Package ‘cito’

August 11, 2022

Type Package

Date 2022-07-25

Title Building and Training Neural Networks

Version 1.0.0

Description Building and training custom neural networks in the typical R syntax. The 'torch' package is used for numerical calculations, which allows for training on CPU as well as on a graphics card.

Encoding UTF-8

RoxygenNote 7.2.0

Depends R (>= 3.5)

Imports coro, checkmate, torch

License GPL (>= 3)

Suggests rmarkdown, knitr, testthat, plotly, ggraph, igraph, stats, ggplot2

VignetteBuilder knitr

BugReports https://github.com/citoverse/cito/issues

NeedsCompilation no

Author Christian Amesöder [aut, cre],
Maximilian Pichler [aut]

Maintainer Christian Amesöder <Christian.Amesoeder@stud.uni-regensburg.de>

Repository CRAN

Date/Publication 2022-08-11 15:10:02 UTC

R topics documented:

ALE *Accumulated Local Effect Plot (ALE)*

Description

Performs an ALE for one or more features.

Usage

```
ALE(
  model,
  variable = NULL,
  data = NULL,
  K = 10,
  type = c("equidistant", "quantile")
)
```

Arguments

`model`	a model created by `dnn`
`variable`	variable as string for which the PDP should be done
`data`	data on which ALE is performed on, if NULL training data will be used.
`K`	number of neighborhoods original feature space gets divided into
`type`	method on how the feature space is divided into neighborhoods.

Details

If the defined variable is a numeric feature, the ALE is performed. Here, the non centered effect for feature j with k equally distant neighborhoods is defined as:

$\hat{\tilde{f}}_{j,ALE}(x) = \sum_{k=1}^{k_j(x)} \frac{1}{n_j(k)} \sum_{i:x_j^{(i)} \in N_j(k)} \left[\hat{f}(z_{k,j}, x_{\setminus j}^{(i)}) - \hat{f}(z_{k-1,j}, x_{\setminus j}^{(i)}) \right]$

Where $N_j(k)$ is the k-th neighborhood and $n_j(k)$ is the number of observations in the k-th neighborhood.

The last part of the equation, $\left[\hat{f}(z_{k,j}, x^{(i)}_{\setminus j}) - \hat{f}(z_{k-1,j}, x^{(i)}_{\setminus j})\right]$ represents the difference in model prediction when the value of feature j is exchanged with the upper and lower border of the current neighborhood.

Value

A list of plots made with 'ggplot2' consisting of an individual plot for each defined variable.

See Also

PDP

Examples

```
if(torch::torch_is_installed()){
library(cito)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris)

ALE(nn.fit, variable = "Petal.Length")
}
```

---

analyze_training *Visualize training of Neural Network*

Description

After training a model with cito, this function helps to analyze the training process and decide on best performing model. Creates a 'plotly' figure which allows to zoom in and out on training graph

Usage

```
analyze_training(object)
```

Arguments

`object`	a model created by dnn

Value

a 'plotly' figure

Examples

```
if(torch::torch_is_installed()){
library(cito)
set.seed(222)
validation_set<- sample(c(1:nrow(datasets::iris)),25)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris[-validation_set,],validation = 0.1)

# show zoomable plot of training and validation losses
analyze_training(nn.fit)

# set model which is used for predictions to model from epoch 22
nn.fit$use_model_epoch <- 22

# Use model on validation set
predictions <- predict(nn.fit, iris[validation_set,])

# Scatterplot
plot(iris[validation_set,]$Sepal.Length,predictions)
}
```

---

cito — *'cito': Building and training neural networks*

Description

Building and training custom neural networks in the typical R syntax. The 'torch' package is used for numerical calculations, which allows for training on CPU as well as on a graphics card. The main function is `dnn` which trains a custom deep neural network.

Installation

in order to install cito please follow these steps:

`install.packges("cito")`

`library(torch)`

`install_torch(reinstall = TRUE)`

`library(cito)`

cito functions

- `dnn`: train deep neural network
- `continue_training`: continues training of an existing cito dnn model for additional epochs
- `PDP`: plot the partial dependency plot for a specific feature
- `ALE`: plot the accumulated local effect plot for a specific feature

Examples

```
vignette("cito", package="cito")
```

---

coef.citodnn *Returns list of parameters the neural network model currently has in use*

Description

Returns list of parameters the neural network model currently has in use

Usage

```
## S3 method for class 'citodnn'
coef(object, ...)
```

Arguments

object	a model created by dnn
...	nothing implemented yet

Value

list of weights of neural network

Examples

```
if(torch::torch_is_installed()){
library(cito)

set.seed(222)
validation_set<- sample(c(1:nrow(datasets::iris)),25)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris[-validation_set,])

# Sturcture of Neural Network
print(nn.fit)

#analyze weights of Neural Network
coef(nn.fit)
}
```

config_lr_scheduler — *Creation of customized learning rate scheduler objects*

Description

Helps create custom learning rate schedulers for dnn.

Usage

```
config_lr_scheduler(
  type = c("lambda", "multiplicative", "one_cycle", "step"),
  verbose = FALSE,
  ...
)
```

Arguments

type	String defining which type of scheduler should be used. See Details.
verbose	If TRUE, additional information about scheduler will be printed to console.
...	additional arguments to be passed to scheduler. See Details.

Details

different learning rate scheduler need different variables, these functions will tell you which variables can be set:

- lambda: lr_lambda
- multiplicative: lr_multiplicative
- one_cycle: lr_one_cycle
- step: lr_step

Value

object of class cito_lr_scheduler to give to dnn

Examples

```
if(torch::torch_is_installed()){
library(cito)

# create learning rate scheduler object
scheduler <- config_lr_scheduler(type = "step",
                        step_size = 30,
                        gamma = 0.15,
                        verbose = TRUE)
```

```
# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris, lr_scheduler = scheduler)

}
```

---

config_optimizer *Creation of customized optimizer objects*

Description

Helps you create custom optimizer for dnn. It is recommended to set learning rate in dnn.

Usage

```
config_optimizer(
  type = c("adam", "adadelta", "adagrad", "rmsprop", "rprop", "sgd"),
  verbose = FALSE,
  ...
)
```

Arguments

type	character string defining which optimizer should be used. See Details.
verbose	If TRUE, additional information about scheduler will be printed to console
...	additional arguments to be passed to optimizer. See Details.

Details

different optimizer need different variables, this function will tell you how the variables are set. For more information see the corresponding functions:

- adam: optim_adam
- adadelta: optim_adadelta
- adagrad: optim_adagrad
- rmsprop: optim_rmsprop
- rprop: optim_rprop
- sgd: optim_sgd

Value

object of class cito_optim to give to dnn

Examples

```
if(torch::torch_is_installed()){
library(cito)

# create optimizer object
opt <- config_optimizer(type = "adagrad",
                        lr_decay = 1e-04,
                        weight_decay = 0.1,
                        verbose = TRUE)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris, optimizer = opt)

}
```

---

continue_training — *Continues training of a model for additional periods*

Description

Continues training of a model for additional periods

Usage

```
continue_training(
  model,
  epochs = 32,
  continue_from = NULL,
  data = NULL,
  device = "cpu",
  verbose = TRUE,
  changed_params = NULL
)
```

Arguments

`model`	a model created by dnn
`epochs`	additional epochs the training should continue for
`continue_from`	define which epoch should be used as starting point for training, 0 if last epoch should be used
`data`	matrix or data.frame if not provided data from original training will be used
`device`	device on which network should be trained on, either "cpu" or "cuda"
`verbose`	print training and validation loss of epochs
`changed_params`	list of arguments to change compared to original training setup, see dnn which parameter can be changed

Value

a model of class cito.dnn same as created by dnn

Examples

```
if(torch::torch_is_installed()){
library(cito)

set.seed(222)
validation_set<- sample(c(1:nrow(datasets::iris)),25)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris[-validation_set,], epochs = 32)

# continue training for another 32 epochs
nn.fit<- continue_training(nn.fit,epochs = 32)

# Use model on validation set
predictions <- predict(nn.fit, iris[validation_set,])
}
```

---

dnn *DNN*

Description

fits a custom deep neural network. dnn() supports the formula syntax and allows to customize the neural network to a maximal degree. So far, only Multilayer Perceptrons are possible. To learn more about Deep Learning, see here

Usage

```
dnn(
  formula,
  data = NULL,
 loss = c("mae", "mse", "softmax", "cross-entropy", "gaussian", "binomial", "poisson"),
  hidden = c(10L, 10L, 10L),
 activation = c("relu", "leaky_relu", "tanh", "elu", "rrelu", "prelu", "softplus",
   "celu", "selu", "gelu", "relu6", "sigmoid", "softsign", "hardtanh", "tanhshrink",
    "softshrink", "hardshrink", "log_sigmoid"),
  validation = 0,
  bias = TRUE,
  lambda = 0,
  alpha = 0.5,
  dropout = 0,
  optimizer = c("adam", "adadelta", "adagrad", "rmsprop", "rprop", "sgd"),
```

```
  lr = 0.01,
  batchsize = 32L,
  shuffle = FALSE,
  epochs = 32,
  plot = TRUE,
  verbose = TRUE,
  lr_scheduler = NULL,
  device = c("cpu", "cuda"),
  early_stopping = FALSE
)
```

Arguments

formula	an object of class "formula": a description of the model that should be fitted
data	matrix or data.frame
loss	loss after which network should be optimized. Can also be distribution from the stats package or own function
hidden	hidden units in layers, length of hidden corresponds to number of layers
activation	activation functions, can be of length one, or a vector of different activation functions for each layer
validation	percentage of data set that should be taken as validation set (chosen randomly)
bias	whether use biases in the layers, can be of length one, or a vector (number of hidden layers + 1 (last layer)) of logicals for each layer.
lambda	strength of regularization: lambda penalty, $\lambda * (L1 + L2)$ (see alpha)
alpha	add L1/L2 regularization to training $(1 - \alpha) * \|weights\| + \alpha \|\|weights\|\|^2$ will get added for each layer. Can be single integer between 0 and 1 or vector of alpha values if layers should be regularized differently.
dropout	dropout rate, probability of a node getting left out during training (see nn_dropout)
optimizer	which optimizer used for training the network, for more adjustments to optimizer see config_optimizer
lr	learning rate given to optimizer
batchsize	number of samples that are used to calculate one learning rate step
shuffle	if TRUE, data in each batch gets reshuffled every epoch
epochs	epochs the training goes on for
plot	plot training loss
verbose	print training and validation loss of epochs
lr_scheduler	learning rate scheduler created with config_lr_scheduler
device	device on which network should be trained on.
early_stopping	if set to integer, training will stop if validation loss worsened between current defined past epoch.

Details

In a Multilayer Perceptron (MLP) network every neuron is connected with all neurons of the previous layer and connected to all neurons of the layer afterwards. The value of each neuron is calculated with:

$a(\sum_j w_j * a_j)$

Where w_j is the weight and a_j is the value from neuron j to the current one. a() is the activation function, e.g. $relu(x) = max(0, x)$ As regularization methods there is dropout and elastic net regularization available. These methods help you avoid over fitting.

Training on graphic cards: If you want to train on your cuda devide, you have to install the NVIDIA CUDA toolkit version 11.3. and cuDNN 8.4. beforehand. Make sure that you have xactly these versions installed, since it does not wor kwith other version. For more information see mlverse: 'torch'

Value

an S3 object of class `"cito.dnn"` is returned. It is a list containing everything there is to know about the model and its training process. The list consists of the following attributes:

`net`	An object of class "nn_sequential" "nn_module", originates from the torch package and represents the core object of this workflow.
`call`	The original function call
`loss`	A list which contains relevant information for the target variable and the used loss function
`data`	Contains data used for training the model
`weigths`	List of weights for each training epoch
`use_model_epoch`	Integer, which defines which model from which training epoch should be used for prediction.
`loaded_model_epoch`	Integer, shows which model from which epoch is loaded currently into model$net.
`model_properties`	A list of properties of the neural network, contains number of input nodes, number of output nodes, size of hidden layers, activation functions, whether bias is included and if dropout layers are included.
`training_properties`	A list of all training parameters that were used the last time the model was trained. It consists of learning rate, information about an learning rate scheduler, information about the optimizer, number of epochs, whether early stopping was used, if plot was active, lambda and alpha for L1/L2 regularization, batchsize, shuffle, was the data set split into validation and training, which formula was used for training and at which epoch did the training stop.
`losses`	A data.frame containing training and validation losses of each epoch

See Also

`predict.citodnn`, `plot.citodnn`, `coef.citodnn`,`print.citodnn`, `summary.citodnn`, `continue_training`, `analyze_training`, `PDP`, `ALE`,

Examples

```
if(torch::torch_is_installed()){
library(cito)

set.seed(222)
validation_set<- sample(c(1:nrow(datasets::iris)),25)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris[-validation_set,])

# Sturcture of Neural Network
print(nn.fit)

# Use model on validation set
predictions <- predict(nn.fit, iris[validation_set,])

# Scatterplot
plot(iris[validation_set,]$Sepal.Length,predictions)
# MAE
mean(abs(predictions-iris[validation_set,]$Sepal.Length))

# Get variable importances
summary(nn.fit)

# Partial dependencies
PDP(nn.fit, variable = "Petal.Length")

# Accumulated local effect plots
ALE(nn.fit, variable = "Petal.Length")

}
```

PDP *Partial Dependence Plot (PDP)*

Description

Calculates the Partial Dependency Plot for one feature, either numeric or categorical. Returns it as a plot.

Usage

```
PDP(model, variable = NULL, data = NULL, ice = FALSE, resolution.ice = 20)
```

Arguments

model	a model created by dnn

`variable`	variable as string for which the PDP should be done. If none is supplied it is done for all variables.
`data`	specify new data PDP should be performed . If NULL, PDP is performed on the training data.
`ice`	Individual Conditional Dependence will be shown if TRUE
`resolution.ice`	resolution in which ice will be computed

Details

Performs the estimation of the partial function $\hat{f}_S$

$\hat{f}_S(x_S) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)})$

with a Monte Carlo Estimation:

$\hat{f}_S(x_S) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)})$

If a categorical feature is analyzed, all data instances are used and set to each level. Then an average is calculated per category and put out in a bar plot.

If ice is set to true additional the individual conditional dependence will be shown and the original PDP will be colored yellow. These lines show, how each individual data sample reacts to changes in the feature. This option is not available for categorical features. Unlike PDP the ICE curves are computed with a value grid instead of utilizing every value of every data entry.

Value

A list of plots made with 'ggplot2' consisting of an individual plot for each defined variable.

See Also

`ALE`

Examples

```
if(torch::torch_is_installed()){
library(cito)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris)

PDP(nn.fit, variable = "Petal.Length")
}
```

`plot.citodnn` *Creates graph plot which gives an overview of the network architecture.*

Description

Creates graph plot which gives an overview of the network architecture.

Usage

```
## S3 method for class 'citodnn'
plot(x, node_size = 1, scale_edges = FALSE, ...)
```

Arguments

x	a model created by dnn
node_size	size of node in plot
scale_edges	edge weight gets scaled according to other weights (layer specific)
...	no further functionality implemented yet

Value

A plot made with 'ggraph' + 'igraph' that represents the neural network

Examples

```
if(torch::torch_is_installed()){
library(cito)

set.seed(222)
validation_set<- sample(c(1:nrow(datasets::iris)),25)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris[-validation_set,])

plot(nn.fit)
}
```

`predict.citodnn` *Predict from a fitted dnn model*

Description

Predict from a fitted dnn model

Usage

```
## S3 method for class 'citodnn'
predict(object, newdata = NULL, type = c("link", "response"), ...)
```

Arguments

`object`	a model created by dnn
`newdata`	new data for predictions
`type`	link or response
`...`	additional arguments

Value

prediction matrix

Examples

```
if(torch::torch_is_installed()){
library(cito)

set.seed(222)
validation_set<- sample(c(1:nrow(datasets::iris)),25)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris[-validation_set,])

# Use model on validation set
predictions <- predict(nn.fit, iris[validation_set,])
# Scatterplot
plot(iris[validation_set,]$Sepal.Length,predictions)
# MAE
mean(abs(predictions-iris[validation_set,]$Sepal.Length))
}
```

`print.citodnn` *Print class citodnn*

Description

Print class citodnn

Usage

```
## S3 method for class 'citodnn'
print(x, ...)
```

Arguments

`x`	a model created by `dnn`
`...`	additional arguments

Value

prediction matrix

original object x gets returned

Examples

```
if(torch::torch_is_installed()){
library(cito)

set.seed(222)
validation_set<- sample(c(1:nrow(datasets::iris)),25)

# Build and train  Network
nn.fit<- dnn(Sepal.Length~., data = datasets::iris[-validation_set,])

# Sturcture of Neural Network
print(nn.fit)
}
```

print.summary.citodnn *Print method for class summary.citodnn*

Description

Print method for class summary.citodnn

Usage

```
## S3 method for class 'summary.citodnn'
print(x, ...)
```

Arguments

`x`	a summary object created by `summary.citodnn`
`...`	additional arguments

Value

original object x gets returned

residuals.citodnn *Extract Model Residuals*

Description

Returns residuals of training set.

Usage

```
## S3 method for class 'citodnn'
residuals(object, ...)
```

Arguments

`object`	a model created by `dnn`
`...`	no additional arguments implemented

Value

residuals of training set

summary.citodnn *Summarize Neural Network of class citodnn*

Description

Performs a Feature Importance calculation based on Permutations

Usage

```
## S3 method for class 'citodnn'
summary(object, n_permute = 256, ...)
```

Arguments

object	a model of class citodnn created by dnn
n_permute	number of permutations performed, higher equals more accurate importance results
...	additional arguments

Details

Performs the feature importance calculation as suggested by Fisher, Rudin, and Dominici (2018). For each feature n permutation get done and original and permuted predictive mean squared error (e_{perm} & e_{orig}) get evaluated with $FI_j = e_{perm}/e_{orig}$. Based on Mean Squared Error.

Value

summary.glm returns an object of class "summary.citodnn", a list with components

Index