Package 'nnlib2Rcpp'

March 19, 2020

Type Package

Title Collection of Artificial Neural Networks

Version 0.1.0

Author Vasilis N. Nikolaidis

Maintainer Nikolaidis Vasilis <vnnikolaidis@gmail.com>

Description A collection of artificial neural networks. Currently includes versions of BP, Autoencoder, MAM, LVQ, SOM. All NNs are created using nnlib2 C++ NN library and accessed via RCpp.

LazyData true

LinkingTo Rcpp

Imports Rcpp , methods

License MIT + file LICENSE

RcppModules

RoxygenNote 7.0.2

Encoding UTF-8

R topics documented:

`Autoencoder` *Autoencoder NN*

Description

Artificial 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`	Back-Propagation ANN learning rate.
`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

Autoencoder employs a BP-based 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 C++ Artificial Neural Network library.)

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.

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, TRUE)

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

`BP_create` *Create a Back-Propagation (BP) ANN*

Description

Create a Back-Propagation (BP) ANN so it can be trained and used. Once created, this will be the current active BP ANN.

Usage

```
BP_create(input_data_dim, output_dim, learning_rate, num_hidden_layers, hidden_layer_size)
```

Arguments

`input_data_dim`	Length of input vectors (thus dimension of input vector space and size of input layer)
`output_dim`	Length of output vectors (thus dimension of output vector space and size of output layer)
`learning_rate`	a number (preferably greater than 0 and less than 1) used in training.
`num_hidden_layers`	number of hidden layers to be created between input and output layers.

`hidden_layer_size`
: number of nodes in the hidden layers (all hidden layers are of the same length in this model).

Details

This is one of several functions used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Value

boolean, TRUE if BP is succesfully created.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

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

`BP_create`, `BP_train_single`, `BP_train_set`, `BP_recall_set`, `BP_print`, `BP_save_to_file`, `BP_load_from_file`, `BP_destroy`, `BP_NN`.

Examples

```
BP_create(4,2,0.89,1,4);
# ...train and use the BP Neural Network (not shown here).
BP_destroy()
```

`BP_destroy` *Deletes the active Back-Propagation ANN*

Description

Deletes a Back-Propagation ANN created via `BP_create` or `BP_load_from_file`.

Usage

```
BP_destroy()
```

Details

This is one of several functions used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

See Also

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy, BP_NN.

Examples

```
BP_create(4,2,0.89,1,4);
# ...train and use the BP Neural Network (not shown here).
BP_destroy()
```

BP_load_from_file *Retrieve a BP ANN from file.*

Description

Loads a BP ANN (created via BP_save_to_file) from specified file, replacing the currently active BP (if any).

Usage

```
BP_load_from_file(filename)
```

Arguments

filename	name of file containing the BP ANN (created via BP_save_to_file).

Details

This is one of several functions used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Value

boolean, TRUE if BP is succesfully loaded.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

See Also

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy, BP_NN.

Examples

```
BP_create(4,2,0.89,1,4);
# ...train and use the BP Neural Network (not shown here).
BP_save_to_file("TEST")
BP_load_from_file("TEST")
BP_destroy()
# remove the TEST file if you dont need it.
file.remove("TEST")
```

BP_NN-class *Class* "BP_NN Backpropagation NN"

Description

Back-Propagation NN as a module. This is an alternative (and probably better) functionaly equivalent way to employ a BP supervised NN without using functions (such as BP_create, BP_train_single, BP_train_set, BP_destroy etc.)

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( ... )`: Setup BP and encode input-output datasets in the NN. Parameters are: NumericMatrix tr_dataset_in, NumericMatrix tr_dataset_out, double learning_rate, int training_epochs, int hidden_layers, int hidden_layer_size

`recall(data_in)`: Get output for a dataset (NumericMatrix data_in) using BP NN.

`train_single (...)`: Encode a single input-output vector pair in current BP NN.

`print()`: prints NN structure.

`load(filename)`: retrieves the NN stored in file.

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

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

Note

The R object employs a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs. Since the 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 C++ Artificial Neural Network library.)

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

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy.

Examples

```
showClass("BP_NN")

# 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_NN")

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)")
```

`BP_print` *Print the current active BP ANN*

Description

Prints the current active BP ANN (if any).

Usage

```
BP_print()
```

Details

Prints the current active BP ANN created via `BP_create` or `BP_load_from_file`. This is one of several functions used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

See Also

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy.

Examples

```
BP_create(4,2,0.89,1,4);
BP_print()
BP_destroy()
```

BP_recall_set *Apply a (trained) BP ANN to data*

Description

Present the data as input to a (trained) BP ANN and produce corresponding output.

Usage

```
BP_recall_set(data_in)
```

Arguments

data_in	numeric matrix, containing input vectors as rows (for operation to succeed, they should have length equal to the size of the input layer in the current active BP).

Details

This is the main function for using the active BP once it is trained, as it accepts an input dataset and produces corresponding output. It is one of several functions used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Value

numeric matrix, containing corresponding vectors returned from the BP ANS.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

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

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy, BP_NN.

Examples

```
# prepare input-output data sets to be used for training
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)

# for each case, assign 1 to the column corresponding to its class, keep 0 otherwise
# (note: there must be a better R syntax way to do this)

for(r in 1:train_num_cases) train_set_data_out[r,train_class_ids[r]]=1

# finally create the BP Neural Network

BP_create(
train_num_variables,
train_num_classes,
learning_rate = 0.83,
num_hidden_layers = 1,
hidden_layer_size = 5 )

# ...train the BP Neural Network

BP_train_set(
```

```
train_set,
train_set_data_out,
training_epochs = 5000 );

# ...and later use the trained BP Neural Network

# BP has been trained. It can be used to recall input-output pairs.
# Let's see how well it recalls the data set it was trained with:

bp_output <- BP_recall_set(train_set)

# show output
cat("\n- Recalling training set returns:\n")
print(bp_output)
cat("- Using this demo's encoding, original class is:\n")
print(train_class_ids)
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_set(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")

# ...when no longer needed, delete the BP Neural Network

BP_destroy()
```

`BP_save_to_file` *Save the current active BP ANN to file.*

Description

Saves the current state of the active BP ANN (created via `BP_create` or `BP_load_from_file`) to a specified file.

Usage

```
BP_save_to_file(filename)
```

Arguments

filename	name of file to be created.

Details

This is one of several functions used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Value

boolean, TRUE if BP is succesfully saved.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

See Also

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy, BP_NN.

Examples

```
BP_create(4,2,0.89,1,4);
# ...train and use the BP Neural Network (not shown here).
BP_save_to_file("TEST")
BP_load_from_file("TEST")
BP_destroy()
# remove the TEST file if you dont need it.
file.remove("TEST")
```

BP_train_set *Train the active BP*

Description

Presents an entire dataset (and corresponding outputs) to the active BO for training.

Usage

```
BP_train_set(dataset_in, dataset_out, training_epochs)
```

Arguments

dataset_in	numeric matrix, containing input vectors as rows (for training to succeed, they should have length equal to the size of the input layer in the current active BP).
dataset_out	numeric matrix, containing corresponding (desired) output vectors as rows (for training to succeed, they should have length equal to the size of the output layer in the current active BP). It is recommended that these values fall in range 0 to 1.
training_epochs	number of training epochs, aka presentations of all training data to ANN during training.

Details

This is the main function for training the active BP, as it accepts entire input-output datasets. Since the 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 function is one of several used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Value

Returns a real number, indicator of training error.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

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

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy, BP_NN.

Examples

```
# prepare input-output data sets to be used for training
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)

# for each case, assign 1 to the column corresponding to its class, keep 0 otherwise
# (note: there must be a better R syntax way to do this)

for(r in 1:train_num_cases) train_set_data_out[r,train_class_ids[r]]=1

# finally create the BP Neural Network

BP_create(
train_num_variables,
train_num_classes,
learning_rate = 0.83,
num_hidden_layers = 1,
hidden_layer_size = 5 )

# ...train the BP Neural Network

BP_train_set(
train_set,
train_set_data_out,
training_epochs = 5000 );

# ...and later use the trained BP Neural Network

# BP has been trained. It can be used to recall input-output pairs.
# Let's see how well it recalls the data set it was trained with:

bp_output <- BP_recall_set(train_set)

# show output
cat("\n- Recalling training set returns:\n")
print(bp_output)
cat("- Using this demo's encoding, original class is:\n")
print(train_class_ids)
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_set(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")

# ...when no longer needed, delete the BP Neural Network

BP_destroy()
```

`BP_train_single` *Train the active BP with an input-output vector pair.*

Description

Trains the active BP with by presenting the given input-output vector pair (data_in and data_out) once, i.e. performs a single iteration (note that several such iterations are usually needed for training to be sufficient).

Usage

```
BP_train_single(data_in, data_out)
```

Arguments

`data_in`	a numeric vector containing input data (for training to succeed, it should have length equal to the size of the input layer in the current active BP). It is recommended that these values fall in 0 to 1 range.
`data_out`	a numeric vector containing output data (for training to succeed, it should have length equal to the size of the output layer in the current active BP)

Details

Since the 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 function is one of several used to handle BP supervised ANNs in this package. A BP NN object must be created by invoking the BP_create (or BP_load_from_file) function before it can be used . Any further calls of BP_create would replace the current BP NN by a new one. The BP NN object should also be deleted when no longer needed by invoking the BP_destroy function.

Value

Returns a real number, indicator of training error.

Note

These functions employ a Back-Propagation (BP) multilayer perceptron NN described in Simpson (1991) as the vanilla back-propagation algorithm, to store input-output vector pairs.

(This function uses Rcpp to employ bp_nn class in nnlib2 C++ Artificial Neural Network library.)

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

BP_create, BP_train_single, BP_train_set, BP_recall_set, BP_print, BP_save_to_file, BP_load_from_file, BP_destroy, BP_NN.

Examples

```
BP_create(4,2,0.89,1,4);
# ...train and use the BP Neural Network (not shown here).
in1 = c(.1,0,.4,.5)
ou1 = c(.3,.4)
BP_train_single(in1,ou1)
BP_destroy()
```

LVQ *LVQ*

Description

Learning Vector Quantization, supervised-training Artificial Neural Network (ANN)

Usage

```
LVQ(data, desired_cluster_ids, number_of_training_epochs, test_data, show_nn = FALSE)
```

Arguments

data	training data, a numeric matrix, (2d, cases in rows, variables in columns). Data should be in 0 to 1 range.
desired_cluster_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.
number_of_training_epochs	number of training epochs, aka presentations of all training data to ANN during training.
test_data	numeric matrix (2d) containing data to be recalled from the trained ANN. Should have same number or columns as the training data.
show_nn	boolean, option to print an outline of the (trained) ANN internal structure.

Value

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

Note

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

(This function uses Rcpp to employ lvq_nn class in nnlib2 C++ Artificial Neural Network library.)

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.

See Also

LVQ_NN

Examples

```
# LVQ and SOM expect 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)

# use cluster indexing starting from 0.
iris_ids<-as.integer(iris$Species)-1;

# now run LVQ:
returned_ids<-LVQ(iris_s,iris_ids,100,iris_s)
print(returned_ids)
plot(iris_s, pch=returned_ids, main="LVQ recalled clusters")
```

LVQ_NN-class *Class* "LVQ_NN"

Description

Learning Vector Quantization NN as a module. This is an alternative (and probably more versatile) functionaly equivalent way to employ an LVQ supervised NN without using the function LVQ.

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

`recall(data_in):` Get output (classification) for a dataset (NumericMatrix data_in) using LVQ NN.

`print():` prints NN structure.

`load(filename):` retrieves the NN stored in file.

`save(filename):` saves the status of 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 function 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 C++ Artificial Neural Network library.)

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

LVQ

Examples

```
showClass("LVQ_NN")

# LVQ and SOM expect data in 0 to 1 range, so scale some 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)

iris_desired_cluster_ids<-as.integer(iris$Species)-1; # use cluster indexing starting from 0.

# now create and use the NN
lvq<-new("LVQ_NN")
```

```
lvq$encode(iris_s,iris_desired_cluster_ids,100)

# just recalling the same data to check how well LVQ was trained...
lvq_recalled_cluster_ids<-lvq$recall(iris_s);
plot(iris_s, pch=lvq_recalled_cluster_ids, main="LVQ recalled clusters (module)")
```

MAM *Matrix Associative Memory*

Description

A single Matrix Associative Memory implemented as a (supervised) Artificial Neural Network.

Usage

```
MAM(train_data_in, train_data_out, test_data_in, show_nn)
```

Arguments

train_data_in	data to be encoded in MAM, a numeric matrix (2d, of n rows). Each row will be paired to the corresponding train_data_out row, forming an input-output vector pair.
train_data_out	data to be encoded in MAM, a numeric matrix (2d, also of n rows). Each row will be paired to the corresponding train_data_in row, forming an input-output vector pair.
test_data_in	data to be recalled from MAM, a numeric matrix (2d, same number of columns as train_data_in).
show_nn	boolean, option to print an outline of the (trained) ANN internal structure.

Value

Returns recalled data from MAM, a numeric matrix (2d, same number of rows as test_data_in, same number of columns as train_data_out).

Note

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

(This function uses Rcpp to employ mam_nn class in nnlib2 C++ Artificial Neural Network library.)

Author(s)

Vasilis N. Nikolaidis <vnnikolaidis@gmail.com>

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.

test_data_out <- MAM (iris_s,iris_data_out,iris_s,TRUE)

# find which MAM output has the largest value and use this as the final cluster tag.
mam_recalled_cluster_ids = apply(test_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)
```

SOM *Self-Organizing Map*

Description

Unsupervised clustering ANN, based on the Self-Organizing Map (SOM) introduced by Kohonen.

Usage

```
SOM(
  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 SOM employs a simplified 1-D version of the Self-Organizing-Map NN for clustering data (Kohonen 1988). This SOM variant is described as Unsupervised Learning LVQ in Simpson (1991). 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. Since 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 C++ Artificial Neural Network library.)

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.

Examples

```
# LVQ and SOM expect 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)

som_cluster_ids<-SOM(iris_s,5,100)
plot(iris_s, pch=som_cluster_ids, main="SOM clustered Iris data")
```

Index