Package 'qrnn'

February 29, 2024

Type Package

Title Quantile Regression Neural Network

Version 2.1.1

Description Fit quantile regression neural network models with optional left censoring, partial monotonicity constraints, generalized additive model constraints, and the ability to fit multiple non-crossing quantile functions following Cannon (2011) <doi:10.1016/j.cageo.2010.07.005> and Cannon (2018) <doi:10.1007/s00477-018-1573-6>.

Date 2024-02-29

License GPL-2

LazyLoad yes

Repository CRAN

NeedsCompilation no

Author Alex J. Cannon [aut, cre] (<https://orcid.org/0000-0002-8025-3790>)

Maintainer Alex J. Cannon <alex.cannon@ec.gc.ca>

Date/Publication 2024-02-29 22:30:12 UTC

R topics documented:

---

qrnn-package *Quantile Regression Neural Network*

Description

This package implements the quantile regression neural network (QRNN) (Taylor, 2000; Cannon, 2011; Cannon, 2018), which is a flexible nonlinear form of quantile regression. While low level modelling functions are available, it is recommended that the `mcqrnn.fit` and `mcqrnn.predict` wrappers be used for most applications. More information is provided below.

The goal of quantile regression is to estimate conditional quantiles of a response variable that depend on covariates in some form of regression equation. The QRNN adopts the multi-layer perceptron neural network architecture. The implementation follows from previous work on the estimation of censored regression quantiles, thus allowing predictions for mixed discrete-continuous variables like precipitation (Friederichs and Hense, 2007). A differentiable approximation to the quantile regression cost function is adopted so that a simplified form of the finite smoothing algorithm (Chen, 2007) can be used to estimate model parameters. This approximation can also be used to force the model to solve a standard least squares regression problem or an expectile regression problem (Cannon, 2018). Weight penalty regularization can be added to help avoid overfitting, and ensemble models with bootstrap aggregation are also provided.

An optional monotone constraint can be invoked, which guarantees monotonic non-decreasing behaviour of model outputs with respect to specified covariates (Zhang, 1999). The input-hidden layer weight matrix can also be constrained so that model relationships are strictly additive (see `gam.style`; Cannon, 2018). Borrowing strength by using a composite model for multiple regression quantiles (Zou et al., 2008; Xu et al., 2017) is also possible (see `composite.stack`). Weights can be applied to individual cases (Jiang et al., 2012).

Applying the monotone constraint in combination with the composite model allows one to simultaneously estimate multiple non-crossing quantiles (Cannon, 2018); the resulting monotone composite QRNN (MCQRNN) is provided by the `mcqrnn.fit` and `mcqrnn.predict` wrapper functions. Examples for `qrnn.fit` and `qrnn2.fit` show how the same functionality can be achieved using the low level `composite.stack` and fitting functions.

QRNN models with a single layer of hidden nodes can be fitted using the `qrnn.fit` function. Predictions from a fitted model are made using the `qrnn.predict` function. The function `gam.style` can be used to visualize and investigate fitted covariate/response relationships from `qrnn.fit` (Plate et al., 2000). Note: a single hidden layer is usually sufficient for most modelling tasks. With added monotonicity constraints, a second hidden layer may sometimes be beneficial (Lang, 2005; Minin et al., 2010). QRNN models with two hidden layers are available using the `qrnn2.fit` and `qrnn2.predict` functions. For non-crossing quantiles, the `mcqrnn.fit` and `mcqrnn.predict` wrappers also allow models with one or two hidden layers to be fitted and predictions to be made from the fitted models.

In general, mcqrnn.fit offers a convenient, single function for fitting multiple quantiles simultaneously. Note, however, that default settings in mcqrnn.fit and other model fitting functions are not optimized for general speed, memory efficiency, or accuracy and should be adjusted for a particular regression problem as needed. In particular, the approximation to the quantile regression cost function eps.seq, the number of trials n.trials, and number of iterations iter.max can all influence fitting speed (and accuracy), as can changing the optimization algorithm via method. Non-crossing quantiles are implemented by stacking multiple copies of the x and y data, one copy per value of tau. Depending on the dataset size, this can lead to large matrices being passed to the optimization routine. In the adam adaptive stochastic gradient descent method, the minibatch size can be adjusted to help offset this cost. Model complexity is determined via the number of hidden nodes, n.hidden and n.hidden2, as well as the optional weight penalty penalty; values of these hyperparameters are crucial to obtaining a well performing model.

When using mcqrnn.fit, it is also possible to estimate the full quantile regression process by specifying a single integer value for tau. In this case, tau is the number of random samples used in the stochastic estimation. For more information, see Tagasovska and Lopez-Paz (2019). It may be necessary to restart the optimization multiple times from the previous weights and biases, in which case init.range can be set to the weights values from the previously completed optimization run. For large datasets, it is recommended that the adam method with an appropriate integer tau and minibatch size be used for optimization.

If models for multiple quantiles have been fitted, for example by mcqrnn.fit or multiple calls to either qrnn.fit or qrnn2.fit, the (experimental) dquantile function and its companion functions are available to create proper probability density, distribution, and quantile functions (Quiñonero-Candela et al., 2006; Cannon, 2011). Alternative distribution, quantile, and random variate functions based on the Nadaraya-Watson estimator (Passow and Donner, 2020) are also available in [p,q,r]quantile.nw. These can be useful for assessing probabilistic calibration and evaluating model performance.

Note: the user cannot easily change the output layer transfer function to be different than hramp, which provides either the identity function or a ramp function to accommodate optional left censoring. Some applications, for example fitting smoothed binary quantile regression models for a binary target variable (Kordas, 2006), require an alternative like the logistic sigmoid. While not straightforward, it is possible to change the output layer transfer function by switching off scale.y in the call to the fitting function and reassigning hramp and hramp.prime as follows:

```
library(qrnn)

# Use the logistic sigmoid as the output layer transfer function
To.logistic <- function(x, lower, eps) 0.5 + 0.5*tanh(x/2)
environment(To.logistic) <- asNamespace("qrnn")
assignInNamespace("hramp", To.logistic, ns="qrnn")

# Change the derivative of the output layer transfer function
To.logistic.prime <- function(x, lower, eps) 0.25/(cosh(x/2)^2)
environment(To.logistic.prime) <- asNamespace("qrnn")
assignInNamespace("hramp.prime", To.logistic.prime, ns="qrnn")
```

Details

Package:	qrnn
Type:	Package
License:	GPL-2
LazyLoad:	yes

References

Cannon, A.J., 2011. Quantile regression neural networks: implementation in R and application to precipitation downscaling. Computers & Geosciences, 37: 1277-1284. doi:10.1016/j.cageo.2010.07.005

Cannon, A.J., 2018. Non-crossing nonlinear regression quantiles by monotone composite quantile regression neural network, with application to rainfall extremes. Stochastic Environmental Research and Risk Assessment, 32(11): 3207-3225. doi:10.1007/s00477-018-1573-6

Chen, C., 2007. A finite smoothing algorithm for quantile regression. Journal of Computational and Graphical Statistics, 16: 136-164.

Friederichs, P. and A. Hense, 2007. Statistical downscaling of extreme precipitation events using censored quantile regression. Monthly Weather Review, 135: 2365-2378.

Jiang, X., J. Jiang, and X. Song, 2012. Oracle model selection for nonlinear models based on weighted composite quantile regression. Statistica Sinica, 22(4): 1479-1506.

Kordas, G., 2006. Smoothed binary regression quantiles. Journal of Applied Econometrics, 21(3): 387-407.

Lang, B., 2005. Monotonic multi-layer perceptron networks as universal approximators. International Conference on Artificial Neural Networks, Artificial Neural Networks: Formal Models and Their Applications-ICANN 2005, pp. 31-37.

Minin, A., M. Velikova, B. Lang, and H. Daniels, 2010. Comparison of universal approximators incorporating partial monotonicity by structure. Neural Networks, 23(4): 471-475.

Passow, C., R.V. Donner, 2020. Regression-based distribution mapping for bias correction of climate model outputs using linear quantile regression. Stochastic Environmental Research and Risk Assessment, 34: 87-102.

Plate, T., J. Bert, J. Grace, and P. Band, 2000. Visualizing the function computed by a feedforward neural network. Neural Computation, 12(6): 1337-1354.

Quiñonero-Candela, J., C. Rasmussen, F. Sinz, O. Bousquet, B. Scholkopf, 2006. Evaluating Predictive Uncertainty Challenge. Lecture Notes in Artificial Intelligence, 3944: 1-27.

Tagasovska, N., D. Lopez-Paz, 2019. Single-model uncertainties for deep learning. Advances in Neural Information Processing Systems, 32, NeurIPS 2019. doi:10.48550/arXiv.1811.00908

Taylor, J.W., 2000. A quantile regression neural network approach to estimating the conditional density of multiperiod returns. Journal of Forecasting, 19(4): 299-311.

Xu, Q., K. Deng, C. Jiang, F. Sun, and X. Huang, 2017. Composite quantile regression neural network with applications. Expert Systems with Applications, 76, 129-139.

Zhang, H. and Zhang, Z., 1999. Feedforward networks with monotone constraints. In: International Joint Conference on Neural Networks, vol. 3, p. 1820-1823. doi:10.1109/IJCNN.1999.832655

Zou, H. and M. Yuan, 2008. Composite quantile regression and the oracle model selection theory. The Annals of Statistics, 1108-1126.

`adam` *Adaptive stochastic gradient descent optimization algorithm (Adam)*

Description

From Kingma and Ba (2015): "We introduce Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which Adam was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss AdaMax, a variant of Adam based on the infinity norm."

Usage

```
adam(f, p, x, y, w, tau, ..., iterlim=5000, iterbreak=iterlim,
     alpha=0.01, minibatch=nrow(x), beta1=0.9, beta2=0.999,
     epsilon=1e-8, print.level=10)
```

Arguments

`f`	the function to be minimized, including gradient information contained in the `gradient` attribute.
`p`	the starting parameters for the minimization.
`x`	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
`y`	response column matrix with number of rows equal to the number of samples.
`w`	vector of weights with length equal to the number of samples.
`tau`	vector of desired tau-quantile(s) with length equal to the number of samples.
`...`	additional parameters passed to the `f` cost function.
`iterlim`	the maximum number of iterations before the optimization is stopped.
`iterbreak`	the maximum number of iterations without progress before the optimization is stopped.
`alpha`	size of the learning rate.
`minibatch`	number of samples in each minibatch.

beta1	controls the exponential decay rate used to scale the biased first moment estimate.
beta2	controls the exponential decay rate used to scale the biased second raw moment estimate.
epsilon	smoothing term to avoid division by zero.
print.level	the level of printing which is done during optimization. A value of 0 suppresses any progress reporting, whereas positive values report the value of f every print.level iterations.

Value

A list with elements:

estimate	The best set of parameters found.
minimum	The value of f corresponding to estimate.

References

Kingma, D.P. and J. Ba, 2015. Adam: A method for stochastic optimization. The International Conference on Learning Representations (ICLR) 2015. http://arxiv.org/abs/1412.6980

---

censored.mean *A hybrid mean/median function for left censored variables*

Description

Returns the median if the majority of values are censored and the mean otherwise.

Usage

```
censored.mean(x, lower, trim=0)
```

Arguments

x	numeric vector.
lower	left censoring point.
trim	fraction of observations to be trimmed from each end of x before the mean is computed.

See Also

qrnn.fit, qrnn.predict

Examples

```
x <- c(0, 0, 1, 2, 3)
print(censored.mean(x, lower=0))
x.cens <- c(0, 0, 0, 1, 2)
print(censored.mean(x.cens, lower=0))
```

composite.stack *Reformat data matrices for composite quantile regression*

Description

Returns stacked x and y matrices and tau vector, which can be passed to qrnn.fit to fit composite quantile regression and composite QRNN models (Zou et al., 2008; Xu et al., 2017). In combination with the partial monotonicity constraints, stacking can be used to fit multiple non-crossing quantile functions (see mcqrnn). More details are provided in Cannon (2018).

Usage

```
composite.stack(x, y, tau)
```

Arguments

x	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
y	response column matrix with number of rows equal to the number of samples.
tau	vector of tau-quantiles.

References

Cannon, A.J., 2018. Non-crossing nonlinear regression quantiles by monotone composite quantile regression neural network, with application to rainfall extremes. Stochastic Environmental Research and Risk Assessment, 32(11): 3207-3225. doi:10.1007/s00477-018-1573-6

Xu, Q., K. Deng, C. Jiang, F. Sun, and X. Huang, 2017. Composite quantile regression neural network with applications. Expert Systems with Applications, 76, 129-139.

Zou, H. and M. Yuan, 2008. Composite quantile regression and the oracle model selection theory. The Annals of Statistics, 1108-1126.

See Also

qrnn.fit, mcqrnn

Examples

```
x <- as.matrix(iris[,"Petal.Length",drop=FALSE])
y <- as.matrix(iris[,"Petal.Width",drop=FALSE])

cases <- order(x)
x <- x[cases,,drop=FALSE]
y <- y[cases,,drop=FALSE]

tau <- seq(0.05, 0.95, by=0.05)
x.y.tau <- composite.stack(x, y, tau)
binary.tau <- dummy.code(as.factor(x.y.tau$tau))
```

```
set.seed(1)

# Composite QR
fit.cqr <- qrnn.fit(cbind(binary.tau, x.y.tau$x), x.y.tau$y,
                    tau=x.y.tau$tau, n.hidden=1, n.trials=1,
                    Th=linear, Th.prime=linear.prime)
pred.cqr <- matrix(qrnn.predict(cbind(binary.tau, x.y.tau$x), fit.cqr),
                   ncol=length(tau))
coef.cqr <- lm.fit(cbind(1, x), pred.cqr)$coef
colnames(coef.cqr) <- tau
print(coef.cqr)

# Composite QRNN
fit.cqrnn <- qrnn.fit(x.y.tau$x, x.y.tau$y, tau=x.y.tau$tau,
                      n.hidden=1, n.trials=1, Th=sigmoid,
                      Th.prime=sigmoid.prime)
pred.cqrnn <- qrnn.predict(x.y.tau$x, fit.cqrnn)
pred.cqrnn <- matrix(pred.cqrnn, ncol=length(tau), byrow=FALSE)

matplot(x, pred.cqrnn, col="red", type="l")
points(x, y, pch=20)
```

---

dummy.code *Convert a factor to a matrix of dummy codes*

Description

Converts a factor (categorical) variable to a matrix of dummy codes using a 1 of C-1 binary coding scheme.

Usage

```
dummy.code(x)
```

Arguments

x	a factor variable.

Value

a matrix with the number of rows equal to the number of cases in x and the number of columns equal to one minus the number of factors in x. The last factor serves as the reference group.

Examples

```
print(dummy.code(iris$Species))
```

gam.style *Modified generalized additive model plots for interpreting QRNN models*

Description

Generalized additive model (GAM)-style effects plots provide a graphical means of interpreting relationships between covariates and conditional quantiles predicted by a QRNN. From Plate et al. (2000): The effect of the ith input variable at a particular input point `Delta.i.x` is the change in `f` resulting from changing `X1` to `x1` from `b1` (the baseline value [...]) while keeping the other inputs constant. The effects are plotted as short line segments, centered at (`x.i`, `Delta.i.x`), where the slope of the segment is given by the partial derivative. Variables that strongly influence the function value have a large total vertical range of effects. Functions without interactions appear as possibly broken straight lines (linear functions) or curves (nonlinear functions). Interactions show up as vertical spread at a particular horizontal location, that is, a vertical scattering of segments. Interactions are present when the effect of a variable depends on the values of other variables.

Usage

```
gam.style(x, parms, column, baseline=mean(x[,column]),
          epsilon=1e-5, seg.len=0.02, seg.cols="black",
          plot=TRUE, return.results=FALSE, trim=0,
          ...)
```

Arguments

`x`	matrix with number of rows equal to the number of samples and number of columns equal to the number of covariate variables.
`parms`	list returned by `qrnn.fit`.
`column`	column of `x` for which effects plots should be returned.
`baseline`	value of `x[,column]` to be used as the baseline for calculation of covariate effects; defaults to `mean(x[,column])`.
`epsilon`	step-size used in the finite difference calculation of the partial derivatives.
`seg.len`	length of effects line segments expressed as a fraction of the range of `x[,column]`.
`seg.cols`	colors of effects line segments.
`plot`	if `TRUE` (the default) then an effects plots for the given model is produced.
`return.results`	if `TRUE` then values of effects and partial derivatives are returned.
`trim`	if `plot=TRUE` and `parms` is for a model with `n.ensemble` > 1, value of `trim` passed to `censored.mean`.
`...`	further arguments to be passed to `plot`.

Value

A list with elements:

`effects`	a matrix of covariate effects.
`partials`	a matrix of covariate partial derivatives.

References

Cannon, A.J. and I.G. McKendry, 2002. A graphical sensitivity analysis for interpreting statistical climate models: Application to Indian monsoon rainfall prediction by artificial neural networks and multiple linear regression models. International Journal of Climatology, 22:1687-1708.

Plate, T., J. Bert, J. Grace, and P. Band, 2000. Visualizing the function computed by a feedforward neural network. Neural Computation, 12(6): 1337-1354.

See Also

qrnn.fit, qrnn.predict

Examples

```
## YVR precipitation data with seasonal cycle and NCEP/NCAR Reanalysis
## covariates
data(YVRprecip)

y <- YVRprecip$precip
x <- cbind(sin(2*pi*seq_along(y)/365.25),
           cos(2*pi*seq_along(y)/365.25),
           YVRprecip$ncep)

## Fit QRNN, additive QRNN (QADD), and quantile regression (QREG)
## models for the conditional 75th percentile
set.seed(1)
train <- c(TRUE, rep(FALSE, 49))
w.qrnn <- qrnn.fit(x=x[train,], y=y[train,,drop=FALSE],
                   n.hidden=2, tau=0.75, iter.max=500,
                   n.trials=1, lower=0, penalty=0.01)
w.qadd <- qrnn.fit(x=x[train,], y=y[train,,drop=FALSE],
                   n.hidden=ncol(x), tau=0.75, iter.max=250,
                   n.trials=1, lower=0, additive=TRUE)
w.qreg <- qrnn.fit(x=x[train,], y=y[train,,drop=FALSE],
                   tau=0.75, iter.max=100, n.trials=1,
                   lower=0, Th=linear, Th.prime=linear.prime)

## GAM-style plots for slp, sh700, and z500
for (column in 3:5) {
    gam.style(x[train,], parms=w.qrnn, column=column,
              main="QRNN")
    gam.style(x[train,], parms=w.qadd, column=column,
              main="QADD")
    gam.style(x[train,], parms=w.qreg, column=column,
              main="QREG")
}
```

huber *Huber norm and Huber approximations to the ramp and tilted absolute value functions*

Description

Huber norm function providing a hybrid L1/L2 norm. Huber approximations to the ramp `hramp` and tilted absolute value `tilted.approx` functions. `huber.prime`, `hramp.prime`, and `tilted.approx.prime` provide the corresponding derivatives.

Usage

```
huber(x, eps)
huber.prime(x, eps)
hramp(x, lower, eps)
hramp.prime(x, lower, eps)
tilted.approx(x, tau, eps)
tilted.approx.prime(x, tau, eps)
```

Arguments

`x`	numeric vector.
`eps`	epsilon value used in `huber` and related functions.
`tau`	desired tau-quantile.
`lower`	left censoring point.

See Also

`tilted.abs`, `qrnn.cost`

Examples

```
x <- seq(-10, 10, length=100)
plot(x, huber(x, eps=1), type="l", col="black", ylim=c(-2, 10), ylab="")
lines(x, hramp(x, lower=0, eps=1), col="red")
lines(x, tilted.approx(x, tau=0.1, eps=1), col="blue")
lines(x, huber.prime(x, eps=1), col="black", lty=2)
lines(x, hramp.prime(x, lower=0, eps=1), lty=2, col="red")
lines(x, tilted.approx.prime(x, tau=0.1, eps=1), lty=2, col="blue")
```

mcqrnn *Monotone composite quantile regression neural network (MCQRNN) for simultaneous estimation of multiple non-crossing quantiles*

Description

High level wrapper functions for fitting and making predictions from a monotone composite quantile regression neural network (MCQRNN) model for multiple non-crossing regression quantiles (Cannon, 2018).

Uses `composite.stack` and monotonicity constraints in `qrnn.fit` or `qrnn2.fit` to fit MCQRNN models with one or two hidden layers. Note: `Th` must be a non-decreasing function to guarantee non-crossing.

Following Tagasovska and Lopez-Paz (2019), it is also possible to estimate the full quantile regression process by specifying a single integer value for `tau`. In this case, tau is the number of random samples used in the stochastic estimation. It may be necessary to restart the optimization multiple times from the previous weights and biases, in which case `init.range` can be set to the weights values from the previously completed optimization run. For large datasets, it is recommended that the `adam` method with an appropriate `minibatch` size be used for optimization.

Usage

```
mcqrnn.fit(x, y, n.hidden=2, n.hidden2=NULL, w=NULL,
           tau=c(0.1, 0.5, 0.9), iter.max=5000, n.trials=5,
           lower=-Inf, init.range=c(-0.5, 0.5, -0.5, 0.5, -0.5, 0.5),
           monotone=NULL, eps.seq=2^seq(-8, -32, by=-4), Th=sigmoid,
           Th.prime=sigmoid.prime, penalty=0, n.errors.max=10,
           trace=TRUE, method=c("nlm", "adam"), scale.y=TRUE, ...)
mcqrnn.predict(x, parms, tau=NULL)
```

Arguments

`x`	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
`y`	response column matrix with number of rows equal to the number of samples.
`n.hidden`	number of hidden nodes in the first hidden layer.
`n.hidden2`	number of hidden nodes in the second hidden layer; `NULL` fits a model with a single hidden layer.
`w`	if `tau` specifies a finite number of tau-quantiles, a vector of weights with length equal to the number of samples times the length of `tau`; see `composite.stack`. Otherwise, a vector of weights with length equal to the number of samples. `NULL` gives equal weight to each sample.
`tau`	desired tau-quantiles; `NULL` in `mcqrnn.predict` uses values from the original call to `mcqrnn.fit`. If `tau` is an integer, specifies the number of random samples used for stochastic estimation of the full quantile regression process.

iter.max	maximum number of iterations of the optimization algorithm.
n.trials	number of repeated trials used to avoid local minima.
lower	left censoring point.
init.range	initial weight range for input-hidden, hidden-hidden, and hidden-output weight matrices. If supplied with a list of weight matrices from a prior run of mcqrnn.fit, will restart model fitting with these values.
monotone	column indices of covariates for which the monotonicity constraint should hold.
eps.seq	sequence of eps values for the finite smoothing algorithm.
Th	hidden layer transfer function; use sigmoid, elu, relu, lrelu, softplus, or other non-decreasing function.
Th.prime	derivative of the hidden layer transfer function Th.
penalty	weight penalty for weight decay regularization.
n.errors.max	maximum number of nlm optimization failures allowed before quitting.
trace	logical variable indicating whether or not diagnostic messages are printed during optimization.
method	character string indicating which optimization algorithm to use when n.hidden2 != NULL.
scale.y	logical variable indicating whether y should be scaled to zero mean and unit standard deviation.
...	additional parameters passed to the nlm or adam optimization routines.
parms	list containing MCQRNN weight matrices and other parameters.

References

Cannon, A.J., 2011. Quantile regression neural networks: implementation in R and application to precipitation downscaling. Computers & Geosciences, 37: 1277-1284. doi:10.1016/j.cageo.2010.07.005

Cannon, A.J., 2018. Non-crossing nonlinear regression quantiles by monotone composite quantile regression neural network, with application to rainfall extremes. Stochastic Environmental Research and Risk Assessment, 32(11): 3207-3225. doi:10.1007/s00477-018-1573-6

Tagasovska, N., D. Lopez-Paz, 2019. Single-model uncertainties for deep learning. Advances in Neural Information Processing Systems, 32, NeurIPS 2019. doi:10.48550/arXiv.1811.00908

See Also

composite.stack, qrnn.fit, qrnn2.fit, qrnn.predict, qrnn2.predict, adam

Examples

```
x <- as.matrix(iris[,"Petal.Length",drop=FALSE])
y <- as.matrix(iris[,"Petal.Width",drop=FALSE])

cases <- order(x)
x <- x[cases,,drop=FALSE]
y <- y[cases,,drop=FALSE]
```

```
set.seed(1)

## MCQRNN model w/ 2 hidden layers for simultaneous estimation of
## multiple non-crossing quantile functions
fit.mcqrnn <- mcqrnn.fit(x, y, tau=seq(0.1, 0.9, by=0.1),
                         n.hidden=2, n.hidden2=2, n.trials=1,
                         iter.max=500)
pred.mcqrnn <- mcqrnn.predict(x, fit.mcqrnn)

## Estimate the full quantile regression process by specifying
## the number of samples/random values of tau used in training

fit.full <- mcqrnn.fit(x, y, tau=1000L, n.hidden=3, n.hidden2=3,
                       n.trials=1, iter.max=300, eps.seq=1e-6,
                       method="adam", minibatch=64, print.level=100)
# Show how to initialize from previous weights
fit.full <- mcqrnn.fit(x, y, tau=1000L, n.hidden=3, n.hidden2=3,
                       n.trials=1, iter.max=300, eps.seq=1e-6,
                       method="adam", minibatch=64, print.level=100,
                       init.range=fit.full$weights)
pred.full <- mcqrnn.predict(x, fit.full, tau=seq(0.1, 0.9, by=0.1))

par(mfrow=c(1, 2))
matplot(x, pred.mcqrnn, col="blue", type="l")
points(x, y)
matplot(x, pred.full, col="blue", type="l")
points(x, y)
```

`qrnn.cost` *Smooth approximation to the tilted absolute value cost function*

Description

Smooth approximation to the tilted absolute value cost function used to fit a QRNN model. Optional left censoring, monotone constraints, and additive constraints are supported.

Usage

```
qrnn.cost(weights, x, y, n.hidden, w, tau, lower, monotone,
          additive, eps, Th, Th.prime, penalty, unpenalized)
```

Arguments

`weights`	weight vector of length returned by `qrnn.initialize`.
`x`	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
`y`	response column matrix with number of rows equal to the number of samples.
`n.hidden`	number of hidden nodes in the QRNN model.

w	vector of weights with length equal to the number of samples; NULL gives equal weight to each sample.
tau	desired tau-quantile.
lower	left censoring point.
monotone	column indices of covariates for which the monotonicity constraint should hold.
additive	force additive relationships.
eps	epsilon value used in the approximation functions.
Th	hidden layer transfer function; use sigmoid, elu, relu, lrelu, softplus, or other non-decreasing function for a nonlinear model and linear for a linear model.
Th.prime	derivative of the hidden layer transfer function Th.
penalty	weight penalty for weight decay regularization.
unpenalized	column indices of covariates for which the weight penalty should not be applied to input-hidden layer weights.

Value

numeric value indicating tilted absolute value cost function, along with attribute containing vector with gradient information.

See Also

qrnn.fit

---

qrnn.fit *Main function used to fit a QRNN model or ensemble of QRNN models*

Description

Function used to fit a QRNN model or ensemble of QRNN models.

Usage

```
qrnn.fit(x, y, n.hidden, w=NULL, tau=0.5, n.ensemble=1,
         iter.max=5000, n.trials=5, bag=FALSE, lower=-Inf,
         init.range=c(-0.5, 0.5, -0.5, 0.5), monotone=NULL,
         additive=FALSE, eps.seq=2^seq(-8, -32, by=-4),
         Th=sigmoid, Th.prime=sigmoid.prime, penalty=0,
         unpenalized=NULL, n.errors.max=10, trace=TRUE,
         scale.y=TRUE, ...)
```

Arguments

x	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
y	response column matrix with number of rows equal to the number of samples.
n.hidden	number of hidden nodes in the QRNN model.
w	vector of weights with length equal to the number of samples; NULL gives equal weight to each sample.
tau	desired tau-quantile(s).
n.ensemble	number of ensemble members to fit.
iter.max	maximum number of iterations of the optimization algorithm.
n.trials	number of repeated trials used to avoid local minima.
bag	logical variable indicating whether or not bootstrap aggregation (bagging) should be used.
lower	left censoring point.
init.range	initial weight range for input-hidden and hidden-output weight matrices.
monotone	column indices of covariates for which the monotonicity constraint should hold.
additive	force additive relationships.
eps.seq	sequence of eps values for the finite smoothing algorithm.
Th	hidden layer transfer function; use sigmoid, elu, relu, lrelu, softplus, or other non-decreasing function for a nonlinear model and linear for a linear model.
Th.prime	derivative of the hidden layer transfer function Th.
penalty	weight penalty for weight decay regularization.
unpenalized	column indices of covariates for which the weight penalty should not be applied to input-hidden layer weights.
n.errors.max	maximum number of nlm optimization failures allowed before quitting.
trace	logical variable indicating whether or not diagnostic messages are printed during optimization.
scale.y	logical variable indicating whether y should be scaled to zero mean and unit standard deviation.
...	additional parameters passed to the nlm optimization routine.

Details

Fit a censored quantile regression neural network model for the tau-quantile by minimizing a cost function based on smooth Huber-norm approximations to the tilted absolute value and ramp functions. Left censoring can be turned on by setting lower to a value greater than -Inf. A simplified form of the finite smoothing algorithm, in which the nlm optimization algorithm is run with values of the eps approximation tolerance progressively reduced in magnitude over the sequence eps.seq, is used to set the QRNN weights and biases. Local minima of the cost function can be avoided by setting n.trials, which controls the number of repeated runs from different starting weights and biases, to a value greater than one.

(Note: if eps.seq is set to a single, sufficiently large value and tau is set to 0.5, then the result will be a standard least squares regression model. The same value of eps.seq and other values of tau leads to expectile regression.)

If invoked, the monotone argument enforces non-decreasing behaviour between specified columns of x and model outputs. This holds if Th and To are monotone non-decreasing functions. In this case, the exp function is applied to the relevant weights following initialization and during optimization; manual adjustment of init.weights or qrnn.initialize may be needed due to differences in scaling of the constrained and unconstrained weights. Non-increasing behaviour can be forced by transforming the relevant covariates, e.g., by reversing sign.

The additive argument sets relevant input-hidden layer weights to zero, resulting in a purely additive model. Interactions between covariates are thus suppressed, leading to a compromise in flexibility between linear quantile regression and the quantile regression neural network.

Borrowing strength by using a composite model for multiple regression quantiles is also possible (see composite.stack). Applying the monotone constraint in combination with the composite model allows one to simultaneously estimate multiple non-crossing quantiles; the resulting monotone composite QRNN (MCQRNN) is demonstrated in mcqrnn.

In the linear case, model complexity does not depend on the number of hidden nodes; the value of n.hidden is ignored and is instead set to one internally. In the nonlinear case, n.hidden controls the overall complexity of the model. As an added means of avoiding overfitting, weight penalty regularization for the magnitude of the input-hidden layer weights (excluding biases) can be applied by setting penalty to a nonzero value. (For the linear model, this penalizes both input-hidden and hidden-output layer weights, leading to a quantile ridge regression model. In this case, kernel quantile ridge regression can be performed with the aid of the qrnn.rbf function.) Finally, if the bag argument is set to TRUE, models are trained on bootstrapped x and y sample pairs; bootstrap aggregation (bagging) can be turned on by setting n.ensemble to a value greater than one. Averaging over an ensemble of bagged models will also tend to alleviate overfitting.

The gam.style function can be used to plot modified generalized additive model effects plots, which are useful for visualizing the modelled covariate-response relationships.

Note: values of x and y need not be standardized or rescaled by the user. All variables are automatically scaled to zero mean and unit standard deviation prior to fitting and parameters are automatically rescaled by qrnn.predict and other prediction functions. Values of eps.seq are relative to the residuals in standard deviation units. Note: scaling of y can be turned off using the scale.y argument.

Value

a list containing elements

weights	a list containing fitted weight matrices
lower	left censoring point
eps.seq	sequence of eps values for the finite smoothing algorithm
tau	desired tau-quantile(s)
Th	hidden layer transfer function
x.center	vector of column means for x
x.scale	vector of column standard deviations for x

y.center	vector of column means for y
y.scale	vector of column standard deviations for y
monotone	column indices indicating covariate monotonicity constraints.
additive	force additive relationships.

References

Cannon, A.J., 2011. Quantile regression neural networks: implementation in R and application to precipitation downscaling. Computers & Geosciences, 37: 1277-1284. doi:10.1016/j.cageo.2010.07.005

Cannon, A.J., 2018. Non-crossing nonlinear regression quantiles by monotone composite quantile regression neural network, with application to rainfall extremes. Stochastic Environmental Research and Risk Assessment, 32(11): 3207-3225. doi:10.1007/s00477-018-1573-6

See Also

qrnn.predict, qrnn.cost, composite.stack, mcqrnn, gam.style

Examples

```
x <- as.matrix(iris[,"Petal.Length",drop=FALSE])
y <- as.matrix(iris[,"Petal.Width",drop=FALSE])

cases <- order(x)
x <- x[cases,,drop=FALSE]
y <- y[cases,,drop=FALSE]

tau <- c(0.05, 0.5, 0.95)

set.seed(1)

## QRNN models for conditional 5th, 50th, and 95th percentiles
w <- p <- vector("list", length(tau))
for(i in seq_along(tau)){
    w[[i]] <- qrnn.fit(x=x, y=y, n.hidden=3, tau=tau[i],
                       iter.max=200, n.trials=1)
    p[[i]] <- qrnn.predict(x, w[[i]])
}

## Monotone composite QRNN (MCQRNN) for simultaneous estimation of
## multiple non-crossing quantile functions
x.y.tau <- composite.stack(x, y, tau)
fit.mcqrnn <- qrnn.fit(cbind(x.y.tau$tau, x.y.tau$x), x.y.tau$y,
                       tau=x.y.tau$tau, n.hidden=3, n.trials=1,
                       iter.max=500, monotone=1)
pred.mcqrnn <- matrix(qrnn.predict(cbind(x.y.tau$tau, x.y.tau$x),
                      fit.mcqrnn), ncol=length(tau))

par(mfrow=c(1, 2))
matplot(x, matrix(unlist(p), nrow=nrow(x), ncol=length(p)), col="red",
        type="l")
points(x, y)
```

```
matplot(x, pred.mcqrnn, col="blue", type="l")
points(x, y)
```

`qrnn.initialize` *Initialize a QRNN weight vector*

Description

Random initialization of the weight vector used during fitting of a QRNN model.

Usage

```
qrnn.initialize(x, y, n.hidden, init.range=c(-0.5, 0.5, -0.5, 0.5))
```

Arguments

`x`	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
`y`	response column matrix with number of rows equal to the number of samples.
`n.hidden`	number of hidden nodes in the QRNN model.
`init.range`	initial weight range for input-hidden and hidden-output weight matrices.

`qrnn.predict` *Evaluate quantiles from trained QRNN model*

Description

Evaluate a fitted QRNN model or ensemble of models, resulting in a list containing the predicted quantiles.

Usage

```
qrnn.predict(x, parms)
```

Arguments

`x`	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
`parms`	list containing QRNN input-hidden and hidden-output layer weight matrices and other parameters from `qrnn.fit`.

Value

a list with number of elements equal to that of `parms`, each containing a column matrix of predicted quantiles.

See Also

qrnn.fit

Examples

```
x <- as.matrix(iris[,"Petal.Length",drop=FALSE])
y <- as.matrix(iris[,"Petal.Width",drop=FALSE])

cases <- order(x)
x <- x[cases,,drop=FALSE]
y <- y[cases,,drop=FALSE]
y[y < 0.5] <- 0.5

set.seed(1)
parms <- qrnn.fit(x=x, y=y, n.hidden=3, tau=0.5, lower=0.5,
                  iter.max=500, n.trials=1)
p <- qrnn.predict(x=x, parms=parms)

matplot(x, cbind(y, p), type=c("p", "l"), pch=1, lwd=1)
```

---

qrnn.rbf *Radial basis function kernel*

Description

Evaluate a kernel matrix based on the radial basis function kernel. Can be used in conjunction with qrnn.fit with Th set to linear and penalty set to a nonzero value for kernel quantile ridge regression.

Usage

```
qrnn.rbf(x, x.basis, sigma)
```

Arguments

x	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
x.basis	covariate matrix with number of rows equal to the number of basis functions and number of columns equal to the number of variables.
sigma	kernel width

Value

kernel matrix with number of rows equal to the number of samples and number of columns equal to the number of basis functions.

See Also

qrnn.fit

Examples

```
x <- as.matrix(iris[,"Petal.Length",drop=FALSE])
y <- as.matrix(iris[,"Petal.Width",drop=FALSE])

cases <- order(x)
x <- x[cases,,drop=FALSE]
y <- y[cases,,drop=FALSE]

set.seed(1)
kern <- qrnn.rbf(x, x.basis=x, sigma=1)

parms <- qrnn.fit(x=kern, y=y, tau=0.5, penalty=0.1,
                  Th=linear, Th.prime=linear.prime,
                  iter.max=500, n.trials=1)
p <- qrnn.predict(x=kern, parms=parms)

matplot(x, cbind(y, p), type=c("p", "l"), pch=1, lwd=1)
```

---

qrnn2 *Fit and make predictions from QRNN models with two hidden layers*

Description

Functions used to fit and make predictions from QRNN models with two hidden layers. Note: `Th` must be a non-decreasing function if `monotone != NULL`.

Usage

```
qrnn2.fit(x, y, n.hidden=2, n.hidden2=2, w=NULL, tau=0.5,
          n.ensemble=1, iter.max=5000, n.trials=5, bag=FALSE,
          lower=-Inf, init.range=c(-0.5, 0.5, -0.5, 0.5, -0.5, 0.5),
          monotone=NULL, eps.seq=2^seq(-8, -32, by=-4), Th=sigmoid,
          Th.prime=sigmoid.prime, penalty=0, unpenalized=NULL,
          n.errors.max=10, trace=TRUE, method=c("nlm", "adam"),
          scale.y=TRUE, ...)
qrnn2.predict(x, parms)
```

Arguments

`x`	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of variables.
`y`	response column matrix with number of rows equal to the number of samples.
`n.hidden`	number of hidden nodes in the first hidden layer.
`n.hidden2`	number of hidden nodes in the second hidden layer.
`w`	vector of weights with length equal to the number of samples; `NULL` gives equal weight to each sample.

tau	desired tau-quantile(s).
n.ensemble	number of ensemble members to fit.
iter.max	maximum number of iterations of the optimization algorithm.
n.trials	number of repeated trials used to avoid local minima.
bag	logical variable indicating whether or not bootstrap aggregation (bagging) should be used.
lower	left censoring point.
init.range	initial weight range for input-hidden, hidden-hidden, and hidden-output weight matrices.
monotone	column indices of covariates for which the monotonicity constraint should hold.
eps.seq	sequence of eps values for the finite smoothing algorithm.
Th	hidden layer transfer function; use sigmoid, elu, relu, lrelu, softplus, or other non-decreasing function for a nonlinear model and linear for a linear model.
Th.prime	derivative of the hidden layer transfer function Th.
penalty	weight penalty for weight decay regularization.
unpenalized	column indices of covariates for which the weight penalty should not be applied to input-hidden layer weights.
n.errors.max	maximum number of nlm optimization failures allowed before quitting.
trace	logical variable indicating whether or not diagnostic messages are printed during optimization.
method	character string indicating which optimization algorithm to use.
scale.y	logical variable indicating whether y should be scaled to zero mean and unit standard deviation.
...	additional parameters passed to the nlm or adam optimization routines.
parms	list containing QRNN weight matrices and other parameters from qrnn2.fit.

References

Cannon, A.J., 2011. Quantile regression neural networks: implementation in R and application to precipitation downscaling. Computers & Geosciences, 37: 1277-1284. doi:10.1016/j.cageo.2010.07.005

Cannon, A.J., 2018. Non-crossing nonlinear regression quantiles by monotone composite quantile regression neural network, with application to rainfall extremes. Stochastic Environmental Research and Risk Assessment, 32(11): 3207-3225. doi:10.1007/s00477-018-1573-6

See Also

qrnn.fit, qrnn.predict, qrnn.cost, composite.stack, mcqrnn, adam

Examples

```
x <- as.matrix(iris[,"Petal.Length",drop=FALSE])
y <- as.matrix(iris[,"Petal.Width",drop=FALSE])

cases <- order(x)
x <- x[cases,,drop=FALSE]
y <- y[cases,,drop=FALSE]

tau <- c(0.05, 0.5, 0.95)

set.seed(1)

## QRNN models w/ 2 hidden layers (tau=0.05, 0.50, 0.95)
w <- p <- vector("list", length(tau))
for(i in seq_along(tau)){
    w[[i]] <- qrnn2.fit(x=x, y=y, n.hidden=3, n.hidden2=3,
                        tau=tau[i], iter.max=200, n.trials=1)
    p[[i]] <- qrnn2.predict(x, w[[i]])
}

## MCQRNN model w/ 2 hidden layers for simultaneous estimation of
## multiple non-crossing quantile functions
x.y.tau <- composite.stack(x, y, tau)
fit.mcqrnn <- qrnn2.fit(cbind(x.y.tau$tau, x.y.tau$x), x.y.tau$y,
                        tau=x.y.tau$tau, n.hidden=3, n.hidden2=3,
                        n.trials=1, iter.max=500, monotone=1)
pred.mcqrnn <- matrix(qrnn2.predict(cbind(x.y.tau$tau, x.y.tau$x),
                      fit.mcqrnn), ncol=length(tau))

par(mfrow=c(1, 2))
matplot(x, matrix(unlist(p), nrow=nrow(x), ncol=length(p)), col="red",
        type="l")
points(x, y)
matplot(x, pred.mcqrnn, col="blue", type="l")
points(x, y)
```

`quantile.dtn` *Interpolated quantile distribution with exponential tails and Nadaraya-Watson quantile distribution*

Description

`dquantile` gives a probability density function (pdf) by combining step-interpolation of probability densities for specified `tau`-quantiles (`quant`) with exponential lower/upper tails (Quiñonero-Candela, 2006; Cannon, 2011). Point mass (e.g., as might occur when using censored QRNN models) can be defined by setting `lower` to the left censoring point. `pquantile` gives the cumulative distribution function (cdf); the `integrate` function is used for values outside the range of

quant. The inverse cdf is given by qquantile; the uniroot function is used for values outside the range of tau. rquantile is used for generating random variates.

Alternative formulations (without left censoring) based on the Nadaraya-Watson estimator [p,q,r]quantile.nw are also provided (Passow and Donner, 2020).

Note: these functions have not been extensively tested or optimized and should be considered experimental.

Usage

```
dquantile(x, tau, quant, lower=-Inf)
pquantile(q, tau, quant, lower=-Inf, ...)
pquantile.nw(q, tau, quant, h=0.001, ...)
qquantile(p, tau, quant, lower=-Inf,
          tol=.Machine$double.eps^0.25, maxiter=1000,
          range.mult=1.1, max.error=100, ...)
qquantile.nw(p, tau, quant, h=0.001)
rquantile(n, tau, quant, lower=-Inf,
          tol=.Machine$double.eps^0.25, maxiter=1000,
          range.mult=1.1, max.error=100, ...)
rquantile.nw(n, tau, quant, h=0.001)
```

Arguments

x, q	vector of quantiles.
p	vector of cumulative probabilities.
n	number of random samples.
tau	ordered vector of cumulative probabilities associated with quant argument.
quant	ordered vector of quantiles associated with tau argument.
lower	left censoring point.
tol	tolerance passed to uniroot.
h	bandwidth for Nadaraya-Watson kernel.
maxiter	maximum number of iterations passed to uniroot.
range.mult	values of lower and upper in uniroot are initialized to quant[1]-range.mult*diff(range(quant)) and quant[length(quant)]+range.mult*diff(range(quant)) respectively; range.mult is squared, lower and upper are recalculated, and uniroot is rerun if the current values lead to an exception.
max.error	maximum number of uniroot errors allowed before termination.
...	additional arguments passed to integrate or uniroot.

Value

dquantile gives the pdf, pquantile gives the cdf, qquantile gives the inverse cdf (or quantile function), and rquantile generates random deviates.

References

Cannon, A.J., 2011. Quantile regression neural networks: implementation in R and application to precipitation downscaling. Computers & Geosciences, 37: 1277-1284. doi:10.1016/j.cageo.2010.07.005

Passow, C., R.V. Donner, 2020. Regression-based distribution mapping for bias correction of climate model outputs using linear quantile regression. Stochastic Environmental Research and Risk Assessment, 34:87-102. doi:10.1007/s00477-019-01750-7

Quiñonero-Candela, J., C. Rasmussen, F. Sinz, O. Bousquet, B. Scholkopf, 2006. Evaluating Predictive Uncertainty Challenge. Lecture Notes in Artificial Intelligence, 3944: 1-27.

See Also

integrate, uniroot, qrnn.predict

Examples

```
## Normal distribution
tau <- c(0.01, seq(0.05, 0.95, by=0.05), 0.99)
quant <- qnorm(tau)

x <- seq(-3, 3, length=500)
plot(x, dnorm(x), type="l", col="red", lty=2, lwd=2,
     main="pdf")
lines(x, dquantile(x, tau, quant), col="blue")

q <- seq(-3, 3, length=20)
plot(q, pnorm(q), type="b", col="red", lty=2, lwd=2,
     main="cdf")
lines(q, pquantile(q, tau, quant),
      col="blue")

abline(v=1.96, lty=2)
abline(h=pnorm(1.96), lty=2)
abline(h=pquantile(1.96, tau, quant), lty=3)
abline(h=pquantile.nw(1.96, tau, quant, h=0.01), lty=3)

p <- c(0.001, 0.01, 0.025, seq(0.05, 0.95, by=0.05),
       0.975, 0.99, 0.999)
plot(p, qnorm(p), type="b", col="red", lty=2, lwd=2,
     main="inverse cdf")
lines(p, qquantile(p, tau, quant), col="blue")

## Distribution with point mass at zero
tau.0 <- c(0.3, 0.5, 0.7, 0.8, 0.9)
quant.0 <- c(0, 5, 7, 15, 20)

r.0 <- rquantile(500, tau=tau.0, quant=quant.0, lower=0)
x.0 <- seq(0, 40, by=0.5)
d.0 <- dquantile(x.0, tau=tau.0, quant=quant.0, lower=0)
p.0 <- pquantile(x.0, tau=tau.0, quant=quant.0, lower=0)
q.0 <- qquantile(p.0, tau=tau.0, quant=quant.0, lower=0)
```

```
par(mfrow=c(2, 2))
plot(r.0, pch=20, main="random")
plot(x.0, d.0, type="b", col="red", main="pdf")
plot(x.0, p.0, type="b", col="blue", ylim=c(0, 1),
     main="cdf")
plot(p.0, q.0, type="b", col="green", xlim=c(0, 1),
     main="inverse cdf")
```

`tilted.abs` *Tilted absolute value function*

Description

Tilted absolute value function. Also known as the check function, hinge function, or the pinball loss function.

Usage

```
tilted.abs(x, tau)
```

Arguments

`x`	numeric vector.
`tau`	desired tau-quantile.

See Also

`tilted.approx`

Examples

```
x <- seq(-2, 2, length=200)
plot(x, tilted.abs(x, tau=0.75), type="l")
```

`transfer` *Transfer functions and their derivatives*

Description

The `sigmoid`, exponential linear `elu`, `softplus`, `lrelu`, and `relu` functions can be used as the hidden layer transfer function for a nonlinear QRNN model. `sigmoid` is used by default. The `linear` function is used as the hidden layer transfer function for linear QRNN models. `sigmoid.prime`, `elu.prime`, `softplus.prime`, `lrelu.prime`, `relu.prime`, and `linear.prime` provide the corresponding derivatives.

Usage

```
sigmoid(x)
sigmoid.prime(x)
elu(x, alpha=1)
elu.prime(x, alpha=1)
softplus(x, alpha=2)
softplus.prime(x, alpha=2)
logistic(x)
logistic.prime(x)
lrelu(x)
lrelu.prime(x)
relu(x)
relu.prime(x)
linear(x)
linear.prime(x)
```

Arguments

x	numeric vector.
alpha	transition parameter for elu and softplus functions.

Examples

```
x <- seq(-10, 10, length=100)
plot(x, sigmoid(x), type="l", col="black", ylab="")
lines(x, sigmoid.prime(x), lty=2, col="black")
lines(x, elu(x), col="red")
lines(x, elu.prime(x), lty=2, col="red")
lines(x, softplus(x), col="blue")
lines(x, softplus.prime(x), lty=2, col="blue")
lines(x, logistic(x), col="brown")
lines(x, logistic.prime(x), lty=2, col="brown")
lines(x, lrelu(x), col="orange")
lines(x, lrelu.prime(x), lty=2, col="orange")
lines(x, relu(x), col="pink")
lines(x, relu.prime(x), lty=2, col="pink")
lines(x, linear(x), col="green")
lines(x, linear.prime(x), lty=2, col="green")
```

YVRprecip *Daily precipitation data at Vancouver Int'l Airport (YVR)*

Description

Daily precipitation totals (mm) at Vancouver Int'l Airport (YVR) for the period 1971-2000.

Covariates for a simple downscaling task include daily sea-level pressures (Pa), 700-hPa specific humidities (kg/kg), and 500-hPa geopotential heights (m) from the NCEP/NCAR Reanalysis (Kalnay et al., 1996) grid point centered on 50 deg. N and 237.5 deg. E.

NCEP/NCAR Reanalysis data provided by the NOAA/OAR/ESRL PSD, Boulder, Colorado, USA, from their Web site at https://psl.noaa.gov/.

References

Kalnay, E. et al., 1996. The NCEP/NCAR 40-year reanalysis project, Bulletin of the American Meteorological Society, 77: 437-470.

Examples

```
## YVR precipitation data with seasonal cycle and NCEP/NCAR Reanalysis
## covariates

data(YVRprecip)
y <- YVRprecip$precip
x <- cbind(sin(2*pi*seq_along(y)/365.25),
           cos(2*pi*seq_along(y)/365.25),
           YVRprecip$ncep)

## Fit QRNN and quantile regression models for the conditional 75th
## percentile using the final 3 years of record for training and the
## remaining years for testing.
train <- as.numeric(format(YVRprecip$date, "%Y")) >= 1998
test <- !train

set.seed(1)
w.qrnn <- qrnn.fit(x=x[train,], y=y[train,,drop=FALSE],
                   n.hidden=1, tau=0.75, iter.max=200,
                   n.trials=1, lower=0)
p.qrnn <- qrnn.predict(x=x[test,], parms=w.qrnn)
w.qreg <- qrnn.fit(x=x[train,], y=y[train,,drop=FALSE],
                   tau=0.75, n.trials=1, lower=0,
                   Th=linear, Th.prime=linear.prime)
p.qreg <- qrnn.predict(x=x[test,], parms=w.qreg)

## Tilted absolute value cost function on test dataset
qvs.qrnn <- mean(tilted.abs(y[test]-p.qrnn, 0.75))
qvs.qreg <- mean(tilted.abs(y[test]-p.qreg, 0.75))
cat("Cost QRNN", qvs.qrnn, "\n")
cat("Cost QREG", qvs.qreg, "\n")

## Plot first year of test dataset
plot(y[test][1:365], type="h", xlab="Day", ylab="Precip. (mm)")
points(p.qrnn[1:365], col="red", pch=19)
points(p.qreg[1:365], col="blue", pch=19)
```

Index