Package 'deeplearning'

August 29, 2016

Type Package

Title An Implementation of Deep Neural Network for Regression and Classification

Description An implementation of deep neural network with rectifier linear units trained with stochastic gradient descent method and batch normalization. A combination of these methods have achieved state-of-the-art performance in ImageNet classification by overcoming the gradient saturation problem experienced by many deep architecture neural network models in the past. In addition, batch normalization and dropout are implemented as a means of regularization. The deeplearning package is inspired by the darch package and uses its class DArch.

Version 0.1.0

Date 2016-04-10

LazyData TRUE

URL https://github.com/rz1988/deeplearning

BugReports https://github.com/rz1988/deeplearning/issues

Depends R (>= 3.2.4), methods, darch (>= 0.10.0),

Imports plotly, futile.logger, graphics, stats

License GPL (>= 2)

RoxygenNote 5.0.1

NeedsCompilation no

Author Zhi Ruan [aut, cre],
Martin Drees [cph]

Maintainer Zhi Ruan <ryan.zhiruan@gmail.com>

Repository CRAN

Date/Publication 2016-04-11 18:09:06

R topics documented:

`applyDropoutMask` *Applies the given dropout mask to the given data row-wise.*

Description

This function multiplies each row with the dropout mask. To apply the dropout mask by row, it can simply be multiplied with the data matrix. This does not work of the mask is to be applied row-wise, hence this function.

Usage

```
applyDropoutMask(data, mask)
```

Arguments

data	Data to which the dropout mask should be applied
mask	The dropout mask, a vector of 0 and 1.

Value

Data with applied dropout mask

References

Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Nitish Srivastava

See Also

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

AR *Calculates the Accuracy Ratio of a classifier*

Description

This function calculates the Accuracy Ratio of a binary classification model

Usage

```
AR(x, ...)
```

Arguments

x	model
...	additional inputs

AR.DArch *Calculates the Accruacy Ratio of a given set of probability*

Description

This function calculates the Accuracy Ratio of a trained darch instance

Usage

```
## S3 method for class 'DArch'
AR(x, input = x@dataSet@data, target = x@dataSet@targets,
  ...)
```

Arguments

x	a DArch instance
input	the input matrix
target	binary response
...	additional inputs

`AR.default` *Calculates the Accruacy Ratio of a given set of probability*

Description

This function calculates the Accuracy Ratio of a binary classification model output against its targets

Usage

```
## Default S3 method:
AR(x, target, ...)
```

Arguments

x	a list of model output in the form of probabilities
target	binary response
...	additional inputs

`AR.numeric` *Calculates the Accruacy Ratio of a given set of probability*

Description

This function calculates the Accuracy Ratio of a binary classification model output against its targets

Usage

```
## S3 method for class 'numeric'
AR(x, target, ...)
```

Arguments

x	a list of model output in the form of probabilities
target	binary response
...	additional inputs

backpropagate_delta_bn *Calculates the delta functions using backpropagation*

Description

function that calculates the delta function of a darch object with batch normalization

Usage

```
backpropagate_delta_bn(darch, trainData, targetData,
  errorFunc = meanSquareErr, with_BN = TRUE)
```

Arguments

darch	a darch instance
trainData	training input
targetData	training target
errorFunc	error function to minimize during training. Right now mean squared erros and cross entropy errors are supported.
with_BN	traing with batch normalization on or off

References

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey Ioffe, Christian Szegedy

See Also

http://jmlr.org/proceedings/papers/v37/ioffe15.pdf Pg 4

batch_normalization *Batch Normalization Function that normalizes the input before applying non-linearity*

Description

This function normalizes the distribution of inputs to hidden layers in a neural network

Usage

```
batch_normalization(x, gamma, beta, mu = NULL, sigma_2 = NULL,
  epsilon = exp(-12))
```

Arguments

x	weighted sum of outputs from the previous layer
gamma	the gamma coefficient
beta	the beta coefficient
mu	the mean of the input neurons. If NULL, it will be caluclated in the function.
sigma_2	the variance of the input nerurons. If NULL, it will be calcualted in the function.
epsilon	a constant added to the variance for numerical stability

References

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey Ioffe, Christian Szegedy

See Also

http://jmlr.org/proceedings/papers/v37/ioffe15.pdf Pg 4

---

batch_normalization_differential

Function that calcualtes the differentials in the batch normalization mode

---

Description

Calculates the differentials in batch normalization

Usage

```
batch_normalization_differential(delta_y, mu, sigma_2, x, x_hat, y, gamma, beta,
  epsilon = exp(-12), with_BN = T)
```

Arguments

delta_y	derivative wrt y
mu	mean of the input
sigma_2	variance of the input
x	input
x_hat	normalized input
y	transformed input after batch normalization
gamma	gamma coefficient
beta	beta coefficient
epsilon	the contant added to the variance for numeric stability
with_BN	logical value, set to TRUE to turn on batch normalization

References

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey Ioffe, Christian Szegedy

See Also

http://jmlr.org/proceedings/papers/v37/ioffe15.pdf Pg 4

`calcualte_population_mu_sigma` *Calculates the mu and sigmas of a darch instance*

Description

This function calculates the mu and sigmas of hidden layers in a darch instance

Usage

```
calcualte_population_mu_sigma(darch, input)
```

Arguments

`darch`	a darch instance
`input`	input data

`classification_error` *Calculates the classification error*

Description

This function calculates the classification error

Usage

```
classification_error(output, target)
```

Arguments

`output`	the output of a classifier in the form of probability. Probability > 1 will be treated as positive (target = 1).
`target`	the target variable

convert_categorical *Data proprosess function that covnerts a categorical input to continuous input or vectorize it*

Description

Proprosess a data set. It converts categorical data into binary variables if it is unordered or continuous variable from 0 to 1 if it is ordinal

Usage

```
convert_categorical(x, type = "ordinal", ordered_list = list(),
  var_name = "var", ...)
```

Arguments

x	input variable
type	ordinal or other
ordered_list	the rank ordering of an ordinal variable. Users are expected to provide a complete list of the rank ordering. Otherwise, a default rank ordering will be used.
var_name	the name of the input variable. This is used to to create vectorized input variables
...	other inputs

crossEntropyErr *Calculates the cross entropy error*

Description

This function calculates the cross entropy error and its first order derivatives

Usage

```
crossEntropyErr(output, target)
```

Arguments

output	the output value
target	the target value

`finetune_SGD_bn` *Updates a deep neural network's parameters using stochastic gradient descent method and batch normalization*

Description

This function finetunes a DArch network using SGD approach

Usage

```
finetune_SGD_bn(darch, trainData, targetData, learn_rate_weight = exp(-10),
  learn_rate_bias = exp(-10), learn_rate_gamma = exp(-10),
  errorFunc = meanSquareErr, with_BN = T)
```

Arguments

`darch`	a darch instance
`trainData`	training input
`targetData`	training target
`learn_rate_weight`	leanring rate for the weight matrices
`learn_rate_bias`	learning rate for the biases
`learn_rate_gamma`	learning rate for the gammas
`errorFunc`	the error function to minimize during training
`with_BN`	logical value, T to train the neural net with batch normalization

Value

a darch instance with parameters updated with stochastic gradient descent

`generateDropoutMask` *Generates the dropout mask for the deep neural network*

Description

This function generates the dropout mask for the deep neural network

Usage

```
generateDropoutMask(length, dropoutRate)
```

Arguments

length,	the dimension of the layer
dropoutRate,	the dropout rate

References

Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Nitish Srivastava

See Also

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

---

generateDropoutMasksForDarch

Generates dropout masks for dnn

Description

This function generates dropout maks for dnn

Usage

```
generateDropoutMasksForDarch(darch, dropout_input, dropout_hidden)
```

Arguments

darch,	a DArch instance
dropout_input,	the dropout rate for the input layer
dropout_hidden,	the dropout rate for the hidden layer

References

Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Nitish Srivastava

See Also

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

matMult *Calculates the outer product of two matricies*

Description

Calcualtes the outer product of two matrices

Usage

```
matMult(data, weight)
```

Arguments

data	the date matrix
weight	the weight matrix

meanSquareErr *Calculates the mean squared error*

Description

This function calculates the mean squared error and its first order derivatives

Usage

```
meanSquareErr(output, target)
```

Arguments

output	the output value
target	the target value

new_dnn *Creats a new instance of darch class*

Description

This function creates a new instance of darch class

Usage

```
new_dnn(layer_structure, layer_functions = NULL,
  output_layer_default = linearUnitDerivative,
  hidden_layer_default = rectified_linear_unit_function,
  weight_initiliazaiton = generateWeights)
```

Arguments

layer_structure	a int vector that specifies the number and width of layers
layer_functions	a list of activation functions used by each layer
output_layer_default	the activation function for the output layer
hidden_layer_default	the activation function for the hidden layers
weight_initiliazaiton	function that initialize a layer's weight matrix

Examples

```
# create a new deep neural network for classificaiton
dnn_regression <- new_dnn(
 c(2, 50, 50, 20, 1),
 # The layer structure of the deep neural network.
 # The first element is the number of input variables.
 # The last element is the number of output variables.
 hidden_layer_default = rectified_linear_unit_function,
 # for hidden layers, use rectified_linear_unit_function
 output_layer_default = sigmoidUnitDerivative
 # for classification, use sigmoidUnitDerivative function
)

# create a new deep neural network for classificaiton
dnn_regression <- new_dnn(
 c(2, 50, 50, 20, 1),
 # The layer structure of the deep neural network.
 # The first element is the number of input variables.
 # The last element is the number of output variables.
 hidden_layer_default = rectified_linear_unit_function,
```

```
 # for hidden layers, use rectified_linear_unit_function
 output_layer_default = linearUnitDerivative
 # for regression, use linearUnitDerivative function
)
```

---

print_weight *Prints out the weight of a deep neural network*

Description

This function prints out the weight in a heat map, 3D surface, or histogram

Usage

```
print_weight(darch, num_of_layer, show_derivative = F, type = "heatmap")
```

Arguments

darch	DArch instance
num_of_layer	the number of the layer to print
show_derivative	T to show the weight value. F to show the percentage weight change in the finetuning stage. This helps spot the network saturation problem.
type	type of the graph. It supports "heatmap", "surface", and "histogram"

Examples

```
# Example of Regression

input <- matrix(runif(1000), 500, 2)
input_valid <- matrix(runif(100), 50, 2)
target <- rowSums(input + input^2)
target_valid <- rowSums(input_valid + input_valid^2)
# create a new deep neural network for classificaiton
dnn_regression <- new_dnn(
 c(2, 50, 50, 20, 1),  # The layer structure of the deep neural network.
 # The first element is the number of input variables.
 # The last element is the number of output variables.
 hidden_layer_default = rectified_linear_unit_function,
 # for hidden layers, use rectified_linear_unit_function
 output_layer_default = linearUnitDerivative
 # for regression, use linearUnitDerivative function
)

# print the layer weights
# this function can print heatmap, histogram, or a surface
print_weight(dnn_regression, 1, type = "heatmap")

print_weight(dnn_regression, 2, type = "surface")
```

```
print_weight(dnn_regression, 3, type = "histogram")
```

`rectified_linear_unit_function` *Rectified Linear Unit Function*

Description

This functions calculates the value and the derivative of a rectified linear function. Reference Vinod Nair, Geoffrey Hinton, Rectified Linear Units Improve Restricted Boltzmann Machines

Usage

```
rectified_linear_unit_function(data, weights)
```

Arguments

`data`	the data matrix for calculation
`weights`	the connection (weight matrix/filter) and the bias

Value

A list of function values and derivatives

`reset_population_mu_sigma` *Resets the mu and sigmas of a darch instance to 0 and 1*

Description

This function resets the mu and sigmas of hidden layers in a darch instance to 0 and 1

Usage

```
reset_population_mu_sigma(darch)
```

Arguments

`darch`	a darch instance

`rsq` *Calculate the RSQ of a regression model Utilitiy function that calcualtes RSQ of a model. It measures the goodness-of- fit of a regression model.*

Description

Calculate the RSQ of a regression model Utilitiy function that calcualtes RSQ of a model. It measures the goodness-of- fit of a regression model.

Usage

```
rsq(x, ...)
```

Arguments

`x`	Regression Model
`...`	Additional Input

`rsq.DArch` *Utilitiy function that calcualtes RSQ of a DArch instance*

Description

Calcualte a regression model's RSQ of a deep neural network

Usage

```
## S3 method for class 'DArch'
rsq(x, input = x@dataSet@data, target = x@dataSet@targets,
  ...)
```

Arguments

`x`	DArch Model
`input`	Input data
`target`	Target data
`...`	addtional inputs

rsq.lm *Utilitiy function that calcualtes RSQ of a linear model*

Description

Calcualte a regression model's RSQ

Usage

```
## S3 method for class 'lm'
rsq(x, input, target, ...)
```

Arguments

x	linear Model
input	Input data
target	Target data
...	additional inputs

run_dnn *Execution function that runs in the batch normalization mode*

Description

This function calcualtes the output of a deep neural network with input data

Usage

```
run_dnn(darch, data)
```

Arguments

darch	a darch instance
data	input data

`train_dnn` *Train a deep neural network*

Description

This function trains a deep neural network

Usage

```
train_dnn(darch, input, target, input_valid = NULL, target_valid = NULL,
  ..., learn_rate_weight = exp(-10), learn_rate_bias = exp(-10),
  learn_rate_gamma = 1, batch_size = 10, batch_normalization = TRUE,
  dropout_input = 0, dropout_hidden = 0, momentum_initial = 0.6,
  momentum_final = 0.9, momentum_switch = 100, num_epochs = 0,
  error_function = meanSquareErr, report_classification_error = FALSE)
```

Arguments

`darch`	a darch instance
`input`	input data for training
`target`	target data for training
`input_valid`	input data for validation
`target_valid`	target data for validation
`...`	additional input
`learn_rate_weight`	learning rate for the weight matrices
`learn_rate_bias`	learning rate for the biases
`learn_rate_gamma`	learning rate for the gamma
`batch_size`	batch size during training
`batch_normalization`	logical value that determines whether to turn on batch normalization during training. Recommneded value: T
`dropout_input`	dropout ratio at input layer. Recommneded value: 0.2
`dropout_hidden`	dropout ratio at hidden layers. Recommended value: 0.5
`momentum_initial`	momentum ratio during training. Recommended value: 0.6
`momentum_final`	final momentum during training. Recommended value: 0.9
`momentum_switch`	afther which epoch the final momentum ratio is used during training
`num_epochs`	number of iterations of the training
`error_function`	error function to minimize during training
`report_classification_error`	logical value. T to report the classification error during training

Value

a trained deep neural network (darch instance)

Examples

```
# Example of Regression

input <- matrix(runif(1000), 500, 2)
input_valid <- matrix(runif(100), 50, 2)
target <- rowSums(input + input^2)
target_valid <- rowSums(input_valid + input_valid^2)
# create a new deep neural network for classificaiton
dnn_regression <- new_dnn(
 c(2, 50, 50, 20, 1),  # The layer structure of the deep neural network.
 # The first element is the number of input variables.
 # The last element is the number of output variables.
 hidden_layer_default = rectified_linear_unit_function,
 # for hidden layers, use rectified_linear_unit_function
 output_layer_default = linearUnitDerivative
 # for regression, use linearUnitDerivative function
)

 dnn_regression <- train_dnn(
 dnn_regression,

 # training data
 input, # input variable for training
 target, # target variable for training
 input_valid, # input variable for validation
 target_valid, # target variable for validation

 # training parameters
 learn_rate_weight = exp(-8) * 10,
 # learning rate for weights, higher if use dropout
 learn_rate_bias = exp(-8) * 10,
 # learning rate for biases, hihger if use dropout
 learn_rate_gamma = exp(-8) * 10,
 # learning rate for the gamma factor used
 batch_size = 10,
 # number of observations in a batch during training.
 # Higher for faster training. Lower for faster convergence
 batch_normalization = TRUE,
 # logical value, T to use batch normalization
 dropout_input = 0.2,
  # dropout ratio in input.
 dropout_hidden = 0.5,
 # dropout ratio in hidden layers
 momentum_initial = 0.6,
 # initial momentum in Stochastic Gradient Descent training
 momentum_final = 0.9,
 # final momentum in Stochastic Gradient Descent training
 momentum_switch = 100,
```

```
 # after which the momentum is switched from initial to final momentum
 num_epochs = 5,
  # number of iterations in training
  # increase numbef of epochs to 100 for better model fit


 # Error function
 error_function = meanSquareErr,
 # error function to minimize during training. For regression, use meanSquareErr
 report_classification_error = FALSE
 # whether to print classification error during training
)


# the prediciton by dnn_regression
pred <- predict(dnn_regression)

# calculate the r-squared of the prediciton
rsq(dnn_regression)


# calcualte the r-squared of the prediciton in validation
rsq(dnn_regression, input = input_valid, target = target_valid)

# print the layer weights
# this function can print heatmap, histogram, or a surface
print_weight(dnn_regression, 1, type = "heatmap")

print_weight(dnn_regression, 2, type = "surface")

print_weight(dnn_regression, 3, type = "histogram")


# Examples of classification

input <- matrix(runif(1000), 500, 2)
input_valid <- matrix(runif(100), 50, 2)
target <- (cos(rowSums(input + input^2)) > 0.5) * 1
target_valid <- (cos(rowSums(input_valid + input_valid^2)) > 0.5) * 1

# create a new deep neural network for classificaiton
dnn_classification <- new_dnn(
 c(2, 50, 50, 20, 1),  # The layer structure of the deep neural network.
 # The first element is the number of input variables.
 # The last element is the number of output variables.
 hidden_layer_default = rectified_linear_unit_function,
 # for hidden layers, use rectified_linear_unit_function
 output_layer_default = sigmoidUnitDerivative
 # for classification, use sigmoidUnitDerivative function
)

dnn_classification <- train_dnn(
 dnn_classification,
```

```
 # training data
 input, # input variable for training
 target, # target variable for training
 input_valid, # input variable for validation
 target_valid, # target variable for validation

 # training parameters
 learn_rate_weight = exp(-8) * 10,
 # learning rate for weights, higher if use dropout
 learn_rate_bias = exp(-8) * 10,
 # learning rate for biases, hihger if use dropout
 learn_rate_gamma = exp(-8) * 10,
 # learning rate for the gamma factor used
 batch_size = 10,
 # number of observations in a batch during training.
 # Higher for faster training. Lower for faster convergence
 batch_normalization = TRUE,
 # logical value, T to use batch normalization
 dropout_input = 0.2,
 # dropout ratio in input.
 dropout_hidden = 0.5,
 # dropout ratio in hidden layers
 momentum_initial = 0.6,
 # initial momentum in Stochastic Gradient Descent training
 momentum_final = 0.9,
 # final momentum in Stochastic Gradient Descent training
 momentum_switch = 100,
 # after which the momentum is switched from initial to final momentum
 num_epochs = 5,
 # number of iterations in training
 # increase num_epochs to 100 for better model fit

 # Error function
 error_function = crossEntropyErr,
 # error function to minimize during training. For regression, use crossEntropyErr
 report_classification_error = TRUE
 # whether to print classification error during training
)

# the prediciton by dnn_regression
pred <- predict(dnn_classification)

hist(pred)

# calculate the r-squared of the prediciton
AR(dnn_classification)

# calcualte the r-squared of the prediciton in validation
AR(dnn_classification, input = input_valid, target = target_valid)
```

`verticalize` *Creates a matrix by repeating a row vector N times*

Description

helper function that repeat a row vector N times

Usage

```
verticalize(vector, N)
```

Arguments

`vector`	the row vector
`N`	number of rows in the output matirx

Value

a matrix

Index