Package 'nnlib2Rcpp'

May 8, 2020

Type Package

Title A Collection of Neural Networks

Version 0.1.2

Author Vasilis Nikolaidis [aut, cph, cre] (<https://orcid.org/0000-0003-1471-8788>)

Maintainer Vasilis Nikolaidis <vnikolaidis@us.uop.gr>

Description Another collection of neural networks. Includes versions of 'BP', 'Autoencoder', 'LVQ' (supervised and unsupervised), 'MAM'.

LazyData true

LinkingTo Rcpp

Imports Rcpp , methods

License MIT + file LICENSE

URL https://github.com/VNNikolaidis/nnlib2Rcpp

Encoding UTF-8

R topics documented:

`Autoencoder` *Autoencoder NN*

Description

A neural network for autoencoding data, projects data to a new set of variables.

Usage

```
Autoencoder(
  data_in,
  desired_new_dimension,
  number_of_training_epochs,
  learning_rate,
  num_hidden_layers = 1L,
  hidden_layer_size = 5L,
  show_nn = FALSE)
```

Arguments

data_in	data to be autoencoded, a numeric matrix, (2d, cases in rows, variables in columns). It is recommended to be in [0 1] range.
desired_new_dimension	number of new variables to be produced (effectively the size of the special hidden layer that outputs the new variable values, thus the dimension of the output vector space).
number_of_training_epochs	number of training epochs, aka presentations of all training data to ANN during training.
learning_rate	the learning rate parameter of the Back-Propagation (BP) NN.
num_hidden_layers	number of hidden layers on each side of the special layer.
hidden_layer_size	number of nodes (Processing Elements or PEs) in each hidden layer
show_nn	boolean, option to print an outline of the (trained) ANN internal structure.

Value

Returns a numeric matrix containing the projected data.

Note

This Autoencoder NN employs a BP-type NN to perform a data pre-processing step baring similarities to PCA since it too can be used for dimensionality reduction (Kramer 1991)(DeMers and Cottrell 1993)(Hinton and Salakhutdinov 2006). Unlike PCA, an autoencoding NN can also expand the feature-space dimensions (as feature expansion methods do). The NN maps input vectors to themselves via a special hidden layer (the coding layer, usually of different size than the input vector length) from which the new data vectors are produced. Note: The internal BP PEs in computing layers apply the logistic sigmoid threshold function, and their output is in [0 1] range. It is recommended to use this range in your data as well. More for this particular autoencoder implementation can be found in (Nikolaidis, Makris, and Stavroyiannis 2013). The method is not deterministic and the mappings may be non-linear, depending on the NN topology.

(This function uses Rcpp to employ 'bpu_autoencoder_nn' class in nnlib2.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

References

Nikolaidis V.N., Makris I.A, Stavroyiannis S, "ANS-based preprocessing of company performance indicators." Global Business and Economics Review 15.1 (2013): 49-58.

See Also

BP.

Examples

```
iris_s <- as.matrix(scale(iris[1:4]))
output_dim <- 2
epochs <- 100
learning_rate <- 0.73
num_hidden_layers <-2
hidden_layer_size <- 5

out_data <-  Autoencoder( iris_s, output_dim,
                          epochs, learning_rate,
                          num_hidden_layers, hidden_layer_size, FALSE)

plot( out_data,pch=21,
      bg=c("red","green3","blue")[unclass(iris$Species)],
      main="Randomly autoencoded Iris data")
```

BP-class *Class* "BP"

Description

Supervised Back-Propagation (BP) NN module, for encoding input-output mappings.

Extends

Class "RcppClass", directly.

All reference classes extend and inherit methods from "envRefClass".

Fields

`.CppObject:` Object of class `C++Object` ~~

`.CppClassDef:` Object of class `activeBindingFunction` ~~

`.CppGenerator:` Object of class `activeBindingFunction` ~~

Methods

`encode( data_in, data_out, learning_rate, training_epochs, hidden_layers, hidden_layer_size ):`
Setup a new BP NN and encode input-output data pairs. Parameters are:

- `data_in`: numeric matrix, containing input vectors as rows. . It is recommended that these values are in 0 to 1 range.
- `data_out`: numeric matrix, containing corresponding (desired) output vectors. It is recommended that these values are in 0 to 1 range.

- learning_rate: a number (preferably greater than 0 and less than 1) used in training.
- training_epochs: number of training epochs, aka single presentation iterations of all training data pairs to the NN during training.
- hidden_layers: number of hidden layers to be created between input and output layers.
- hidden_layer_size: number of nodes (processing elements or PEs) in the hidden layers (all hidden layers are of the same length in this model).

Note: to encode additional input-output vector pairs in an existing BP, use train_single or train_multiple methods (see below).

recall(data_in): Get output for a dataset (numeric matrix data_in) from the (trained) BP NN.

setup(input_dim, output_dim, learning_rate, hidden_layers, hidden_layer_size): Setup the BP NN so it can be trained and used. Note: this is not needed if using encode. Parameters are:

- input_dim: integer length of input vectors.
- output_dim: integer length of output vectors.
- learning_rate: a number (preferably greater than 0 and less than 1) used in training.
- hidden_layers: number of hidden layers to be created between input and output layers.
- hidden_layer_size: number of nodes (processing elements or PEs) in the hidden layers (all hidden layers are of the same length in this model).

train_single (data_in, data_out): Encode an input-output vector pair in the BP NN. Only performs a single training iteration (multiple may be required for proper encoding). Vector sizes should be compatible to the current NN (as resulted from the encode or setup methods). Returns error level indicator value.

train_multiple (data_in, data_out, training_epochs): Encode multiple input-output vector pairs stored in corresponding datasets. Performs multiple iterations in epochs (see encode). Vector sizes should be compatible to the current NN (as resulted from the encode or setup methods). Returns error level indicator value.

print(): print NN structure.

load(filename): retrieve the NN from specified file.

save(filename): save the NN to specified file.

The following methods are inherited (from the corresponding class): objectPointer ("RcppClass"), initialize ("RcppClass"), show ("RcppClass")

Note

This R module contains an internal Back-Propagation (BP) multilayer perceptron NN (described in Simpson (1991) as the vanilla back-propagation algorithm), which can be used to store input-output vector pairs. Since the nodes (PEs) in computing layers of this BP implementation apply the logistic sigmoid threshold function, their output is in [0 1] range (and so should the desired output vector values).

(This object uses Rcpp to employ 'bp_nn' class in nnlib2.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

References

Simpson, P. K. (1991). Artificial neural systems: Foundations, paradigms, applications, and implementations. New York: Pergamon Press.

See Also

Autoencoder.

Examples

```
# create some data...
iris_s                 <- as.matrix(scale(iris[1:4]))

# use a randomply picked subset of (scaled) iris data for training
training_cases         <- sample(1:nrow(iris_s), nrow(iris_s)/2,replace=FALSE)
train_set              <- iris_s[training_cases,]
train_class_ids        <- as.integer(iris$Species[training_cases])
train_num_cases        <- nrow(train_set)
train_num_variables    <- ncol(train_set)
train_num_classes      <- max(train_class_ids)

# create output dataset to be used for training, Here we encode class as 0s and 1s
train_set_data_out <- matrix(
          data = 0,
          nrow = train_num_cases,
          ncol = train_num_classes)

# now for each case, assign a 1 to the column corresponding to its class, 0 otherwise
# (there must be a better R way to do this)
for(r in 1:train_num_cases) train_set_data_out[r,train_class_ids[r]]=1

# done with data, let's use BP...
bp<-new("BP")

bp$encode(train_set,train_set_data_out,0.8,10000,2,4)

# let's test by recalling the original training set...
bp_output <- bp$recall(train_set)

cat("- Using this demo's encoding, recalled class is:\n")
print(apply(bp_output,1,which.max))
cat("- BP success in recalling correct class is: ",
  sum(apply(bp_output,1,which.max)==train_class_ids)," out of ",
  train_num_cases, "\n")

# Let's see how well it recalls the entire Iris set:
bp_output <- bp$recall(iris_s)

# show output
cat("\n- Recalling entire Iris set returns:\n")
print(bp_output)
cat("- Using this demo's encoding, original class is:\n")
print(as.integer(iris$Species))
cat("- Using this demo's encoding, recalled class is:\n")
bp_classification <- apply(bp_output,1,which.max)
print(bp_classification)
cat("- BP success in recalling correct class is: ",
  sum(apply(bp_output,1,which.max)==as.integer(iris$Species)),
  "out of ", nrow(iris_s), "\n")
plot(iris_s, pch=bp_classification, main="Iris classified by a partialy trained BP (module)")
```

LVQs-class *Class* "LVQs"

Description

Supervised Learning Vector Quantization (LVQ) NN module, for data classification.

Extends

Class "RcppClass", directly.

All reference classes extend and inherit methods from "envRefClass".

Fields

`.CppObject:` Object of class `C++Object` ~~

`.CppClassDef:` Object of class `activeBindingFunction` ~~

`.CppGenerator:` Object of class `activeBindingFunction` ~~

Methods

`encode(data, desired_class_ids, training_epochs):` Encode input and output (classification) for a dataset using LVQ NN. Parameters are:

- `data`: training data, a numeric matrix, (2d, cases in rows, variables in columns). Data should be in 0 to 1 range.
- `desired_class_ids` : vector of integers containing a desired class id for each training data case (row). Should contain integers in 0 to n-1 range, where n is the number of classes.
- `training_epochs`: integer, number of training epochs, aka presentations of all training data to the NN during training.

`recall(data_in):` Get output (classification) for a dataset (numeric matrix `data_in`) from the (trained) LVQ NN. The `data_in` dataset should be 2-d containing data cases (rows) to be presented to the NN and is expected to have same number or columns as the original training data. Returns a vector of integers containing a class id for each case (row).

`print():` print NN structure.

`load(filename):` retrieve the NN from specified file.

`save(filename):` save the NN to specified file.

The following methods are inherited (from the corresponding class): objectPointer ("RcppClass"), initialize ("RcppClass"), show ("RcppClass")

Note

The NN used in this module uses supervised training for data classification (described as Supervised Learning LVQ in Simpson (1991)). Data should be scaled in 0 to 1 range.

(This module uses Rcpp to employ 'lvq_nn' class in nnlib2.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

References

Simpson, P. K. (1991). Artificial neural systems: Foundations, paradigms, applications, and implementations. New York: Pergamon Press.

See Also

LVQu (unsupervised LVQ function).

Examples

```
# LVQ expects data in 0 to 1 range, so scale some numeric data...
iris_s<-as.matrix(iris[1:4])
c_min<-apply(iris_s,2,FUN = "min")
c_max<-apply(iris_s,2,FUN = "max")
c_rng<-c_max-c_min
iris_s<-sweep(iris_s,2,FUN="-",c_min)
iris_s<-sweep(iris_s,2,FUN="/",c_rng)

# create a vector of desired class ids (starting from 0):
iris_desired_class_ids<-as.integer(iris$Species)-1;

# now create the NN:
lvq<-new("LVQs")

# and train it:
lvq$encode(iris_s,iris_desired_class_ids,100)

# recall the same data (a simple check of how well the LVQ was trained):
lvq_recalled_class_ids<-lvq$recall(iris_s);
plot(iris_s, pch=lvq_recalled_class_ids, main="LVQ recalled clusters (module)")
```

LVQu *Unsupervised LVQ*

Description

Unsupervised (clustering) Learning Vector Quantization (LVQ) NN.

Usage

```
LVQu(
  data,
  max_number_of_desired_clusters,
  number_of_training_epochs,
  neighborhood_size,
  show_nn )
```

Arguments

data	data to be clustered, a numeric matrix, (2d, cases in rows, variables in columns). Data should be in 0 to 1 range.
max_number_of_desired_clusters	clusters to be produced (at most)

number_of_training_epochs
: number of training epochs, aka presentations of all training data to ANN during training.

neighborhood_size
: integer >=1, specifies affected neighbor output nodes during training. if 1 (Single Winner) the ANN is somewhat similar to k-means.

show_nn
: boolean, option to print an outline of the (trained) ANN internal structure.

Value

Returns a vector of integers containing a cluster id for each data case (row).

Note

Function LVQu employs an unsupervised LVQ for clustering data (Kohonen 1988). This LVQ variant is described as Unsupervised Learning LVQ in Simpson (1991) and is a simplified 1-D version of Self-Organizing-Map (SOM). Its parameter `neighborhood_size` controls the encoding mode (where `neighborhood_size=1` is Single-Winner Unsupervised encoding, similar to k-means, while an odd valued `neighborhood_size > 1` means Multiple-Winner Unsupervised encoding mode). Initial weights are random (uniform distribution) in 0 to 1 range. As these weights represent cluster center coordinates (the class reference vector), it is important that input data is also scaled to this range.

(This function uses Rcpp to employ 'som_nn' class in nnlib2.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

References

Kohonen, T (1988). Self-Organization and Associative Memory, Springer-Verlag.; Simpson, P. K. (1991). Artificial neural systems: Foundations, paradigms, applications, and implementations. New York: Pergamon Press.

Philippidis, TP & Nikolaidis, VN & Kolaxis, JG. (1999). Unsupervised pattern recognition techniques for the prediction of composite failure. Journal of acoustic emission. 17. 69-81.

See Also

LVQs (supervised LVQ module),

Examples

```
# LVQ expects data in 0 to 1 range, so scale...
iris_s<-as.matrix(iris[1:4])
c_min<-apply(iris_s,2,FUN = "min")
c_max<-apply(iris_s,2,FUN = "max")
c_rng<-c_max-c_min
iris_s<-sweep(iris_s,2,FUN="-",c_min)
iris_s<-sweep(iris_s,2,FUN="/",c_rng)

cluster_ids<-LVQu(iris_s,5,100)
plot(iris_s, pch=cluster_ids, main="LVQ-clustered Iris data")
```

MAM-class *Class "MAM"*

Description

A single Matrix Associative Memory (MAM) implemented as a (supervised) NN.

Extends

Class "RcppClass", directly.

All reference classes extend and inherit methods from "envRefClass".

Fields

`.CppObject:` Object of class `C++Object` ~~

`.CppClassDef:` Object of class `activeBindingFunction` ~~

`.CppGenerator:` Object of class `activeBindingFunction` ~~

Methods

`encode( data_in, data_out ):` Setup a new MAM NN and encode input-output data pairs. Parameters are:

- `data_in`: numeric matrix, input data to be encoded in MAM, a numeric matrix (2d, of n rows). Each row will be paired to the corresponding data_out row, forming an input-output vector pair.
- `data_out`: numeric matrix, output data to be encoded in MAM, a numeric matrix (2d, also of n rows). Each row will be paired to the corresponding data_in row, forming an input-output vector pair.

Note: to encode additional input-output vector pairs in an existing MAM, use `train_single` method (see below).

`recall(data):` Get output for a dataset (numeric matrix `data`) from the (trained) MAM NN.

`train_single (data_in, data_out):` Encode an input-output vector pair in the MAM NN. Vector sizes should be compatible to the current NN (as resulted from the `encode` method).

`print():` print NN structure.

`load(filename):` retrieve the NN from specified file.

`save(filename):` save the NN to specified file.

The following methods are inherited (from the corresponding class): objectPointer ("RcppClass"), initialize ("RcppClass"), show ("RcppClass")

Note

The NN in this module uses supervised training to store input-output vector pairs.

(This function uses Rcpp to employ 'mam_nn' class in nnlib2.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

References

Pao Y (1989). Adaptive Pattern Recognition and Neural Networks. Reading, MA (US); Addison-Wesley Publishing Co., Inc.

See Also

BP, LVQs.

Examples

```
iris_s            <- as.matrix(scale(iris[1:4]))
class_ids         <- as.integer(iris$Species)
num_classes       <- max(class_ids)

# create output dataset to be used for training, Here we encode class as -1s and 1s
iris_data_out <- matrix( data = -1, nrow = nrow(iris_s), ncol = num_classes)

# now for each case, assign a 1 to the column corresponding to its class
for(r in 1:nrow(iris_data_out)) iris_data_out[r,class_ids[r]]=1

# Finally apply MAM:
# Encode train pairs in MAM and then get output dataset by recalling the test data.

mam <- new("MAM")

mam$encode(iris_s,iris_data_out)

# test the encoding by recalling the original input data...
mam_data_out <- mam$recall(iris_s)

# find which MAM output has the largest value and use this as the final cluster tag.
mam_recalled_cluster_ids = apply(mam_data_out,1,which.max)

plot(iris_s, pch=mam_recalled_cluster_ids, main="MAM recalled Iris data classes")

cat("MAM recalled these IDs:\n")
print(mam_recalled_cluster_ids)
```

---

nn-class *Neural Network Classes*

Description

This package uses 'Rcpp' to create modules and functions that encapsulate NNs created using 'nnlib2' (a collection of C++ classes for creating NNs).

Note

Due to their relative simplicity MAM and the corresponding 'mam_nn' C++ class are a good starting example of how to implement a new NN using nnlib2 (source code is available on GitHub).

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

See Also

modules BP, MAM, LVQs, functions Autoencoder, LVQu.

Index