

Package ‘DDESONN’

May 7, 2026

Type Package

Title A Deep Dynamic Experimental Self-Organizing Neural Network Framework

Version 7.1.11

Description Provides a fully native R deep learning framework for constructing, training, evaluating, and inspecting Deep Dynamic Ensemble Self Organizing Neural Networks at research scale. The core engine is an object oriented R6 class-based implementation with explicit control over layer layout, dimensional flow, forward propagation, back propagation, and transparent optimizer state updates. The framework does not rely on external deep learning back ends, enabling direct inspection of model state, reproducible numerical behavior, and fine grained architectural control without requiring compiled dependencies or graphics processing unit specific run times. Users can define dimension agnostic single layer or deep multi-layer networks without hard coded architecture limits, with per layer configuration vectors for activation functions, derivatives, dropout behavior, and initialization strategies automatically aligned to network depth through controlled replication or truncation. Reproducible workflows can be executed through high level helpers for fit, run, and predict across binary classification, multi-class classification, and regression modes. Training pipelines support optional self organization, adaptive learning rate behavior, and structured ensemble orchestration in which candidate models are evaluated under user specified performance metrics and selectively promoted or pruned to refine a primary ensemble, enabling controlled ensemble evolution over successive runs. Ensemble evaluation includes fused prediction strategies in which member outputs may be combined through weighted averaging, arithmetic averaging, or voting mechanisms to generate consolidated metrics for research level comparison and reproducible per-seed assessment. The framework supports multiple optimization approaches, including stochastic gradient descent, adaptive moment estimation, and look ahead methods, alongside configurable regularization controls such as L1, L2, and mixed penalties with separate weight and bias update logic. Evaluation features provide threshold tuning, relevance scoring, receiver operating characteristic and precision recall curve generation, area under curve computation, regression error diagnostics,

and report ready metric outputs. The package also includes artifact path management, debug state utilities, structured run level metadata persistence capturing seeds, configuration states, thresholds, metrics, ensemble transitions, fused evaluation artifacts, and model identifiers, as well as reproducible scripts and vignettes documenting end to end experiments.

Kingma and Ba (2015) <[doi:10.48550/arXiv.1412.6980](https://doi.org/10.48550/arXiv.1412.6980)> ``Adam: A Method for Stochastic Optimization".

Hinton et al. (2012) <https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf> ``Neural Networks for Machine Learning (RMSprop lecture notes)".

Duchi et al. (2011) <<https://jmlr.org/papers/v12/duchi11a.html>> ``Adaptive Subgradient Methods for Online Learning and Stochastic Optimization".

Zeiler (2012) <[doi:10.48550/arXiv.1212.5701](https://doi.org/10.48550/arXiv.1212.5701)> ``ADADELTA: An Adaptive Learning Rate Method".

Zhang et al. (2019) <[doi:10.48550/arXiv.1907.08610](https://doi.org/10.48550/arXiv.1907.08610)> ``Lookahead Optimizer: k steps forward, 1 step back".

You et al. (2019) <[doi:10.48550/arXiv.1904.00962](https://doi.org/10.48550/arXiv.1904.00962)> ``Large Batch Optimization for Deep Learning: Training BERT in 76 minutes (LAMB)".

McMahan et al. (2013) <<https://research.google.com/pubs/archive/41159.pdf>> ``Ad Click Prediction: a View from the Trenches (FTRL-Proximal)".

Klambauer et al. (2017) <<https://proceedings.neurips.cc/paper/6698-self-normalizing-neural-networks.pdf>> ``Self-Normalizing Neural Networks (SELU)".

Maas et al. (2013) <https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf> ``Rectifier Non-linearities Improve Neural Network Acoustic Models (Leaky ReLU / rectifiers)".

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports R6, stats, utils, dplyr, openxlsx, tidyr, pROC, PRROC, reshape2, digest, ggplot2

Suggests testthat, knitr, rmarkdown, foreach, quantmod, randomForest, reticulate, zoo, readxl, tibble

VignetteBuilder knitr

URL <https://github.com/MatHatter/DDESONN>

BugReports <https://github.com/MatHatter/DDESONN/issues>

NeedsCompilation no

Author Mathew William Armitage Fok [aut, cre]

Maintainer Mathew William Armitage Fok <quiksilver67213@yahoo.com>

Repository CRAN

Date/Publication 2026-03-11 13:50:03 UTC

Contents

ddesonn_activations	3
DDESONN_activation_defaults	6
ddesonn_activation_derivatives	7
DDESONN_artifacts_root	10
ddesonn_debug_state	11
DDESONN_dropout_defaults	11
DDESONN_fit	12
DDESONN_model	13
DDESONN_optimizer_options	15
DDESONN_plots_dir	16
DDESONN_predict	16
DDESONN_run	18
DDESONN_training_defaults	24
predict.ddesonn_model	25
print.ddesonn_run_result	26
Index	28

ddesonn_activations *Activation functions (DDESONN)*

Description

A collection of activation functions used by DDESONN. These functions operate on numeric vectors/matrices and preserve shape.

Usage

```

binary_activation(x)

custom_binary_activation(x, threshold = -1.08)

custom_activation(z)

bent_identity(x)

relu(x)

softplus(x)

leaky_relu(x, alpha = 0.01)

elu(x, alpha = 1)

tanh(x)

```

```
sigmoid(x)
hard_sigmoid(x)
swish(x)
sigmoid_binary(x)
gelu(x)
selu(x, lambda = 1.0507, alpha = 1.67326)
mish(x)
prelu(x, alpha = 0.01)
softmax(z)
maxout(x, w1 = 0.5, b1 = 1, w2 = -0.5, b2 = 0.5)
bent_relu(x)
bent_sigmoid(x)
arctangent(x)
sinusoid(x)
gaussian(x)
isrlu(x, alpha = 1)
bent_swish(x)
parametric_bent_relu(x, beta = 1)
leaky_bent(x, alpha = 0.01)
inverse_linear_unit(x)
tanh_relu_hybrid(x)
custom_bent_piecewise(x, threshold = 0.5)
sigmoid_sharp(x, temp = 5)
leaky_selu(x, alpha = 0.01, lambda = 1.0507)
```

identity(x)

Arguments

x	Numeric vector/matrix.
threshold	Threshold for piecewise.
z	Numeric vector/matrix.
alpha	Leak factor.
lambda	SELU lambda.
w1	Weight 1.
b1	Bias 1.
w2	Weight 2.
b2	Bias 2.
beta	Beta parameter.
temp	Temperature/sharpness.

Details

Many functions coerce inputs to matrix form and preserve dimensions. Some functions are experimental and may not be suitable for training.

Value

A matrix of 0/1 values with the same dimensions as x.

A matrix of 0/1 values with the same dimensions as x.

A matrix of 0/1 values with the same dimensions as z.

Bent identity transform.

ReLU transform.

Softplus transform.

Leaky ReLU transform.

ELU transform.

tanh transform.

Sigmoid transform.

Hard sigmoid transform.

Swish transform.

0/1 matrix.

GELU transform.

SELU transform.

Mish transform.

PReLU transform.

Row-wise softmax probabilities.

Elementwise max of two affine transforms.
 Bent ReLU transform.
 Bent sigmoid transform.
 atan transform.
 sin transform.
 $\exp(-x^2)$.
 ISRLU transform.
 Bent swish transform.
 Parametric bent ReLU transform.
 Leaky bent transform.
 $x/(1+\text{abs}(x))$.
 Hybrid transform.
 Piecewise transform.
 Sharpened sigmoid.
 Leaky SELU transform.
 Base identity function.

DDESONN_activation_defaults

Legacy alias for [ddesonn_activation_defaults\(\)](#)

Description

Compute sensible activation functions for hidden and output layers based on the modelling mode and stage (training or prediction).

Usage

```
DDESONN_activation_defaults(...)
```

```

ddesonn_activation_defaults(
  mode = c("binary", "multiclass", "regression"),
  hidden_sizes = NULL,
  stage = c("train", "predict")
)

```

Arguments

...	Additional arguments passed through to ddesonn_activation_defaults() .
mode	Problem mode. One of "binary", "multiclass", or "regression".
hidden_sizes	Integer vector describing the hidden layer widths.
stage	Stage for which activations are required. Either "train" or "predict".

Value

Same as `ddesonn_activation_defaults()`.

A list of activation functions suitable for passing into the underlying R6 classes.

Examples

```
ddesonn_activation_defaults("binary", hidden_sizes = c(32, 16))
ddesonn_activation_defaults("regression", hidden_sizes = 64, stage = "predict")
```

ddesonn_activation_derivatives

Activation derivatives (DDESONN)

Description

A collection of derivative functions corresponding to DDESONN activation functions.

Usage

```
binary_activation_derivative(x)
custom_binary_activation_derivative(x, threshold = -1.08)
custom_activation_derivative(z)
bent_identity_derivative(x)
relu_derivative(x)
softplus_derivative(x)
leaky_relu_derivative(x, alpha = 0.01)
elu_derivative(x, alpha = 1)
tanh_derivative(x)
sigmoid_derivative(x)
hard_sigmoid_derivative(x)
swish_derivative(x)
sigmoid_binary_derivative(x)
```

```
gelu_derivative(x)
selu_derivative(x, lambda = 1.0507, alpha = 1.67326)
mish_derivative(x)
maxout_derivative(x, w1 = 0.5, b1 = 1, w2 = -0.5, b2 = 0.5)
prelu_derivative(x, alpha = 0.01)
softmax_derivative(x)
bent_relu_derivative(x)
bent_sigmoid_derivative(x)
arctangent_derivative(x)
sinusoid_derivative(x)
gaussian_derivative(x)
isrlu_derivative(x, alpha = 1)
bent_swish_derivative(x)
parametric_bent_relu_derivative(x, beta = 1)
leaky_bent_derivative(x, alpha = 0.01)
inverse_linear_unit_derivative(x)
tanh_relu_hybrid_derivative(x)
custom_bent_piecewise_derivative(x, threshold = 0.5)
sigmoid_sharp_derivative(x, temp = 5)
leaky_selu_derivative(x, alpha = 0.01, lambda = 1.0507)
identity_derivative(x)
```

Arguments

x	Numeric vector/matrix.
threshold	Threshold.
z	Numeric vector/matrix.
alpha	Leak factor.

Derivative matrix.
Derivative matrix.
Derivative matrix.
Derivative matrix.
Derivative matrix/vector.
Matrix of ones with same dimensions as x .

DDESONN_artifacts_root

Legacy alias for [ddessonn_artifacts_root\(\)](#)

Description

Defaults to a session-scoped temp directory so examples/tests/vignettes do not write into the user home, working directory, or package tree by default.

Usage

```
DDESONN_artifacts_root(...)
```

```
ddessonn_artifacts_root(output_root = NULL)
```

Arguments

...	Additional arguments passed through to ddessonn_artifacts_root() .
output_root	Optional base directory for artifacts. When NULL, a temp-directory location is selected automatically.

Details

Override order (first non-empty wins):

1. output_root argument
2. Sys.getenv("DDESONN_ARTIFACTS_ROOT")
3. getOption("DDESONN_OUTPUT_ROOT")

Value

Same as [ddessonn_artifacts_root\(\)](#).

Absolute path to the artifacts directory (created if missing).

ddesonn_debug_state *Inspect internal DDESONN debug state*

Description

Returns a named list snapshot of objects currently stored in the private package debug/state environment.

Usage

```
ddesonn_debug_state()
```

Value

A named list of objects from DDESONN internal state.

DDESONN_dropout_defaults

Legacy alias for [ddesonn_dropout_defaults\(\)](#)

Description

Produce a simple dropout configuration matching the supplied hidden layer sizes.

Usage

```
DDESONN_dropout_defaults(...)
```

```
ddesonn_dropout_defaults(hidden_sizes)
```

Arguments

... Additional arguments passed through to [ddesonn_dropout_defaults\(\)](#).
hidden_sizes Integer vector describing the hidden layer widths.

Value

Same as [ddesonn_dropout_defaults\(\)](#).
A list of dropout rates for each hidden layer.

Examples

```
ddesonn_dropout_defaults(c(64, 32))
```

DDESONN_fit *Legacy alias for [ddesonnn_fit\(\)](#)*

Description

Train a `ddesonnn_model` (backed by DDESONN) using matrices or data frames, handling label coercion, validation data, and training control defaults.

Usage

```
DDESONN_fit(...)

ddesonnn_fit(
  model,
  x,
  y,
  validation = NULL,
  self_org = NULL,
  ...,
  verbose = FALSE,
  verboseLow = FALSE,
  debug = FALSE
)
```

Arguments

<code>...</code>	Named overrides for entries in ddesonnn_training_defaults() .
<code>model</code>	A model created by ddesonnn_model() .
<code>x</code>	Training features.
<code>y</code>	Training targets/labels.
<code>validation</code>	Optional list containing <code>x</code> and <code>y</code> elements for validation.
<code>self_org</code>	Optional logical override for the legacy self-organization phase. TRUE enables <code>self_organize()</code> during training and FALSE disables it. NULL keeps the configured default (<code>self_org = FALSE</code> in ddesonnn_training_defaults()). Self-organization acts on input-space topology error (how well neighborhood structure is organized), not on the supervised prediction-loss term.
<code>verbose</code>	Logical; emit detailed progress output when TRUE.
<code>verboseLow</code>	Logical; emit important progress output when TRUE.
<code>debug</code>	Logical; emit debug diagnostics when TRUE.

Value

Same as [ddesonnn_fit\(\)](#).

The trained model (invisibly). The underlying R6 object is modified in-place and the last training result is stored under `model$last_training`.

See Also[DDESONN-package](#)**Examples**

```

data <- mtcars
x <- data[, c("disp", "hp", "wt", "qsec", "drat")]
y <- data$am
model <- ddesonn_model(input_size = ncol(x), output_size = 1, hidden_sizes = 8)
ddesonn_fit(model, x, y, num_epochs = 1, lr = 0.05, validation_metrics = FALSE)

# Regression example (mtcars) with explicit scheduler controls.
# If you do NOT want LR decay, set lr_decay_rate = 1.0.
reg_x <- mtcars[, c("disp", "hp", "wt", "qsec", "drat")]
reg_y <- mtcars$mpg
reg_model <- ddesonn_model(
  input_size = ncol(reg_x),          # number of input features
  output_size = 1,                  # one numeric target
  hidden_sizes = c(16, 8),          # hidden-layer widths
  classification_mode = "regression" # problem type
)
ddesonn_fit(
  model = reg_model,                # model object from ddesonn_model()
  x = reg_x,                        # training predictors
  y = reg_y,                        # training target
  num_epochs = 10,                 # training epochs
  lr = 0.05,                        # initial learning rate
  lr_decay_rate = 0.5,              # decay multiplier (use 1.0 to disable)
  lr_decay_epoch = 20L,             # decay step interval in epochs
  lr_min = 1e-5,                   # lower bound for learning rate
  validation_metrics = FALSE        # disable validation metric pass in this example
)

```

DDESONN_model*Legacy alias for [ddesonn_model\(\)](#)*

Description

Initialise a `ddesonn_model` (R6) instance backed by the legacy `DDESONN` class, while handling sensible defaults for activations and node counts.

Usage

```
DDESONN_model(...)
```

```

ddesonn_model(
  input_size,
  output_size,

```

```

hidden_sizes = c(64, 32),
num_networks = 1L,
lambda = 0.00028,
classification_mode = c("binary", "multiclass", "regression"),
ML_NN = TRUE,
activation_functions = NULL,
activation_functions_predict = NULL,
init_method = "he",
custom_scale = 1,
N = NULL,
ensembles = NULL,
ensemble_number = 0L,
verbose = FALSE,
verboseLow = FALSE,
debug = FALSE
)

```

Arguments

...	Additional arguments passed through to <code>ddesonn_model()</code> .
<code>input_size</code>	Number of input features.
<code>output_size</code>	Number of outputs.
<code>hidden_sizes</code>	Integer vector describing hidden layer widths.
<code>num_networks</code>	Number of SONN members to initialise within the ensemble.
<code>lambda</code>	Regularisation strength.
<code>classification_mode</code>	Problem mode: "binary", "multiclass", or "regression".
<code>ML_NN</code>	Logical; whether to initialise a multi-layer SONN.
<code>activation_functions</code>	Optional list of activation functions for training.
<code>activation_functions_predict</code>	Optional list of activation functions used during prediction.
<code>init_method</code>	Weight initialisation scheme passed to the legacy constructor.
<code>custom_scale</code>	Optional scaling factor for the initialiser.
<code>N</code>	Optional total node count. If omitted it is inferred from the architecture.
<code>ensembles</code>	Optional pre-existing ensemble container.
<code>ensemble_number</code>	Identifier used when combining multiple ensembles.
<code>verbose</code>	Logical; emit detailed progress output when TRUE.
<code>verboseLow</code>	Logical; emit important progress output when TRUE.
<code>debug</code>	Logical; emit debug diagnostics when TRUE.

Value

Same as `ddesonn_model()`.

A `ddesonn_model` (R6) instance ready for training.

See Also

[DDESonn-package](#)

Examples

```
model <- ddesonn_model(  
  input_size = 5,  
  output_size = 1,  
  hidden_sizes = c(32, 16),  
  classification_mode = "binary"  
)
```

DDESonn_optimizer_options

Legacy alias for [ddesonn_optimizer_options\(\)](#)

Description

List the optimizer strings understood by the legacy DDESonn training loop.

Usage

```
DDESonn_optimizer_options(...)
```

```
ddesonn_optimizer_options()
```

Arguments

... Additional arguments passed through to [ddesonn_optimizer_options\(\)](#).

Value

Same as [ddesonn_optimizer_options\(\)](#).

A character vector of supported optimiser identifiers.

Examples

```
ddesonn_optimizer_options()
```

DDESONN_plots_dir *Legacy alias for ddesonn_plots_dir()*

Description

Legacy alias for `ddesonn_plots_dir()`
 Resolve the plots directory inside the artifacts root.

Usage

```
DDESONN_plots_dir(...)  
  
ddesonn_plots_dir(output_root = NULL)
```

Arguments

... Additional arguments passed through to `ddesonn_plots_dir()`.
 output_root Optional base directory for artifacts. When NULL, a temp-directory location is selected automatically.

Value

Same as `ddesonn_plots_dir()`.
 Absolute path to the plots directory.

DDESONN_predict *Legacy alias for ddesonn_predict()*

Description

Internal prediction engine / forward-pass primitive that produces ensemble or per-model predictions from a trained `ddesonn_model`. For user-facing inference, prefer `predict.ddesonn_model()`, which wraps this helper to provide a stable API for type/aggregate/threshold handling and return shapes. Multiclass note: For multiclass classification, `y` should be encoded as integer class indices 1..K (or a one-hot matrix whose columns follow the model's class order), otherwise accuracy comparisons may be incorrect.

Usage

```
DDESONN_predict(...)  
  
ddesonn_predict(  
  model,  
  new_data,  
  aggregate = c("mean", "median", "none"),
```

```

    type = c("response", "class"),
    threshold = NULL,
    verbose = FALSE,
    verboseLow = FALSE,
    debug = FALSE
  )

```

Arguments

...	Additional arguments passed through to <code>ddesonn_predict()</code> .
<code>model</code>	A trained model produced by <code>ddesonn_model()</code> .
<code>new_data</code>	New feature matrix or data frame.
<code>aggregate</code>	Aggregation strategy across ensemble members. One of "mean", "median", or "none".
<code>type</code>	Prediction type. "response" returns numeric predictions, while "class" applies thresholding for classification problems.
<code>threshold</code>	Optional threshold override when <code>type = "class"</code> .
<code>verbose</code>	Logical; emit detailed progress output when TRUE.
<code>verboseLow</code>	Logical; emit important progress output when TRUE.
<code>debug</code>	Logical; emit debug diagnostics when TRUE.

Value

Same as `ddesonn_predict()`.

A list containing the aggregated prediction matrix and the per-model outputs when `aggregate = "none"`.

See Also

[DDESONN-package](#)

Examples

```

# =====
# Example 1 – Manual API (minimal, CRAN-safe)
# =====
# This is the base mtcars binary classification example.
# The exact same setup is used again below in the full workflow script.

data <- mtcars
target <- "am"
features <- setdiff(colnames(data), target)

x <- data[, features]
y <- data[[target]]

model <- ddesonn_model(
  input_size = ncol(x),

```

```

    output_size = 1,
    hidden_sizes = c(32, 16),
    classification_mode = "binary",
    activation_functions = c("relu", "relu", "sigmoid"),
    activation_functions_predict = c("relu", "relu", "sigmoid"),
    num_networks = 1
  )

  ddesonn_fit(
    model,
    x,
    y,
    num_epochs = 3,
    lr = 0.02,
    validation_metrics = FALSE
  )

  preds <- ddesonn_predict(model, x)
  head(preds$prediction)

# =====
# Example 2 - Same example, extended (A-D scenarios)
# =====
# This is the SAME mtcars example shown above.
# The only difference is that the full script adds:
# - train/validation/test splitting
# - scaling (fit on training data)
# - ensemble configurations
# - scenario orchestration (A-D)

# View the full version of this same example:
system.file("scripts", "DDESONN_mtcars_A-D_examples.R", package = "DDESONN")

# Repository path:
# /DDESONN/inst/scripts/DDESONN_mtcars_A-D_examples.R

```

DDESONN_run

Legacy alias for [ddesonn_run\(\)](#)

Description

This helper re-creates the four orchestration modes that previously lived in TestDDESONN.R:

Usage

```
DDESONN_run(...)
```

```
ddesonn_run(
  x,
```

```

y,
classification_mode = c("binary", "multiclass", "regression"),
hidden_sizes = c(64, 32),
seeds = 1L,
do_ensemble = FALSE,
num_networks = if (isTRUE(do_ensemble)) 3L else 1L,
num_temp_iterations = 0L,
validation = NULL,
x_valid = NULL,
y_valid = NULL,
model_overrides = list(),
training_overrides = list(),
temp_overrides = NULL,
prediction_data = NULL,
test = NULL,
x_test = NULL,
y_test = NULL,
prediction_type = c("response", "class"),
aggregate = c("mean", "median", "none"),
seed_aggregate = c("mean", "median", "none"),
threshold = NULL,
output_root = NULL,
plot_controls = NULL,
save_models = TRUE,
verbose = FALSE,
verboseLow = FALSE,
debug = FALSE
)

```

Arguments

...	Additional arguments passed through to <code>ddesonn_run()</code> .
x	Training features (the training split) as a data frame, matrix, or tibble.
y	Training labels/targets (the training split).
classification_mode	Overall problem mode. One of "binary", "multiclass", or "regression".
hidden_sizes	Integer vector describing the hidden layer widths.
seeds	Integer vector of seeds. A separate model (or ensemble stack) is trained for each seed.
do_ensemble	Logical flag selecting the ensemble container modes (scenarios C/D). When FALSE, scenarios A/B are executed.
num_networks	Number of ensemble members inside each <code>ddesonn_model()</code> instance.
num_temp_iterations	Number of TEMP iterations to run when <code>do_ensemble = TRUE</code> (scenario D). Ignored otherwise.
validation	Optional validation list with elements x and y.

<code>x_valid</code>	Optional validation features. Overrides <code>validation\$x</code> when set.
<code>y_valid</code>	Optional validation labels. Overrides <code>validation\$y</code> when set.
<code>model_overrides</code>	Named list forwarded to <code>ddesonn_model()</code> allowing custom architectures.
<code>training_overrides</code>	Named list forwarded to <code>ddesonn_fit()</code> for the main run(s). Any argument accepted by <code>ddesonn_fit()</code> may be provided here. Unspecified values fall back to <code>ddesonn_training_defaults()</code> for the given <code>classification_mode</code> and <code>hidden_sizes</code> . See Details and the example showing how to inspect defaults.
<code>temp_overrides</code>	Optional named list forwarded to <code>ddesonn_fit()</code> for TEMP iterations. Defaults to <code>training_overrides</code> . Use this when TEMP candidates should train differently than the main model.
<code>prediction_data</code>	Optional features for prediction. When supplied, predictions are computed for each seed/iteration.
<code>test</code>	Optional test list with elements <code>x</code> and <code>y</code> . When supplied, the final model computes test metrics (loss and, for classification, accuracy) and stores them in <code>result\$test_metrics</code> . The run history (<code>result\$history</code>) mirrors the training metadata (train/validation losses) and appends <code>test_loss</code> when test data is provided.
<code>x_test</code>	Optional test features. Overrides <code>test\$x</code> when set.
<code>y_test</code>	Optional test labels. Overrides <code>test\$y</code> when set.
<code>prediction_type</code>	Passed to <code>ddesonn_predict()</code> .
<code>aggregate</code>	Aggregation strategy within a single model (across ensemble members).
<code>seed_aggregate</code>	Aggregation strategy across seeds. Set to "none" to keep per-seed prediction matrices.
<code>threshold</code>	Optional threshold override for classification prediction.
<code>output_root</code>	Optional directory where legacy-style artifacts are written. When NULL (default) no files are created.
<code>plot_controls</code>	Optional list passed through to <code>ddesonn_fit()</code> as <code>plot_controls</code> . Use this to enable/disable specific report plots or diagnostics (for example, evaluation report settings). The supported structure is defined by <code>ddesonn_fit()</code> ; this function does not create defaults.
<code>save_models</code>	Logical; if TRUE (default) individual models are persisted when <code>output_root</code> is supplied.
<code>verbose</code>	Logical; emit detailed progress output when TRUE.
<code>verboseLow</code>	Logical; emit important progress output when TRUE.
<code>debug</code>	Logical; emit debug diagnostics when TRUE.

Details

- Scenario A – single model (`do_ensemble = FALSE`, `num_networks = 1`).

- Scenario B – single run with multiple members inside a single model (`do_ensemble = FALSE`, `num_networks > 1`).
- Scenario C – main ensemble container (`do_ensemble = TRUE`, `num_temp_iterations = 0`).
- Scenario D – main ensemble plus TEMP iterations (`do_ensemble = TRUE`, `num_temp_iterations > 0`).

The function accepts a training set, optional validation data, and optional prediction features. It repeatedly instantiates `ddesonn_model()` objects, fits them with `ddesonn_fit()`, and (when requested) calls `ddesonn_predict()` to surface aggregated predictions.

Discovering available training overrides

`training_overrides` is a direct pass-through to `ddesonn_fit()`. To see the baseline defaults used by `ddesonn_run()`, call:

```
ddesonn_training_defaults(classification_mode, hidden_sizes)
```

To see all tunable training arguments, see `?ddesonn_fit`.

Value

Same as `ddesonn_run()`.

A list (classed as "ddesonn_run_result") containing the configuration, per-seed models, and optional prediction summaries.

Examples

```
# =====
# DDESONN - FULL example using package data in inst/extdata
# (binary classification; train/valid/test split; scale train-only)
# =====

library(DDESONN)

set.seed(111)

# -----
# 1) Locate package extdata folder (robust across check/install)
# -----
ext_dir <- system.file("extdata", package = "DDESONN")
if (!nzchar(ext_dir)) {
  stop("Could not find DDESONN extdata folder. Is the package installed?",
       call. = FALSE)
}

# -----
# 1b) Find CSVs (recursive + check-dir edge cases)
# -----
csvs <- list.files(
  ext_dir,
  pattern = "\\\\.csv$",
  full.names = TRUE,
  recursive = TRUE
)
```

```

# Defensive fallback for rare nested layouts
if (!length(csvs)) {
  ext_dir2 <- file.path(ext_dir, "inst", "extdata")
  if (dir.exists(ext_dir2)) {
    csvs <- list.files(
      ext_dir2,
      pattern = "\\\\.csv$",
      full.names = TRUE,
      recursive = TRUE
    )
  }
}

if (!length(csvs)) {
  message(sprintf(
    "No .csv files found under: %s - skipping example.",
    ext_dir
  ))
} else {

  hf_path <- file.path(ext_dir, "heart_failure_clinical_records.csv")
  data_path <- if (file.exists(hf_path)) hf_path else csvs[[1]]

  cat("[extdata] using:", data_path, "\\n")

# -----
# 2) Load data
# -----
df <- read.csv(data_path)

# Prefer DEATH_EVENT if present; otherwise infer a binary target
target_col <- if ("DEATH_EVENT" %in% names(df)) {
  "DEATH_EVENT"
} else {
  cand <- names(df)[vapply(df, function(col) {
    v <- suppressWarnings(as.numeric(col))
    if (all(is.na(v))) return(FALSE)
    u <- unique(v[is.finite(v)])
    length(u) <= 2 && all(sort(u) %in% c(0, 1))
  }, logical(1))]
  if (!length(cand)) {
    stop(
      "Could not infer a binary target column. ",
      "Provide a binary CSV in extdata or rename target to DEATH_EVENT.",
      call. = FALSE
    )
  }
  cand[[1]]
}

cat("[data] target_col =", target_col, "\\n")

```

```

# -----
# 3) Build X and y
# -----
y_all <- matrix(as.integer(df[[target_col]]), ncol = 1)

x_df <- df[, setdiff(names(df), target_col), drop = FALSE]
x_all <- as.matrix(x_df)
storage.mode(x_all) <- "double"

# -----
# 4) Split 70 / 15 / 15
# -----
n <- nrow(x_all)
idx <- sample.int(n)

n_train <- floor(0.70 * n)
n_valid <- floor(0.15 * n)

i_tr <- idx[1:n_train]
i_va <- idx[(n_train + 1):(n_train + n_valid)]
i_te <- idx[(n_train + n_valid + 1):n]

x_train <- x_all[i_tr, , drop = FALSE]
y_train <- y_all[i_tr, , drop = FALSE]

x_valid <- x_all[i_va, , drop = FALSE]
y_valid <- y_all[i_va, , drop = FALSE]

x_test <- x_all[i_te, , drop = FALSE]
y_test <- y_all[i_te, , drop = FALSE]

cat(sprintf("[split] train=%d valid=%d test=%d\n",
           nrow(x_train), nrow(x_valid), nrow(x_test)))

# -----
# 5) Scale using TRAIN stats only (no leakage)
# -----
x_train_s <- scale(x_train)
ctr <- attr(x_train_s, "scaled:center")
scl <- attr(x_train_s, "scaled:scale")
scl[!is.finite(scl) | scl == 0] <- 1

x_valid_s <- sweep(sweep(x_valid, 2, ctr, "-"), 2, scl, "/")
x_test_s <- sweep(sweep(x_test, 2, ctr, "-"), 2, scl, "/")

mx <- suppressWarnings(max(abs(x_train_s)))
if (!is.finite(mx) || mx == 0) mx <- 1

x_train <- x_train_s / mx
x_valid <- x_valid_s / mx
x_test <- x_test_s / mx

# -----

```

```

# 6) Run DDESONN
# -----
res <- ddesonn_run(
  x = x_train,
  y = y_train,
  classification_mode = "binary",

  hidden_sizes = c(64, 32),
  seeds = 1L,
  do_ensemble = FALSE,

  validation = list(
    x = x_valid,
    y = y_valid
  ),

  test = list(
    x = x_test,
    y = y_test
  ),

  training_overrides = list(
    init_method = "he",
    optimizer = "adagrad",
    lr = 0.125,
    lambda = 0.00028,

    activation_functions = list(relu, relu, sigmoid),
    dropout_rates = list(0.10),
    loss_type = "CrossEntropy",

    validation_metrics = TRUE,
    num_epochs = 360,
    final_summary_decimals = 6L
  ),

  plot_controls = list(
    evaluate_report = list(
      roc_curve = TRUE,
      pr_curve = FALSE
    )
  )
)
}

```

DDESONN_training_defaults

Legacy alias for [ddesonn_training_defaults\(\)](#)

Description

Build a list of training hyperparameters that mirror the expectations of the legacy DDESONN training loop.

Usage

```
DDESONN_training_defaults(...)

ddesonn_training_defaults(
  mode = c("binary", "multiclass", "regression"),
  hidden_sizes = NULL
)
```

Arguments

...	Additional arguments passed through to <code>ddesonn_training_defaults()</code> .
mode	Problem mode used to determine sensible defaults.
hidden_sizes	Integer vector describing the hidden layer widths.

Value

Same as `ddesonn_training_defaults()`.

A named list that can be modified and supplied to `ddesonn_fit()`.

Examples

```
ddesonn_training_defaults("binary", hidden_sizes = c(32, 16))

# Inspect regression defaults (includes LR decay by default).
cfg_reg <- ddesonn_training_defaults("regression", hidden_sizes = c(16, 8))
cfg_reg$lr
cfg_reg$lr_decay_rate
cfg_reg$lr_decay_epoch
cfg_reg$lr_min

# If you prefer a fixed LR in regression, disable decay explicitly.
cfg_reg$lr_decay_rate <- 1.0
```

predict.ddesonn_model *Predict method for DDESONN models*

Description

This is the canonical user-facing wrapper around `ddesonn_predict()`. It standardizes arguments (type, aggregate, threshold) and normalizes the output structure for inference workflows. Multiclass note: For multiclass classification, `y` should be encoded as integer class indices 1..K (or a one-hot matrix whose columns follow the model's class order), otherwise accuracy comparisons may be incorrect.

Usage

```
## S3 method for class 'ddesonn_model'
predict(
  object,
  newdata,
  ...,
  aggregate = c("mean", "median", "none"),
  type = c("response", "class"),
  threshold = NULL,
  verbose = FALSE,
  verboseLow = FALSE,
  debug = FALSE
)
```

Arguments

object	A ddesonn_model (R6) returned by ddesonn_run() / ddesonn_model().
newdata	Matrix/data.frame of predictors.
...	Unused.
aggregate	Aggregation strategy across ensemble members.
type	Prediction type. "response" returns numeric predictions, while "class" returns class labels for classification problems.
threshold	Optional threshold override when type = "class".
verbose	Logical; emit detailed progress output when TRUE.
verboseLow	Logical; emit important progress output when TRUE.
debug	Logical; emit debug diagnostics when TRUE.

```
print.ddesonn_run_result
```

Print a summary of a DDESONN run result

Description

Print a summary of a DDESONN run result

Usage

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

Arguments

x	A ddesonn_run_result object.
...	Unused.

print.ddesonn_run_result

27

Value

x, invisibly.

Index

arctangent (ddesonn_activations), 3
arctangent_derivative
 (ddesonn_activation_derivatives),
 7

bent_identity (ddesonn_activations), 3
bent_identity_derivative
 (ddesonn_activation_derivatives),
 7

bent_relu (ddesonn_activations), 3
bent_relu_derivative
 (ddesonn_activation_derivatives),
 7

bent_sigmoid (ddesonn_activations), 3
bent_sigmoid_derivative
 (ddesonn_activation_derivatives),
 7

bent_swish (ddesonn_activations), 3
bent_swish_derivative
 (ddesonn_activation_derivatives),
 7

binary_activation
 (ddesonn_activations), 3
binary_activation_derivative
 (ddesonn_activation_derivatives),
 7

custom_activation
 (ddesonn_activations), 3
custom_activation_derivative
 (ddesonn_activation_derivatives),
 7

custom_bent_piecewise
 (ddesonn_activations), 3
custom_bent_piecewise_derivative
 (ddesonn_activation_derivatives),
 7

custom_binary_activation
 (ddesonn_activations), 3
custom_binary_activation_derivative
 (ddesonn_activation_derivatives),
 7

DDESONN-package, 13, 15, 17
DDESONN_activation_defaults, 6
ddesonn_activation_defaults
 (DDESONN_activation_defaults),
 6
ddesonn_activation_defaults(), 6, 7
ddesonn_activation_derivatives, 7
ddesonn_activations, 3
DDESONN_artifacts_root, 10
ddesonn_artifacts_root
 (DDESONN_artifacts_root), 10
ddesonn_artifacts_root(), 10
ddesonn_debug_state, 11
DDESONN_dropout_defaults, 11
ddesonn_dropout_defaults
 (DDESONN_dropout_defaults), 11
ddesonn_dropout_defaults(), 11
DDESONN_fit, 12
ddesonn_fit (DDESONN_fit), 12
ddesonn_fit(), 12, 20, 21, 25
DDESONN_model, 13
ddesonn_model (DDESONN_model), 13
ddesonn_model(), 12–14, 17, 19–21
DDESONN_optimizer_options, 15
ddesonn_optimizer_options
 (DDESONN_optimizer_options), 15
ddesonn_optimizer_options(), 15
DDESONN_plots_dir, 16
ddesonn_plots_dir (DDESONN_plots_dir),
 16
DDESONN_predict, 16
ddesonn_predict (DDESONN_predict), 16
ddesonn_predict(), 16, 17, 20, 21, 25
DDESONN_run, 18
ddesonn_run (DDESONN_run), 18
ddesonn_run(), 18, 19, 21

- DDESONN_training_defaults, [24](#)
- ddesonn_training_defaults
 - (DDESONN_training_defaults), [24](#)
- ddesonn_training_defaults(), [12](#), [20](#), [24](#), [25](#)
- elu(ddesonn_activations), [3](#)
- elu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- gaussian(ddesonn_activations), [3](#)
- gaussian_derivative
 - (ddesonn_activation_derivatives), [7](#)
- gelu(ddesonn_activations), [3](#)
- gelu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- hard_sigmoid(ddesonn_activations), [3](#)
- hard_sigmoid_derivative
 - (ddesonn_activation_derivatives), [7](#)
- identity(ddesonn_activations), [3](#)
- identity_derivative
 - (ddesonn_activation_derivatives), [7](#)
- inverse_linear_unit
 - (ddesonn_activations), [3](#)
- inverse_linear_unit_derivative
 - (ddesonn_activation_derivatives), [7](#)
- isrlu(ddesonn_activations), [3](#)
- isrlu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- leaky_bent(ddesonn_activations), [3](#)
- leaky_bent_derivative
 - (ddesonn_activation_derivatives), [7](#)
- leaky_relu(ddesonn_activations), [3](#)
- leaky_relu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- leaky_selu(ddesonn_activations), [3](#)
- leaky_selu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- maxout(ddesonn_activations), [3](#)
- maxout_derivative
 - (ddesonn_activation_derivatives), [7](#)
- mish(ddesonn_activations), [3](#)
- mish_derivative
 - (ddesonn_activation_derivatives), [7](#)
- parametric_bent_relu
 - (ddesonn_activations), [3](#)
- parametric_bent_relu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- predict.ddesonn_model, [25](#)
- predict.ddesonn_model(), [16](#)
- prelu(ddesonn_activations), [3](#)
- prelu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- print.ddesonn_run_result, [26](#)
- relu(ddesonn_activations), [3](#)
- relu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- selu(ddesonn_activations), [3](#)
- selu_derivative
 - (ddesonn_activation_derivatives), [7](#)
- sigmoid(ddesonn_activations), [3](#)
- sigmoid_binary(ddesonn_activations), [3](#)
- sigmoid_binary_derivative
 - (ddesonn_activation_derivatives), [7](#)
- sigmoid_derivative
 - (ddesonn_activation_derivatives), [7](#)
- sigmoid_sharp(ddesonn_activations), [3](#)
- sigmoid_sharp_derivative
 - (ddesonn_activation_derivatives), [7](#)
- sinusoid(ddesonn_activations), [3](#)

sinusoid_derivative
 (ddesonn_activation_derivatives),
 7

softmax (ddesonn_activations), 3

softmax_derivative
 (ddesonn_activation_derivatives),
 7

softplus (ddesonn_activations), 3

softplus_derivative
 (ddesonn_activation_derivatives),
 7

swish (ddesonn_activations), 3

swish_derivative
 (ddesonn_activation_derivatives),
 7

tanh (ddesonn_activations), 3

tanh_derivative
 (ddesonn_activation_derivatives),
 7

tanh_relu_hybrid (ddesonn_activations),
 3

tanh_relu_hybrid_derivative
 (ddesonn_activation_derivatives),
 7