Package 'darch'

July 2, 2014

Type Package

Title Package for deep architectures and Restricted-Bolzmann-Machines

Version 0.9.1

Date 2013-04-17

Author Martin Drees

Maintainer Martin Drees <mdrees@stud.fh-dortmund.de>

Description The darch package is build on the basis of the code from G. E. Hinton and R. R. Salakhutdinov (available under Matlab Code for deep belief nets : last visit: 01.08.2013). This package is for generating neural networks with many layers (deep architectures) and train them with the method introduced by the publications ``A fast learning algorithm for deep belief nets'' (G. E. Hinton, S. Osindero, Y. W. Teh) and ``Reducing the dimensionality of data with neural networks'' (G. E. Hinton, R. R. Salakhutdinov). This method includes a pre training with the contrastive divergence method publishing by G.E Hinton (2002) and a fine tuning with common known training algorithms like backpropagation or conjugate gradient.

License GPL-2

URL http://github.com/maddin79/darch

Imports futile.logger, ff, methods

Collate 'net.R' 'darch.R' 'backpropagation.R' 'darch.Add.R' 'darch.Getter.R' 'darch.Learn.R' 'darch.Remove.R' 'darch.Reset.R' 'darch.Setter.R' 'darchUnitFunctions.R' 'errorFunctions.R' 'rbm.R' 'generateRBMs.R' 'generateWeights.R' 'loadDArch.R' 'loadRBM.R' 'loadRBMFFWeights.R' 'makeStartEndPoints.R' 'minimize.R' 'minimizeAutoencoder.R' 'minimizeClassifier.R' 'net.Getter.R' 'net.Setter.R' 'newDArch.R' 'rbm.Setter.R' 'rbm.Reset.R' 'newRBM.R' 'package-darch.R' 'rbm.Getter.R' 'rbm.Learn.R' 'rbmUnitFunctions.R' 'rbmUpdate.R' 'readMNIST.R' 'rpropagation.R' 'runDArch.R' 'saveDArch.R' 'saveRBM.R' 'saveRBMFFWeights.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2014-03-16 13:34:52

R topics documented:

`addExecOutput` *Adds an execution output for a DArch object*

Description

This method can be used to save the execution outputs of every layer for the DArch object. The outputs are saved in a list and every time this function is called, the list is extended of one field with the new output.

Usage

```
addExecOutput(darch, output)

## S4 method for signature 'DArch'
addExecOutput(darch, output)
```

Arguments

darch	An instance of the class DArch.
output	The output of the layer.

See Also

DArch

---

addLayer *Adds a layer to the DArch object*

Description

Adds a layer to the given DArch object. The parameter weights and biases will be put together in one matrix.

Usage

```
addLayer(darch, weights, biases, unitFunction)

## S4 method for signature 'DArch'
addLayer(darch, weights, biases, unitFunction)
```

Arguments

darch	An instance of the class DArch.
weights	The weights for the layer.
biases	The biases for the layer.
unitFunction	The functions of the units in the layer.

See Also

DArch, sigmoidUnit, binSigmoidUnit, sigmoidUnitDerivative, linearUnit, linearUnitDerivative, softmaxUnit, softmaxUnitDerivative,

addLayerField *Adds a field to a layer*

Description

Adds a field to the layer given by the index.

Usage

```
addLayerField(darch, index, field)

## S4 method for signature 'DArch'
addLayerField(darch, index, field)
```

Arguments

darch	An instance of the class DArch.
index	The position of the layer.
field	The new field for the layer.

See Also

DArch

backpropagation *Backpropagation learning function*

Description

This function provides the backpropagtion algorithm for deep architectures.

Usage

```
backpropagation(darch,trainData,targetData,epoch)
```

Arguments

darch	An instance of the class DArch.
trainData	The data for training.
targetData	The targets for the data
epoch	Number of epochs for the training

Details

The function is getting the learning parameters from the provided DArch object. It uses the attributes momentum, finalMomentum and momentumSwitch for the calculation of the new weights with momentum. The parameter epoch is provided for the change from momentum to finalMomentum and is compared to momentumSwitch. The attributes learnRateWeights and learnRateBiases will be used for updating the weights. To use the backpropagation function as the fine tuning function the layer functions of the darch DArch object must set to the versions which calculates also the derivatives of the function result.

Value

The trained deep architecture

References

Rumelhart, D., G. E. Hinton, R. J. Williams, Learning representations by backpropagating errors, Nature 323, S. 533-536, DOI: 10.1038/323533a0, 1986.

See Also

DArch rpropagation minimizeAutoencoder minimizeClassifier minimizeClassifier

---

binSigmoidUnit *Binary sigmoid unit function.*

Description

The function calculates the activation and the output from the sigmoid transfer function. It returns a binary matrix where a entry is 1 if the value is bigger than a random number generated with runif.

Usage

```
binSigmoidUnit(data,weights)
```

Arguments

data	The data matrix for the calculation
weights	The weight and bias matrix for the calculation

Value

A list with the binary activation of the unit in the first entry.

See Also

DArch, sigmoidUnit, sigmoidUnitDerivative, linearUnit, linearUnitDerivative, softmaxUnit, softmaxUnitDerivative

crossEntropyError *Cross entropy error function*

Description

The function calculates the cross entropy error from the `original` and `estimate` parameters.

Usage

```
crossEntropyError(original, estimate)
```

Arguments

`original`	The original data matrix
`estimate`	The calculated data matrix

Value

A list with the name of the error function in the first entry and the error value in the second entry

See Also

`quadraticError`, `mseError`

DArch *Class for deep architectures*

Description

This class implements deep architectures and provides the ability to train them with a pre training using contrastive divergence and fine tuning with backpropagation, resilient backpropagation and conjugate gradients.

Details

The class is inherits all attributes from the class `link{Net}`. When creating a new instance with the constructor `newDArch` (recommended), the darch-object contained the number of layers -1 restricted bolzmann machines (`RBM`), which are used for the unsupervised pre training of the network. The `RBM`s are saved in the attribute `rbmList` and can be fetched over the getter method (`getRBMList`. The two attributes `fineTuneFunction` and `executeFunction` containing the functions for the fine tuning (default: `backpropagation`) and for the execution (default: `runDArch`. The training of the network is performed by the two learning functions `preTrainDArch` and `fineTuneDArch`. The first function trains the network with the unsupervised method contrastive divergence. The second function used the function in the attribute `fineTuneFunction` for the fine tuning. After an execution of the network, the outputs of every layer can be found in the attribute `executeOutput`.

Slots

`rbmList`: A list which contains all rbm's for the pre-training.

`layers`: A list with the layer information. In the first field are the weights and in the second field is the unit function.

`learnRateBiases`: The learning rate for the bias weights.

`fineTuneFunction`: Contains the function for the fine tuning.

`executeFunction`: Contains the function for executing the network.

`executeOutput`: A list which contains the outputs of every layer after an execution of the network.

`cancel`: Boolean value which indicates if the network training is canceled.

`executeOutput`: A string containing the message why the network training is stopped.

`cancel`: Indicates if the execution is canceled.

`cancelMessage`: The message when the execution is canceled.

Author(s)

Martin Drees

See Also

`Net`, `RBM`

---

darch *Deep architectures in R*

Description

The darch-package implements Deep-Architecture-Networks and Restricted-Bolzmann-Machines.

Details

The creation of this package is motivated by the papers from G. Hinton et. al. from 2006 (see references for details) and from the matlab source code developed in this context. This package provides the possibility to generate deep architecture networks (darch) like the deep belief networks from Hinton et. al.. The deep architectures can then be trained with the contrastive divergence method. After this pre-training it can be fine tuned with several lerning methods like backpropagation, resilient backpropagtion and conjugate gradients.

Package:	darch
Type:	Package
Version:	0.9.1
Date:	2013-04-17
License:	GPL-2
LazyLoad:	yes

Author(s)

Martin Drees <mdrees@stud.fh-dortmund.de>

Maintainer: Martin Drees <mdrees@stud.fh-dortmund.de>

References

Hinton, G. E., S. Osindero, Y. W. Teh, A fast learning algorithm for deep belief nets, Neural Computation 18(7), S. 1527-1554, DOI: 10.1162/neco.2006.18.7.1527 2006.

Hinton, G. E., R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313(5786), S. 504-507, DOI: 10.1126/science.1127647, 2006.

Examples

```
## Not run:
# Generating the datasets
inputs <- matrix(c(0,0,0,1,1,0,1,1),ncol=2,byrow=TRUE)
outputs <- matrix(c(0,1,1,0),nrow=4)

# Generating the darch
darch <- newDArch(c(2,4,1),batchSize=2)

# Pre-Train the darch
darch <- preTrainDArch(darch,inputs,maxEpoch=1000)

# Prepare the layers for backpropagation training for
# backpropagation training the layer functions must be
# set to the unit functions which calculates the also
# derivatives of the function result.
layers <- getLayers(darch)
for(i in length(layers):1){
  layers[[i]][[2]] <- sigmoidUnitDerivative
}
setLayers(darch) <- layers
rm(layers)

# Setting and running the Fine-Tune function
setFineTuneFunction(darch) <- backpropagation
darch <- fineTuneDArch(darch,inputs,outputs,maxEpoch=1000)

# Running the darch
darch <- darch <- getExecuteFunction(darch)(darch,inputs)
outputs <- getExecOutputs(darch)
cat(outputs[[length(outputs)]])

## End(Not run)
```

fineTuneDArch *Fine tuning function for the deep architecture.*

Description

The fine tuning function for deep architectures. This function use the function saved in the attribute `fineTuneFunction` to train the deep architecture.

Usage

```
fineTuneDArch(darch, trainData, targetData, ..., maxEpoch = 1,
  isBin = FALSE, isClass = TRUE, validData = NULL, validTargets = NULL,
  testData = NULL, testTargets = NULL, stopErr = -Inf,
  stopClassErr = 101, stopValidErr = -Inf, stopValidClassErr = 101)

## S4 method for signature 'DArch'
fineTuneDArch(darch, trainData, targetData, ...,
  maxEpoch = 1, isBin = FALSE, isClass = TRUE, validData = NULL,
  validTargets = NULL, testData = NULL, testTargets = NULL,
  stopErr = -Inf, stopClassErr = 101, stopValidErr = -Inf,
  stopValidClassErr = 101)
```

Arguments

`darch`	A instance of the class `DArch`.
`trainData`	The training data matrix
`targetData`	The expected output matrix for the training data
`...`	Additional parameters for the training function
`maxEpoch`	The number of training iterations
`isBin`	Indicates whether the output data must be interpreted as boolean value. Default is `FALSE`. If it is true, every value over 0.5 is interpreted as 1 and under as 0.
`isClass`	Indicates whether the training is for a classification net. When `TRUE` then statistics for classification will be determind. Default is `TRUE`
`validData`	Data for validating the network. Default is `NULL`
`validTargets`	The expected output for the training data Default is `NULL`
`testData`	Data for testing the network. Default is `NULL`
`testTargets`	The expected output for the training data Default is `NULL`
`stopErr`	Stop criteria for the error on the train data. Default is `-Inf`
`stopClassErr`	Stop criteria for the classification error on the train data. Default is `101`
`stopValidErr`	Stop criteria for the error on the validation data. Default is `-Inf`.
`stopValidClassErr`	Stop criteria for the classification error on the validation data. Default is `101` .

Details

The function trains the given network darch with the function saved in the attribute fineTuneFunction of the DArch-Object. The data (trainData, validData, testData) and belonging classes of the data (targetData, validTargets, testTargets) can be hand over either as matrix or as ff-matrix (see package ff for details). The data and classes for validation and testing are optional. If they are provided the network will be executed with this datasets and statistics will be calculated. This statistics are saved in the stats attribute (see Net). The attribue isBin indicates whether the output data must be interpreted as binary value. If true every value over 0.5 is interpreted as 1 otherwise as 0. Also it is possible to set stop criteria for the training on the error (stopErr, stopValidErr) or the correct classifications (stopClassErr, stopValidClassErr) of the training or validation dataset.

See Also

DArch, Net, backpropagation, rpropagation, minimizeAutoencoder, minimizeClassifier

---

generateRBMs *Generates the rbm's for the pre-training.*

Description

Used the layer sizes from the DArch object to create the RBM objects for the pre-training.

Usage

```
generateRBMs(darch, layers, genWeightFunc = generateWeights)

## S4 method for signature 'DArch'
generateRBMs(darch, layers, genWeightFunc = generateWeights)
```

Arguments

darch	A instance of the class DArch.
layers	An array with the sizes of the layers
genWeightFunc	The function for generating the weight matrices

Value

The DArch object with the generated rbm's

See Also

DArch RBM

generateWeights *Generates a weight matrix.*

Description

This function is the standard method for generating weights for instances of Net. When using another function to generate weights, the function must be like this one.

Usage

```
generateWeights(numUnits1, numUnits2)
```

Arguments

numUnits1	Number of units in the lower layer.
numUnits2	Number of units in the upper layer.

See Also

Net

getBatchSize *Returns the batch size of the Net.*

Description

Returns the batch size of the Net.

Usage

```
getBatchSize(net)

## S4 method for signature 'Net'
getBatchSize(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

getCancel *Returns the cancel value.*

Description

Returns the cancel value.

Usage

```
getCancel(darch)

## S4 method for signature 'DArch'
getCancel(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch

getCancelMessage *Returns the cancel message.*

Description

Returns the message, why the learing is canceled. If the message is not set, the default value "no reason specified" will be returned.

Usage

```
getCancelMessage(darch)

## S4 method for signature 'DArch'
getCancelMessage(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch

getErrorFunction *Returns the error function of the* Net.

Description

Returns the error function of the Net.

Usage

```
getErrorFunction(net)

## S4 method for signature 'Net'
getErrorFunction(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

getExecOutput *Returns the execution output of the layer from the* DArch *object*

Description

Returns the execution output of the layer by the given index. If the index is not set, the output of the last layer will be returned.

Usage

```
getExecOutput(darch,index=1)

## S4 method for signature 'DArch'
getExecOutput(darch, index = NULL)
```

Arguments

darch	A instance of the class DArch.
index	The index of the layer.

See Also

DArch

getExecOutputs *Returns the execution output list of the DArch object*

Description

Returns the execution output of the DArch object. The list contains all outputs of every layer in the network.

Usage

```
getExecOutputs(darch)

## S4 method for signature 'DArch'
getExecOutputs(darch)
```

Arguments

darch	A instance of the class DArch

See Also

DArch

getExecuteFunction *Returns the function for the execution of the DArch network.*

Description

Returns the function for the execution of the DArch network.

Usage

```
getExecuteFunction(darch)

## S4 method for signature 'DArch'
getExecuteFunction(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch

getFF — *Returns if the weights are saved as ff objects*

Description

Returns if the weights are saved as ff objects

Returns the ff state of the Net.

Usage

```
getFF(net)

## S4 method for signature 'Net'
getFF(net)

getFF(net)

## S4 method for signature 'Net'
getFF(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

Net

getFinalMomentum — *Returns the final momentum of the Net.*

Description

Returns the final momentum of the Net.

Usage

```
getFinalMomentum(net)

## S4 method for signature 'Net'
getFinalMomentum(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

---

getFineTuneFunction *Returns the fine tune function for the* DArch *object.*

Description

Returns the fine tune function which is executed by the function fineTuneDArch.

Usage

```
getFineTuneFunction(darch)

## S4 method for signature 'DArch'
getFineTuneFunction(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch, backpropagation, rpropagation, minimizeAutoencoder, minimizeClassifier

---

getGenWeightFunction *Returns the function for generating weight matrices.*

Description

Returns the function for generating weight matrices.

Usage

```
getGenWeightFunction(net)

## S4 method for signature 'Net'
getGenWeightFunction(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

getHiddenBiases *Returns the biases of the hidden units.*

Description

Returns the biases of the hidden units.

Usage

```
getHiddenBiases(rbm)

## S4 method for signature 'RBM'
getHiddenBiases(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The biases of the hidden units.

See Also

RBM

RBM

getHiddenBiasesInc *Returns the update value for the biases of the hidden units.*

Description

Returns the update value for the biases of the hidden units.

Usage

```
getHiddenBiasesInc(rbm)

## S4 method for signature 'RBM'
getHiddenBiasesInc(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The update value for the biases of the hidden units.

See Also

RBM

RBM

`getHiddenUnitStates` *Returns a list with the states of the hidden units.*

Description

Returns a list with the states of the hidden units.

Usage

```
getHiddenUnitStates(rbm)

## S4 method for signature 'RBM'
getHiddenUnitStates(rbm)
```

Arguments

`rbm`	A instance of the class RBM.

Value

The states of the hidden units.

See Also

RBM

getLayer *Returns a layer from the* DArch *object.*

Description

The function returns the layer with the given `index` which contains all weights, functions for the neurons, and possible additional parameters for the training.

Usage

```
getLayer(darch,index=1)

## S4 method for signature 'DArch'
getLayer(darch, index = 1)
```

Arguments

`darch`	A instance of the class DArch.
`index`	The index of the layer.Default is 1.

See Also

DArch

getLayerField *Returns the field of a layer from the* DArch *object.*

Description

The function returns the field given by the `fieldIndex` of the layer given by the `layerIndex`.

Usage

```
getLayerField(darch,layerIndex=1,fieldIndex=3)

## S4 method for signature 'DArch'
getLayerField(darch, layerIndex = 1, fieldIndex = 3)
```

Arguments

`darch`	A instance of the class DArch.
`layerIndex`	The index of the layer.
`fieldIndex`	The index of the field in the layer.

See Also

DArch

getLayerFunction *Returns the neuron function of a layer from the DArch object.*

Description

The function returns the neuron function of the layer with the given index.

Usage

```
getLayerFunction(darch,index=1)

## S4 method for signature 'DArch'
getLayerFunction(darch, index = 1)
```

Arguments

darch	A instance of the class DArch.
index	The index of the layer.Default is 1.

See Also

DArch sigmoidUnit, binSigmoidUnit, sigmoidUnitDerivative, linearUnitDerivative, softmaxUnit, softmaxUnitDerivative

getLayers *Returns the a list of layers from the DArch object.*

Description

The function returns the layers list which contains all weights, functions for the neurons, and possible additional parameters for the training.

Usage

```
getLayers(darch)

## S4 method for signature 'DArch'
getLayers(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch

getLayerWeights *Returns the weights of a layer from the* DArch *object.*

Description

The function returns the weights of the layer with the given index.

Usage

```
getLayerWeights(darch,index=1)

## S4 method for signature 'DArch'
getLayerWeights(darch, index = 1)
```

Arguments

darch	A instance of the class DArch.
index	The index of the layer.Default is 1.

See Also

DArch

getLearnRateBiases *Returns the learning rate for the bias weigths of the* DArch *object.*

Description

Returns the learning rate for the bias weigths of the DArch object.

Usage

```
getLearnRateBiases(darch)

## S4 method for signature 'DArch'
getLearnRateBiases(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch

`getLearnRateBiasHidden` *Returns the learning rate for the hidden biases.*

Description

Returns the learning rate for the hidden biases.

Usage

```
getLearnRateBiasHidden(rbm)

## S4 method for signature 'RBM'
getLearnRateBiasHidden(rbm)
```

Arguments

`rbm`	A instance of the class RBM.

Value

The learning rate for the hidden biases

See Also

RBM

RBM

`getLearnRateBiasVisible` *Returns the learning rate for the visible biases.*

Description

Returns the learning rate for the visible biases.

Usage

```
getLearnRateBiasVisible(rbm)

## S4 method for signature 'RBM'
getLearnRateBiasVisible(rbm)
```

Arguments

`rbm`	A instance of the class RBM.

Value

The learning rate for the visible biases

See Also

RBM

RBM

`getLearnRateWeights` *Returns the learn rate of the weights.*

Description

Returns the learn rate of the weights.

Usage

```
getLearnRateWeights(net)

## S4 method for signature 'Net'
getLearnRateWeights(net)
```

Arguments

`net`	A instance of the class Net.

See Also

Net

`getMomentum` *Returns the momentum of the* Net.

Description

Returns the momentum of the Net.

Usage

```
getMomentum(net)

## S4 method for signature 'Net'
getMomentum(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

`getMomentumSwitch` *Returns the momentum switch of the* Net.

Description

Returns the momentum switch of the Net.

Usage

```
getMomentumSwitch(net)

## S4 method for signature 'Net'
getMomentumSwitch(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

`getNumHidden` *Returns the number of hidden units of the* RBM

Description

Returns the number of hidden units of the RBM

Usage

```
getNumHidden(rbm)

## S4 method for signature 'RBM'
getNumHidden(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The number of hidden units of the RBM

See Also

RBM

RBM

`getNumVisible` *Returns the number of visible units of the* RBM

Description

Returns the number of visible units of the RBM

Usage

```
getNumVisible(rbm)

## S4 method for signature 'RBM'
getNumVisible(rbm)
```

Arguments

`rbm`	A instance of the class RBM.

Value

The number of visible units of the RBM

See Also

RBM

RBM

getOutput *Returns the output of the RBM*

Description

Returns the output of the RBM

Usage

```
getOutput(rbm)

## S4 method for signature 'RBM'
getOutput(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The output of the RBM

See Also

RBM

getPosPhaseData *Returns the data for the positive phaes.*

Description

Returns the data for the positive phaes.

Usage

```
getPosPhaseData(rbm)

## S4 method for signature 'RBM'
getPosPhaseData(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The data for the positive phaes.

See Also

RBM

RBM

---

getRBMList *Returns a list of RBMs of the DArch object.*

Description

This function returns a list of RBMs of the DArch which are needed for the pre training of the network.

Usage

```
getRBMList(darch)

## S4 method for signature 'DArch'
getRBMList(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch

---

getStats *Returns the list of statistics for the network*

Description

The list of statistics can contain values about errors, miss classifications and other usefull things from the pre-training or fine-tuning of a deep architecture.

Usage

```
getStats(net)

## S4 method for signature 'Net'
getStats(net)
```

Arguments

net	A instance of the class Net.

See Also

Net

getVisibleBiases *Returns the biases of the visible units.*

Description

Returns the biases of the visible units.

Usage

```
getVisibleBiases(rbm)

## S4 method for signature 'RBM'
getVisibleBiases(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The biases of the visible units.

See Also

RBM

RBM

getVisibleBiasesInc *Returns the update value for the biases of the visible units.*

Description

Returns the update value for the biases of the visible units.

Usage

```
getVisibleBiasesInc(rbm)

## S4 method for signature 'RBM'
getVisibleBiasesInc(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The update value for the biases of the visible units.

See Also

RBM

RBM

`getVisibleUnitStates` *Returns a list with the states of the visible units.*

Description

Returns a list with the states of the visible units.

Usage

```
getVisibleUnitStates(rbm)

## S4 method for signature 'RBM'
getVisibleUnitStates(rbm)
```

Arguments

`rbm`	A instance of the class RBM.

Value

The states of the visible units.

See Also

RBM

getWeightCost *Returns the weigth cost for the training*

Description

Returns the weigth cost for the training

Usage

```
getWeightCost(rbm)

## S4 method for signature 'RBM'
getWeightCost(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The weigth cost for the training

See Also

RBM

RBM

getWeightInc *Returns the update value for the weights.*

Description

Returns the update value for the weights.

Usage

```
getWeightInc(rbm)

## S4 method for signature 'RBM'
getWeightInc(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The update value for the weights.

See Also

RBM

RBM

`getWeights` *Returns the weigths of the* RBM.

Description

Returns the weigths of the RBM.

Usage

```
getWeights(rbm)

## S4 method for signature 'RBM'
getWeights(rbm)
```

Arguments

`rbm`	A instance of the class RBM.

Value

The weigths of the RBM.

See Also

RBM

RBM

linearUnit *Linear unit function.*

Description

The function calculates the activation of the units and returns it.

Usage

```
linearUnit(data,weights)
```

Arguments

data	The data matrix for the calculation
weights	The weight and bias matrix for the calculation

Value

A list with the linear activation of the unit in the first entry.

See Also

DArch, sigmoidUnit, binSigmoidUnit, sigmoidUnitDerivative, linearUnitDerivative, softmaxUnit, softmaxUnitDerivative

linearUnitDerivative *Linear unit function with unit derivatives.*

Description

The function calculates the activation of the units and returns a list, in which the first entry is the linear activation of the units and the second entry is the derivative of the transfer function.

Usage

```
linearUnitDerivative(data,weights)
```

Arguments

data	The data matrix for the calculation
weights	The weight and bias matrix for the calculation

Value

A list with the linear activation in the first entry and the derivative of the activation in the second entry

See Also

DArch, sigmoidUnit, binSigmoidUnit, sigmoidUnitDerivative, linearUnit, softmaxUnit, softmaxUnitDerivative

---

`linearUnitFunc` *Calculates the linear neuron output no transfer function*

Description

Calculates the linear neuron output with no transfer function from real value input saved in the first entry of the list `dataList`.

Usage

```
linearUnitFunc(rbm, dataList, biases, weights, runParams)
```

Arguments

`rbm`	A instance of the class RBM.
`dataList`	A list with the data matrices for the calculations.
`biases`	The biases for the calculations
`weights`	The weight matrix for the calculations
`runParams`	Parameters which indicates the status of the training.

Details

The return value is a list with the output of the neurons as first entry and binary representation calculated through a comparison of the output with random numbers. The random numbers a generated with the function rnorm.

Value

The real value and binary activations for the units

See Also

DArch

loadDArch *Loads a DArch network*

Description

Loads the DArch object from the filename given through the parameter `name` plus the ending ".net".

Usage

```
loadDArch(name="darch")
```

Arguments

`name`	The name of the file without the ending ".net".

Details

Make sure when you load a DArch object that every file written by the `saveDArch`-Funktion, specially when the parameter `ff` of the saved DArch object is `TRUE`, are in the working directory.

Value

`darch` - The loaded deep architecture

See Also

`saveDArch`, `loadRBMFFWeights`

loadRBM *Loads a RBM network*

Description

Loads the RBM object from the filename given through the parameter `name` plus the ending ".net".

Usage

```
loadRBM(name="rbm")
```

Arguments

`name`	The name of the file without the ending ".net".

Details

Make sure when you load a RBM object that every file written by the `saveRBM`-Funktion, specially when the parameter `ff` of the saved RBM object is `TRUE`, are in the working directory

Value

rbm - The loaded RBM

See Also

saveRBM, loadRBMFFWeights, saveRBMFFWeights

---

loadRBMFFWeights *Loads weigths and biases for a RBM network from a ffData file.*

Description

Loads the weigths and the biases for the given RBM object from the filename given through the parameter name. See ffload for more details

Usage

```
loadRBMFFWeights(rbm,name)
```

Arguments

rbm	A instance of the class RBM.
name	The name of the file without the ending ".net".

Value

rbm - The RBM with the loaded weights and biases.

See Also

ffload, saveRBM, loadRBM, saveRBMFFWeights

---

makeStartEndPoints *Makes start- and end-points for the batches.*

Description

The start- and end-points are used for dividing the data into batches.

Usage

```
makeStartEndPoints(batchSize, numRows)
```

Arguments

batchSize	Desired batch size
numRows	Number of rows of the data

Details

If the data is not divedable by the batchSize the last batch will contain the rest of the data. The function returns a list with in which the first entry is a list with the values for the start and end points for reading the data matrix. The second entry is the number of batches.

See Also

Net

---

minimize *Minimize a differentiable multivariate function.*

Description

This function is a direct translation from the Matlab source code of the minimize function from Carl Edward Rasmussen.

Usage

```
minimize( X, f, length, ...)
```

Arguments

X	Starting point. An Array of the weights.
f	Function for calculating the function value and the partial derivatives
length	Maximum number of line searches or maximum allowed number of function evaluations if negative
...	Additional Parameters for the function f

Details

Minimize a differentiable multivariate function.

Usage: [X, fX, i] <- minimize(X, f, length, P1, P2, P3, ...)

where the starting point is given by "X" (D by 1), and the function named in the string "f", must return a function value and a vector of partial derivatives of f wrt X, the "length" gives the length of the run: if it is positive, it gives the maximum number of line searches, if negative its absolute gives the maximum allowed number of function evaluations. You can (optionally) give "length" a second component, which will indicate the reduction in function value to be expected in the first line-search (defaults to 1.0). The parameters P1, P2, P3, ... are passed on to the function f.

The function returns when either its length is up, or if no further progress can be made (ie, we are at a (local) minimum, or so close that due to numerical problems, we cannot get any closer). NOTE: If the function terminates within a few iterations, it could be an indication that the function values and derivatives are not consistent (ie, there may be a bug in the implementation of your "f" function). The function returns the found solution "X", a vector of function values "fX" indicating the progress made and "i" the number of iterations (line searches or function evaluations, depending on the sign of "length") used. The Polack-Ribiere flavour of conjugate gradients is used to compute search directions, and a line search using quadratic and cubic polynomial approximations and the Wolfe-Powell stopping criteria is used together with the slope ratio method for guessing initial step sizes. Additionally a bunch of checks are made to make sure that exploration is taking place and that extrapolation will not be unboundedly large. See also: checkgrad Copyright (C) 2001 - 2006 by Carl Edward Rasmussen (2006-09-08).

Value

The function returns the found solution "X", a vector of function values "fX" indicating the progress made and "i" the number of iterations (line searches or function evaluations, depending on the sign of "length") used.

See Also

DArch, minimizeAutoencoder, minimizeClassifier

---

minimizeAutoencoder *Conjugate gradient for a autoencoder network*

Description

This function trains a DArch autoencoder network with the conjugate gradient method.

Usage

```
minimizeAutoencoder(darch,trainData,targetData,epoch,length)
```

Arguments

darch	A instance of the class DArch.
trainData	The training data matrix
targetData	The labels for the training data
epoch	The actual epoch of the training
length	Numbers of line search

Details

This function is build on the basis of the code from G. Hinton et. al. (http://www.cs.toronto.edu/~hinton/MatlabForSciencePa
- last visit 06.06.2013) for the fine tuning of deep belief nets. The original code is located in the files 'backpropclassify.m', 'CG_MNIST.m' and 'CG_CLASSIFY_INIT.m'. It implements the fine tuning for a classification net with backpropagation using a direct translation of the `minimize` function from C. Rassmussen (available at http://www.gatsby.ucl.ac.uk/~edward/code/minimize/ - last visit 06.06.2013) to R.

Value

The trained `DArch` object.

See Also

`DArch` `fineTuneDArch`

`minimizeClassifier` *Conjugate gradient for a classification network*

Description

This function trains a `DArch` classifier network with the conjugate gradient method.

Usage

```
minimizeClassifier(darch,trainData,targetData,epoch,length,switchLayers)
```

Arguments

`darch`	A instance of the class `DArch`.
`trainData`	The training data matrix
`targetData`	The labels for the training data
`epoch`	The actual epoch of the training
`length`	Numbers of line search
`switchLayers`	Indicates when to train the full network instead of only the upper two layers

Details

This function is build on the basis of the code from G. Hinton et. al. (http://www.cs.toronto.edu/~hinton/MatlabForSciencePa
- last visit 06.06.2013) for the fine tuning of deep belief nets. The original code is located in the files 'backpropclassify.m', 'CG_MNIST.m' and 'CG_CLASSIFY_INIT.m'. It implements the fine tuning for a classification net with backpropagation using a direct translation of the `minimize` function from C. Rassmussen (available at http://www.gatsby.ucl.ac.uk/~edward/code/minimize/ - last visit 06.06.2013) to R. The parameter `switchLayers` is for the switch between two training type. Like in the original code, the top two layers can be trained alone until `epoch` is equal to `epochSwitch`. Afterwards the entire network will be trained.

Value

The trained DArch object.

See Also

DArch

---

mseError *Mean quared error function*

Description

The function calculates the mean quared error (MSE) from the original and estimate parameters.

Usage

```
mseError(original, estimate)
```

Arguments

original	The original data matrix
estimate	The calculated data matrix

Value

A list with the name of the error function in the first entry and the error value in the second entry

See Also

quadraticError, crossEntropyError

Net *Abtract class for neural networks.*

Description

This is a abstract class for neural networks. It provides some functionalitys used in more than one network type.

Slot

batchSize: Object of class "numeric". The batch size for the training and test data during the learning.

errorFunction: Object of class "function". Function for error calculation.

ff: Object of class "logical". Indicates if the package ff is used to save the network data.

genWeightFunction: Object of class "function". A function for generate random initialised weight matrix.

Author(s)

Martin Drees

See Also

DArch, RBM

---

newDArch *Constructor function for* DArch *objects.*

Description

Generate a new DArch object with the given parameters.

Usage

```
newDArch(layers,batchSize,ff=FALSE,
logLevel=INFO, genWeightFunc=generateWeights)
```

Arguments

layers	Array of layer sizes.
batchSize	Size of the batches
ff	Indicates whether the ff package is used for the weights, biases and outputs
logLevel	The logging level. See setLogLevel for details.
genWeightFunc	The function for generating the weight matrices

Details

It is recommended to use this function for generating a new DArch object, because this function generates and sets all the necessary parameters like the internally used RBM networks, the list of statistiks (stats) etc.

Value

The new DArch object

newRBM *Constructor function for RBM object.*

Description

TODO: Doc ...

Usage

```
newRBM(numVisible, numHidden, batchSize, ff = FALSE, logLevel = INFO,
  genWeightFunc = generateWeights)
```

Arguments

numVisible	Number of visible units.
numHidden	Number of hidden units.
batchSize	Size of the batches
ff	Indicates whether the ff package is used for the weights, biases and outputs
logLevel	The logging level. See setLogLevel for details.
genWeightFunc	The function for generating the weight matrices

Value

The new RBM object

preTrainDArch *Pre trains a DArch network*

Description

This function pre trains a DArch network with the contrastive divergence method

Usage

```
preTrainDArch(darch,trainData,maxEpoch=1,numCD=1,...)

## S4 method for signature 'DArch'
preTrainDArch(darch, trainData, maxEpoch = 1, numCD = 1,
  ...)
```

Arguments

darch	A instance of the class DArch.
trainData	The data matrix for the training
maxEpoch	The number of epochs
numCD	The number of CD iterations
...	Additional parameters for the function trainRBM

Details

The function runs for every RBM in the attribute rbmList the traning function trainRBM copies after the training the weights and biases into the corresponding layer of the DArch network.

See Also

DArch, RBM, trainRBM

---

quadraticError *Quadratic error function*

Description

The function calculates the quadradic error from the original and estimate parameters.

Usage

```
quadraticError(original, estimate)
```

Arguments

original	The original data matrix
estimate	The calculated data matrix

Value

A list with the name of the error function in the first entry and the error value in the second entry

See Also

mseError, crossEntropyError

RBM *Class for Restricted-Bolzmann-Machine*

Description

This class represents a Restricted-Bolzmann-Machine

Slots

`learnRateBiasVisible`: Object of class `"numeric"`. Learning rate of the visible biases.

`learnRateBiasHidden`: Object of class `"numeric"`. Learning rate of the hidden biases.

`weightCost`: Object of class `"numeric"`. Weigth cost for the update of the weigths.

`numHidden`: Object of class `"numeric"`. Number of hidden units.

`numVisible`: Object of class `"numeric"`. Number of visible units.

`weights`: Object of class `"matrix"`. Weight matrix.

`weightInc`: Object of class `"matrix"`. Matrix of update values for the Weight.

`output`: Object of class `"matrix"`. Output matrix of the RBM.

`visibleBiases`: Object of class `"array"`. Visible biases array.

`visibleBiasesInc`: Object of class `"array"`. Array of update values for the visible biases

`visibleUnitFunction`: Object of class `"function"`. Unit function for the visible units.

`visibleUnitStates`: Object of class `"list"`. States of the visible units.

`hiddenBiases`: Object of class `"array"`. Hidden biases array.

`hiddenBiasesInc`: Object of class `"array"`. Array of update values for the hidden biases.

`hiddenUnitFunction`: Object of class `"function"`. Unit function for the hidden units.

`hiddenUnitStates`: Object of class `"list"`. States of the hidden units.

`updateFunction`: Object of class `"function"`. Function for updating the weights and biases.

`posPhaseData`: Object of class `"list"`. Attribute to save the positive phase data during the training.

`ffWeights`: Object of class `"ff_matrix"`. Weight ff matrix. Used when the ff attribute is `TRUE`.

`ffOutput`: Object of class `"ff_matrix"`. Output ff matrix of the RBM. Used when the ff attribute is `TRUE`.

`ffHiddenBiases`: Object of class `"ff_array"`. Hidden biases ff array. Used when the ff attribute is `TRUE`.

`ffVisibleBiases`: Object of class `"ff_array"`. Hidden biases ff array. Used when the ff attribute is `TRUE`.

Author(s)

Martin Drees

See Also

`Net`, `DArch`, `trainRBM`

rbmUpdate *Function for updating the weights and biases of an RBM*

Description

This function updates the weights and biases for an RBM network. It is saved in the attribute updateFunction of the RBM object and called from the training function trainRBM.

Usage

```
rbmUpdate(rbm)
```

Arguments

rbm	A instance of the class RBM.

Value

The updated RBM.

See Also

RBM

readMNIST *Function for generating ff files of the MNIST Database*

Description

This function reads the MNIST-Database, randomized it and saved it in the files "train" for the training data and "test" for test data.

Usage

```
readMNIST(folder)
```

Arguments

folder	The location of the MNIST-Database files.

Details

When the data is read the variables for the training data is trainData and trainLabels and for the test data testData and testLabels. To start the function The files "train-images-idx3-ubyte", "train-labels-idx1-ubyte', "t10k-images-idx3-ubyte", and "t10k-labels-idx1-ubyte" have to be in the folder given by the parameter folder. The folder name must end with a slash.

removeLayerField *Removes a layer from the DArch object*

Description

This function removes the layer with the given index from the DArch object.

Usage

```
removeLayerField(darch, index)

## S4 method for signature 'DArch'
removeLayerField(darch, index)
```

Arguments

darch	A instance of the class DArch.
index	The index of the layer.

Value

The DArch object without the layer.

See Also

DArch

resetDArch *Resets the weights and biases of the DArch object*

Description

This function resets the weights and biases of the DArch object and all RBM objects if the parameter resetRBMs is TRUE.

Usage

```
resetDArch(darch,resetRBMs=TRUE)

## S4 method for signature 'DArch'
resetDArch(darch, resetRBMs = TRUE)
```

Arguments

darch	A instance of the class DArch.
resetRBMs	If true the RBMs are also reseted.

Details

When the parameter resetRBMs is FALSE then the trained weights and biases are copied from the RBM objects to the layers.

See Also

DArch

---

resetExecOutput *Resets the output list of the DArch object*

Description

This function sets the attribute executeOutput of the DArch object to an empty list.

Usage

```
resetExecOutput(darch)

## S4 method for signature 'DArch'
resetExecOutput(darch)
```

Arguments

darch	A instance of the class DArch.

See Also

DArch

---

resetRBM *Resets the weights and biases of the RBM object*

Description

This function resets the weights and biases of the RBM object.

Usage

```
resetRBM(rbm)

## S4 method for signature 'RBM'
resetRBM(rbm)
```

Arguments

rbm	A instance of the class RBM.

See Also

RBM

---

rpropagation *Resilient-Backpropgation training for deep architectures.*

Description

The function traines a deep architecture with the resilient backpropagation algorithm. It is able to use four different types of training (see details). For details of the resilient backpropagation algorith see the references.

Usage

```
rpropagation(darch,trainData,targetData,epoch,method="iRprop+",
decFact=0.5,incFact=1.2,weightDecay=0,
initDelta=0.0125,minDelta=0.000001,maxDelta=50)
```

Arguments

darch	The deep architecture to train
trainData	The training data
targetData	The expected output for the training data
epoch	The number of training iterations
method	The method for the training. Default is "iRprop+"
decFact	Decreasing factor for the training. Default is 0.5.
incFact	Increasing factor for the training Default is 1.2.
weightDecay	Weight decay for the training. Default is 0
initDelta	Initialisation value for the update. Default is 0.0125.
minDelta	Lower bound for step size. Default is 0.000001
maxDelta	Upper bound for step size. Default is 50

Details

The code for the calculation of the weight change is a translation from the matlab code from the Rprop Optimization Toolbox implemented by R. Calandra (see References).

Value

`darch` - The trained deep architecture

References

M. Riedmiller, H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In Proceedings of the IEEE International Conference on Neural Networks, pp 586-591. IEEE Press, 1993.

C. Igel , M. Huesken. Improving the Rprop Learning Algorithm, Proceedings of the Second International Symposium on Neural Computation, NC 2000, ICSC Academic Press, Canada/Switzerland, pp. 115-121., 2000.

Kohavi, R., A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, Proceedings of the 14th Int. Joint Conference on Artificial Intelligence 2, S. 1137-1143, Morgan Kaufmann, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

See Also

`DArch`

`runDArch` *Execute the darch*

Description

Runs the darch in a feed forward manner and saves the generated outputs for every layer in the list `executeOutput` from the darch. To get the outputs call

Usage

```
runDArch(darch, data)
```

Arguments

darch	A instance of the class DArch.
data	The input data to execute the darch on.

Value

The DArch object with the calculated outputs

See Also

DArch

---

saveDArch *Saves a DArch network*

Description

Saves the DArch object to the filename given through the parameter name plus the ending ".net".

Usage

```
saveDArch(darch,name="darch",saveRBM=TRUE)

## S4 method for signature 'DArch'
saveDArch(darch, name = "darch", saveRBM = TRUE)
```

Arguments

darch	A instance of the class DArch.
name	The name for the file. Default value is "darch".
saveRBM	Boolean value to indicate if the RBM's are saved.

Details

If the field ff of the DArch object is TRUE then the weights are saved in seperate ff-files named by the parameter name plus the string "-W" and the number of the layer. In the same way the weights from the RBM's of the DArch are saved, but only if the parameter saveRBM is TRUE. For more information about the how the weights and biases from the RBM's are saved see saveRBMFFWeights. If the parameter saveRBM is FALSE the field rbmList of the DArch object ist overwritten by an empty list.

See Also

loadDArch, saveRBMFFWeights

saveRBM *Saves a RBM network*

Description

Saves the RBM object to the filename given through the parameter name plus the ending ".net".

Usage

```
saveRBM(rbm,name="rbm")

## S4 method for signature 'RBM'
saveRBM(rbm, name = "rbm")
```

Arguments

rbm	A instance of the class RBM.
name	The name for the file. Default value is "rbm".

Details

If the field ff of the RBM object is TRUE then the weights are saved in seperate ff-files through the funktion saveRBMFFWeights.

See Also

loadRBM, saveRBMFFWeights loadRBMFFWeights

saveRBMFFWeights *Saves weights and biases of a RBM network into a ffData file.*

Description

Saves the weigths and the biases for the given RBM object to the filename given through the parameter name.

Usage

```
saveRBMFFWeights(rbm,name="saveName")

## S4 method for signature 'RBM'
saveRBMFFWeights(rbm, name = "saveName")
```

Arguments

rbm	A instance of the class RBM.
name	The name for the file.

Details

The weights and biases are saved in one file with the name given through the parameter `name` and the string "-WB". See `ffsave` for more details.

See Also

`ffsave`, `loadRBM`, `saveRBM`, `loadRBMFFWeights`

---

setBatchSize<- *Sets the batch size of the* `Net`.

Description

Sets the batch size of the `Net`.

Usage

```
setBatchSize(net) <- value
```

Arguments

`net`	A instance of the class `Net`.
`value`	Object of the class `numeric`.

See Also

`Net`

---

setCancel<- *Set whether the learning shall be canceled.*

Description

Set whether the learning shall be canceled.

Usage

```
setCancel(darch) <- value
```

Arguments

`darch`	A instance of the class `DArch`.
`value`	Boolean value if the learning shall canceled

See Also

`DArch`

setCancelMessage<- *Sets the cancel message.*

Description

Sets the cancel message.

Usage

```
setCancelMessage(darch) <- value
```

Arguments

darch	A instance of the class DArch.
value	The message for the termination

See Also

DArch

setErrorFunction<- *Sets the error function of the Net.*

Description

Sets the error function of the Net.

Usage

```
setErrorFunction(net) <- value
```

Arguments

net	A instance of the class Net.
value	Object of the class function.

See Also

Net

setExecuteFunction<- *Sets the execution function for the network*

Description

Sets the execution function for the network

Usage

```
setExecuteFunction(darch)  <- value
```

Arguments

darch	A instance of the class DArch.
value	The execution function for the network.

See Also

DArch

setFF<- *Sets if the weights are saved as ff objects*

Description

Sets if the weights are saved as ff objects

Usage

```
setFF(net) <- value
```

Arguments

net	A instance of the class Net.
value	Boolean value which indicates if the weights are saved as ff objects

See Also

Net

setFinalMomentum<- *Sets the final momentum of the* Net.

Description

Sets the final momentum of the Net.

Usage

```
setFinalMomentum(net) <- value
```

Arguments

net	A instance of the class Net.
value	Object of the class numeric.

See Also

Net

setFineTuneFunction<- *Sets the fine tuning function for the network*

Description

Sets the fine tuning function for the network

Usage

```
setFineTuneFunction(darch)  <- value
```

Arguments

darch	A instance of the class DArch.
value	The fine tuning function for the network.

See Also

DArch

setGenWeightFunction<- *Sets the function for generating weight matrices.*

Description

The function have to return a matrix with number of units in the lower layer as number of rows and number of units in the upper layer as the nubmer of columns.

Usage

```
setGenWeightFunction(net) <- value
```

Arguments

net	A instance of the class Net.
value	Object of the class function.

See Also

Net

setHiddenBiases<- *Sets the biases of the hidden units for the RBM object*

Description

Sets the biases of the hidden units for the RBM object

Usage

```
setHiddenBiases(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The biases of the hidden units for the RBM object.

See Also

RBM

setHiddenBiasesInc<- *Sets the update value for the biases of the hidden units*

Description

Sets the update value for the biases of the hidden units

Usage

```
setHiddenBiasesInc(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The update value for the biases of the hidden units.

See Also

RBM

setHiddenUnitFunction<- *Sets the unit function of the hidden units*

Description

Sets the unit function of the hidden units

Usage

```
setHiddenUnitFunction(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The unit function of the hidden units

See Also

RBM

setHiddenUnitStates<- *Sets the states of the hidden units*

Description

Sets the states of the hidden units

Usage

```
setHiddenUnitStates(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The states of the hidden units

See Also

RBM

setLayer<- *Sets a layer with the given index for the network*

Description

Sets a layer with the given index for the network

Usage

```
setLayer(darch,index) <- value
```

Arguments

darch	A instance of the class DArch.
index	The index of the layer
value	The layer for the network.

See Also

DArch

setLayerField<- *Sets a field in a layer.*

Description

Sets the field on position `fieldIndex` of the layer given by the `layerIndex` to the `value`.

Usage

```
setLayerField(darch,layerIndex,fieldIndex) <- value
```

Arguments

`darch`	A instance of the class DArch.
`layerIndex`	The index of the layer
`fieldIndex`	The index of the field
`value`	The value for the layer field

Value

The darch with the updated layer

See Also

DArch

setLayerFunction<- *Sets the function for a layer with the given index*

Description

Sets the function for a layer with the given index

Usage

```
setLayerFunction(darch,index) <- value
```

Arguments

`darch`	A instance of the class DArch.
`index`	The index of the layer.
`value`	The function for the layer.

See Also

DArch

setLayers<- *Sets the layers for the network*

Description

Sets the layers for the network

Usage

```
setLayers(darch) <- value
```

Arguments

darch	A instance of the class DArch.
value	The layers for the network.

See Also

DArch

setLayerWeights<- *Sets the weights of a layer with the given index*

Description

Sets the weights of a layer with the given index

Usage

```
setLayerWeights(darch,index) <- value
```

Arguments

darch	A instance of the class DArch.
index	The index of the layer.
value	The weights for the layer.

See Also

DArch

setLearnRateBiases<- *Sets the learning rate for the biases*

Description

Sets the learning rate for the biases

Usage

```
setLearnRateBiases(darch)  <- value
```

Arguments

darch	A instance of the class DArch.
value	The learning rate for the biases.

See Also

DArch

setLearnRateBiasHidden<- *Sets the learnig rates of the biases for the hidden units*

Description

Sets the learnig rates of the biases for the hidden units

Usage

```
setLearnRateBiasHidden(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The learnig rates of the biases for the hidden units

See Also

RBM

setLearnRateBiasVisible<- *Sets the learnig rates of the biases for the visible units*

Description

Sets the learnig rates of the biases for the visible units

Usage

```
setLearnRateBiasVisible(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The learnig rates of the biases for the visible units

See Also

RBM

setLearnRateWeights<- *Sets the learning rate for the weights.*

Description

Sets the learning rate for the weights.

Usage

```
setLearnRateWeights(net) <- value
```

Arguments

net	A instance of the class Net.
value	Object of the class numeric.

See Also

Net

setLogLevel<- *Sets the log level for the* Net.

Description

The log levels a defined by the futile.logger package. The following levels a available:

TRACE
DEBUG
INFO
WARN
ERROR
FATAL

Usage

```
setLogLevel(net) <- value
```

Arguments

net	A instance of the class Net.
value	Object of the class numeric.

See Also

Net

`setMomentum<-` *Sets the momentum of the* Net.

Description

Sets the momentum of the Net.

Usage

```
setMomentum(net) <- value
```

Arguments

net	A instance of the class Net.
value	Object of the class numeric.

See Also

Net

setMomentumSwitch<- *Sets the momentum switch of the* Net.

Description

Sets the momentum switch of the Net.

Usage

```
setMomentumSwitch(net) <- value
```

Arguments

net	A instance of the class Net.
value	Object of the class numeric.

See Also

Net

setNumHidden<- *Sets the number of hidden units*

Description

Sets the number of hidden units

Usage

```
setNumHidden(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The number of hidden units

See Also

RBM

setNumVisible<- *Sets the number of visible units*

Description

Sets the number of visible units

Usage

```
setNumVisible(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The number of visible units

See Also

RBM

setOutput<- *Sets the output of the RBM object*

Description

Sets the output of the RBM object

Usage

```
setOutput(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The output of the RBM object

See Also

RBM

setPosPhaseData<- *Sets the positive phase data for the training*

Description

Sets the positive phase data for the training

Usage

```
setPosPhaseData(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The positive phase data for the training

See Also

RBM

setRBMList<- *Sets the list of RBMs*

Description

Sets the list of RBMs

Usage

```
setRBMList(darch) <- value
```

Arguments

darch	A instance of the class DArch.
value	The list of RBMs.

See Also

DArch

setStats<- *Adds a list of statistics to the network*

Description

The list of statistics can contain values about errors, miss classifications and other usefull things from the pre-training or fine-tuning of a deep architecture.

Usage

```
setStats(net) <- value
```

Arguments

net	A instance of the class Net.
value	Statistics for the Net.

setUpdateFunction<- *Sets the update function of the RBM object*

Description

Sets the update function of the RBM object

Usage

```
setUpdateFunction(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The update function of the RBM object.

See Also

RBM

setVisibleBiases<- *Sets the biases of the visible units for the RBM object*

Description

Sets the biases of the visible units for the RBM object

Usage

```
setVisibleBiases(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The biases of the visible units for the RBM object.

See Also

RBM

setVisibleBiasesInc<- *Sets the update value for the biases of the visible units*

Description

Sets the update value for the biases of the visible units

Usage

```
setVisibleBiasesInc(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The update value for the biases of the visible units.

See Also

RBM

`setVisibleUnitFunction<-`

Sets the unit function of the visible units

Description

Sets the unit function of the visible units

Usage

```
setVisibleUnitFunction(rbm) <- value
```

Arguments

`rbm`	A instance of the class RBM.
`value`	The unit function of the visible units

See Also

RBM

`setVisibleUnitStates<-`

Sets the states of the visible units

Description

Sets the states of the visible units

Usage

```
setVisibleUnitStates(rbm) <- value
```

Arguments

`rbm`	A instance of the class RBM.
`value`	The states of the visible units

See Also

RBM

setWeightCost<- *Sets the weight costs for the training*

Description

Sets the weight costs for the training

Usage

```
setWeightCost(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The weight costs for the training

See Also

RBM

setWeightInc<- *Sets the update values for the weights*

Description

Sets the update values for the weights

Usage

```
setWeightInc(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The update values for the weights

See Also

RBM

setWeights<- *Sets the weights of the RBM object*

Description

Sets the weights of the RBM object

Usage

```
setWeights(rbm) <- value
```

Arguments

rbm	A instance of the class RBM.
value	The weights of the RBM object.

See Also

RBM

sigmoidUnit *Sigmoid unit function.*

Description

The function calculates the activation and returns the result through the sigmoid transfer function.

Usage

```
sigmoidUnit(data,weights)
```

Arguments

data	The data matrix for the calculation
weights	The weight and bias matrix for the calculation

Value

A list with the activation of the unit in the first entry.

See Also

DArch, binSigmoidUnit, sigmoidUnitDerivative, linearUnitDerivative, softmaxUnit, softmaxUnitDerivative

sigmoidUnitDerivative *Sigmoid unit function with unit derivatives.*

Description

The function calculates the activation and returns a list which the first entry is the result through the sigmoid transfer function and the second entry is the derivative of the transfer function.

Usage

```
sigmoidUnitDerivative(data,weights)
```

Arguments

data	The data matrix for the calculation
weights	The weight and bias matrix for the calculation

Value

A list with the activation in the first entry and the derivative of the transfer function in the second entry

See Also

DArch, sigmoidUnit, binSigmoidUnit, linearUnit, linearUnitDerivative, softmaxUnit, softmaxUnitDerivative,

sigmUnitFunc *Calculates the neuron output with the sigmoid function*

Description

Calculates the neuron output with the sigmoid function from binary input saved in the second entry of the list dataList.

Usage

```
sigmUnitFunc(rbm, dataList, biases, weights, runParams)
```

Arguments

rbm	A instance of the class RBM.
dataList	A list with the data matrices for the calculations.
biases	The biases for the calculations
weights	The weight matrix for the calculations
runParams	Parameters which indicates the status of the training.

Details

The return value is a list with the output of the sigmoid function as first entry and binary representation calculated through a comparison of the output with random numbers. The random numbers a generated with the function runif.

Value

The real value and binary activations for the units

See Also

DArch

---

sigmUnitFuncSwitch *Calculates the neuron output with the sigmoid function*

Description

Calculates the neuron output with the sigmoid function either from the real value or binary input saved in the list dataList.

Usage

```
sigmUnitFuncSwitch(rbm, dataList, biases, weights, runParams)
```

Arguments

rbm	A instance of the class RBM.
dataList	A list with the data matrices for the calculations.
biases	The biases for the calculations
weights	The weight matrix for the calculations
runParams	Parameters which indicates the status of the training. "actualCD" and "finishCD" are needed (see trainRBM)

Details

The return value is a list with the output of the sigmoid function as first entry and binary representation calculated through a comparison of the output with random numbers. The random numbers a generated with the function runif. If the parameter runParams["actualCD"] or runParams["finishCD"] is equal one, the calculation is made with the real value data (dataList[[1]]), otherwise with the binary representations (dataList[[2]]).

Value

The real value and binary activations for the units

See Also

DArch

`softmaxUnit` *Softmax unit function.*

Description

The function calculates the activation of the units and returns a list, in which the first entry is the result through the softmax transfer function.

Usage

```
softmaxUnit(data,weights)
```

Arguments

`data`	The data matrix for the calculation
`weights`	The weight and bias matrix for the calculation

Value

A list with the softmax activation in the first entry

See Also

DArch, sigmoidUnit, binSigmoidUnit, sigmoidUnitDerivative, linearUnit, linearUnitDerivative, softmaxUnitDerivative

---

`softmaxUnitDerivative` *Softmax unit function with unit derivatives.*

Description

The function calculates the activation of the units and returns a list, in which the first entry is the result through the softmax transfer function and the second entry is the derivative of the transfer function.

Usage

```
softmaxUnitDerivative(data,weights)
```

Arguments

`data`	The data matrix for the calculation
`weights`	The weight and bias matrix for the calculation

Value

A list with the softmax activation in the first entry and the derivative of the transfer function in the second entry

See Also

DArch, sigmoidUnit, binSigmoidUnit, sigmoidUnitDerivative, linearUnit, linearUnitDerivative, softmaxUnit

---

trainRBM *Trains a RBM with contrastive divergence*

Description

The function trains a restricted bolzmann machine (RBM) with the contrastive divergence method.

Usage

```
trainRBM(rbm,trainData,maxEpoch=1,numCD=1,...)

## S4 method for signature 'RBM'
trainRBM(rbm, trainData, maxEpoch = 1, numCD = 1, ...)
```

Arguments

rbm	A instance of the class RBM.
trainData	The data matrix for the training
maxEpoch	The number of training iterations
numCD	Number of contrastive divergence iterations
...	Additional parameters for the unit functions

Details

This function is build on the basis of the code from G. Hinton et. al. (http://www.cs.toronto.edu/~hinton/MatlabForSciencePa
- last visit 06.06.2013) for the pre training of deep belief nets. The original code is located in the files 'rbm.m' and 'rbmhidlinear.m'. It iterates in every epoche over the batches and calculates the updates for the weights. If it is the first CD iteration or the CD iterations are finished, the hidden units are calculated with the real value activations of the visible units, otherwise with the binary activations. To tell the unit functions the actual state of the training, the function generates a array with the following running parameters and passes them to the units: Maximal epochs: "maxEpoch", Actual epochs: "actualEpoch", Number of batches: "numBatches", Actual batch: "actualBatch", Maximal CD iterations: "numCD", Actual CD iteration: "actualCD", CD is finished: "finishCD". (see source code from sigmUnitFuncSwitch for an example).

See Also

RBM

Index