Package 'VAExprs'

October 18, 2022

Type Package

Title Generating Samples of Gene Expression Data with Variational Autoencoders

Description A fundamental problem in biomedical research is the low number of observations, mostly due to a lack of available biosamples, prohibitive costs, or ethical reasons. By augmenting a few real observations with artificially generated samples, their analysis could lead to more robust and higher reproducible. One possible solution to the problem is the use of generative models, which are statistical models of data that attempt to capture the entire probability distribution from the observations. Using the variational autoencoder (VAE), a well-known deep generative model, this package is aimed to generate samples with gene expression data, especially for single-cell RNA-seq data. Furthermore, the VAE can use conditioning to produce specific cell types or subpopulations. The conditional VAE (CVAE) allows us to create targeted samples rather than completely random ones.

Version 1.2.1

Date 2022-05-18

LazyData TRUE

Depends keras, mclust

Imports SingleCellExperiment, SummarizedExperiment, tensorflow, scater, CatEncoders, DeepPINCS, purrr, DiagrammeR, stats

Suggests SC3, knitr, testthat, reticulate, rmarkdown

License Artistic-2.0

biocViews Software, GeneExpression, SingleCell

NeedsCompilation no

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/VAExprs

git_branch RELEASE_3_15

git_last_commit 528bf0a

git_last_commit_date 2022-05-18

Date/Publication 2022-10-18

Author Dongmin Jung [cre, aut] (<https://orcid.org/0000-0001-7499-8422>)

Maintainer Dongmin Jung <dmdmjung@gmail.com>

R topics documented:

`fit_vae` *Variational autoencoder model fitting*

Description

A fundamental problem in biomedical research is the low number of observations available. Augmenting a few real observations with generated in silico samples could lead to more robust analysis. Here, the variational autoencoder (VAE) is used for the realistic generation of single-cell RNA-seq data. Also, the conditional variational autoencoder (CVAE) can be used if labels of samples are available. This function allows us to fit variational autoencoders with the standard Gaussian prior to expression data. It is assumed that there will likely be no clusters in the latent space representation of variational autoencoders.

Usage

```
fit_vae(object = NULL,
        x_train = NULL,
        x_val = NULL,
        y_train = NULL,
        y_val = NULL,
        encoder_layers,
        decoder_layers,
        latent_dim = 2,
        regularization = 1,
        epochs,
        batch_size,
        preprocessing = list(
            x_train = NULL,
            x_val = NULL,
            y_train = NULL,
            y_val = NULL,
            minmax = NULL,
            lenc = NULL),
        use_generator = FALSE,
        optimizer = "adam",
        validation_split = 0, ...)
```

Arguments

`object`	SummarizedExperiment object
`x_train`	expression data for train, where each row is a cell and each column is a gene
`x_val`	expression data for validation, where each row is a cell and each column is a gene
`y_train`	labels for train
`y_val`	labels for validation
`encoder_layers`	list of layers for encoder
`decoder_layers`	list of layers for decoder
`latent_dim`	dimension of latent vector (default: 2)
`regularization`	regularization parameter, which is nonnegative (default: 1)
`epochs`	number of epochs
`batch_size`	batch size
`preprocessing`	list of preprocessed results, they are set to NULL as default • x_train : expression data for train • x_val : expression data for validation • y_train : labels for train • y_val : labels for validation • minmax : result of min-max normalization • lenc : encoded labels
`use_generator`	use data generator if TRUE (default: FALSE)
`optimizer`	name of optimizer (default: adam)
`validation_split`	proportion of validation data, it is ignored when there is a validation set (default: 0)
`...`	additional parameters for the "fit" or "fit_generator"

Value

`model`	trained VAE model
`encoder`	trained encoder model
`decoder`	trained decoder model
`preprocessing`	preprocessed results

Author(s)

Dongmin Jung

References

Marouf, M., Machart, P., Bansal, V., Kilian, C., Magruder, D. S., Krebs, C. F., & Bonn, S. (2020). Realistic in silico generation and augmentation of single-cell RNA-seq data using generative adversarial networks. Nature communications, 11(1), 1-12.

See Also

SummarizedExperiment::assay, SummarizedExperiment::colData, scater::logNormCounts, keras::fit, keras::fit_generator, keras::compile, CatEncoders::LabelEncoder.fit, CatEncoders::transform, DeepPINCS::multiple_sampling_generator

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    ### simulate differentially expressed genes
    set.seed(1)
    g <- 3
    n <- 100
    m <- 1000
    mu <- 5
    sigma <- 5
    mat <- matrix(rnorm(n*m*g, mu, sigma), m, n*g)
    rownames(mat) <- paste0("gene", seq_len(m))
    colnames(mat) <- paste0("cell", seq_len(n*g))
    group <- factor(sapply(seq_len(g), function(x) {
        rep(paste0("group", x), n)
    }))
    names(group) <- colnames(mat)
    mu_upreg <- 6
    sigma_upreg <- 10
    deg <- 100
    for (i in seq_len(g)) {
        mat[(deg*(i-1) + 1):(deg*i), group == paste0("group", i)] <-
            mat[1:deg, group==paste0("group", i)] + rnorm(deg, mu_upreg, sigma_upreg)
    }
    # positive expression only
    mat[mat < 0] <- 0
    x_train <- as.matrix(t(mat))


    ### model
    batch_size <- 32
    original_dim <- 1000
    intermediate_dim <- 512
    epochs <- 2
    # VAE
    vae_result <- fit_vae(x_train = x_train,
                          encoder_layers = list(layer_input(shape = c(original_dim)),
                                                layer_dense(units = intermediate_dim,
                                                            activation = "relu")),
                          decoder_layers = list(layer_dense(units = intermediate_dim,
                                                            activation = "relu"),
                                                layer_dense(units = original_dim,
                                                            activation = "sigmoid")),
                          epochs = epochs, batch_size = batch_size,
                          validation_split = 0.5,
                          use_generator = FALSE,
                          callbacks = keras::callback_early_stopping(
```

```
                            monitor = "val_loss",
                            patience = 10,
                            restore_best_weights = TRUE))


    ### from preprocessing
    vae_result_preprocessing <- fit_vae(preprocessing = vae_result$preprocessing,
                             encoder_layers = list(layer_input(shape = c(original_dim)),
                                                   layer_dense(units = intermediate_dim,
                                                               activation = "relu")),
                             decoder_layers = list(layer_dense(units = intermediate_dim,
                                                               activation = "relu"),
                                                   layer_dense(units = original_dim,
                                                               activation = "sigmoid")),
                                       epochs = epochs, batch_size = batch_size,
                                       validation_split = 0.5,
                                       use_generator = FALSE,
                                       callbacks = keras::callback_early_stopping(
                                           monitor = "val_loss",
                                           patience = 10,
                                           restore_best_weights = TRUE))
}
```

`gen_exprs` *Generate samples with expression data*

Description

This function generate expression data by drawing samples from the latent vectors following the standard multivariate Gaussian distribution (the standard multivariate normal distribution) for convenience. However, this assumption for the prior may not be appropriate because there may be underlying distinctions between groups of samples. Any density function can be modeled by the Gaussian mixture model. Here, by using the library "mclust", the finite Gaussian mixture is applied for such sampling. Note that the Gaussian mixture model is not used for fitting in the function "fit_vae".

Usage

```
gen_exprs(x, num_samples,
        batch_size, use_generator = FALSE)
```

Arguments

`x`	result of the function "fit_vae"
`num_samples`	number of samples to be generated
`batch_size`	batch size
`use_generator`	use data generator if TRUE (default: FALSE)

Value

x_gen	generated expression data, where each row is a cell and each column is a gene
y_gen	geneated labels
x_train	real expression data, where each row is a cell and each column is a gene
y_train	real labels
latent_vector	latent vector from real expression data

Author(s)

Dongmin Jung

See Also

mclust::mclustBIC, mclust::mclustModel, mclust::sim, DeepPINCS::multiple_sampling_generator, CatEncoders::inverse.transform

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    ### simulate differentially expressed genes
    set.seed(1)
    g <- 3
    n <- 100
    m <- 1000
    mu <- 5
    sigma <- 5
    mat <- matrix(rnorm(n*m*g, mu, sigma), m, n*g)
    rownames(mat) <- paste0("gene", seq_len(m))
    colnames(mat) <- paste0("cell", seq_len(n*g))
    group <- factor(sapply(seq_len(g), function(x) {
        rep(paste0("group", x), n)
    }))
    names(group) <- colnames(mat)
    mu_upreg <- 6
    sigma_upreg <- 10
    deg <- 100
    for (i in seq_len(g)) {
        mat[(deg*(i-1) + 1):(deg*i), group == paste0("group", i)] <-
            mat[1:deg, group==paste0("group", i)] + rnorm(deg, mu_upreg, sigma_upreg)
    }
    # positive expression only
    mat[mat < 0] <- 0
    x_train <- as.matrix(t(mat))


    ### model
    batch_size <- 32
    original_dim <- 1000
    intermediate_dim <- 512
    epochs <- 2
```

```
    # VAE
    vae_result <- fit_vae(x_train = x_train,
                          encoder_layers = list(layer_input(shape = c(original_dim)),
                                                layer_dense(units = intermediate_dim,
                                                            activation = "relu")),
                          decoder_layers = list(layer_dense(units = intermediate_dim,
                                                            activation = "relu"),
                                                layer_dense(units = original_dim,
                                                            activation = "sigmoid")),
                          epochs = epochs, batch_size = batch_size,
                          validation_split = 0.5,
                          use_generator = FALSE,
                          callbacks = keras::callback_early_stopping(
                              monitor = "val_loss",
                              patience = 10,
                              restore_best_weights = TRUE))
    # plot
    plot_vae(vae_result$model)


    ### generate samples
    set.seed(1)
    gen_sample_result <- gen_exprs(vae_result, num_samples = 100)
}
```

---

plot_aug *Visualization for augmented data*

Description

For augmented data, we can create plots for specific types of dimension reduction.

Usage

```
plot_aug(x, plot_fun, ...)
```

Arguments

x	result of the function "gen_exprs"
plot_fun	"PCA", "MDS", "TSNE", "UMAP", "NMF", or "DiffusionMap"
...	additional parameters for the reduced dimension plots such as "scater::runPCA"

Value

plot for augmented data

Author(s)

Dongmin Jung

See Also

SingleCellExperiment::SingleCellExperiment, scater::logNormCounts, scater::runPCA, scater::runMDS, scater::runTSNE, scater::runUMAP, scater::runNMF, scater::runDiffusionMap, scater::plotPCA, scater::plotMDS, scater::plotTSNE, scater::plotUMAP, scater::plotNMF, scater::plotDiffusionMap

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    ### simulate differentially expressed genes
    set.seed(1)
    g <- 3
    n <- 100
    m <- 1000
    mu <- 5
    sigma <- 5
    mat <- matrix(rnorm(n*m*g, mu, sigma), m, n*g)
    rownames(mat) <- paste0("gene", seq_len(m))
    colnames(mat) <- paste0("cell", seq_len(n*g))
    group <- factor(sapply(seq_len(g), function(x) {
        rep(paste0("group", x), n)
    }))
    names(group) <- colnames(mat)
    mu_upreg <- 6
    sigma_upreg <- 10
    deg <- 100
    for (i in seq_len(g)) {
        mat[(deg*(i-1) + 1):(deg*i), group == paste0("group", i)] <-
            mat[1:deg, group==paste0("group", i)] + rnorm(deg, mu_upreg, sigma_upreg)
    }
    # positive expression only
    mat[mat < 0] <- 0
    x_train <- as.matrix(t(mat))


    ### model
    batch_size <- 32
    original_dim <- 1000
    intermediate_dim <- 512
    epochs <- 2
    # VAE
    vae_result <- fit_vae(x_train = x_train,
                          encoder_layers = list(layer_input(shape = c(original_dim)),
                                                layer_dense(units = intermediate_dim,
                                                            activation = "relu")),
                          decoder_layers = list(layer_dense(units = intermediate_dim,
                                                            activation = "relu"),
                                                layer_dense(units = original_dim,
                                                            activation = "sigmoid")),
                          epochs = epochs, batch_size = batch_size,
                          validation_split = 0.5,
                          use_generator = FALSE,
                          callbacks = keras::callback_early_stopping(
```

```
                            monitor = "val_loss",
                            patience = 10,
                            restore_best_weights = TRUE))
    # plot
    plot_vae(vae_result$model)


    ### generate samples
    set.seed(1)
    gen_sample_result <- gen_exprs(vae_result, num_samples = 100)
    # plot
    plot_aug(gen_sample_result, "PCA")
}
```

---

plot_vae *Visualization for the variational autoencoder*

Description

You can create a plot of the VAE model. This plot can help you check that the model is connected the way you intended. The node colors indicate the components of the VAE.

Usage

```
plot_vae(x, node_color = list(encoder_col = "tomato",
                              mean_vector_col = "orange",
                              stddev_vector_col = "lavender",
                              latent_vector_col = "lightblue",
                              decoder_col = "palegreen",
                              condition_col = "gray"))
```

Arguments

x	VAE model
node_color	node colors for encoder(default: tomato), mean vector(default: orange), standard deviation vector(default: lavender), latent_vector(default: lightblue), decoder(default: palegreen), and condition(default: gray)

Value

plot for the model architecture

Author(s)

Dongmin Jung

See Also

purrr::map, purrr::map_chr, purrr::pluck, purrr::imap_dfr, DiagrammeR::grViz

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    ### simulate differentially expressed genes
    set.seed(1)
    g <- 3
    n <- 100
    m <- 1000
    mu <- 5
    sigma <- 5
    mat <- matrix(rnorm(n*m*g, mu, sigma), m, n*g)
    rownames(mat) <- paste0("gene", seq_len(m))
    colnames(mat) <- paste0("cell", seq_len(n*g))
    group <- factor(sapply(seq_len(g), function(x) {
        rep(paste0("group", x), n)
    }))
    names(group) <- colnames(mat)
    mu_upreg <- 6
    sigma_upreg <- 10
    deg <- 100
    for (i in seq_len(g)) {
        mat[(deg*(i-1) + 1):(deg*i), group == paste0("group", i)] <-
            mat[1:deg, group==paste0("group", i)] + rnorm(deg, mu_upreg, sigma_upreg)
    }
    # positive expression only
    mat[mat < 0] <- 0
    x_train <- as.matrix(t(mat))


    ### model
    batch_size <- 32
    original_dim <- 1000
    intermediate_dim <- 512
    epochs <- 2
    # VAE
    vae_result <- fit_vae(x_train = x_train,
                          encoder_layers = list(layer_input(shape = c(original_dim)),
                                                layer_dense(units = intermediate_dim,
                                                            activation = "relu")),
                          decoder_layers = list(layer_dense(units = intermediate_dim,
                                                            activation = "relu"),
                                                layer_dense(units = original_dim,
                                                            activation = "sigmoid")),
                          epochs = epochs, batch_size = batch_size,
                          validation_split = 0.5,
                          use_generator = FALSE,
                          callbacks = keras::callback_early_stopping(
                              monitor = "val_loss",
                              patience = 10,
                              restore_best_weights = TRUE))
    # plot
    plot_vae(vae_result$model)
}
```

Index