Package 'dnn'

November 6, 2026

Type Package

Title Deep Neural Network Tools for Probability and Statistic Models

Version 0.0.8

Date 2026-11-06

Author Bingshu E. Chen [aut, cre],
Patrick Norman [aut, ctb],
Wenyu Jiang [ctb],
Wanlu Li [ctb]

Maintainer Bingshu E. Chen <bingshu.chen@queensu.ca>

Depends R (>= 3.5.0), ggplot2, lpl(>= 0.12), Rcpp, survival

Imports methods

LinkingTo Rcpp, RcppArmadillo

Description Contains a robust set of tools designed for constructing deep neural networks, which are highly adaptable with user-defined loss function and probability models. It includes several practical applications, such as the (deepAFT) model, which utilizes a deep neural network approach to enhance the accelerated failure time (AFT) model for survival data. Another example is the (deepGLM) model that applies deep neural network to the generalized linear model (glm), accommodating data types with continuous, categorical and Poisson distributions.

License GPL (>= 2)

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-11-06 04:10:02 UTC

Contents

dnn-package 2
activation 3
bwdNN 5
deepAFT 6

deepGLM 9
deepSurv 11
dnnControl 13
dnnFit 15
dNNmodel 17
fwdNN 19
hyperTuning 20
ibs.deepAFT 22
msePICW 23
optimizerSGD 24
plot 25
predict 26
print 27
residuals 28
rmst.deepSurv 29
survfit 31

Index **33**

---

dnn-package *An R package for the deep neural networks probability and statistics models*

Description

This package provides tools for deep neural network which allow user define loss function for complex outcome data with probability and statistics models such as generalized linear models, accelerated failure time (AFT) models, and Cox proportional hazards models.

It contains the essential building blocks such as feed forward network and back propagation. This gives users the flexibility to write their own loss function (i.e. cost function) and train the neural network.

Details

{dnn} is a R package for deep learning neural network with probability models that use the negative of the log-likelihood as the loss function. It provides functions for feed forward network from covariates to the output layer and back propagation to find the derivatives of the weight parameters. Different optimization methods such as stochastic gradient descent (SGD), Momentum and ADAM can be used to train the network.

Currently, { dnn } can be install by the package source file 'dnn.tar.gz', use

install.packages("dnn.tar.gz", repos = NULL, type = "source")

users can use the following steps to install the most recent version of 'dnn' package:

1. First, you need to install the 'devtools' package. You can skip this step if you have 'devtools' installed in your R. Invoke R and then type

install.packages("devtools")

2. Load the devtools package.

library(devtools)

3. Install "dnn" package from github with R command

install_github("statapps/dnn")

A stable version of View the "dnn" package is also available from the Comprehensive R Archive Network (https://CRAN.R-project.org/package=dnn) and can be installed using R command

install.packages("dnn")

Author(s)

Bingshu E. Chen

Maintainer: Bingshu E. Chen <bingshu.chen@queensu.ca>

See Also

dNNmodel, bwdNN, fwdNN, deepAFT, deepGLM, deepSurv, glm, coxph, survreg

Examples

```
# Create the models with 3 layers
  model = dNNmodel(units=c(8, 6, 1), activation = c('elu', 'relu', 'sigmoid'),
          input_shape = c(3))
  print(model)
#
# Feed forward network with dummy data x
  x = matrix(runif(15), nrow = 5, ncol = 3)
  cache = fwdNN(x, model)
#
# Back propagation with dummy dy = dL/dyhat and minin batch for SGD
  dy = as.matrix(runif(5, -0.1, 0.1), nrow = 5)
  dW = bwdNN(dy, cache, model)
#
# Gradient descent with SGD
  lr_rate = 0.0001
  sgd = function(w, dw) {w-lr_rate*dw}
  model$params = mapply(sgd, w = model$params, dw = dW)
```

`activation` *Activation function*

Description

Different type of activation functions and the corresponding derivatives

Usage

```
sigmoid(x)
elu(x)
relu(x)
lrelu(x)
idu(x)
dsigmoid(y, dy = 1)
dsoftmax(y, dy = 1)
delu(y, dy = 1)
drelu(y, dy = 1)
dlrelu(y, dy = 1)
dtanh(y, dy = 1)   #activation function tanh(x) is already available in R
```

Arguments

x	A numeric matrix input of the activation function
y	A numeric matrix respresentating the output of the activatin function, serving as the input of the derivative of the activation function
dy	The gradient of the loss with respect to the output y. Defaults to 1. Must match the dimensions of y.

Details

Each function returns either the activation function (e.g. sigmoid, relu) or its derivative (e.g. dsigmoid, drelu).

For dsoftmax, the function computes the Jacobian-vector product. When dy represents the gradient of the loss L with respect to the softmax output P, the gradient with respect to the input x is:

$$\frac{\partial L}{\partial x} = P \odot (dy - \sum(P \odot dy))$$

where $\odot$ denotes element-wise multiplication.

Value

An activation function is applied to x and returns a matrix the same size as x. The detail formula for each activation function is:

sigmoid	return 1/(1+exp(-x))
elu	return x for x>0 and exp(x)-1 for x<0
relu	return x for x>0 and 0 for x<0
lrelu	return x for x>0 and 0.1*x for x<0
tanh	return tanh(x)
idu	return (x)

Author(s)

Bingshu E. Chen

See Also

softmax, bwdNN, fwdNN, dNNmodel, optimizerSGD, optimizerNAG

Examples

```
# Specify a dnn nodel with user define activation function in layer 2.
model = dNNmodel(units=c(8, 6, 1), activation= c('relu', 'softmax', 'sigmoid'),
        input_shape = c(3))
print(model)
```

---

bwdNN *Back propagation for dnn Models*

Description

{bwdNN} is an R function for back propagation in DNN network.

Usage

```
#
# To apply back propagation in with a feed forward model
#
# use
#
   bwdNN(dy, cache, model)
#
# to calculate derivative of dL/dW
```

Arguments

dy	the derivative of the cost function with respect to the output layer of the fwdNN function.
cache	the cached output of fwdNN.
model	a model return from dNNmodel function.

Details

Here 'dy' plays an import role in the back propagation { bwdNN } since the probability model's loss function takes the output layer of the { dnn } (denote as yhat) as one of its parameter. Then 'dy' equals to the partial derivative of the loss function (-Log Likelihood) with respect to yhat, that is, dy = dL/d(yhat). For example, if the 'dnn' predicts the probability (yhat = p) for the mixture of two populations f1 and f2, then the likelihood function is f = p*f1 + (1-p)*f2, and the loss function is L = -log(p*f1+(1-p)*f2). Hence, dy = dL/dp = -(f1-f2)/f.

'cache' is the cache of each input layer generated from the { fwdNN } function.

The function { bwdCheck } calculates the numerical derivatives of dL/dW, which can be used to check if the back propagation is correct or not, see example below.

Value

A list contains the derivatives dL/dW of weight parameter W is returned.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

See Also

dNNmodel, fwdNN, plot.dNNmodel, print.dNNmodel, summary.dNNmodel,

Examples

```
### define a dnn model, calculate the feed forward network
   model = dNNmodel(units = c(8, 6, 1),
           activation = c("elu", "sigmoid", "sigmoid"), input_shape = 3)
   print(model)
   x = matrix(runif(15), nrow = 5, ncol = 3)
   cache = fwdNN(x, model)
   # dy = dL/dp, where L is the cost function such as the
   # log-likehood and p is the output layer parameter of the DNN
   dy = as.matrix(runif(5, -0.1, 0.1), nrow = 5)  # a dummy dy for bwdNN input
   y  = predict(model, x) + dy

   # back propagation
   dW = bwdNN(dy, cache, model)
   dw = bwdCheck(x, y, model)
   print(dW[[1]])
   print(dw[[1]])
```

---

deepAFT *Deep learning for the accelerated failure time (AFT) model*

Description

Fit a deep learning survival regression model. These are location-scale models for an arbitrary transform of the time variable; the most common cases use a log transformation, leading to accelerated failure time models.

Usage

```
deepAFT(x, ...)

## S3 method for class 'formula'
deepAFT(formula, model, data, control = list(...), method =
                 c("BuckleyJames", "ipcw", "transform"), ...)

## Default S3 method:
```

```
deepAFT(x, y, model, control, ...)

## S3 method for class 'ipcw'
deepAFT(x, y, model, control, ...)
# use:
#   deepAFT.ipcw(x, y, model, control)
# or
#   class(x) = "ipcw"
#   deepAFT(x, y, model, control)
#
## S3 method for class 'trans'
deepAFT(x, y, model, control, ...)
# use:
#   class(x) = "transform"
#   deepAFT(x, y, model, control)
```

Arguments

formula	a formula expression as for other regression models. The response is usually a survival object as returned by the 'Surv' function. See the documentation for 'Surv', 'lm' and 'formula' for details.
model	a deep neural network model, created by function dNNmodel().
data	a data.frame in which to interpret the variables named in the formula.
x	Covariates for the AFT model
y	Response Surv object for the AFT model
method	methods to handle censoring data in deep AFT model fit, 'BuckleyJames' for Buckley and James method, 'ipcw' for inverse probability censoring weights method. 'transform' for transformation based on book of Fan and Gijbels (1996, page 168)
control	a list of control values, in the format produced by 'dnnControl'. The default value 'dnnControl()'
...	optional arguments

Details

When all observations are uncensored, the Buckley-James imputation step returns the observed times directly; no censoring-time maximum is needed.

For the IPCW method, the marginal censoring distribution can be estimated for datasets with fewer than 20 observations; there is no minimum sample-size threshold. The internal grouped nonparametric calculation permits at most five nonempty censoring groups, independently of sample size. Groups with one distinct observed time use their fitted survival value directly. The existing replacement of zero censoring survival by the smallest positive estimate within the group is retained. If no positive estimate exists, IPCW reports that the weights are undefined.

See "Deep learning with R" for details on how to build a deep learning model.

The following parameters in 'dnnControl' will be used to control the model fit process.

'epochs': number of deep learning epochs, default is 100.

'batch_size': batch size, default is 128. 'NaN' may be generated if batch size is too small and there is not event in a batch.

'verbose': verbose = 1 for print out verbose during the model fit, 0 for not print.

'epsilon': epsilon for convergence check, default is epsilon = 0.001.

'max.iter': number of maximum iteration, default is max.iter = 100.

When the variance for covariance matrix X is too large, please use xbar = apply(x, 2, stndx) to standardize X.

Value

An object of class "deepAFT" is returned. The deepAFT object contains the following list components:

`x`	Covariates for the AFT model
`y`	Survival object for the AFT model, y = Surv(time, event)
`model`	A fitted artificial neural network (ANN) model
`mean.ipt`	mean survival or censoring time
`predictor`	predictor score mu = f(x)
`risk`	risk score = exp(predictor)
`method`	method for deepAFT fitting, either Buckley-James, IPCW or transformed model

Note

For right censored survival time only

Author(s)

Chen, B. E. and Norman P.

References

Buckley, J. and James, I. (1979). Linear regression with cencored data. Biometrika, 66, page 429-436.

Norman, P., Li, W., Jiang, W. and Chen, B. E. (2024). deepAFT: A non-linear accelerated failure time model with artificial neural network. Statistics in Medicine, 43, page 3689-3701.

See Also

`dNNmodel`, `dnnControl`, `deepSurv`, `print.deepAFT`, `survreg`, `ibs.deepAFT`

Examples

```
## Example for deep learning model for AFT survival data
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(15), nrow = 5, ncol = 3)
  time = exp(x[, 1])
  status = c(1, 0, 1, 1, 1)
  fit = deepAFT(Surv(time, status) ~ x, model)
```

---

deepGLM *Deep learning for the generalized linear models*

Description

Fit generalized linear models (Gaussian, Binomial and Poisson) using deep learning neural network (DNN). The glm formula is specified by giving a symbolic description of the predictor and a description of the error distribution.

Usage

```
deepGlm(formula, model, family = c("gaussian", "binomial",
        "poisson"), data, epochs = 200, lr_rate = 1e-04,
         batch_size = 64, alpha = 0.7, lambda = 1, verbose = 0,
         weights = 1, ...)
```

Arguments

formula	a formula expression as for other regression models. The response is usually an object for glm response variable. See the documentation for 'glm', 'lm' and 'formula' for details.
model	a deep neural network model, created by function dNNmodel().
family	a description of the error distribution and link function to be used in the model. This can be either a character string of 'gaussian', 'binomial', or 'poisson', naming a family function, or result of a call to a family function (See 'family' for details of family functions).)
data	a data.frame in which to interpret the variables named in the formula.
epochs	number of deep learning epochs, default is 200.
batch_size	batch size, default is 64. 'NaN' may be generated if batch size is too small and there is not event in a batch.
lr_rate	learning rate for the gradient descent algorithm, default is lr_rate = 1e-04.
weights	the scalar 1 (the default) for unit weights, or a finite, nonnegative numeric vector with one value per observation. A one-column numeric matrix is converted to a vector. NULL and other matrix shapes are not accepted.

alpha	momentum rate for the gradient descent method, alpha takes value in [0, 1), default is alpha = 0.70.
lambda	L2 regularization parameter for deep learning.
verbose	verbose = 1 for print out verbose during the model fit, 0 for not print.
...	optional arguments

Details

Weights are used as supplied and are not normalized. Scalar 1 means unit weights for all observations. Weight vectors need not sum to 1.

For the binomial family, neural-network outputs represent probabilities. Fitting minimizes weighted cross-entropy (the negative log likelihood), with weights applied to both success and failure contributions.

For the Poisson family, the neural-network output represents a positive Poisson rate. Fitting minimizes the weighted negative log likelihood, omitting response-only log-factorial constants. Weights scale observation contributions; they do not scale the predicted rates themselves.

See dNNmodel for details on how to specify a deep learning model.

The following parameters in 'dnnControl' will be used to control the model fit process.

'epochs': number of deep learning epochs, default is 200.

'verbose': verbose = 1 for print out verbose during the model fit, 0 for not print.

When the variance for covariance matrix X is too large, please use xbar = scale(x) to standardize X.

Value

An object of class "deepGlm" is returned. The deepGlm object contains the following list components:

x	Covariates for glm model
y	Object for glm model
model	dnn model
predictor	predictor score mu = f(x)
risk	risk score = exp(predictor)

Note

For glm models with Gaussian, Binomial and Poisson only

Author(s)

Bingshu E. Chen

References

Chollet, F. and Allaire J. J. (2017). Deep learning with R. Manning.

See Also

dNNmodel, deepAFT, deepSurv, dnnControl, glm, predict.deepGlm, print.deepGlm

Examples

```
## Example for deep learning for glm models
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(15), nrow = 5, ncol = 3)
  y = exp(x[, 1] + rnorm(5))

  fit = deepGlm(y ~ x, model, family = "gaussian")
```

---

deepSurv *Deep learning for the Cox proportional hazards model*

Description

Fit a survival regression model under the Cox proportional hazards assumption using deep learning neural network (DNN).

Usage

```
deepSurv(formula, model, data, epochs = 100, lr_rate = 0.0001,
       batch_size = 64, alpha = 0.7, lambda = 1, verbose = 0,
       epsilon = 0.001, weights = 1, ...)
```

Arguments

formula	a formula expression as for other regression models. The response is usually a survival object as returned by the 'Surv' function. See the documentation for 'Surv', 'lm' and 'formula' for details.
model	a deep neural network model, created by function dNNmodel().
data	a data.frame in which to interpret the variables named in the formula.
epochs	number of deep learning epochs, default is 100.
batch_size	batch size, default is 64. 'NaN' may be generated if batch size is too small and there is not event in a batch.
lr_rate	learning rate for the gradient descent algorithm, default is lr_rate = 1e-04.
weights	the scalar 1 (the default) for unit weights, or a finite, nonnegative numeric vector with one value per observation. A one-column numeric matrix is converted to a vector. NULL and other matrix shapes are not accepted.
alpha	momentum rate for the gradient descent method, alpha takes value in [0, 1), default is alpha = 0.70.

lambda	L2 regularization parameter for deep learning.
verbose	verbose = 1 for print out verbose during the model fit, 0 for not print.
epsilon	epsilon for convergence check, default is epsilon = 0.001.
...	optional arguments

Details

Cox partial likelihood uses the Breslow method for tied observed times, including events and censored observations at the same time in the risk set. The baseline cumulative hazard also uses Breslow increments and returns one row per distinct observed time, with grouped event and censoring counts. Direct calls to the fitters with Cox loss require finite times sorted from largest to smallest; deepSurv performs this sorting automatically. Tie groups are cached for full-batch fitting and rebuilt for sampled mini-batches. Mini-batch Cox loss uses risk sets within the sampled batch.

Weights are used as supplied and are not normalized. Scalar 1 means unit weights for all observations. Weight vectors need not sum to 1.

See dNNmodel for details on how to build a deep learning model.

For a deepSurv model, the activation function for the output layer of the dNN model shall be an identity function 'idu'.

When the variance for covariance matrix X is too large, please use xbar = scale(x) to standardize X.

Value

An object of class "deepSurv" is returned. The deepSurv object contains the following list components:

x	Covariates for Cox model
y	Surv object for Cox model
model	dnn model with fitted weights
predictor	predictor score = f(x)
risk	risk score = exp(predictor)

Note

For right censored survival time only

Author(s)

Chen, B. E. wrote the R code using the partial likelihood cost function proposed by Katzman et al (2018).

References

Katzman JL, Shaham U, Cloninger A, Bates J, Jiang T, Kluger Y. DeepSurv: Personalized treatment recommender system using a Cox proportional hazards deep neural network. BMC Medical Research Methodology 2018; 18: 24.

See Also

deepAFT, deepGlm, dNNmodel, print.deepSurv, survreg, coxph, Surv

Examples

```
## Example for deep learning proportional hazards survival model
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "idu"),
                   input_shape = 3)
  n = 15
  x = matrix(runif(n*3), nrow = n, ncol = 3)
  time = rexp(n, rate = 3*(0.1+abs(x[, 1]-0.5)))
  status = rbinom(n, 1, 0.8)
  fit = deepSurv(Surv(time, status) ~ x, model = model, epochs = 50)
```

---

dnnControl *Auxiliary function for* dnnFit *dnnFit*

Description

dnnControl is an auxiliary function for dnnFit. Typically only used internally by the dnn package, may be used to construct a control argument for the deep learning neural network model to specify parameters such as a loss function.

Usage

```
dnnControl(loss = c("mse", "cox", "bin", "log", "mae"), epochs = 300,
    batch_size = 64, verbose = 0, lr_rate = 0.0001,
    alpha = 0.5, lambda = 1.0, epsilon = 0.01, max.iter = 100,
    weights = 1)
```

Arguments

loss	loss function for the neural network model, "mse" for mean square error (guassian glm model), "mae" for mean absolute error, "cox" for the Cox partial likelihood (proportional hazards model), "bin" for cross-entropy (binomial glm model), "log" for log-linear (poisson glm model).
epochs	positive integer number of deep learning epochs, default is 300.
batch_size	positive integer batch size, default is 64. 'NaN' may be generated if batch size is too small and there is not event in a batch.
lr_rate	learning rate, default is 0.0001.
weights	the scalar 1 (the default) for unit weights, or a finite, nonnegative numeric vector with one value per observation. A one-column numeric matrix is converted to a vector. NULL and other matrix shapes are not accepted.
alpha	alpha decay rate for momentum gradient descent, default is 0.5.

lambda	regularization term for dnn weighting parameters, 0.5*lambda*W*W), default is 1.0.
verbose	verbose = 1 for print out verbose during the model fit, 0 for not print.
epsilon	epsilon for convergence check, default is epsilon = 0.01.
max.iter	positive integer maximum iteration count, default is max.iter = 100. This is used in the deepAFT function

Details

Epochs, batch size, and maximum iteration counts must be finite positive integers no greater than the platform integer limit. Zero, negative, fractional, missing, and infinite counts are rejected before fitting.

Weights are used as supplied and are not normalized. Scalar 1 means unit weights for all observations. Weight vectors need not sum to 1.

dnnControl is used in model fitting of "dnnFit". Additional loss functions will be added to the library in the future.

Value

This function checks the internal consistency and returns a list of values as input to control model fitting of "dnnFit".

Author(s)

Chen, B. E. (bingshu.chen@queensu.ca)

References

Norman, P., Li, W., Jiang, W. and Chen, B. E. (2024). deepAFT: A non-linear accelerated failure time model with artificial neural network. Statistics in Medicine.

See Also

deepAFT, deepGLM, deepSurv, dnnFit

Examples

```
## Example for dnnControl
##
# model = dNNmodel()

  control = dnnControl(loss='mse')

# can also be used in
# fit = dnnFit(y ~ x, model, control)
# print(fit)
```

dnnFit *Fitting a Deep Learning model with a given loss function*

Description

dnnFit is used to train a deep learning neural network model based on a specified loss function.

Usage

```
dnnFit(x, y, model, control)
```

Arguments

x	covariates for the neural network model
y	output (target) value for neural network model
model	the neural network model, see below for details
control	a list of control values created by dnnControl(). Both fitters reuse its validated epochs and batch_size values without repeating the count checks.

Details

Cox partial likelihood uses the Breslow method for tied observed times, including events and censored observations at the same time in the risk set. Direct calls to the fitters with Cox loss require finite times sorted from largest to smallest; deepSurv performs this sorting automatically. Tie groups are cached for full-batch fitting and rebuilt for sampled mini-batches. Mini-batch Cox loss uses risk sets within the sampled batch.

The 'dnnFit' function takes the input data, the target values, the network architecture, and the loss function as arguments, and returns a trained model that minimizes the loss function. The function also supports various options for regularization and optimization of the model.

See dNNmodel for details on how to specify a deep learning model.

Parameters in dnnControl will be used to control the model fit process. The loss function can be specified as dnnControl(loss = "lossFunction"). Currently, the following loss functions are supported:

'mse': Weighted square error loss = 0.5*sum(weights*(y-yhat)^2)

'cox': Cox partial likelihood loss = -sum(delta*(yhat - log(S0)))

'bin': Weighted cross-entropy = -sum(weights*(y*log(p) + (1-y)*log(1-p)))

'log': Poisson cost = sum(weights*(lambda-y*log(lambda)))

'mae': Weighted absolute error loss = sum(weights*abs(y-yhat))

Weights supplied through dnnControl(weights = ...) must be the scalar 1 (the default) or a finite, nonnegative numeric vector of length equal to the sample size. An $n \times 1$ numeric matrix is converted to a vector before validation. Other matrix shapes, NULL, and per-output weight arrays are not accepted. Each observation's weight is shared across all response outputs. Weights need not sum to 1 and are not normalized. Scalar 1 means unit weights for all observations. Mini-batches select the

corresponding observation weights. Weight validation and column-matrix conversion are performed before training rather than at every loss evaluation.

For MSE loss, both implementations use the same weighted half-squared-error objective.

The package stops with an error when forward predictions, loss values, or output gradients become non-finite, including numerical overflow from extremely large values or invalid boundary arithmetic. Rescale the data or adjust model parameters and output activations. The underlying loss formulas are unchanged; no numerical stabilization or probability clipping is applied.

For MAE loss in `dnnFit`, the negative output subgradient used by backpropagation is `weights * sign(y - yhat)`. The subgradient is chosen as zero when the response equals the prediction. Zero-weight observations contribute neither loss nor gradient. The loss is a sum, not a normalized mean.

For binomial loss (`loss = "bin"`), predictions are probabilities. Both implementations report the weighted negative log likelihood and apply weights to the complete success-and-failure contribution. The negative output derivative used in backpropagation is `weights * (y / yhat - (1 - y) / (1 - yhat))`. Native binomial fitting uses the same observation-weight contract as MSE fitting.

For Poisson loss (`loss = "log"`), predictions are positive rates, not log rates. The objective omits the response-only log-factorial constant. Weights multiply each observation's loss contribution rather than changing its predicted rate. Both R and native fitting use the negative output derivative `weights * (y / yhat - 1)` for backpropagation. Native Poisson fitting uses the same observation-weight contract as MSE fitting.

After training, `dnnFit` disables dropout before evaluating final predictions, loss, residuals, and `dvi` on the full training data. These diagnostics therefore match the returned prediction model rather than the last mini-batch. The `history` and `dW` fields retain their training-iteration meanings; they are not recomputed as final diagnostics.

Dropout is disabled for Rcpp fitting. A positive dropout rate produces a warning and is reset to zero; stored dropout masks are cleared. This applies to `dnnFit` when `model$Rcpp = TRUE`, and to all `dnnFit2` calls because that fitter always uses native code. The supplied input model is not modified by `dnnFit2`. R-only fitting retains its existing dropout option.

Single-layer networks (no hidden layer) are supported by both fitters and by the R backpropagation implementation. Optimizer state is initialized for the supplied architecture. Native Cox fitting accepts the default `weights = 1`; Cox sample weights remain unsupported.

The native `dnnFit2` fitter uses momentum optimization regardless of the `model$optimizer` setting. To select NAG or Adam, use `dnnFit` with `model$Rcpp = TRUE`.

The native `dnnFit2` fitter trains a copy of the supplied model. The input model is not modified in place; use the returned `model` component to obtain the trained weights. Reusing the same input model therefore starts subsequent fits from the same supplied weights.

When the variance for covariance matrix X is too large, please use xbar = scale(x) to standardize X.

Value

An object of class "dnnFit" is returned. The dnnFit object contains the following list components:

`cost`	loss evaluated on the full training data using the final prediction model.
`dW`	the gradient at the final epoch dW = dL/dW.
`fitted.values`	predictor value mu = f(x).
`history`	a cost history at each epoch.

lp	predictor value mu = f(x).
logLik	-2*log Likelihood = cost.
model	a dNNmodel object with fitted weights.
residuals	loss-specific negative output derivative evaluated using the final model and all training observations. For weighted MSE this is weights*(y-yhat); for Cox loss it is status-H0(t)exp(lp).
dvi	elementwise square of the returned residuals, dvi = dy*dy.

Author(s)

Chen, B. E. (bingshu.chen@queensu.ca)

References

Chollet, F. and Allaire J. J. (2017). Deep learning with R. Manning.

Norman, P., Li, W., Jiang, W. and Chen, B. E. (2024). deepAFT: A non-linear accelerated failure time model with artificial neural network. Statistics in Medicine.

See Also

deepAFT, deepGLM, deepSurv, dnnControl

Examples

```
## Example for dnnFit with MSE loss function to do a non-linear regression
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(15), nrow = 5, ncol = 3)
  y = exp(x[, 1])
  control = dnnControl(loss='mse')
  fit = dnnFit(x, y, model, control)
```

---

dNNmodel *Specify a deep neural network model*

Description

{dNNmodel} is an R function to create a deep neural network model that is to be used in the feed forward network { fwdNN } and back propagation { bwdNN }.

Usage

```
dNNmodel(units, input_shape, activation=NULL, type = NULL,
      N = NULL, dropout_rate = 0, Rcpp=TRUE,
      optimizer = c("momentum", "nag", "adam"))
```

Arguments

units	number of nodes for each layer
activation	activation function
input_shape	the number of columns of input X, default is NULL.
N	the number of training sample, default is NULL.
dropout_rate	dropout rate, default is 0. The existing dropout option is available only with Rcpp = FALSE. If Rcpp = TRUE, a positive rate produces a warning and is reset to zero.
type	default is "dense", currently only support dense layer.
Rcpp	use Rcpp (C++ for R) to speed up the fwdNN and bwdNN, default is "TRUE".
optimizer	optimizer used in SGD, default is "momentum".

Details

dNNmodel returns an object of class "dNNmodel".

The function "print" (i.e., "print.dNNmodel") can be used to print a summary of the dnn model,

The function "summary" (i.e., "summary.dNNmodel") can be used to print a summary of the dnn model,

Value

An object of class "dNNmodel" is a list containing at least the following components:

units	number of nodes for each layer
activation	activation function
drvfun	derivative of the activation function
params	the initial values of the parameters, to be updated in model training.
input_shape	the number of columns of input X, default is NULL.
N	the number of training sample, default is NULL.
type	default is "dense", currently only support dense layer.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

See Also

plot.dNNmodel, print.dNNmodel, summary.dNNmodel, fwdNN, bwdNN, optimizerSGD, optimizerNAG,

Examples

```
### To define a dnn model
 model = dNNmodel(units = c(8, 6, 1), activation = c("relu", "sigmoid", "sigmoid"),
         input_shape = c(3))
```

fwdNN *Feed forward and back propagation for dnn Models*

Description

{fwdNN} is an R function for feed forward network.

Usage

```
   fwdNN(X, model)
#
# to calculate a feed feedward model
#
```

Arguments

X	For "dNNmodel", X is a design matrix of dimension n * p.
model	a model return from dNNmodel function.

Details

'cache' is the cache of each input layer, will be used in the bwdNN function.

Value

The function fwdNN return a list containing at least the following components:

cache	a list contains the values of each output layer after activation function transformation and adding the intercept term (i.e. the bias term). The intercept does not add to the output layer in the cache.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

See Also

bwdNN, plot.dNNmodel, print.dNNmodel, summary.dNNmodel,

Examples

```
### define a dnn model, calculate the feed forward network
   model = dNNmodel(units = c(8, 6, 1), activation = c("elu", "sigmoid", "sigmoid"),
                    input_shape = 3)

### feed forward with a dummy x matrix
   x = matrix(runif(15), nrow = 5, ncol = 3)
   cache = fwdNN(x, model)
```

hyperTuning *A function for tuning of the hyper parameters*

Description

{ hyperTuning} is a tuning tool to find the optimal hyper parameter for the ANN model.

Usage

```
   hyperTuning(x, y, model, ER = c("cindex", "mse"),
          method = c('BuckleyJames', 'ipcw', 'transform', 'deepSurv'),
          lower = NULL, upper = NULL, node = FALSE,
          K = 5, R = 25)
### additional function used in hyperTuning is cross-validation prediction error
#
#  CVpredErr(x, y, model, control, method)
#
```

Arguments

x	Covariates for the deep neural network model
y	Surv object for the deep neural network model
model	A deep neural network model, created by function dNNmodel().
ER	Prediction error measurement to be used in the cross vaditation, can be either a concordance index (cindex) or a mean square error (mse), default is cindex
method	Methods to handle censoring data in deep AFT model fit, 'BuckleyJames' for the Buckley and James method, 'ipcw' for the inverse probability censoring weights method. 'transform' for the transformation method based on book of Fan and Gijbels (1996, page 168). 'deepSurv' for the deepSurv model(Katzman, 2017)
node	Tuning the number of nodes in each hidden layer, default is FALSE
K	integer number of cross-validation folds, between 2 and the sample size; default is K = 5.
lower, upper	Bounds on the hyper parameters for the deep learning method. If NULL, then the default value for lower = dnnControl(alpha = 0.5, lambda = 1.0, lr_rate = 0.0001), upper = dnnControl(alpha = 0.97, lambda = 10, lr_rate = 0.001).
R	positive integer number of attempted random draws from the hyperparameter space, including attempts whose cross-validation fails; default is R = 25.

Details

The reported prediction error is the mean of the fold scores. The reported SE is their sample standard deviation divided by the square root of the number of folds. This is a nominal standard error of the fold-score mean; it does not correct for dependence caused by overlapping training sets and should not be interpreted as a generalization-error confidence interval. Invalid fold counts or tuning-trial counts are rejected before tuning. Failed cross-validation attempts count toward R. If no candidate

is selected after these attempts, tuning stops with an error. When node = TRUE, each candidate is rebuilt with freshly initialized parameter matrices matching its sampled hidden-layer widths. The output-layer width and the supplied model's other architecture settings are preserved. Hidden layers with at least two nodes are sampled from two through their original width; one-node hidden layers remain unchanged. Models without hidden layers retain their architecture. The returned model is the candidate associated with the best cross-validation score and returned control values; it is an initialization, not a model fitted on the full dataset.

A random search method is used to optimal hyper parameter (Bergstra and Bengio, 2012). The function { CVpredErr} will be call to calculate the cross-validation prediction error for the given x and y with the specified method from the input argument.

Value

A list of "model" and "dnnControl" is returned. The list contains at least the following components,

model	The "model" contains the optimal number of nodes for each hidden layer in the model specified by dNNmodel
control	The "control" contains the optimal tuning parameters with list components the same as those created by dnnControl

Author(s)

Chen, B. E. (chenbe@queensu.ca)

References

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. The Journal of Machine Learning Research. 13, page 281-305.

See Also

deepAFT, deepGLM, deepSurv, dnnFit

Examples

```
### Tuning the hyper parameter for a deepAFT model:
#### cross-validation take a long time to run.

  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(45), nrow = 15, ncol = 3)
  time = exp(x[, 1])
  status = rbinom(15, 1, 0.5)
  y = Surv(time, status)
  ctl = dnnControl(epochs = 30)
  hyperTuning(x, y, model, method = "BuckleyJames", K = 2, R = 2, lower = ctl)
```

`ibs.deepAFT` *Calculate integrated Brier Score for deepAFT*

Description

The function ibs is used to calculate integrated Brier Score for deepAFT.

Usage

```
### To calculate Brier score for the original fitted data or the new data
### with new outcomes
## S3 method for class 'deepAFT'
ibs(object, newdata=NULL, newy = NULL, ...)
```

Arguments

`object`	the results of a deepAFT fit.
`newdata`	optional argument, if no null, new data and new y will be used for calculation.
`newy`	optional argument, used together with new data.
`...`	other unused arguments.

Details

Observations may be supplied in any row order. Survival indicators and IPCW weights use comparisons of observed times rather than row positions. At an evaluation time t, subjects with observed time greater than t contribute the survival term; observed events at or before t contribute the event term. Subjects censored at or before t contribute zero. Tied times therefore have identical Brier scores.

The returned `bs` rows are ordered by increasing observed time, retaining duplicate times. The reported `ibs` is the unnormalized sum of each Brier score multiplied by the preceding interval width, starting at zero. Duplicate times have zero interval width. The existing censoring-survival convention and its zero-survival replacement of `0.0001` are retained.

ibs is called to calculate integrate Brier score for the deepAFT model `deepAFT`.

Value

A list contains the integrate Brier score and the Brier score is returned:

`ibs`	Integerate Brier score
`bs`	Brier score

Author(s)

Bingshu E. Chen

See Also

deepAFT

`msePICW` *Mean Square Error (mse) for a fitted survival Object*

Description

Compute Mean Square Error (mse) values for a fitted AFT model survival object using IPCW method.

Usage

```
## S3 method for 'mseIPCW'
mseIPCW(object, newdata, newy)
```

Arguments

`object`	the results of a model fit using a deepAFT or a survreg function.
`newdata`	optional new data at which to do predictions. If absent, predictions are for the dataframe used in the original fit.
`newy`	optional new outcome variable y. newy is reqired if newdata is not null.

Details

predict is called to predict object from a deepAFT deepAFT or a survreg model.

IPCW method is used to calcuate the mean square error for censored survival time.

Value

mseIPCW returns the mse for the predicted survival data.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

See Also

predict, deepAFT, deepSurv, survfit.deepSurv

optimizerSGD *Functions to optimize the gradient descent of a cost function*

Description

Different type of optimizer functions such as SGD, Momentum, AdamG and NAG.

Usage

```
optimizerMomentum(V, dW, W, alpha = 0.63, lr = 1e-4, lambda = 1)
```

Arguments

V	Momentum V = alpha*V - lr*(dW + lambda*W); W = W + V. NAG V = alpha*(V - lr*(dW + lambda*W); W = W + V - lr*(dW + lambda*W)
dW	derivative of cost with respect to W, can be founde by dW = bwdNN2(dy, cache, model),
W	weights for DNN model, optimizerd by W = W + V
alpha	Momentum rate 0 < alpha < 1, default is alpah = 0.5.
lr	learning rate, default is lr = 0.001.
lambda	regulation rate for cost + 0.5*lambda*‖W‖, default is lambda = 1.0.

Details

For SGD with momentum, use

V = 0; obj = optimizerMomentum(V, dW, W); V = obj$V; W = obj$W

For SDG with MAG

V = 0; obj = optimizerNAG(V, dW, W); V = obj$V; W = obj$W

Value

return and updated W and other parameters such as V, V1 and V2 that will be used on SGD.

Author(s)

Bingshu E. Chen

See Also

activation, bwdNN, fwdNN, dNNmodel, dnnFit

plot *Plot methods in dnn package*

Description

Plot function for plotting of R objects in the dnn package.

Several different type of plots can be produced for the deep learning mdels. Plot method is used to provide a summary of outputs from "deepAFT", "deepGLM", "deepSurv" and "dnn".

Use "methods(plot)" and the documentation for these for other plot methods.

Usage

```
## S3 method for class 'dNNmodel'
plot(x, ...)
## S3 method for class 'deepAFT'
plot(x, type = c("predicted", "residuals", "baselineKM"), ...)
```

Arguments

x	a class of "dNNmodel".
type	type of plot in deepAFT object, "predicted" to plot the linear predicted values, "residuals" to plot residuals, "baselineKM" to plot baseline Kaplan-Meier survival curve.
...	other options used in plot().

Details

plot.deepAFT is called to plot the fitted deep learning AFT model.

plot.dNNmodel is called to plot fitted dnn model

The default method, plot.default has its own help page. Use methods("plot") to get all the methods for the plot generic.

Value

No return value, called to plot a figure.

Author(s)

Bingshu E. Chen

See Also

The default method for plot `plot.default`. `glm`

predict *Predicted Values for a deepAFT or a deepSurv Object*

Description

Compute the predicted values for a deepAFT or a deepSurv object

Usage

```
## S3 method for class 'predict'
## S3 method for class 'dSurv'
predict(object, newdata, newy=NULL, ...)
```

Arguments

object	the results of a model fit using the deepAFT or the deepSurv function.
newdata	optional new data at which to do predictions. If absent, predictions are for the dataframe used in the original fit.
newy	optional new outcome variable y.
...	other options used in predict().

Details

When newy is supplied for a deepSurv model, martingale residuals use $\delta - \exp(\eta)H_0(t)$ at the original observed times. The baseline cumulative hazard is evaluated as a step function, with zero cumulative hazard before the first baseline time. DeepAFT models instead use the AFT time transformation.

predict.dSurv is called to predict object from a deepAFT deepAFT or a deepSurv deepSurv model.

The default method, predict has its own help page. Use methods("predict") to get all the methods for the predict generic.

Value

predict.dSurv returns a list of the predicted values, risk scores and the cumulative hazard function.

lp	covariate effect predictor beta(X), wherebeta() is the fitted regression function and X is the covariate matrix.
risk	risk score, risk = exp(-lp) for the deepAFT model and risk = exp(lp) for the deepSurv model. When new y is provided, both lp and risk will be ordered by survival time of the new y.
cumhaz	the fitted cumulative hzard function for the training data with covariate X = 0 will be reported if new y is missing.
time	time for cumulative hazard function. Time from new y will be used if new y is provided.
cindex	the concordance index for the fitted model or the predicted model if new y are provided.
c.index	the numerical value of the cindex.

Author(s)

Bingshu E. Chen

See Also

The default method for predict `predict`, `deepAFT`, `deepSurv`, `survfit.deepSurv`

`print` *print a summary of fitted deep learning model object*

Description

print is used to provide a short summary of outputs from `deepAFT`, `deepSurv`, `deepGLM`, and `dNNmodel`.

Usage

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

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

Arguments

`x`	a class returned from deepAFT, deepSurv, deepGLM model fit or a dNNmodel
`object`	a class of deepAFT object
`...`	other options used in print()

Details

print.deepAFT is called to print object or summary of object from the deep learning AFT models `deepAFT`. summary(fit) provides detail summary of 'deepAFT' model fit, including predictors, baseline survival function for T0=T/exp(mu), and martingale residuals for the fitted model.

print.dNNmodel is called to print object or summary of object from the dNNmodel.

The default method, print.default has its own help page. Use methods("print") to get all the methods for the print generic.

Value

An object of class "summary.deepAFT" is returned. The object contains the following list components:

location	location parameter exp(mu), to predice the mean value of survival time.
sfit	survfit object of the baselie survival function of T0=T/exp(mu).
cindex	Concordance index of the fitted deepAFT model.
resid	martingle residuals of the fitted deepAFT model.
method	the model used to fit the deepAFT model.

Author(s)

Bingshu E. Chen

See Also

The default method for print `print.default`. Other methods include `coxph`, `survreg`, `deepAFT`, `summary`

`residuals` *Calculate Residuals for a deepAFT or deepSurv Fit*

Description

Calculates martingale, deviance or Cox-Snell residuals for a previously fitted deepAFT or deepSurv model.

Usage

```
## S3 method for class 'deepAFT'
## S3 method for class 'dSurv'
residuals(object, type = c("martingale", "deviance", "coxSnell"), ...)
```

Arguments

object	the results of a deepAFT or deepSurv fit.
type	character string indicating the type of residual desired. Possible values are "martingale", "deviance". Only enough of the string to determine a unique match is required.
...	other unused arguments.

Details

For a deepSurv model, the Cox-Snell residual is $H_i(T_i) = \exp(\eta_i)H_0(T_i)$, where η_i is the neural-network predictor and H_0 is the fitted baseline cumulative hazard. It is evaluated as a step function at the original observed time, with zero cumulative hazard before the first baseline time. The martingale residual is $\delta_i - H_i(T_i)$. AFT time rescaling is not used for deepSurv models.

residuals.deepAFT is called to compute baseline survival function S_T0(t) from the deepAFT model `deepAFT`, where T0 = T/exp(mu), or log(T) = log(T) - mu.

The default method, residuals has its own help page. Use methods("residuals") to get all the methods for the residuals generic.

Value

Fitted AFT survival estimates are preserved, including the final estimate. When the fitted survival is zero, the Cox-Snell residual is `Inf`, and the martingale and deviance residuals are `-Inf`.

For martingale and deviance residuals, the returned object is a vector with one element for each subject. The row order will match the input data for the original fit.

See `residuals` for more detail about other output values.

Note

For deviance residuals, the status variable may need to be reconstructed.

Author(s)

Bingshu E. Chen

See Also

The default method for residuals `residuals`, `predict.dSurv`, `survfit.deepSurv`, and `deepAFT`.

`rmst.deepSurv` *The restricted mean survival time (RMST)*

Description

Calculate the restricted mean survival time (RMST) for deepSurv survival objects.

Usage

```
## S3 method for class 'deepSurv'
rmst(object, newdata = NULL, risk = NULL, tau = NULL, ...)
```

Arguments

object	this is a survival object created by deepSurv.
risk	the predicted from the deepSUrv model, i.e. risk = exp(predict(model, newdata)).
newdata	optional new data at which the RMST is calculated. If absent, RMST is for the dataframe used in the original model fit.
tau	the upper bound of the restricted mean survival time calculation.
...	additional arguments to be passed to the functions such as rmst.coxph, rmst.lple, rmst.Surv etc.

Details

The restricted mean survival time (RMST) is the mean of the truncated survival time at some finite value tau. The RMST is defined as,

RMST(tau) = E(min(T, tau)) = int_0 ^tau S(t)dt,

where S(t) = P(T>t) is the survival function of the random variable T.

When risk or newdata is supplied, RMST is calculated by summing interval widths multiplied by the fitted survival probability, $S(t) = \exp\{-\text{risk}H_0(t)\}$. The baseline cumulative hazard is treated as a step function and is zero before the first baseline time. Thus RMST equals tau when the horizon precedes the first event. Beyond the last baseline time, the final fitted survival probability is held constant. The horizon must be finite and nonnegative, and risk values must be finite and nonnegative.

If both risk and newdata are NULL, the function calculates RMST directly from the response using rmst.

Value

A value or vector of the restricted mean survival time is returned.

Author(s)

Bingshu E. Chen

See Also

rmst, rmst.coxph, coxph, Surv

Examples

```
set.seed(29)
n     = 5
time  = rexp(n, 1)
event = rbinom(n, 1, 0.75)
x     = rnorm(n)
y     = Surv(time, event)

### calculate the restricted mean survival time at tau = 0.5
```

```
rms   = rmst(y, tau = 0.5)

### calculate the RMST for deepSurv
#fit   = deepSurv(y~x, model = model)
#RMST  = rmst(fit, risk = c(1, 2, 3), tau = 2)
```

`survfit` *Compute a Survival Curve from a deepAFT or a deepSurv Model*

Description

Computes the predicted survival function of a previously fitted deepAFT or deepSurv model.

Usage

```
## S3 method for class 'deepAFT' or 'deepSurv'
## S3 method for class 'deepAFT'
survfit(formula, se.fit=TRUE, conf.int=.95, ...)
## S3 method for class 'deepSurv'
survfit(formula, se.fit=TRUE, conf.int=.95, ...)
```

Arguments

`formula`	a deepAFT or deepSurv fit object.
`se.fit`	a logical value indicating whether standard errors shall be computed. Default is TRUE
`conf.int`	the level for a two-sided confidence interval on the survival curve. Default is 0.95
`...`	other unused arguments.

Details

survfit.dSurv is called to compuate baseline survival function S0(t) for T0 from the deepAFT model `deepAFT`, where T0 = T/exp(mu), or log(T) = log(T) - mu.

For the deepSurv model `deepSurv`, survfit.dSurv evaluates the Nelson-Aalen estimate of the baseline survival function with covariate X = 0.

The default method, survfit has its own help page. Use methods("survfit") to get all the methods for the survfit generic.

Value

survfit.dSurv returns a list of predicted baseline survival function, cumulative hazard function and residuals.

`surv`	Predicted baseline survival function S0(t).
`cumhaz`	Baseline cumulative hazard function, -log(surv).

hazard	Baseline hazard function.
varhaz	Variance of the baseline hazard.
residuals	Martingale residuals of the deepAFT or deepSurv model.
std.err	Standard error for the cumulative hazard function, if se.fit = TRUE.

See survfit for more detail about other output values such as upper, lower, conf.type. Confidence interval is based on log-transformation of survival function.

Author(s)

Bingshu E. Chen

See Also

The default method for survfit survfit, predict.dSurv

Index

∗ **Back propagation**
activation, 3
bwdNN, 5
∗ **Deep Neural Networks**
dnn-package, 2
∗ **Deep Neural Network**
activation, 3
bwdNN, 5
dNNmodel, 17
fwdNN, 19
∗ **Feed forward**
activation, 3
fwdNN, 19
∗ **Hyper parameter**
hyperTuning, 20
∗ **IPCW**
msePICW, 23
∗ **Integrated Brier Score**
ibs.deepAFT, 22
∗ **MSE**
msePICW, 23
∗ **Restricted Mean Survival Time**
rmst.deepSurv, 29
∗ **Survival Analysis**
rmst.deepSurv, 29
∗ **activation function**
activation, 3
∗ **dNNmodel**
deepAFT, 6
∗ **deepAFT**
deepAFT, 6
hyperTuning, 20
plot, 25
print, 27
∗ **deepGLM**
deepGLM, 9
plot, 25
print, 27
∗ **deepSurv**
deepSurv, 11
hyperTuning, 20
plot, 25
print, 27
∗ **dnnControl**
deepAFT, 6
∗ **dnnFit**
dnnControl, 13
dnnFit, 15
∗ **dnn**
deepGLM, 9
deepSurv, 11
∗ **optimizer AdamG**
optimizerSGD, 24
∗ **optimizer Momentum**
optimizerSGD, 24
∗ **optimizer NAG**
optimizerSGD, 24
∗ **optimizer SDG**
optimizerSGD, 24
∗ **plot**
plot, 25
∗ **predict**
predict, 26
∗ **print**
print, 27
∗ **residuals**
residuals, 28
∗ **summary**
plot, 25
print, 27
∗ **survfit**
survfit, 31

activation, 3, *24*

bwdCheck *(*bwdNN*)*, 5
bwdNN, *3*, *5*, 5, *18*, *19*, *24*
bwdNN2 *(*bwdNN*)*, 5

coxph, *3*, *13*, *28*, *30*
CVpredErr *(*hyperTuning*)*, 20

deepAFT, *3*, 6, *11*, *13, 14*, *17*, *21–23*, *26–29*, *31*
deepGLM, *3*, 9, *14*, *17*, *21*, *27*
deepGlm, *13*
deepGlm *(*deepGLM*)*, 9
deepSurv, *3*, *8*, *11*, 11, *14*, *17*, *21*, *23*, *26, 27*, *31*
delu *(*activation*)*, 3
didu *(*activation*)*, 3
dlrelu *(*activation*)*, 3
dnn *(*dnn-package*)*, 2
dnn-doc *(*dnn-package*)*, 2
dnn-package, 2
dnnControl, *8*, *11*, 13, *15*, *17*, *21*
dnnFit, *13, 14*, 15, *21*, *24*
dnnFit2 *(*dnnFit*)*, 15
dNNmodel, *3*, *5, 6*, *8*, *10–13*, *15*, 17, *21*, *24*, *27*
drelu *(*activation*)*, 3
dsigmoid *(*activation*)*, 3
dsoftmax *(*activation*)*, 3
dtanh *(*activation*)*, 3

elu *(*activation*)*, 3

fwdNN, *3*, *5, 6*, *18*, 19, *24*
fwdNN2 *(*fwdNN*)*, 19

glm, *3*, *11*, *25*

hyperTuning, 20

ibs.deepAFT, *8*, 22
idu *(*activation*)*, 3

lrelu *(*activation*)*, 3

mseIPCW *(*msePICW*)*, 23
msePICW, 23

optimizerAdamG *(*optimizerSGD*)*, 24
optimizerMomentum *(*optimizerSGD*)*, 24
optimizerNAG, *5*, *18*
optimizerNAG *(*optimizerSGD*)*, 24
optimizerSGD, *5*, *18*, 24

plot, 25
plot.default, *25*
plot.dNNmodel, *6*, *18, 19*
predict, *23*, 26, *27*
predict.deepGlm, *11*
predict.deepGlm *(*deepGLM*)*, 9
predict.dNNmodel *(*fwdNN*)*, 19
predict.dSurv, *29*, *32*
print, 27
print.deepAFT, *8*
print.deepGlm, *11*
print.deepSurv, *13*
print.default, *28*
print.dNNmodel, *6*, *18, 19*

relu *(*activation*)*, 3
residuals, 28, *29*
residuals.deepGlm *(*deepGLM*)*, 9
rmst, *30*
rmst.coxph, *30*
rmst.deepSurv, 29

sigmoid *(*activation*)*, 3
softmax, *5*
summary, *28*
summary.deepAFT *(*print*)*, 27
summary.deepGlm *(*deepGLM*)*, 9
summary.deepSurv *(*deepSurv*)*, 11
summary.dNNmodel, *6*, *18, 19*
summary.dNNmodel *(*print*)*, 27
Surv, *13*, *30*
survfit, 31, *32*
survfit.deepSurv, *23*, *27*, *29*
survreg, *3*, *8*, *13*, *28*