Package 'GenProSeq'

May 21, 2026

Type Package

Title Generating Protein Sequences with Deep Generative Models

Description Generative modeling for protein engineering is key to solving fundamental problems in synthetic biology, medicine, and material science. Machine learning has enabled us to generate useful protein sequences on a variety of scales. Generative models are machine learning methods which seek to model the distribution underlying the data, allowing for the generation of novel samples with similar properties to those on which the model was trained. Generative models of proteins can learn biologically meaningful representations helpful for a variety of downstream tasks. Furthermore, they can learn to generate protein sequences that have not been observed before and to assign higher probability to protein sequences that satisfy desired criteria. In this package, common deep generative models for protein sequences, such as variational autoencoder (VAE), generative adversarial networks (GAN), and autoregressive models are available. In the VAE and GAN, the Word2vec is used for embedding. The transformer encoder is applied to protein sequences for the autoregressive model.

Version 1.17.0

Date 2024-02-06

LazyData FALSE

Depends keras, mclust, R (>= 4.2)

Imports tensorflow, word2vec, DeepPINCS, ttgsea, CatEncoders, reticulate, stats

Suggests VAExprs, stringdist, knitr, testthat, rmarkdown

License Artistic-2.0

biocViews Software, Proteomics

NeedsCompilation no

VignetteBuilder knitr

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

git_branch devel

git_last_commit 31769aa

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-20

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

Maintainer Dongmin Jung <dmdmjung@gmail.com>

Contents

ART *Autoregressive language model with Transformer*

Description

The autoregressive generative model predicts the next amino acid in a protein given the amino acid sequence up to that point. The autoregressive model generates proteins one amino acid at a time. For one step of generation, it takes a context sequence of amino acids as input and outputs a probability distribution over amino acids. We sample from that distribution and then update the context sequence with the sampled amino acid. The Transformer is used as an encoder model. The AR with the Transformer model can be trained by the function "fit_ART", and then the function "gen_ART" generates protein sequences.

Usage

```
fit_ART(prot_seq,
        length_seq,
        embedding_dim,
        num_heads,
        ff_dim,
        num_transformer_blocks,
        layers = NULL,
        prot_seq_val = NULL,
        epochs,
        batch_size,
        preprocessing = list(
            x_train = NULL,
            x_val = NULL,
            y_train = NULL,
            y_val = NULL,
            lenc = NULL,
            length_seq = NULL,
            num_AA = NULL,
            embedding_dim = NULL,
            removed_prot_seq = NULL,
            removed_prot_seq_val = NULL),
        use_generator = FALSE,
        optimizer = "adam",
        metrics = "accuracy",
```

```
        validation_split = 0, ...)

gen_ART(x,
        seed_prot,
        length_AA,
        method = NULL,
        b = NULL,
        t = 1,
        k = NULL,
        p = NULL)
```

Arguments

prot_seq	amino acid sequence
length_seq	length of sequence used as input
embedding_dim	dimension of the dense embedding
num_heads	number of attention heads
ff_dim	hidden layer size in feedforward network inside transformer
num_transformer_blocks	number of transformer blocks
layers	list of layers between the transformer encoder and the output layer (default: NULL)
prot_seq_val	amino acid sequence for validation (default: NULL)
epochs	number of epochs
batch_size	batch size
preprocessing	list of preprocessed results, they are set to NULL as default x_train, y_train, lenc, length_seq, num_AA, and embedding_dim must be required for training • x_train : embedded sequence data for train, result of the function "DeepPINCS::get_seq_encode_pad" • x_val : embedded sequence data for validation, result of the function "DeepPINCS::get_seq_encode_pad" • y_train : labels for train • y_val : labels for validation • lenc : encoded labels, result of the function "DeepPINCS::get_seq_encode_pad" • length_seq : length of sequence • num_AA : number of amino acids, result of the function "DeepPINCS::get_seq_encode_pad" • embedding_dim : dimension of the dense embedding • removed_prot_seq : index for removed protein sequences while checking • removed_prot_seq_val : index for removed protein sequences of validation
use_generator	use data generator if TRUE (default: FALSE)
optimizer	name of optimizer (default: adam)
metrics	name of metrics (default: accuracy)
validation_split	proportion of validation data, it is ignored when there is a validation set (default: 0)
...	additional parameters for the "fit"

x	result of the function "fit_ART"
seed_prot	sequence to be used as a seed protein
length_AA	length of amino acids to be generated
method	"greedy", "beam", "temperature", "top_k", or "top_p"
b	beam size in the beam search
t	temperature in the temperature sampling (default: 1)
k	number of amino acids in the top-k sampling
p	minimum probabilty for the set of amino acids in the top-p sampling

Value

model	trained ART model
preprocessing	preprocessed results

Author(s)

Dongmin Jung

References

Deepak, P., Chakraborty, T., & Long, C. (2021). Data Science for Fake News: Surveys and Perspectives (Vol. 42). Springer.

Liu, Z., Lin, Y., & Sun, M. (2020). Representation learning for natural language processing. Springer.

Madani, A., McCann, B., Naik, N., Keskar, N. S., Anand, N., Eguchi, R. R., Huang, P., & Socher, R. (2020). Progen: Language modeling for protein generation. arXiv:2004.03497.

See Also

keras::fit, keras::compile, ttgsea::sampling_generator, DeepPINCS::multiple_sampling_generator, DeepPINCS::seq_preprocessing, DeepPINCS::get_seq_encode_pad, CatEncoders::LabelEncoder.fit, CatEncoders::transform, CatEncoders::inverse.transform

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    prot_seq <- DeepPINCS::SARS_CoV2_3CL_Protease

    # model parameters
    length_seq <- 10
    embedding_dim <- 16
    num_heads <- 2
    ff_dim <- 16
    num_transformer_blocks <- 2
    batch_size <- 32
    epochs <- 2

    # ART
    ART_result <- fit_ART(prot_seq = prot_seq,
                          length_seq = length_seq,
                          embedding_dim = embedding_dim,
                          num_heads = num_heads,
```

```
                        ff_dim = ff_dim,
                        num_transformer_blocks = num_transformer_blocks,
                        layers = list(layer_dropout(rate = 0.1),
                                      layer_dense(units = 32, activation = "relu"),
                                      layer_dropout(rate = 0.1)),
                        prot_seq_val = prot_seq,
                        epochs = epochs,
                        batch_size = batch_size,
                        use_generator = TRUE,
                        callbacks = callback_early_stopping(
                            monitor = "val_loss",
                            patience = 10,
                            restore_best_weights = TRUE))

    seed_prot <- "SGFRKMAFPS"
    gen_ART(ART_result, seed_prot, length_AA = 20, method = "greedy")
    gen_ART(ART_result, seed_prot, length_AA = 20, method = "beam", b = 5)
    gen_ART(ART_result, seed_prot, length_AA = 20, method = "temperature", t = 0.1)
    gen_ART(ART_result, seed_prot, length_AA = 20, method = "top_k", k = 3)
    gen_ART(ART_result, seed_prot, length_AA = 20, method = "top_p", p = 0.75)


    ### from preprocessing
    ART_result2 <- fit_ART(num_heads = 4,
                        ff_dim = 32,
                        num_transformer_blocks = 3,
                        layers = list(layer_dropout(rate=0.1),
                                      layer_dense(units=32, activation="relu"),
                                      layer_dropout(rate=0.1)),
                        epochs = epochs,
                        batch_size = batch_size,
                        preprocessing = ART_result$preprocessing,
                        use_generator = TRUE,
                        callbacks = callback_early_stopping(
                                monitor = "val_loss",
                                patience = 50,
                                restore_best_weights = TRUE))

    gen_ART(ART_result2, seed_prot, length_AA = 20, method = "greedy")
    gen_ART(ART_result2, seed_prot, length_AA = 20, method = "beam", b = 5)
    gen_ART(ART_result2, seed_prot, length_AA = 20, method = "temperature", t = 0.1)
    gen_ART(ART_result2, seed_prot, length_AA = 20, method = "top_k", k = 3)
    gen_ART(ART_result2, seed_prot, length_AA = 20, method = "top_p", p = 0.75)
}
```

`example_luxA` *Example Data for Protein Sequences*

Description

The data consist of selected amino acid sequences of the luxA. There are 2283 aligned sequences of length 360.

Usage

```
example_luxA
```

Value

aligned amino acid sequences

Author(s)

Dongmin Jung

Source

Hawkins-Hooker, A., Depardieu, F., Baur, S., Couairon, G., Chen, A., & Bikard, D. (2020). Generating functional protein variants with variational autoencoders. bioRxiv.

---

example_PTEN *Example Data for Protein Sequences*

Description

The data consist of selected amino acid sequences of the PTEN. There are 912 aligned sequences of length 403.

Usage

```
example_PTEN
```

Value

aligned amino acid sequences

Author(s)

Dongmin Jung

Source

Frazer, J., Notin, P., Dias, M., Gomez, A., Brock, K., Gal, Y., & Marks, D. (2020). Large-scale clinical interpretation of genetic variants using evolutionary data and deep learning. bioRxiv.

GAN *Generative adversarial network for generating protein sequences*

Description

The generative adversarial network (GAN) is made up of a discriminator and a generator that compete in a two-player minimax game. The objective of the generator is to produce an output that is so close to real that it confuses the discriminator in being able to differentiate the fake data from the real data. The conditional GAN (CGAN) is based on vanilla GAN with additional conditional input to generator and discriminator. The auxiliary classifier GAN (ACGAN) is an extension of CGAN that adds conditional input only to the generator. The Word2vec is applied to amino acids for embedding. The GAN or ACGAN model can be trained by the function "fit_GAN", and then the function "gen_GAN" generates protein sequences from the trained model.

Usage

```
fit_GAN(prot_seq,
        label = NULL,
        length_seq,
        embedding_dim,
        embedding_args = list(),
        latent_dim = NULL,
        intermediate_generator_layers,
        intermediate_discriminator_layers,
        prot_seq_val = NULL,
        label_val = NULL,
        epochs,
        batch_size,
        preprocessing = list(
            x_train = NULL,
            x_val = NULL,
            y_train = NULL,
            y_val = NULL,
            lenc = NULL,
            length_seq = NULL,
            num_seq = NULL,
            embedding_dim = NULL,
            embedding_matrix = NULL,
            removed_prot_seq = NULL,
            removed_prot_seq_val = NULL,
            latent_dim = NULL),
        optimizer = "adam",
        validation_split = 0)

gen_GAN(x,
        label = NULL,
        num_seq,
        remove_gap = TRUE)
```

Arguments

prot_seq	aligned amino acid sequence
label	label (default: NULL)
length_seq	length of sequence
embedding_dim	dimension of the dense embedding
embedding_args	list of arguments for "word2vec::word2vec" but for dim, min_count and split
latent_dim	dimension of latent vector (default: NULL)
intermediate_generator_layers	list of intermediate layers for generator, without input layer
intermediate_discriminator_layers	list of intermediate layers for discriminator, without output layer
prot_seq_val	amino acid sequence for validation (default: NULL)
label_val	label for validation (default: NULL)
epochs	number of epochs

batch_size	batch size
preprocessing	list of preprocessed results, they are set to NULL as default x_train, length_seq, num_seq, embedding_dim and embedding_matrix must be required for training • x_train : embedded sequence data for train • x_val : embedded sequence data for validation • y_train : labels for train • y_val : labels for validation • lenc : encoded labels • length_seq : length of sequence • num_seq : number of sequences for train • embedding_dim : dimension of the dense embedding • embedding_matrix : embedding matrix • removed_prot_seq : index for removed protein sequences while checking • removed_prot_seq_val : index for removed protein sequences of validation • latent_dim : dimension of latent vector
optimizer	name of optimizer (default: adam)
validation_split	proportion of validation data, it is ignored when there is a validation set (default: 0)
x	result of the function "fit_GAN"
num_seq	number of sequences to be generated
remove_gap	remove gaps from sequences (default: TRUE)

Value

model	trained GAN model
generator	trained generator model
discriminator	trained discriminator model
preprocessing	preprocessed results
gen_seq	generated sequence data
label	labels for generated sequence data

Author(s)

Dongmin Jung

References

Liebowitz, J. (Ed.). (2020). Data Analytics and AI. CRC Press.

Pedrycz, W., & Chen, S. M. (Eds.). (2020). Deep Learning: Concepts and Architectures. Springer.

Suguna, S. K., Dhivya, M., & Paiva, S. (Eds.). (2021). Artificial Intelligence (AI): Recent Trends and Applications. CRC Press.

Sun, S., Mao, L., Dong, Z., & Wu, L. (2019). Multiview machine learning. Springer.

See Also

keras::train_on_batch, keras::evaluate, keras::compile, CatEncoders::LabelEncoder.fit, CatEncoders::transform, CatEncoders::inverse.transform

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    data("example_PTEN")
    # model parameters
    length_seq <- 403
    embedding_dim <- 8
    latent_dim <- 4
    epochs <- 2
    batch_size <- 64

    # GAN
    GAN_result <- fit_GAN(prot_seq = example_PTEN,
                          length_seq = length_seq,
                          embedding_dim = embedding_dim,
                          latent_dim = latent_dim,
                          intermediate_generator_layers = list(
                              layer_dense(units = 16),
                              layer_dense(units = 128)),
                          intermediate_discriminator_layers = list(
                              layer_dense(units = 128, activation = "relu"),
                              layer_dense(units = 16, activation = "relu")),
                          prot_seq_val = example_PTEN,
                          epochs = epochs,
                          batch_size = batch_size)
    set.seed(1)
    gen_prot_GAN <- gen_GAN(GAN_result, num_seq = 100)


    ### from preprocessing
    GAN_result2 <- fit_GAN(preprocessing = GAN_result$preprocessing,
                           intermediate_generator_layers = list(
                               layer_dense(units = 16),
                               layer_dense(units = 128)),
                           intermediate_discriminator_layers = list(
                               layer_dense(units = 128, activation = "relu"),
                               layer_dense(units = 16, activation = "relu")),
                           epochs = epochs,
                           batch_size = batch_size)
    gen_prot_GAN <- gen_GAN(GAN_result2, num_seq = 100)
}
```

`prot_seq_check` *Check a protein sequence*

Description

The protein sequence dataset is filtered by eliminating sequences containing the non-amino acid characters (digits and blank spaces) from the amino acid sequences. A valid amino acid sequence means a string that only contains capital letters of an alphabet and a hyphen for a gap.

Usage

```
prot_seq_check(prot_seq, label = NULL)
```

Arguments

prot_seq	amino acid sequences
label	label (default: NULL)

Value

valid sequences

Author(s)

Dongmin Jung

References

Mukhopadhyay, C. S., Choudhary, R. K., & Iquebal, M. A. (2017). Basic Applied Bioinformatics. John Wiley & Sons.

Examples

```
data("example_PTEN")
prot_seq_check(example_PTEN[1])
```

---

prot_vec *Converting from protein sequences to vectors or vice versa.*

Description

By using the word2vec model, amino acids are mapped to vectors of real numbers. Conceptually, it involves a mathematical embedding from a space with many dimensions per amino acid to a continuous vector space with a much lower dimension.

Usage

```
prot2vec(prot_seq, embedding_dim, embedding_matrix = NULL, ...)
vec2prot(prot_vec, embedding_matrix)
```

Arguments

prot_seq	protein sequences
prot_vec	protein embedding vectors
embedding_dim	dimension of embedding vectors
embedding_matrix	embedding matrix (default: NULL)
...	arguments for "word2vec::word2vec" but for dim, min_count and split

Value

prot_seq	protein sequences
prot_vec	protein embedding vectors
embedding_matrix	embedding matrix

Author(s)

Dongmin Jung

References

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. arXiv:1310.4546.

Chang, M. (2020). Artificial intelligence for drug development, precision medicine, and healthcare.

See Also

word2vec::word2vec, word2vec::word2vec_similarity

Examples

```
data("example_PTEN")
prot_seq <- example_PTEN[1:10]
prot2vec_result <- prot2vec(prot_seq = prot_seq, embedding_dim = 8)
vec2prot_result <- vec2prot(prot_vec = prot2vec_result$prot_vec,
                            embedding_matrix = prot2vec_result$embedding_matrix)
```

---

transformer *Transformer model*

Description

The Transformer architecture is a nonrecurrent architecture with a series of attention-based blocks. Each block is composed of a multi-head attention layer and a position-wise feedforward layer with an add and normalize layer in between. These layers process input sequences simultaneously, in parallel, independently of sequential order.

Usage

```
layer_embedding_token_position(x, maxlen, vocab_size, embed_dim)
layer_transformer_encoder(x, embed_dim, num_heads, ff_dim, num_transformer_blocks)
```

Arguments

x	layer object
maxlen	maximum of sequence size
vocab_size	vacabulary size
embed_dim	embedding size for each token
num_heads	number of attention heads
ff_dim	hidden layer size in feedforward network inside transformer
num_transformer_blocks	number of transformer blocks

Value

layer object

Author(s)

Dongmin Jung

References

Lappin, S. (2021). Deep learning and linguistic representation. CRC Press.

Liu, Z., Lin, Y., & Sun, M. (2020). Representation learning for natural language processing. Springer.

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    num_AA <- 20
    length_seq <- 10
    embedding_dim <- 16
    num_heads <- 2
    ff_dim <- 16
    num_transformer_blocks <- 2

    inputs <- layer_input(shape = length_seq)
    x <- inputs %>%
        layer_embedding_token_position(maxlen = length_seq,
                                       vocab_size = num_AA,
                                       embed_dim = embedding_dim) %>%
        layer_transformer_encoder(embed_dim = embedding_dim,
                                  num_heads = num_heads,
                                  ff_dim = ff_dim,
                                  num_transformer_blocks = num_transformer_blocks) %>%
        layer_global_average_pooling_1d()
}
```

VAE *Variational autoencoder for generating protein sequences*

Description

The variational autoencoder (VAE) is a class of autoencoder where the encoder module is used to learn the parameter of a distribution and the decoder is used to generate examples from samples drawn from the learned distribution. The conditional variational autoencoder (CVAE) is designed to generate desired samples by including additional conditioning information. Since there may be underlying distinctions between groups of samples, the Gaussian mixture model is used for sequence generation. The Word2vec is applied to amino acids for embedding. The VAE or CVAE model can be trained by the function "fit_VAE", and then the function "gen_VAE" generates protein sequences from the trained model.

Usage

```
fit_VAE(prot_seq,
        label = NULL,
        length_seq,
        embedding_dim,
        embedding_args = list(),
```

```
          latent_dim = 2,
          intermediate_encoder_layers,
          intermediate_decoder_layers,
          prot_seq_val = NULL,
          label_val = NULL,
          regularization = 1,
          epochs,
          batch_size,
          preprocessing = list(
              x_train = NULL,
              x_val = NULL,
              y_train = NULL,
              y_val = NULL,
              lenc = NULL,
              length_seq = NULL,
              embedding_dim = NULL,
              embedding_matrix = NULL,
              removed_prot_seq = NULL,
              removed_prot_seq_val = NULL),
          use_generator = FALSE,
          optimizer = "adam",
          validation_split = 0, ...)

gen_VAE(x,
          label = NULL,
          num_seq,
          remove_gap = TRUE,
          batch_size,
          use_generator = FALSE)
```

Arguments

prot_seq	aligned amino acid sequence
label	label (default: NULL)
length_seq	length of sequence
embedding_dim	dimension of the dense embedding
embedding_args	list of arguments for "word2vec::word2vec" but for dim, min_count and split
latent_dim	dimension of latent vector (default: 2)
intermediate_encoder_layers	list of intermediate layers for encoder, without input layer
intermediate_decoder_layers	list of intermediate layers for decoder, without output layer
regularization	regularization parameter, which is nonnegative (default: 1)
prot_seq_val	amino acid sequence for validation (default: NULL)
label_val	label for validation (default: NULL)
epochs	number of epochs
batch_size	batch size
preprocessing	list of preprocessed results, they are set to NULL as default x_train, length_seq, embedding_dim and embedding_matrix must be required for training

- x_train : embedded sequence data for train
- x_val : embedded sequence data for validation
- y_train : labels for train
- y_val : labels for validation
- lenc : encoded labels
- length_seq : length of sequence
- embedding_dim : dimension of the dense embedding
- embedding_matrix : embedding matrix
- removed_prot_seq : index for removed protein sequences while checking
- removed_prot_seq_val : index for removed protein sequences of validation

`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"

`x` result of the function "fit_VAE"

`num_seq` number of sequences to be generated

`remove_gap` remove gaps from sequences (default: TRUE)

Value

`model` trained VAE model

`encoder` trained encoder model

`decoder` trained decoder model

`preprocessing` preprocessed results

`gen_seq` generated sequence data

`label` labels for generated sequence data

`latent_vector` latent vector from embedded sequence data

Author(s)

Dongmin Jung

References

Cinelli, L. P., Marins, M. A., da Silva, E. A. B., & Netto, S. L. (2021). Variational Methods for Machine Learning with Applications to Deep Networks. Springer.

Liebowitz, J. (Ed.). (2020). Data Analytics and AI. CRC Press.

See Also

keras::fit, keras::compile, reticulate::array_reshape, mclust::mclustBIC, mclust::mclustModel, mclust::sim, DeepPINCS::multiple_sampling_generator, CatEncoders::LabelEncoder.fit, CatEncoders::transform, CatEncoders::inverse.transform

Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    data("example_luxA")
    label <- substr(example_luxA, 3, 3)

    # model parameters
    length_seq <- 360
    embedding_dim <- 8
    batch_size <- 128
    epochs <- 2

    # CVAE
    VAE_result <- fit_VAE(prot_seq = example_luxA,
                          label = label,
                          length_seq = length_seq,
                          embedding_dim = embedding_dim,
                          embedding_args = list(iter = 20),
                          intermediate_encoder_layers = list(layer_dense(units = 128),
                                                             layer_dense(units = 16)),
                          intermediate_decoder_layers = list(layer_dense(units = 16),
                                                             layer_dense(units = 128)),
                          prot_seq_val = example_luxA,
                          label_val = label,
                          epochs = epochs,
                          batch_size = batch_size,
                          use_generator = FALSE,
                          optimizer = keras::optimizer_adam(clipnorm = 0.1),
                          callbacks = keras::callback_early_stopping(
                              monitor = "val_loss",
                              patience = 10,
                              restore_best_weights = TRUE))
    gen_prot_VAE_I <- gen_VAE(VAE_result, label = rep("I", 100), num_seq = 100)
    gen_prot_VAE_L <- gen_VAE(VAE_result, label = rep("L", 100), num_seq = 100)


    ### from preprocessing
    VAE_result2 <- fit_VAE(intermediate_encoder_layers = list(layer_dense(units = 128),
                                                              layer_dense(units = 16)),
                           intermediate_decoder_layers = list(layer_dense(units = 16),
                                                              layer_dense(units = 128)),
                           epochs = epochs, batch_size = batch_size,
                           preprocessing = VAE_result$preprocessing,
                           use_generator = FALSE,
                           optimizer = keras::optimizer_adam(clipnorm = 0.1),
                           callbacks = keras::callback_early_stopping(
                               monitor = "val_loss",
                               patience = 10,
                               restore_best_weights = TRUE))
    gen_prot_VAE2_I <- gen_VAE(VAE_result2, label = rep("I", 100), num_seq = 100)
    gen_prot_VAE2_L <- gen_VAE(VAE_result2, label = rep("L", 100), num_seq = 100)
}
```

Index