first patience epochs (a warm-up), because early validation losses are noisy.

All inputs are checked before the model is changed. Calling vae_fit() again on the same model continues training from the current weights, with the scaling recomputed from the new data; build a new model with vae_build() to start afresh.

## Value

The fitted model, invisibly (it is also updated in place). Use vae_history() for the per-epoch losses.

## Examples

```
if (vae_available()) {
  X <- matrix(rnorm(2000), nrow = 200, ncol = 10)
  y <- X[, 1] - X[, 2] + rnorm(200, sd = 0.1)
  model <- vae_build(input_dim = ncol(X), hidden_enc = 32L, hidden_dec = 32L,
                     latent_dim = 4L, seed = 1L)
  vae_fit(model, X, y, validation_split = 0.2, epochs = 30L)
}
```

## vae_history *Training history of a fitted VAEReg*

### Description

Training history of a fitted VAEReg

### Usage

```
vae_history(model)
```

### Arguments

| | |
|---|---|
| model | Python VAEReg fitted with vae_fit(). |

### Value

A list of per-epoch values of the last vae_fit() call: loss, recon_loss, kl_loss, y_loss, and with validation data the same names prefixed by val_ (and the learning rate, learning_rate). y_loss is on the standardised scale of y. The history is kept only in the R session where the model was fitted; vae_save() does not store it.

### Examples

```
if (vae_available()) {
  X <- matrix(rnorm(2000), nrow = 200, ncol = 10)
  model <- vae_build(input_dim = 10L, hidden_enc = 32L, hidden_dec = 32L,
                     latent_dim = 4L, seed = 1L)
  vae_fit(model, X, X[, 1], validation_split = 0.2, epochs = 20L)
  h <- vae_history(model)
  plot(h$val_y_loss, type = "l", xlab = "Epoch", ylab = "Validation y loss")
}
```

## vae_install *Install the Python backend for soilVAE*

### Description

Creates a 'Python' virtual environment (or conda environment) and installs 'TensorFlow' and 'Keras' in it (about 1 GB of downloads). Run it once. soilVAE uses the "r-soilvae" environment automatically in later sessions; for another name, call vae_configure(venv = envname) (or conda = envname) at the start of each session.

## Usage

```
vae_install(
  envname = "r-soilvae",
  method = c("virtualenv", "conda"),
  python_version = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| envname | Name of the environment to create or reuse. |
| method | "virtualenv" (default) or "conda". |
| python_version | 'Python' version for a new environment. The default uses any installed 'Python' from 3.9 to 3.13 for a virtual environment, and 3.11 for conda. |
| ... | Further arguments passed to reticulate::py_install(). |

## Details

If 'Python' is already running in the R session, restart R after installing.

## Value

The environment name, invisibly.

## Examples

```
## Not run:
  vae_install()
  vae_available()

## End(Not run)
```

---

# vae_predict *Predict y with VAEReg*

---

## Description

Deterministic: the prediction is computed from the latent mean, so repeated calls on the same data return identical values. New data are scaled with the training statistics stored by vae_fit(), and results are in the original units of y.

## Usage

```
vae_predict(model, X)
```

### Arguments

| | |
|---|---|
| model | Fitted Python VAEReg. |
| X | Numeric matrix or data frame with the same columns as the training data (finite values only), or a numeric vector for a single sample. If both have column names, the columns are matched by name. |

### Value

Numeric vector of predicted values.

### Examples

```
if (vae_available()) {
  X <- matrix(rnorm(2000), nrow = 200, ncol = 10)
  y <- X[, 1] - X[, 2] + rnorm(200, sd = 0.1)
  model <- vae_build(input_dim = ncol(X), hidden_enc = 32L, hidden_dec = 32L,
                     latent_dim = 4L, seed = 1L)
  vae_fit(model, X, y, epochs = 30L)
  head(vae_predict(model, X))
}
```

---

## vae_save *Save and load a fitted VAEReg*

---

### Description

Saves the architecture, the weights, the predictor and target scaling and the column names in a single Keras file, so the loaded model predicts exactly like the saved one. The training history is not saved.

### Usage

```
vae_save(model, file)

vae_load(file)
```

### Arguments

| | |
|---|---|
| model | Fitted Python VAEReg. |
| file | Path of a `.keras` file. |

### Value

`vae_save()` returns `file` invisibly; `vae_load()` returns the model.

## Examples

```
if (vae_available()) {
  X <- matrix(rnorm(2000), nrow = 200, ncol = 10)
  model <- vae_build(input_dim = ncol(X), hidden_enc = 32L, hidden_dec = 32L,
                     latent_dim = 4L, seed = 1L)
  vae_fit(model, X, X[, 1], epochs = 20L)
  f <- tempfile(fileext = ".keras")
  vae_save(model, f)
  model2 <- vae_load(f)
  all.equal(vae_predict(model2, X), vae_predict(model, X))
}
```

# Index

select_best_from_grid, 2

tune_vae_train_val, 3

vae_available, 4
vae_build, 4
vae_configure, 6
vae_encode, 6
vae_fit, 7
vae_history, 9
vae_install, 9
vae_load *(*vae_save*)*, 11
vae_predict, 10
vae_save, 11