Package ‘darch’

July 20, 2016

Type Package
Title Package for Deep Architectures and Restricted Boltzmann Machines
Version 0.12.0
Date 2016-07-19
Author Martin Drees [aut, cre, cph],
      Johannes Rueckert [ctb],
      Christoph M. Friedrich [ctb],
      Geoffrey Hinton [cph],
      Ruslan Salakhutdinov [cph],
      Carl Edward Rasmussen [cph],
Maintainer Martin Drees <mdrees@stud.fh-dortmund.de>
Description The darch package is built on the basis of the code from G. E.
      Hinton and R. R. Salakhutdinov (available under Matlab Code for deep belief
      nets). 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 (2006) <DOI:10.1162/neco.2006.18.7.1527>) and ``Reducing the
      dimensionality of data with neural networks'' (G. E. Hinton, R. R.
      Salakhutdinov (2006) <DOI:10.1126/science.1127647>). This method includes a
      pre training with the contrastive divergence method published by G.E Hinton
      (2002) <DOI:10.1162/089976602760128018> and a fine tuning with common known
      training algorithms like backpropagation or conjugate gradients.
      Additionally, supervised fine-tuning can be enhanced with maxout and
      dropout, two recently developed techniques to improve fine-tuning for deep
      learning.
License GPL (>= 2) | file LICENSE
URL https://github.com/maddin79/darch
BugReports https://github.com/maddin79/darch/issues
Depends R (>= 3.0.0)
Imports stats, methods, ggplot2, reshape2, futile.logger (>= 1.4.1),
      caret, Rcpp (>= 0.12.3)
LinkingTo Rcpp
Suggests  foreach, doRNG, NeuralNetTools, gputools, testthat, plyr (>= 1.8.3.9000)

Collate  'RcppExports.R' 'autosave.R' 'net.Class.R' 'darch.Class.R'
'backpropagation.R' 'benchmark.R' 'bootstrap.R' 'caret.R'
'dataset.R' 'darch.Learn.R' 'darch.R' 'darch.Setter.R'
'darchUnitFunctions.R' 'dropout.R' 'errorFunctions.R'
'gbm.Class.R' 'generateRBMs.R' 'generateWeightsFunctions.R'
'loadDArch.R' 'log.R' 'makeStartEndPoints.R' 'minimize.R'
'minimizeAutoencoder.R' 'minimizeClassifier.R' 'mnist.R'
'momentum.R' 'net.Getter.R' 'newDArch.R' 'params.R' 'plot.R'
'rbmUnitFunctions.R' 'rbmUpdate.R' 'rpropagation.R'
'runDArch.R' 'saveDArch.R' 'test.R' 'util.R'
'weightUpdateFunctions.R'

RoxygenNote  5.0.1

NeedsCompilation  yes

Repository  CRAN

Date/Publication  2016-07-20 00:33:10

R topics documented:

backpropagation .................................................. 3
crossEntropyError .................................................. 4
darch ................................................................. 5
darchBench ........................................................... 13
darchModelInfo ...................................................... 14
darchTest ............................................................. 15
exponentialLinearUnit ............................................. 16
generateWeightsGlorotNormal .................................... 17
generateWeightsGlorotUniform ................................... 18
generateWeightsHeNormal ........................................ 19
generateWeightsHeUniform ....................................... 20
generateWeightsNormal .......................................... 21
generateWeightsUniform .......................................... 22
linearUnit ............................................................ 23
maxoutUnit ............................................................ 23
maxoutWeightUpdate ............................................... 25
minimizeAutoencoder .............................................. 26
minimizeClassifier ............................................... 27
mseError .............................................................. 29
plot.DArch ............................................................ 30
predict.DArch ........................................................ 31
print.DArch ........................................................... 32
provideMNIST ........................................................ 32
rectifiedLinearUnit ................................................. 33
backpropagation

Description
This function provides the backpropagation algorithm for deep architectures.

Usage
backpropagation(darch, trainData, targetData,
bp.learnRate = getParameter(".bp.learnRate", rep(1, times = length(darch@layers))),
bp.learnRateScale = getParameter(".bp.learnRateScale"),
nesterovMomentum = getParameter(".darch.nesterovMomentum"),
dropout = getParameter(".darch.dropout", rep(0, times = length(darch@layers) + 1), darch),
dropConnect = getParameter(".darch.dropout.dropConnect"),
matMult = getParameter(".matMult"),
debugMode = getParameter(".debug", F),
...)

Arguments
- darch: An instance of the class DArch.
- trainData: The training data (inputs).
- targetData: The target data (outputs).
- bp.learnRate: Learning rates for backpropagation, length is either one or the same as the number of weight matrices when using different learning rates for each layer.
- bp.learnRateScale: The learn rate is multiplied by this value after each epoch.
- nesterovMomentum: See darch.nesterovMomentum parameter of darch.
- dropout: See darch.dropout parameter of darch.
- dropConnect: See darch.dropout.dropConnect parameter of darch.
- matMult: Matrix multiplication function, internal parameter.
- debugMode: Whether debug mode is enabled, internal parameter.
- ... Further parameters.
The only backpropagation-specific, user-relevant parameters are \texttt{bp.learnRate} and \texttt{bp.learnRateScale}; they can be passed to the \texttt{darch} function when enabling backpropagation as the fine-tuning function. \texttt{bp.learnRate} defines the backpropagation learning rate and can either be specified as a single scalar or as a vector with one entry for each weight matrix, allowing for per-layer learning rates. \texttt{bp.learnRateScale} is a single scalar which contains a scaling factor for the learning rate(s) which will be applied after each epoch.

Backpropagation supports dropout and uses the weight update function as defined via the \texttt{darch.weightUpdateFunction} parameter of \texttt{darch}.

### Value

The trained deep architecture

### References


### See Also

\texttt{darch}

Other fine-tuning functions: \texttt{minimizeAutoencoder}, \texttt{minimizeClassifier}, \texttt{rpropagation}

### Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.fineTuneFunction = "backpropagation")
## End(Not run)
```

### Description

The function calculates the cross entropy error from the \texttt{original} and \texttt{estimate} parameters.

### Usage

\texttt{crossEntropyError(original, estimate)}

### Arguments

- \texttt{original} The original data matrix.
- \texttt{estimate} The calculated data matrix.
darch

Details

This function can be used for the darch.errorFunction parameter of the darch function, but is only a valid error function if used in combination with the softmaxUnit activation function! It is not a valid value for the parameter rbm.errorFunction.

Value

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

References


See Also

Other error functions: mseError, rmseError

Examples

## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.errorFunction = "crossEntropyError")

## End(Not run)

---

**darch**

*Fit a deep neural network*

Description

Fit a deep neural network with optional pre-training and one of various fine-tuning algorithms.

Usage

darch(x, ...)

## Default S3 method:
darch(x, y, layers = 10, ..., autosave = F,
autosave.epochs = round(darch.numEpochs/20),
autosave.dir = "./darch.autosave", autosave.trim = F, bp.learnRate = 1,
bp.learnRateScale = 1, bootstrap = F, bootstrap.unique = T,
bootstrap.num = 0, cg.length = 2, cg.switchLayers = 1, darch = NULL,
darch.batchSize = 1, darch.dither = F, darch.dropout = 0,
darch.dropout.dropConnect = F, darch.dropout.momentMatching = 0,
darch.dropout.oneMaskPerEpoch = F, darch.elu.alpha = 1,
darch.errorFunction = if (darch.isClass) crossEntropyError else mseError,
darch.finalMomentum = 0.9, darch.fineTuneFunction = backpropagation,
darch.initialMomentum = 0.5, darch.isClass = T,
darch.maxout.poolSize = 2, darch.maxout.unitFunction = linearUnit,
darch.momentumRampLength = 1, darch.nesterovMomentum = T,
darch.numEpochs = 100, darch.returnBestModel = T,
darch.returnBestModel.validationErrorFactor = 1 - exp(-1),
darch.stopClassErr = -Inf, darch.stopErr = -Inf,
darch.stopValidClassErr = -Inf, darch.stopValidErr = -Inf,
darch.trainLayers = T, darch.unitFunction = sigmoidUnit,
darch.weightDecay = 0,
darch.weightUpdateFunction = weightDecayWeightUpdate, dataSet = NULL,
dataSetValid = NULL,
generateWeightsFunction = generateWeightsGlorotUniform, gputools = F,
gputools.deviceId = 0, logLevel = NULL, normalizeWeights = F,
normalizeWeightsBound = 15, paramsList = list(),
preProc.factorToNumeric = F, preProc.factorToNumeric.targets = F,
preProc.fullRank = T, preProc.fullRank.targets = F,
preProc.orderedToFactor.targets = T, preProc.params = F,
preProc.targets = F, rbm.allData = F, rbm.batchSize = 1,
rbm.consecutive = T, rbm.errorFunction = mseError,
rbm.finalMomentum = 0.9, rbm.initialMomentum = 0.5, rbm.lastLayer = 0,
rbm.learnRate = 1, rbm.learnRateScale = 1, rbm.momentumRampLength = 1,
rbm.numCD = 1, rbm.numEpochs = 0, rbm.unitFunction = sigmoidUnitRbm,
rbm.updateFunction = rbmUpdate, rbm.weightDecay = 2e-04, retainData = F,
rprop.decFact = 0.5, rprop.incFact = 1.2, rprop.initDelta = 1/80,
rprop.maxDelta = 50, rprop.method = "iRprop+", rprop.minDelta = 1e-06,
seed = NULL, shuffleTrainData = T, weights.max = 0.1,
weights.mean = 0, weights.min = -0.1, weights.sd = 0.01,
xValid = NULL, yValid = NULL)

## S3 method for class 'formula'
darch(x, data, layers, ..., xValid = NULL, dataSet = NULL,
dataSetValid = NULL, logLevel = NULL, paramsList = list(),
darch = NULL)

## S3 method for class 'Dataset'
darch(x, ...)

### Arguments

- **x**
  - Input data matrix or **data.frame** (darch.default) or **formula** (darch.formula) or **Dataset** (darch.Dataset).
- **...**
  - Additional parameters.
- **y**
  - Target data matrix or **data.frame**, if x is an input data matrix or **data.frame**.
- **layers**
  - Vector containing one integer for the number of neurons of each layer. Defaults to c(a, 10, b), where a is the number of columns in the training data and b the number of columns in the targets. If this has length 1, it is used as the number of neurons in the hidden layer, not as the number of layers!
**autosave** Logical indicating whether to activate automatically saving the **DArch** instance to a file during fine-tuning.

**autosave.epochs** After how many epochs should auto-saving happen, by default after every 5 1, the network will only be saved once when the fine-tuning is done.

**autosave.dir** Directory for the autosave files, the file names will be e.g. autosave_010.net for the DArch instance after 10 epochs

**autosave.trim** Whether to trim the network before saving it. This will remove the dataset and the layer weights, resulting in a network that is no longer usable for predictions or training. Useful when only statistics and settings need to be stored.

**bp.learnRate** Learning rates for backpropagation, length is either one or the same as the number of weight matrices when using different learning rates for each layer.

**bp.learnRateScale** The learn rate is multiplied by this value after each epoch.

**bootstrap** Logical indicating whether to use bootstrapping to create a training and validation data set from the given training data.

**bootstrap.unique** Logical indicating whether to take only unique samples for the training (TRUE, default) or take all drawn samples (FALSE), which will result in a bigger training set with duplicates. **Note:** This is ignored if bootstrap.num is greater than 0.

**bootstrap.num** If this is greater than 0, bootstrapping will draw this number of training samples without replacement.

**cg.length** Numbers of line search

**cg.switchLayers** Indicates when to train the full network instead of only the upper two layers

**darch** Existing **DArch** instance for which training is to be resumed. **Note:** When enabling pre-training, previous training results will be lost, see explanation for parameter rbm.numEpochs.

**darch.batchSize** Batch size, i.e. the number of training samples that are presented to the network before weight updates are performed, for fine-tuning.

**darch.dither** Whether to apply **dither** to numeric columns in the training input data.

**darch.dropout** Dropout rates. If this is a vector it will be treated as the dropout rates for each individual layer. If one element is missing, the input dropout will be set to 0. When enabling darch.dropout.dropConnect, this vector needs an additional element (one element per weight matrix between two layers as opposed to one element per layer excluding the last layer).

**darch.dropout.dropConnect** Whether to use DropConnect instead of dropout for the hidden layers. Will use darch.dropout as the dropout rates.

**darch.dropout.momentMatching** How many iterations to perform during moment matching for dropout inference, 0 to disable moment matching.
darch.dropout.oneMaskPerEpoch
Whether to generate a new mask for each batch (FALSE, default) or for each epoch (TRUE).

darch.elu.alpha
Alpha parameter for the exponential linear unit function. See exponentialLinearUnit.

darch.errorFunction
Error function during fine-tuning. Possible error functions include mseError, rmseError, and crossEntropyError.

darch.finalMomentum
Final momentum during fine-tuning.

darch.finetuneFunction
Fine-tuning function. Possible values include backpropagation (default), rpropagation, minimizeClassifier and minimizeAutoencoder (unsupervised).

darch.initialMomentum
Initial momentum during fine-tuning.

darch.isClass
Whether output should be treated as class labels during fine-tuning and classification rates should be printed.

darch.maxout.poolSize
Pool size for maxout units, when using the maxout activation function. See maxoutUnit.

darch.maxout.unitFunction
Inner unit function used by maxout. See darch.unitFunction for possible unit functions.

darch.momentumRampLength
After how many epochs, relative to the overall number of epochs trained, should the momentum reach darch.finalMomentum? A value of 1 indicates that the darch.finalMomentum should be reached in the final epoch, a value of 0.5 indicates that darch.finalMomentum should be reached after half of the training is complete. Note that this will lead to bumps in the momentum ramp if training is resumed with the same parameters for darch.initialMomentum and darch.finalMomentum. Set darch.momentumRampLength to 0 to avoid this problem when resuming training.

darch.nesterovMomentum
Whether to use Nesterov Accelerated Momentum. (NAG) for gradient descent based fine-tuning algorithms.

darch.numEpochs
Number of epochs of fine-tuning.

darch.returnBestModel
Logical indicating whether to return the best model at the end of training, instead of the last.

darch.returnBestModel.validationErrorFactor
When evaluating models with validation data, how high should the validation error be valued, compared to the training error? This is a value between 0 and 1. By default, this value is 1 - exp(-1). The training error factor and the validation error factor will always add to 1, so if you pass 1 here, the training error will be ignored, and if you pass 0 here, the validation error will be ignored.
Darch

**darch.stopClassErr**
When the classification error is lower than or equal to this value, training is stopped (0..100).

**darch.stopErr**
When the value of the error function is lower than or equal to this value, training is stopped.

**darch.stopValidClassErr**
When the classification error on the validation data is lower than or equal to this value, training is stopped (0..100).

**darch.stopValidErr**
When the value of the error function on the validation data is lower than or equal to this value, training is stopped.

**darch.trainLayers**
Either TRUE to train all layers or a mask containing TRUE for all layers which should be trained and FALSE for all layers that should not be trained (no entry for the input layer).

**darch.unitFunction**
Layer function or vector of layer functions of length number of layers - 1. Note that the first entry signifies the layer function between layers 1 and 2, i.e. the output of layer 2. Layer 1 does not have a layer function, since the input values are used directly. Possible unit functions include `linearUnit`, `sigmoidUnit`, `tanhUnit`, `rectifiedLinearUnit`, `softplusUnit`, `softmaxUnit`, and `maxoutUnit`.

**darch.weightDecay**
Weight decay factor, defaults to 0. All weights will be multiplied by (1 - darch.weightDecay) prior to each weight update.

**darch.weightUpdateFunction**
Weight update function or vector of weight update functions, very similar to darch.unitFunction. Possible weight update functions include `weightDecayWeightUpdate` and `maxoutWeightUpdate`. Note that `maxoutWeightUpdate` must be used on the layer after the maxout activation function!

**dataSet**
`DataSet` instance, passed from darch.DataSet(), may be specified manually.

**dataSetValid**
`DataSet` instance containing validation data.

**generateWeightsFunction**
Weight generation function or vector of layer generation functions of length number of layers - 1. Possible weight generation functions include `generateWeightsUniform` (default), `generateWeightsNormal`, `generateWeightsGlorotNormal`, `generateWeightsGlorotUniform`, `generateWeightsHeNormal`, and `generateWeightsHeUniform`.

**gputools**
Logical indicating whether to use gputools for matrix multiplication, if available.

**gputools.deviceId**
Integer specifying the device to use for GPU matrix multiplication. See `chooseGpu`.

**logLevel**
`futile.logger` log level. Uses the currently set log level by default, which is `futile.logger::flog.info` if it was not changed. Other available levels include, from least to most verbose: FATAL, ERROR, WARN, DEBUG, and TRACE.

**normalizeWeights**
Logical indicating whether to normalize weights (L2 norm = normalizeWeightsBound).
normalizeWeightsBound
Upper bound on the L2 norm of incoming weight vectors. Used only if normalizeWeights is TRUE.

paramsList
List of parameters, can include and does overwrite specified parameters listed above. Primary for convenience or for use in scripts.

preProc.factorToNumeric
Whether all factors should be converted to numeric.

preProc.factorToNumeric.targets
Whether all factors should be converted to numeric in the target data.

preProc.fullRank
Whether to use full rank encoding. See preProcess for details.

preProc.fullRank.targets
Whether to use full rank encoding for target data. See preProcess for details.

preProc.orderedToFactor.targets
Whether ordered factors in the target data should be converted to unordered factors. Note: Ordered factors are converted to numeric by dummyVars and no longer usable for classification tasks.

preProc.params
List of parameters to pass to the preProcess function for the input data or FALSE to disable input data pre-processing.

preProc.targets
Whether target data is to be centered and scaled. Unlike preProc.params, this is just a logical turning pre-processing for target data on or off, since this pre-processing has to be reverted when predicting new data. Most useful for regression tasks. Note: This will skew the raw network error.

rbm.allData
Logical indicating whether to use training and validation data for pre-training. Note: This also applies when using bootstrapping.

rbm.batchSize
Pre-training batch size.

rbm.consecutive
Logical indicating whether to train the RBMs one at a time for rbm.numEpochs epochs (TRUE, default) or alternatingly training each RBM for one epoch at a time (FALSE).

rbm.errorFunction
Error function during pre-training. This is only used to estimate the RBM error and does not affect the training itself. Possible error functions include mseError and rmseError.

rbm.finalMomentum
Final momentum during pre-training.

rbm.initialMomentum
Initial momentum during pre-training.

rbm.lastLayer
Numeric indicating at which layer to stop the pre-training. Possible values include 0, meaning that all layers are trained; positive integers, meaning to stop training after the RBM where rbm.lastLayer forms the visible layer; negative integers, meaning to stop the training at rbm.lastLayer RBMs from the top RBM.

rbm.learnRate
Learning rate during pre-training.
**darch**

**rbm.learnRateScale**
The learn rates will be multiplied with this value after each epoch.

**rbm.momentumRampLength**
After how many epochs, relative to rbm.numEpochs, should the momentum reach rbm.finalMomentum? A value of 1 indicates that the rbm.finalMomentum should be reached in the final epoch, a value of 0.5 indicates that rbm.finalMomentum should be reached after half of the training is complete.

**rbm.numCD**
Number of full steps for which contrastive divergence is performed. Increasing this will slow training down considerably.

**rbm.numEpochs**
Number of pre-training epochs. **Note:** When passing a value other than 0 here and also passing an existing DArch instance via the darch parameter, the weights of the network will be completely reset! Pre-training is essentially a form of advanced weight initialization and it makes no sense to perform pre-training on a previously trained network.

**rbm.unitFunction**
Unit function during pre-training. Possible functions include sigmoidUnitRbm (default), tanhUnitRbm, and linearUnitRbm.

**rbm.updateFunction**
Update function during pre-training. Currently, darch only provides rbmUpdate.

**rbm.weightDecay**
Pre-training weight decay. Weights will be multiplied by (1 - rbm.weightDecay) prior to each weight update.

**retainData**
Logical indicating whether to store the training data in the DArch instance after training or when saving it to disk.

**rprop.decFact**
Decreasing factor for the training. Default is 0.6.

**rprop.incFact**
Increasing factor for the training. Default is 1.2.

**rprop.initDelta**
Initialisation value for the update. Default is 0.0125.

**rprop.maxDelta**
Upper bound for step size. Default is 50.

**rprop.method**
The method for the training. Default is "iRprop+"

**rprop.minDelta**
Lower bound for step size. Default is 0.000001

**seed**
Allows the specification of a seed which will be set via set.seed. Used in the context of darchBench.

**shuffleTrainData**
Logical indicating whether to shuffle training data before each epoch.

**weights.max**
Max parameter to the runif function.

**weights.mean**
Mean parameter to the rnorm function.

**weights.min**
Min parameter to the runif function.

**weights.sd**
Sd parameter to the rnorm function.

**xValid**
Validation input data matrix or data.frame.

**yValid**
Validation target data matrix or data.frame, if x is a data matrix or data.frame.

**data**
Data.frame containing the dataset, if x is a formula.
Details

The darch package implements Deep Architecture Networks and restricted Boltzmann machines. 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 learning methods like backpropagation, resilient backpropagation and conjugate gradients as well as more recent techniques like dropout and maxout.

See https://github.com/maddin79/darch for further information, documentation, and releases.

Value

Fitted DArch instance

Author(s)

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

References


See Also

Other darch interface functions: darchBench, darchTest, plot.DArch, predict.DArch, print.DArch
Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris)
print(model)
predictions <- predict(model, newdata = iris, type = "class")
cat(paste("Incorrect classifications:", sum(predictions != iris[,5])))

trainData <- matrix(c(0,0,0,1,1,0,1,1, ncol = 2, byrow = TRUE)
trainTargets <- matrix(c(0,1,1,0, nrow = 4)
model2 <- darch(trainData, trainTargets, layers = c(2, 10, 1),
   darch.numEpochs = 500, darch.stopClassErr = 0, retainData = T)
e <- darchTest(model2)
cat(paste("Incorrect classifications on all examples: ", e[3], " (",
e[2], ",%)

plot(model2)

## End(Not run)
```

## More examples can be found at

# https://github.com/maddin79/darch/tree/v0.12.0/examples

---

darchBench

**Benchmarking wrapper for darch**

### Description

Simple benchmarking function which wraps around the `darch` function for users who can’t or don’t want to use the caret package for benchmarking. This function requires the `foreach` package to work, and will perform parallel benchmarks if an appropriate backend was registered beforehand.

### Usage

```r
darchBench(..., bench.times = 1, bench.save = F, bench.dir = ".darch.benchmark", bench.continue = T, bench.delete = F, bench.seed = NULL, output.capture = bench.save, logLevel = NULL)
```

### Arguments

- `...` Parameters to the `darch` function
- `bench.times` How many benchmark runs to perform
- `bench.save` Whether to save benchmarking results to a directory
- `bench.dir` Path (relative or absolute) including directory where benchmark results are saved if bench.save is true
**bench.continue**  Whether the benchmark is to be continued from an earlier run. If TRUE, existing benchmark results are looked for in the directory given in bench.dir and new results are appended. If both this and bench.continue are FALSE and the directory given in bench.dir does already exist, the training will be aborted with an error.

**bench.delete**  Whether to delete the contents of bench.dir if bench.continue is FALSE. Caution: This will attempt to delete ALL files in the given directory, use at your own risk!

**bench.seeds**  Vector of seeds, one for each run. Will be passed to darch.

**output.capture**  Whether to capture R output in .Rout files in the given directory. This is the only way of gaining access to the R output since the foreach loop will not print anything to the console. Will be ignored if bench.save is FALSE.

**logLevel**  futile.logger log level. Uses the currently set log level by default, which is futile.logger::flog.info if it was not changed. Other available levels include, from least to most verbose: FATAL, ERROR, WARN, DEBUG, and TRACE.

**Value**

List of DArch instances; the results of each call to darch.

**See Also**

Other darch interface functions: darchTest, darch.plot.DArch, predict.DArch, print.DArch

**Examples**

```r
## Not run:
data(iris)
modellist <- darchBench(Species ~ ., iris, c(0, 50, 0),
                         preProc.params = list(method = c("center", "scale")),
                         darch.unitFunction = c("sigmoidUnit", "softmaxUnit"),
                         darch.numEpochs = 30, bench.times = 10, bench.save = T)

## End(Not run)
```

---

**darchModelInfo**  Creates a custom caret model for darch.

**Description**

This function creates a caret model description to enable training DArch instances with the `train` function. See the documentation on custom caret models for further information and examples on how to create valid params and grid values.

**Usage**

darchModelInfo(params = NULL, grid = NULL)
Arguments

params  
data.frame of parameters or NULL to use a simple default (bp.learnRate).

grid  
Function which produces a data.frame containing a grid of parameter combinations or NULL to use a simple default.

Value

A valid caret model which can be passed to train.

See Also

Caret custom models

Examples

```r
## Not run:
data(iris)
tc <- trainControl(method = "boot", number = 5, allowParallel = F,
                     verboseIter = T)

parameters <- data.frame(parameter = c("layers", "bp.learnRate", "darch.unitFunction"),
                           class = c("character", "numeric", "character"),
                           label = c("Network structure", "Learning rate", "unitFunction"))

grid <- function(x, y, len = NULL, search = "grid")
{
  df <- expand.grid(layers = c("c(0,20,0)","c(0,10,10,0)","c(0,10,5,5,0)"),
                    bp.learnRate = c(1,2,5,10))

  df[["darch.unitFunction"]]<- rep(c("c(tanhUnit, softmaxUnit)",
                                     "c(tanhUnit, tanhUnit, softmaxUnit)",
                                     "c(tanhUnit, tanhUnit, tanhUnit, softmaxUnit)")
                                      ,4)

  df
}
caretModel <- train(Species ~ ., data = iris, tuneLength = 12, trControl = tc,
                     method = darchModelInfo(parameters, grid), preProc = c("center", "scale"),
                     darch.numEpochs = 15, darch.batchSize = 6, testing = T, ...)

## End(Not run)
```

darchTest  
Test classification network.

Description

Forward-propagate given data through the deep neural network and return classification accuracy using the given labels.
exponentialLinearUnit

Usage

darchTest(darch, newdata = NULL, targets = T)

Arguments

darch  
   DArch instance.
newdata  
   New data to use, NULL to use training data.
targets  
   Labels for the data, NULL to use training labels (only possible when data is NULL as well).

Details

This is primarily a convenience function similar to predict.DArch with classification performance measurements instead of network output, and it returns a list of accuracy indicators (raw network error, percentage of incorrect classifications and absolute number of incorrect classifications).

Value

Vector containing error function output, percentage of incorrect classifications and absolute number of incorrect classifications.

See Also

Other darch interface functions: darchBench, darch, plot.DArch, predict.DArch, print.DArch

Examples

## Not run:
data(iris)
model <- darch(Species ~ ., iris, retainData = T)
classificationStats <- darchTest(model)

## End(Not run)

exponentialLinearUnit  

Exponential linear unit (ELU) function with unit derivatives.

Description

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

Usage

exponentialLinearUnit(input, alpha = getParameter(".darch.elu.alpha", 1, ...), ...)
**generateWeightsGlorotNormal**

**Arguments**

input  
Input for the activation function.

alpha  
ELU hyperparameter.

...  
Additional parameters.

**Value**

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

**References**


**See Also**

Other darch unit functions: `linearUnit`, `maxoutUnit`, `rectifiedLinearUnit`, `sigmoidUnit`, `softmaxUnit`, `softplusUnit`, `tanhUnit`

**Examples**

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.unitFunction = "exponentialLinearUnit",
              darch.elu.alpha = 2)
## End(Not run)
```

---

**generateWeightsGlorotNormal**

*Glorot normal weight initialization*

**Description**

This function is used to generate random weights and biases using Glorot normal weight initialization as described in Glorot & Bengio, AISTATS 2010.

**Usage**

```r
generateWeightsGlorotNormal(numUnits1, numUnits2, weights.mean = getParameter(".weights.mean", 0, ...), ...)
```
generateWeightsGlorotUniform

Arguments

numUnits1 Number of units in the lower layer.
numUnits2 Number of units in the upper layer.
weights.mean mean parameter to the rnorm function.
... Additional parameters, used for parameter resolution and passed to generateWeightsNormal.

Value

Weight matrix.

References


See Also

Other weight generation functions: generateWeightsGlorotUniform, generateWeightsHeNormal, generateWeightsHeUniform, generateWeightsNormal, generateWeightsUniform

Examples

## Not run:
data(iris)
model <- darch(Species ~ ., iris, generateWeightsFunction = "generateWeightsGlorotNormal",
               weights.mean = .1)

## End(Not run)

generateWeightsGlorotUniform

Glorot uniform weight initialization

Description

This function is used to generate random weights and biases using Glorot uniform weight initialization as described in Glorot & Bengio, AISTATS 2010.

Usage

generateWeightsGlorotUniform(numUnits1, numUnits2, ...)

Arguments

numUnits1 Number of units in the lower layer.
numUnits2 Number of units in the upper layer.
... Additional parameters, used for parameter resolution and passed to generateWeightsUniform.
Value

Weight matrix.

References


See Also

Other weight generation functions: generateWeightsGlorotNormal, generateWeightsHeNormal, generateWeightsHeUniform, generateWeightsNormal, generateWeightsUniform

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, generateWeightsFunction = "generateWeightsGlorotUniform")
## End(Not run)
```

---

**generateWeightsHeNormal**

*He normal weight initialization*

Description

This function is used to generate random weights and biases using He normal weight initialization as described in He et al., [http://arxiv.org/abs/1502.01852](http://arxiv.org/abs/1502.01852).

Usage

```r
generateWeightsHeNormal(numUnits1, numUnits2,
weights.mean = getParameter(“.weights.mean”, 0, ...), ...)
```

Arguments

- `numUnits1` Number of units in the lower layer.
- `numUnits2` Number of units in the upper layer.
- `weights.mean` Mean parameter to the `rnorm` function.
- `...` Additional parameters, used for parameter resolution and passed to `generateWeightsNormal`.

Value

Weight matrix.
References


See Also

Other weight generation functions: generateWeightsGlorotNormal, generateWeightsGlorotUniform, generateWeightsHeUniform, generateWeightsNormal, generateWeightsUniform

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, generateWeightsFunction = "generateWeightsHeNormal",
               weights.mean = .1)

## End(Not run)
```

**generateWeightsHeUniform**

*He uniform weight initialization*

Description

This function is used to generate random weights and biases using He uniform weight initialization as described in He et al., [http://arxiv.org/abs/1502.01852](http://arxiv.org/abs/1502.01852).

Usage

`generateWeightsHeUniform(numUnits1, numUnits2, ...)`

Arguments

- `numUnits1` Number of units in the lower layer.
- `numUnits2` Number of units in the upper layer.
- `...` Additional parameters, used for parameter resolution and passed to `generateWeightsUniform`.

Value

Weight matrix.

References

generateWeightsNormal

See Also

Other weight generation functions: generateWeightsGlorotNormal, generateWeightsGlorotUniform, generateWeightsHeNormal, generateWeightsNormal, generateWeightsUniform

Examples

## Not run:
data(iris)
model <- darch(Species ~ ., iris, generateWeightsFunction = "generateWeightsHeUniform")

## End(Not run)

generateWeightsNormal  Generates a weight matrix using rnorm.

Description

This function is the standard method for generating weights for instances of Net. It uses rnorm to do so.

Usage

generateWeightsNormal(numUnits1, numUnits2,
  weights.mean = getParameter(".weights.mean", 0, ...),
  weights.sd = getParameter(".weights.sd", 0.01, ...), ...)

Arguments

numUnits1    Number of units in the lower layer.
numUnits2    Number of units in the upper layer.
weights.mean mean parameter to the rnorm function.
weights.sd   sd parameter to the rnorm function.
...          Additional parameters, used for parameter resolution.

Value

Weight matrix.

See Also

Other weight generation functions: generateWeightsGlorotNormal, generateWeightsGlorotUniform, generateWeightsHeNormal, generateWeightsHeUniform, generateWeightsUniform
generateWeightsUniform

Generates a weight matrix using runif

Description

This function is used to generate random weights and biases using runif.

Usage

generateWeightsUniform(numUnits1, numUnits2,
weights.min = getParameter(".weights.min", -0.1, ...),
weights.max = getParameter(".weights.max", 0.1, ...), ...)

Arguments

numUnits1 Number of units in the lower layer.
numUnits2 Number of units in the upper layer.
weights.min min parameter to the runif function.
weights.max max parameter to the runif function.
... Additional parameters, used for parameter resolution.

Value

Weight matrix.

See Also

Other weight generation functions: generateWeightsGlorotNormal, generateWeightsGlorotUniform, generateWeightsHeNormal, generateWeightsHeUniform, generateWeightsNormal

Examples

## Not run:
data(iris)
model <- darch(Species ~ ., iris, generateWeightsFunction = "generateWeightsNormal",
weights.mean = .1, weights.sd = .05)

## End(Not run)
linearUnit

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

linearUnit(input, ...)

Arguments

input

Input for the activation function.

... Additional parameters, not used.

Value

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

See Also

Other darch unit functions: exponentialLinearUnit, maxoutUnit, rectifiedLinearUnit, sigmoidUnit, softmaxUnit, softplusUnit, tanhUnit

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.unitFunction = "linearUnit")
## End(Not run)
```

maxoutUnit

Maxout / LWTA 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 maxout transfer function and the second entry is the derivative of the transfer function.
maxoutUnit

Usage

maxoutUnit(input, ..., poolSize = getParameter(".darch.maxout.poolSize", 2, ...), unitFunc = getParameter(".darch.maxout.unitFunction", linearUnit, ...), dropoutMask = vector())

Arguments

- **input**: Input for the activation function.
- **...**: Additional parameters, passed on to inner unit function.
- **poolSize**: The size of each maxout pool.
- **unitFunc**: Inner unit function for maxout.
- **dropoutMask**: Vector containing the dropout mask.

Details

Maxout sets the activations of all neurons but the one with the highest activation within a pool to 0. If this is used without `maxoutWeightUpdate`, it becomes the local-winner-takes-all algorithm, as the only difference between the two is that outgoing weights are shared for maxout.

Value

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

References


See Also

Other darch unit functions: `exponentialLinearUnit`, `linearUnit`, `rectifiedLinearUnit`, `sigmoidUnit`, `softmaxUnit`, `softplusUnit`, `tanhUnit`

Examples

```r
## Not run:
data(iris)
# LWTA:
model <- darch(Species ~ ., iris, c(0, 50, 0),
darch.unitFunction = c("maxoutUnit", "softmaxUnit"),
darch.maxout.poolSize = 5, darch.maxout.unitFunction = "sigmoidUnit")
```
maxoutWeightUpdate

 Updates the weight on maxout layers

Description
On maxout layers, only the weights of active units are altered, additionally all weights within a pool must be the same.

Usage
maxoutWeightUpdate(darch, layerIndex, weightsInc, biasesInc, ..., weightDecay = getParameter(".darch.weightDecay", 0, darch), poolSize = getParameter(".darch.maxout.poolSize", 2, darch))

Arguments
darch DArch instance.
layerIndex Layer index within the network.
weightsInc Matrix containing scheduled weight updates from the fine-tuning algorithm.
biasesInc Bias weight updates.
... Additional parameters, not used.
weightDecay Weights are multiplied by (1 - weightDecay) before each update. Corresponds to the darch.weightDecay parameter of darch.default.
poolSize Size of maxout pools, see parameter darch.maxout.poolSize of darch.

Value
The updated weights.

References

See Also
Other weight update functions: weightDecayWeightUpdate
Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, c(0, 50, 0),
darch.unitFunction = c("maxOutUnit", "softmaxUnit"),
darch.maxout.poolSize = 5, darch.maxout.unitFunction = "sigmoidUnit",
darch.weightupdatefunction = c("weightDecayWeightUpdate", "maxOutWeightUpdate"))
## End(Not run)
```

---

minimizeAutoencoder  Conjugate gradient for an autoencoder network

Description

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

Usage

```r
minimizeAutoencoder(darch, trainData, targetData,
cg.length = getParameter(".cg.length"),
dropout = getParameter(".darch.dropout"),
dropConnect = getParameter(".darch.dropout.dropConnect"),
matMult = getParameter(".matMult"), debugMode = getParameter(".debug"),
...)
```

Arguments

- **darch**: A instance of the class DArch.
- **trainData**: The training data matrix.
- **targetData**: The labels for the training data.
- **cg.length**: Numbers of line search
- **dropout**: See darch.dropout parameter of darch.
- **dropConnect**: See darch.dropout.dropConnect parameter of darch.
- **matMult**: Matrix multiplication function, internal parameter.
- **debugMode**: Whether debug mode is enabled, internal parameter.
- **...**: Further parameters.

Details

This function is built on the basis of the code from G. Hinton et. al. ([link](http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html) - last visit 2016-04-30) 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 link](http://www.gatsby.ucl.ac.uk/~edward/code/minimize/) - last visit 2016-04-30) to R.
minimizeAutoencoder supports dropout but does not use the weight update function as defined via the darch.weightUpdateFunction parameter of darch, so that weight decay, momentum etc. are not supported.

Value
The trained DArch object.

See Also
darch, fineTuneDArch
Other fine-tuning functions: backpropagation, minimizeClassifier, rpropagation

Examples
## Not run:
data(iris)
model <- darch(Species ~ ., iris, c(6,10,2,10,6), darch.isClass = F,
preProc.params = list(method=c("center", "scale")),
darch.numEpochs = 20, darch.batchSize = 6, darch.unitFunction = tanhUnit
darch.finetuneFunction = "minimizeAutoencoder")

## End(Not run)

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,
   cg.length = getParameter(".cg.length"),
   cg.switchLayers = getParameter(".cg.length"),
   dropout = getParameter(".darch.dropout"),
   dropConnect = getParameter(".darch.dropout.dropConnect"),
   matMult = getParameter(".matMult"), debugMode = getParameter(".debug"),
   ...
)

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

cg.switchLayers
Indicates when to train the full network instead of only the upper two layers.

dropout
See darch.dropout parameter of darch.
dropConnect
See darch.dropout.dropConnect parameter of darch.
matMult
Matrix multiplication function, internal parameter.
debugMode
Whether debug mode is enabled, internal parameter.
...
Further parameters.

Details

This function is build on the basis of the code from G. Hinton et. al. (http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html - last visit 2016-04-30) 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 2016-04-30) to R. The parameter cg.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.

minimizeClassifier supports dropout but does not use the weight update function as defined via the darch.weightUpdateFunction parameter of darch, so that weight decay, momentum etc. are not supported.

Value

The trained DArch object.

See Also

darch, fineTuneDArch

Other fine-tuning functions: backpropagation, minimizeAutoencoder, rpropagation

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, 
preProc.params = list(method = c("center", "scale")),
darch.unitFunction = c("sigmoidUnit", "softmaxUnit"),
darch.fineTuneFunction = "minimizeClassifier",
cg.length = 3, cg.switchLayers = 5)

## End(Not run)
```
mseError

Mean squared error function

Description

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

Usage

mseError(original, estimate)

Arguments

original The original data matrix.
estimate The calculated data matrix.

Details

This function is a valid value for both darch parameters rbm.errorFunction and darch.errorFunction.

Value

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

See Also

Other error functions: crossEntropyError, rmseError

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, rbm.errorFunction = "mseError",
               darch.errorFunction = "mseError")

## End(Not run)
```
plot.DArch

Plot DArch statistics or structure.

Description
This function provides different plots depending on the type parameter:

Usage
```r
## S3 method for class 'DArch'
plot(x, y = "raw", ..., type = y)
```

Arguments
- `x` : DArch instance.
- `y` : See type.
- `...` : Additional parameters, passed to plotting functions.
- `type` : Which type of plot to create, one of raw, class, time, momentum, and net.

Details
- raw. Plots the raw network error (e.g. MSE), this is the default
- class. Plots the classification error
- time. Plots the times needed for each epoch
- momentum. Plots the momentum ramp
- net. Calls `plotnet` to plot the network

Value
The plotted graph.

See Also
Other darch interface functions: `darchBench`, `darchTest`, `darch`, `predict.DArch`, `print.DArch`

Examples
```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris)
plot(model)
plot(model, "class")
plot(model, "time")
plot(model, "momentum")
plot(model, "net")

## End(Not run)
```
predict.DArch

Forward-propagate data.

Description
Forward-propagate given data through the deep neural network.

Usage
## S3 method for class 'DArch'
predict(object, ..., newdata = NULL, type = "raw",
inputLayer = 1, outputLayer = 0)

Arguments
- **object**  
  DArch instance
- **...**  
  Further parameters, if newdata is NULL, the first unnamed parameter will be used for newdata instead.
- **newdata**  
  New data to predict, NULL to return latest network output
- **type**  
  Output type, one of: raw, bin, class, or character. raw returns the layer output, bin returns 1 for every layer output > 0.5, 0 otherwise, and class returns 1 for the output unit with the highest activation, otherwise 0. Additionally, when using class, class labels are returned when available. character is the same as class, except using character vectors instead of factors.
- **inputLayer**  
  Layer number (> 0). The data given in newdata will be fed into this layer. Note that absolute numbers count from the input layer, i.e. for a network with three layers, 1 would indicate the input layer.
- **outputLayer**  
  Layer number (if > 0) or offset (if <= 0) relative to the last layer. The output of the given layer is returned. Note that absolute numbers count from the input layer, i.e. for a network with three layers, 1 would indicate the input layer.

Value
Vector or matrix of networks outputs, output type depending on the type parameter.

See Also
Other darch interface functions: darchBench, darchTest, darch, plot.DArch, print.DArch

Examples
## Not run:
data(iris)
model <- darch(Species ~ ., iris, retainData = T)
predict(model)

## End(Not run)
Description
Print verbose information about a DArch instance.

Usage
## S3 method for class 'DArch'
print(x, ...)

Arguments
- x: DArch instance
- ...: Further parameters, not used.

Details
Information printed include darch parameters and a summary of training statistics.

See Also
Other darch interface functions: darchBench, darchTest, darch, plot.DArch, predict.DArch

Examples
## Not run:
data(iris)
model <- darch(Species ~ ., iris)
print(model)

## End(Not run)

Description
Provides MNIST data set in the given folder.

Usage
provideMNIST(folder = "data/", download = F)
Arguments

folder    Folder name, including a trailing slash.
download Logical indicating whether download is allowed.

Value

Boolean value indicating success or failure.

Examples

```r
## Not run:
provideMNIST("mnist/", download = T)
```

## End(Not run)

### rectifiedLinearUnit

Rectified 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 rectified linear activation of the units and the second entry is the derivative of the transfer function.

Usage

```r
rectifiedLinearUnit(input, ...)
```

Arguments

input Input for the activation function.
... Additional parameters, not used.

Value

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

References

See Also

Other darch unit functions: exponentialLinearUnit, linearUnit, maxoutUnit, sigmoidUnit, softmaxUnit, softplusUnit, tanhUnit

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.unitFunction = "rectifiedLinearUnit")

## End(Not run)
```

---

#### rmseError

**Root-mean-square error function**

**Description**

The function calculates the root-mean-square error (RMSE) from the original and estimate parameters.

**Usage**

```r
rmseError(original, estimate)
```

**Arguments**

- `original`: The original data matrix.
- `estimate`: The calculated data matrix.

**Details**

This function is a valid value for both `darch` parameters `rbm.errorFunction` and `darch.errorFunction`.

**Value**

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

**See Also**

Other error functions: crossEntropyError, mseError

**Examples**

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, rbm.errorFunction = "rmseError",
               darch.errorFunction = "rmseError")

## End(Not run)
```
**rpropagation**

Resilient backpropagation training for deep architectures.

**Description**

The function trains 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 algorithm see the references.

**Usage**

```r
rpropagation(darch, trainData, targetData, 
  rprop.method = getParameter(".rprop.method"),
  rprop.decFact = getParameter(".rprop.decFact"),
  rprop.incFact = getParameter(".rprop.incFact"),
  rprop.initDelta = getParameter(".rprop.initDelta"),
  rprop.minDelta = getParameter(".rprop.minDelta"),
  rprop.maxDelta = getParameter(".rprop.maxDelta"),
  nesterovMomentum = getParameter(".darch.nesterovMomentum"),
  dropout = getParameter(".darch.dropout"),
  dropConnect = getParameter(".darch.dropout.dropConnect"),
  errorFunction = getParameter(".darch.errorFunction"),
  matMult = getParameter(".matMult"),
  debugMode = getParameter(".debug", F, darch), ...)
```

**Arguments**

- **darch** The deep architecture to train
- **trainData** The training data
- **targetData** The expected output for the training data
- **rprop.method** The method for the training. Default is "iRprop+
- **rprop.decFact** Decreasing factor for the training. Default is 0.6.
- **rprop.incFact** Increasing factor for the training. Default is 1.2.
- **rprop.initDelta** Initialisation value for the update. Default is 0.0125.
- **rprop.minDelta** Lower bound for step size. Default is 0.000001
- **rprop.maxDelta** Upper bound for step size. Default is 50
- **nesterovMomentum** See `darch.nesterovMomentum` parameter of `darch`.
- **dropout** See `darch.dropout` parameter of `darch`.
- **dropConnect** See `darch.dropout.dropConnect` parameter of `darch`.
- **errorFunction** See `darch.errorFunction` parameter of `darch`.
- **matMult** Matrix multiplication function, internal parameter.
- **debugMode** Whether debug mode is enabled, internal parameter.
- **...** Further parameters.
Details

RPROP supports dropout and uses the weight update function as defined via the darch.weightUpdateFunction parameter of darch.

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).

Copyright (c) 2011, Roberto Calandra. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. 4. If used in any scientific publications, the publication has to refer specifically to the work published on this webpage.

This software is provided by us "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for particular purpose are disclaimed. In no event shall the copyright holders or any contributor be liable for any direct, indirect, incidental, special, exemplary, or consequential damages however caused and on any theory of liability whether in contract, strict liability or tort arising in any way out of the use of this software, even

The possible training methods (parameter rprop.method) are the following (see References for details):

- Rprop+: Rprop with Weight-Backtracking
- Rprop-: Rprop without Weight-Backtracking
- iRprop+: Improved Rprop with Weight-Backtracking
- iRprop-: Improved Rprop without Weight-Backtracking

Value

DArch - The trained deep architecture

References


See Also

darch
sigmoidUnit

Other fine-tuning functions: backpropagation, minimizeAutoencoder, minimizeClassifier

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.fineTuneFunction = "rpropagation",
preProc.params = list(method = c("center", "scale")),
darch.unitFunction = c("softplusUnit", "softmaxUnit"),
rprop.method = "iRprop+", rprop.decFact = .5, rprop.incFact = 1.2,
rprop.initDelta = 1/100, rprop.minDelta = 1/1000000, rprop.maxDelta = 50)
## End(Not run)
```

sigmoidUnit  
*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

`sigmoidUnit(input, ...)`

Arguments

- `input`  
  Input for the activation function.
- `...`  
  Additional parameters, not used.

Value

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

See Also

Other darch unit functions: exponentialLinearUnit, linearUnit, maxoutUnit, rectifiedLinearUnit, softmaxUnit, softplusUnit, tanhUnit

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.unitFunction = "sigmoidUnit")
## End(Not run)
```
softmaxUnit

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

softmaxUnit(input, ...)

Arguments

input Input for the activation function.
...
Additional parameters, not used.

Value

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

References


See Also

Other darch unit functions: exponentialLinearUnit, linearUnit, maxoutUnit, rectifiedLinearUnit,
sigmoidUnit, softplusUnit, tanhUnit

Examples

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris,
               darch.unitFunction = c("sigmoidUnit", "softmaxUnit"))
## End(Not run)
```
softplusUnit

**softplusUnit**

*Softplus 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 softmax activation of the units and the second entry is the derivative of the transfer function. Softplus is a smoothed version of the rectified linear activation function.

**Usage**

```r
softplusUnit(input, ...)
```

**Arguments**

- **input**: Input for the activation function.
- **...**: Additional parameters, not used.

**Value**

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

**References**


**See Also**

Other darch unit functions: `exponentialLinearUnit`, `linearUnit`, `maxoutUnit`, `rectifiedLinearUnit`, `sigmoidUnit`, `softmaxUnit`, `tanhUnit`

**Examples**

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.unitFunction = "softplusUnit")

## End(Not run)
```
tanhUnit  

*Continuous Tan-Sigmoid unit function.*

**Description**

Calculates the unit activations and returns them in a list.

**Usage**

```
tanhUnit(input, ...) 
```

**Arguments**

- `input`  
  Input for the activation function.
- `...`  
  Additional parameters, not used.

**Value**

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

**See Also**

Other darch unit functions: `exponentialLinearUnit`, `linearUnit`, `maxoutUnit`, `rectifiedLinearUnit`, `sigmoidUnit`, `softmaxUnit`, `softplusUnit`

**Examples**

```r
## Not run:
data(iris)
model <- darch(Species ~ ., iris, darch.unitFunction = "tanhUnit")

## End(Not run)
```

---

weightDecayWeightUpdate  

*Updates the weight using weight decay.*

**Description**

Multiplies the weights by `(1 - weightDecay)` before applying the scheduled weight changes.

**Usage**

```
weightDecayWeightUpdate(darch, layerIndex, weightsInc, biasesInc, ..., weightDecay = getParameter(".darch.weightDecay", 0, darch), debug = getParameter(".debug", F, darch))
```
Arguments

- **darch**: DArch instance.
- **layerIndex**: Layer index within the network.
- **weightsInc**: Matrix containing scheduled weight updates from the fine-tuning algorithm.
- **biasesInc**: Bias weight updates.
- **weightDecay**: Weights are multiplied by \((1 - \text{weightDecay})\) before each update. Corresponds to the `darch.weightDecay` parameter of `darch.default`.
- **debug**: Internal debugging flag.

Value

updated weights

See Also

Other weight update functions: `maxoutWeightUpdate`

Examples

```r
## Not run:
model <- darch(Species ~ ., iris, c(0, 50, 0),
               darch.weightUpdateFunction = "weightDecayWeightUpdate")

## End(Not run)
```
Index

backpropagation, 3, 8, 27, 28, 37
chooseGpu, 9
crossEntropyError, 4, 8, 29, 34
DArch, 3, 7, 11, 12, 16, 25–28, 30–32, 36, 41
darch, 3–5, 5, 13, 14, 16, 25–32, 34–36
darch.default, 25, 41
darchBench, 11, 12, 13, 16, 30–32
darchModelInfo, 14
data.frame, 6, 11, 15
DataSet, 6, 9
dummyVars, 10
exponentialLinearUnit, 8, 16, 23, 24, 34, 37–40
fineTuneDArch, 27, 28
formula, 6, 11
futile.logger, 9, 14
generateWeightsGlorotNormal, 9, 17, 19–22
generateWeightsGlorotUniform, 9, 18, 18, 20–22
generateWeightsHeNormal, 9, 18, 19, 19, 21, 22
generateWeightsHeUniform, 9, 18–20, 20, 21, 22
generateWeightsNormal, 9, 18–21, 21, 22
generateWeightsUniform, 9, 18–21, 22
linearUnit, 9, 17, 23, 24, 34, 37–40
linearUnitRbm, 11
maxoutUnit, 8, 9, 17, 23, 23, 34, 37–40
maxoutWeightUpdate, 9, 24, 25, 41
minimize, 26, 28
minimizeAutoencoder, 4, 8, 26, 28, 37
minimizeClassifier, 4, 8, 27, 27, 37
mseError, 5, 8, 10, 29, 34
Net, 21
plot.DArch, 12, 14, 16, 30, 31, 32
plotnet, 30
predict.DArch, 12, 14, 16, 30, 31, 32
preProcess, 10
print.DArch, 12, 14, 16, 30, 31, 32
provideMNIST, 32
rbmUpdate, 11
readMNIST, 32
rectifiedLinearUnit, 9, 17, 23, 24, 33, 37–40
rmseError, 5, 8, 10, 29, 34
rnorm, 11, 18, 19, 21
rpropagation, 4, 8, 27, 28, 35
runif, 11, 22
set.seed, 11
sigmoidUnit, 9, 17, 23, 24, 34, 37, 38–40
sigmoidUnitRbm, 11
softmaxUnit, 5, 9, 17, 23, 24, 34, 37, 38, 39, 40
softplusUnit, 9, 17, 23, 24, 34, 37, 38, 39, 40
tanhUnit, 9, 17, 23, 24, 34, 37–39, 40
tanhUnitRbm, 11
train, 14, 15
weightDecayWeightUpdate, 9, 25, 40

42