# Package: metANN (via r-universe)

May 16, 2026

**Title** Metaheuristic and Gradient-Based Optimization for Neural Network Training and Continuous Problems

**Version** 0.1.0

**Description** Provides tools for general-purpose continuous optimization and feed-forward artificial neural network training using metaheuristic and gradient-based optimization algorithms. The package supports benchmark function optimization, regression, binary classification, and multi-class classification with multilayer perceptrons. The package implements several optimization methods, including particle swarm optimization Kennedy and Eberhart (1995) <doi:10.1109/ICNN.1995.488968>, differential evolution Storn and Price (1997) <doi:10.1023/A:1008202821328>, grey wolf optimizer Mirjalili et al. (2014) <doi:10.1016/j.advengsoft.2013.12.007>, secretary bird optimization Fu et al. (2024) <doi:10.1007/s10462-024-10729-y>, and Adam Kingma and Ba (2015) <doi:10.48550/arXiv.1412.6980>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**URL** `https://github.com/burakdilber/metANN`

**BugReports** `https://github.com/burakdilber/metANN/issues`

**Repository** https://burakdilber.r-universe.dev

**Date/Publication** 2026-05-11 19:24:34 UTC

**RemoteUrl** https://github.com/burakdilber/metann

**RemoteRef** HEAD

**RemoteSha** 9fa689ff6ec12805ee5c24951e8b1bfed6030a4a

# Contents

---

`activation_leaky_relu` *Leaky Rectified Linear Unit Activation Function*

---

## Description

Creates a leaky rectified linear unit activation function object.

## Usage

```
activation_leaky_relu(alpha = 0.01)
```

## Arguments

| | |
|---|---|
| `alpha` | A non-negative numeric value controlling the slope for negative inputs. |

### Value

An object of class "met_activation".

### Examples

```
act <- activation_leaky_relu(alpha = 0.01)
act$fn(c(-1, 0, 1))
```

---

## activation_linear *Linear Activation Function*

---

### Description

Creates a linear activation function object.

### Usage

```
activation_linear()
```

### Value

An object of class "met_activation".

### Examples

```
act <- activation_linear()
act$fn(c(-1, 0, 1))
```

---

## activation_relu *Rectified Linear Unit Activation Function*

---

### Description

Creates a rectified linear unit activation function object.

### Usage

```
activation_relu()
```

### Value

An object of class "met_activation".

### References

Nair, V., and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. Proceedings of the 27th International Conference on Machine Learning, 807–814.

## Examples

```
act <- activation_relu()
act$fn(c(-1, 0, 1))
```

---

`activation_sigmoid` *Sigmoid Activation Function*

---

## Description

Creates a sigmoid activation function object.

## Usage

```
activation_sigmoid()
```

## Value

An object of class "`met_activation`".

## Examples

```
act <- activation_sigmoid()
act$fn(c(-1, 0, 1))
```

---

`activation_softmax` *Softmax Activation Function*

---

## Description

Creates a softmax activation function object.

## Usage

```
activation_softmax()
```

## Value

An object of class "`met_activation`".

## References

Bridle, J. S. (1990). Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In Neurocomputing: Algorithms, Architectures and Applications, 227–236. Springer.

## Examples

```
act <- activation_softmax()
act$fn(c(1, 2, 3))
```

## activation_tanh — *Hyperbolic Tangent Activation Function*

### Description

Creates a hyperbolic tangent activation function object.

### Usage

```
activation_tanh()
```

### Value

An object of class "met_activation".

### Examples

```
act <- activation_tanh()
act$fn(c(-1, 0, 1))
```

## as_activation — *Convert Character Input to an Activation Object*

### Description

Converts a character string such as "relu" into the corresponding activation function object.

### Usage

```
as_activation(activation)
```

### Arguments

| | |
|---|---|
| activation | A character string or an object of class "met_activation". |

### Value

An object of class "met_activation".

### Examples

```
as_activation("relu")
as_activation(activation_leaky_relu(alpha = 0.05))
```

## as_loss — *Convert Character Input to a Loss Object*

### Description

Converts a character string such as "mse" into the corresponding loss function object.

### Usage

```
as_loss(loss)
```

### Arguments

| | |
|---|---|
| loss | A character string or an object of class "met_loss". |

### Value

An object of class "met_loss".

### Examples

```
as_loss("mse")
as_loss(loss_huber(delta = 1.5))
```

## as_metric — *Convert Character Input to a Metric Object*

### Description

Converts a character string such as "rmse" into the corresponding metric function object.

### Usage

```
as_metric(metric)
```

### Arguments

| | |
|---|---|
| metric | A character string or an object of class "met_metric". |

### Value

An object of class "met_metric".

### Examples

```
as_metric("rmse")
as_metric(metric_accuracy())
```

## as_metrics — *Convert Multiple Inputs to Metric Objects*

### Description

Converts a character vector or a list of metric objects into a list of metric objects.

### Usage

```
as_metrics(metrics)
```

### Arguments

| | |
|---|---|
| `metrics` | A character vector, a single metric object, or a list of metric objects. |

### Value

A list of objects of class `"met_metric"`.

### Examples

```
as_metrics(c("rmse", "mae", "r2"))
as_metrics(list(metric_accuracy(), metric_f1()))
```

## as_optimizer — *Convert Character Input to an Optimizer Object*

### Description

Converts a character string such as `"pso"` into the corresponding optimizer object.

### Usage

```
as_optimizer(optimizer)
```

### Arguments

| | |
|---|---|
| `optimizer` | A character string or an object of class `"met_optimizer"`. |

### Value

An object of class `"met_optimizer"`.

### Examples

```
as_optimizer("pso")
as_optimizer(optimizer_adam())
```

## available_activations *List Available Activation Functions*

### Description

Returns the names of activation functions currently available in the metANN package.

### Usage

```
available_activations()
```

### Value

A character vector of activation function names.

### Examples

```
available_activations()
```

## available_gradient_optimizers *List Available Gradient-Based Optimizers*

### Description

Returns the names of gradient-based optimizer objects currently available in the metANN package.

### Usage

```
available_gradient_optimizers()
```

### Value

A character vector of gradient-based optimizer names.

### Examples

```
available_gradient_optimizers()
```

`available_losses` *List Available Loss Functions*

### Description

Returns the names of loss functions currently available in the metANN package.

### Usage

```
available_losses()
```

### Value

A character vector of loss function names.

### Examples

```
available_losses()
```

`available_metaheuristics` *List Available Metaheuristic Optimizers*

### Description

Returns the names of metaheuristic optimization algorithms currently available in the metANN package.

### Usage

```
available_metaheuristics()
```

### Value

A character vector of metaheuristic optimizer names.

### Examples

```
available_metaheuristics()
```

## available_metrics — *List Available Performance Metrics*

### Description

Returns the names of performance metrics currently available in the metANN package.

### Usage

```
available_metrics()
```

### Value

A character vector of metric names.

### Examples

```
available_metrics()
```

## available_optimizers — *List Available Optimizers*

### Description

Returns the names of optimization algorithms currently available in the metANN package.

### Usage

```
available_optimizers()
```

### Value

A character vector of optimizer names.

### Examples

```
available_optimizers()
```

## coef.met_optimize_result *Extract the Best Parameters from a metANN Optimization Result*

### Description

Extract the Best Parameters from a metANN Optimization Result

### Usage

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

### Arguments

| | |
|---|---|
| `object` | A metANN optimization result object. |
| `...` | Additional arguments, currently unused. |

### Value

A numeric vector containing the best solution found.

## coef.metann *Extract Weights from a metANN Model*

### Description

Extract Weights from a metANN Model

### Usage

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

### Arguments

| | |
|---|---|
| `object` | A fitted metANN model. |
| `...` | Additional arguments, currently unused. |

### Value

A numeric vector of fitted network weights.

## count_parameters — *Count the Number of Trainable Parameters in an MLP Architecture*

### Description

Computes the total number of weights and bias terms required by a multilayer perceptron architecture.

### Usage

```
count_parameters(architecture, input_dim = NULL)
```

### Arguments

| | |
|---|---|
| architecture | An object created by `mlp_architecture()`. |
| input_dim | Optional positive integer specifying the number of input features. If `NULL`, `architecture$input_dim` is used. |

### Value

A positive integer giving the total number of parameters.

### Examples

```
arch <- mlp_architecture(
  input_dim = 4,
  layers = list(
    dense_layer(5, activation = "relu"),
    dense_layer(1, activation = "linear")
  )
)
count_parameters(arch)
```

## decode_weights — *Decode an MLP Weight Vector*

### Description

Converts a numeric parameter vector into layer-wise weight matrices and bias vectors.

### Usage

```
decode_weights(weights, architecture, input_dim = NULL)
```

### Arguments

| | |
|---|---|
| `weights` | A numeric vector of MLP parameters. |
| `architecture` | An object created by `mlp_architecture()`. |
| `input_dim` | Optional positive integer specifying the number of input features. If `NULL`, `architecture$input_dim` is used. |

### Value

A list containing layer-wise weight matrices and bias vectors.

### Examples

```
arch <- mlp_architecture(
  input_dim = 2,
  layers = list(dense_layer(3), dense_layer(1, activation = "linear"))
)
w <- initialize_weights(arch, seed = 123)
decoded <- decode_weights(w, arch)
```

---

## dense_layer *Create a Dense Layer*

---

### Description

Creates a fully connected dense layer object for use in metANN architectures.

### Usage

```
dense_layer(
  units,
  activation = "relu",
  use_bias = TRUE,
  trainable = TRUE,
  name = NULL
)
```

### Arguments

| | |
|---|---|
| `units` | A positive integer specifying the number of neurons in the layer. |
| `activation` | A character string or a `"met_activation"` object. |
| `use_bias` | Logical. Whether to include a bias term in the layer. |
| `trainable` | Logical. Whether the layer parameters should be trainable. |
| `name` | An optional character string specifying the layer name. |

### Value

An object of class `"met_dense_layer"`.

## Examples

```
dense_layer(10, activation = "relu")
dense_layer(1, activation = activation_linear())
```

---

`evaluate` *Evaluate a metANN Model*

---

## Description

Evaluates a fitted metANN model on new data.

## Usage

```
evaluate(object, newdata, y_true = NULL, metrics = NULL, threshold = 0.5, ...)
```

## Arguments

| | |
|---|---|
| `object` | A fitted object of class "`metann`". |
| `newdata` | New data used for evaluation. For formula-based models, this should be a data frame containing the response variable. For x-y models, this should be a numeric matrix or numeric data frame. |
| `y_true` | Optional true response values. Required for x-y models. For formula-based models, if `NULL`, the response is extracted from `newdata`. |
| `metrics` | Optional performance metrics. If `NULL`, the metrics stored in the fitted model are used. |
| `threshold` | Classification threshold for binary classification. |
| `...` | Additional arguments passed to `predict()`. |

## Value

An object of class "`metann_evaluation`".

## Examples

```
fit <- met_mlp(
  formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length,
  data = iris,
  hidden_layers = c(5),
  optimizer = optimizer_pso(pop_size = 10, max_iter = 10),
  seed = 123,
  verbose = FALSE
)

evaluate(fit, newdata = iris)
```

## forward_pass *Forward Pass for an MLP*

### Description

Computes predictions from input data, an MLP architecture, and a parameter vector.

### Usage

```
forward_pass(x, weights, architecture)
```

### Arguments

| | |
|---|---|
| `x` | A numeric matrix or data frame of input features. |
| `weights` | A numeric vector of MLP parameters. |
| `architecture` | An object created by `mlp_architecture()`. |

### Value

A numeric matrix containing network outputs.

### Examples

```
x <- matrix(rnorm(10), nrow = 5, ncol = 2)
arch <- mlp_architecture(
  input_dim = 2,
  layers = list(
    dense_layer(3, activation = "relu"),
    dense_layer(1, activation = "linear")
  )
)
w <- initialize_weights(arch, seed = 123)
forward_pass(x, w, arch)
```

## initialize_weights *Initialize MLP Weights*

### Description

Creates a numeric vector of randomly initialized weights and bias terms for an MLP architecture.

### Usage

```
initialize_weights(
  architecture,
  input_dim = NULL,
  method = c("uniform", "normal"),
  lower = -0.5,
  upper = 0.5,
  mean = 0,
  sd = 0.1,
  seed = NULL
)
```

### Arguments

| | |
|---|---|
| architecture | An object created by `mlp_architecture()`. |
| input_dim | Optional positive integer specifying the number of input features. If `NULL`, `architecture$input_dim` is used. |
| method | Initialization method. Currently `"uniform"` and `"normal"` are supported. |
| lower | Lower bound for uniform initialization. |
| upper | Upper bound for uniform initialization. |
| mean | Mean for normal initialization. |
| sd | Standard deviation for normal initialization. |
| seed | Optional random seed. |

### Value

A numeric vector containing initialized parameters.

### Examples

```
arch <- mlp_architecture(
  input_dim = 3,
  layers = list(dense_layer(2), dense_layer(1, activation = "linear"))
)
initialize_weights(arch, seed = 123)
```

---

## is_activation — *Check Whether an Object is a metANN Activation*

---

### Description

Check Whether an Object is a metANN Activation

### Usage

```
is_activation(x)
```

## Arguments

| | |
|---|---|
| x | An object. |

## Value

TRUE if x is a metANN activation object; otherwise FALSE.

## Examples

```
is_activation(activation_relu())
```

---

is_architecture *Check Whether an Object is a metANN Architecture*

---

## Description

Check Whether an Object is a metANN Architecture

## Usage

```
is_architecture(x)
```

## Arguments

| | |
|---|---|
| x | An object. |

## Value

TRUE if x is a metANN architecture object; otherwise FALSE.

## Examples

```
arch <- mlp_architecture(list(dense_layer(1)))
is_architecture(arch)
```

---

`is_dense_layer` *Check Whether an Object is a Dense Layer*

---

### Description

Check Whether an Object is a Dense Layer

### Usage

```
is_dense_layer(x)
```

### Arguments

| | |
|---|---|
| `x` | An object. |

### Value

`TRUE` if `x` is a dense layer object; otherwise `FALSE`.

### Examples

```
is_dense_layer(dense_layer(5))
```

---

`is_layer` *Check Whether an Object is a metANN Layer*

---

### Description

Check Whether an Object is a metANN Layer

### Usage

```
is_layer(x)
```

### Arguments

| | |
|---|---|
| `x` | An object. |

### Value

`TRUE` if `x` is a metANN layer object; otherwise `FALSE`.

### Examples

```
is_layer(dense_layer(5))
```

## is_loss — *Check Whether an Object is a metANN Loss*

### Description

Check Whether an Object is a metANN Loss

### Usage

```
is_loss(x)
```

### Arguments

| | |
|---|---|
| `x` | An object. |

### Value

`TRUE` if `x` is a metANN loss object; otherwise `FALSE`.

### Examples

```
is_loss(loss_mse())
```

## is_metric — *Check Whether an Object is a metANN Metric*

### Description

Check Whether an Object is a metANN Metric

### Usage

```
is_metric(x)
```

### Arguments

| | |
|---|---|
| `x` | An object. |

### Value

`TRUE` if `x` is a metANN metric object; otherwise `FALSE`.

### Examples

```
is_metric(metric_rmse())
```

## is_mlp_architecture *Check Whether an Object is an MLP Architecture*

### Description

Check Whether an Object is an MLP Architecture

### Usage

```
is_mlp_architecture(x)
```

### Arguments

| | |
|---|---|
| x | An object. |

### Value

TRUE if x is an MLP architecture object; otherwise FALSE.

### Examples

```
arch <- mlp_architecture(list(dense_layer(1)))
is_mlp_architecture(arch)
```

## is_optimizer *Check Whether an Object is a metANN Optimizer*

### Description

Check Whether an Object is a metANN Optimizer

### Usage

```
is_optimizer(x)
```

### Arguments

| | |
|---|---|
| x | An object. |

### Value

TRUE if x is a metANN optimizer object; otherwise FALSE.

### Examples

```
is_optimizer(optimizer_pso())
```

loss_binary_crossentropy *Binary Cross-Entropy Loss*

## Description

Creates a binary cross-entropy loss function object.

## Usage

```
loss_binary_crossentropy(epsilon = 1e-15)
```

## Arguments

| | |
|---|---|
| epsilon | A small positive numeric value used for numerical stability. |

## Value

An object of class "met_loss".

## References

Bridle, J. S. (1990). Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In Neurocomputing: Algorithms, Architectures and Applications, 227–236. Springer.

## Examples

```
loss <- loss_binary_crossentropy()
loss$fn(c(0, 1, 1), c(0.1, 0.8, 0.9))
```

loss_crossentropy *Categorical Cross-Entropy Loss*

## Description

Creates a categorical cross-entropy loss function object.

## Usage

```
loss_crossentropy(epsilon = 1e-15)
```

## Arguments

| | |
|---|---|
| epsilon | A small positive numeric value used for numerical stability. |

## Value

An object of class "met_loss".

## References

Bridle, J. S. (1990). Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In Neurocomputing: Algorithms, Architectures and Applications, 227–236. Springer.

## Examples

```
loss <- loss_crossentropy()
y_true <- matrix(c(1, 0, 0, 0, 1, 0), nrow = 2, byrow = TRUE)
y_pred <- matrix(c(0.8, 0.1, 0.1, 0.2, 0.7, 0.1), nrow = 2, byrow = TRUE)
loss$fn(y_true, y_pred)
```

---

# loss_huber *Huber Loss*

---

## Description

Creates a Huber loss function object.

## Usage

```
loss_huber(delta = 1)
```

## Arguments

| | |
|---|---|
| delta | A positive numeric value controlling the transition point between squared and absolute loss behavior. |

## Value

An object of class "met_loss".

## References

Huber, P. J. (1964). Robust Estimation of a Location Parameter. The Annals of Mathematical Statistics, 35(1), 73–101. doi:10.1214/aoms/1177703732

## Examples

```
loss <- loss_huber(delta = 1)
loss$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## loss_log_cosh *Log-Cosh Loss*

### Description

Creates a log-cosh loss function object.

### Usage

```
loss_log_cosh()
```

### Value

An object of class "met_loss".

### Examples

```
loss <- loss_log_cosh()
loss$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## loss_mae *Mean Absolute Error Loss*

### Description

Creates a mean absolute error loss function object.

### Usage

```
loss_mae()
```

### Value

An object of class "met_loss".

### Examples

```
loss <- loss_mae()
loss$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## loss_mse *Mean Squared Error Loss*

### Description

Creates a mean squared error loss function object.

### Usage

```
loss_mse()
```

### Value

An object of class "met_loss".

### Examples

```
loss <- loss_mse()
loss$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## met_mlp *Train a Feed-Forward Multilayer Perceptron*

### Description

Convenience wrapper around metann() for training feed-forward multilayer perceptrons.

### Usage

```
met_mlp(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  architecture = NULL,
  hidden_layers = NULL,
  activation = "relu",
  output_activation = NULL,
  task = c("auto", "regression", "classification"),
  optimizer = optimizer_pso(),
  loss = NULL,
  metrics = NULL,
  seed = NULL,
  verbose = TRUE
)
```

## Arguments

- `formula` Optional model formula.
- `data` Optional data frame used with `formula`.
- `x` Optional numeric input matrix or data frame.
- `y` Optional response vector.
- `architecture` Optional MLP architecture object.
- `hidden_layers` Integer vector giving the number of units in each hidden layer.
- `activation` Activation function for hidden layers.
- `output_activation` Optional output activation function. If `NULL`, it is selected automatically based on the task.
- `task` One of `"auto"`, `"regression"`, or `"classification"`.
- `optimizer` Optimizer object.
- `loss` Optional loss function. If `NULL`, it is selected automatically based on the task.
- `metrics` Optional performance metrics. If `NULL`, default metrics are selected automatically based on the task.
- `seed` Optional random seed.
- `verbose` Logical. If `TRUE`, progress information is printed.

## Value

An object of class `"metann"`.

## References

Montana, D. J., and Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. Proceedings of the 11th International Joint Conference on Artificial Intelligence, 762–767.

Ilonen, J., Kamarainen, J.-K., and Lampinen, J. (2003). Differential Evolution Training Algorithm for Feed-Forward Neural Networks. Neural Processing Letters, 17, 93–105. doi:10.1023/A:1022995128597

Karaboga, D., and Ozturk, C. (2009). Neural Networks Training by Artificial Bee Colony Algorithm on Pattern Classification. Neural Network World, 19(3), 279–292.

Mirjalili, S. (2015). How Effective is the Grey Wolf Optimizer in Training Multi-Layer Perceptrons. Applied Intelligence, 43, 150–161. doi:10.1007/s10489-014-0645-7

Dilber, B., and Ozdemir, A. F. (2026). A novel approach to training feed-forward multi-layer perceptrons with recently proposed secretary bird optimization algorithm. Neural Computing and Applications, 38(5). doi:10.1007/s00521-026-11874-x

## Examples

```
fit <- met_mlp(
  formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length,
  data = iris,
  hidden_layers = c(5),
  optimizer = optimizer_pso(pop_size = 10, max_iter = 10),
```

```
  seed = 123,
  verbose = FALSE
)

fit
```

---

met_optimize *General-Purpose Optimization*

---

## Description

Performs continuous optimization using metaheuristic or gradient-based optimization algorithms.

## Usage

```
met_optimize(
  fn,
  optimizer = optimizer_pso(),
  lower,
  upper,
  gr = NULL,
  initial = NULL,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| fn | Objective function to be minimized. It must accept a numeric vector as its first argument and return a single numeric value. |
| optimizer | Optimizer object created by functions such as `optimizer_pso()`, `optimizer_sboa()`, `optimizer_sgd()`, or `optimizer_adam()`. |
| lower | Numeric vector of lower bounds. |
| upper | Numeric vector of upper bounds. |
| gr | Optional gradient function. Required for gradient-based optimizers such as `optimizer_sgd()` and `optimizer_adam()`. It must accept a numeric vector as its first argument and return a numeric vector of the same length. |
| initial | Optional numeric vector of initial parameter values. If `NULL`, a random initial point is generated within the given bounds for gradient-based optimizers. |
| seed | Optional random seed. |
| verbose | Logical. If `TRUE`, progress information is printed. |
| ... | Additional arguments passed to `fn` and, when applicable, `gr`. |

## Value

An object of class "met_optimize_result".

## Examples

```
sphere <- function(x) sum(x^2)

result <- met_optimize(
  fn = sphere,
  optimizer = optimizer_pso(pop_size = 10, max_iter = 20),
  lower = rep(-5, 2),
  upper = rep(5, 2),
  seed = 123,
  verbose = FALSE
)

result

grad_sphere <- function(x) 2 * x

result_adam <- met_optimize(
  fn = sphere,
  gr = grad_sphere,
  optimizer = optimizer_adam(learning_rate = 0.1, epochs = 20),
  lower = rep(-5, 2),
  upper = rep(5, 2),
  initial = rep(4, 2),
  seed = 123,
  verbose = FALSE
)

result_adam
```

---

metann *Train an Artificial Neural Network with metANN*

---

## Description

Trains a feed-forward multilayer perceptron using metaheuristic or gradient-based optimization algorithms. The function supports regression and classification tasks through either an x-y interface or a formula-data interface.

## Usage

```
metann(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
```

```
    architecture = NULL,
    hidden_layers = NULL,
    activation = "relu",
    output_activation = NULL,
    task = c("auto", "regression", "classification"),
    optimizer = optimizer_pso(),
    loss = NULL,
    metrics = NULL,
    seed = NULL,
    verbose = TRUE
  )
```

## Arguments

| | |
|---|---|
| formula | Optional formula specifying the model. |
| data | Optional data frame containing the variables in formula. |
| x | Optional numeric matrix or data frame of input features. |
| y | Optional response vector or one-column matrix. |
| architecture | Optional MLP architecture created by mlp_architecture(). |
| hidden_layers | Optional integer vector specifying hidden layer sizes. Used when architecture is not supplied. |
| activation | Activation function used for hidden layers when hidden_layers is supplied. It can be a single value or a vector with the same length as hidden_layers. |
| output_activation | Optional activation function used for the output layer when hidden_layers is supplied. If NULL, it is selected automatically based on the task. |
| task | One of "auto", "regression", or "classification". If "auto", the task is detected from the response variable. |
| optimizer | A character string or a metANN optimizer object. |
| loss | Optional character string or metANN loss object. If NULL, it is selected automatically based on the task. |
| metrics | Optional character vector, metric object, or list of metric objects. If NULL, default metrics are selected automatically based on the task. |
| seed | Optional random seed. |
| verbose | Logical. If TRUE, optimization or training progress is printed. |

## Value

An object of class "metann".

## References

Montana, D. J., and Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. Proceedings of the 11th International Joint Conference on Artificial Intelligence, 762–767.

Ilonen, J., Kamarainen, J.-K., and Lampinen, J. (2003). Differential Evolution Training Algorithm for Feed-Forward Neural Networks. Neural Processing Letters, 17, 93–105. doi:10.1023/A:1022995128597

Karaboga, D., and Ozturk, C. (2009). Neural Networks Training by Artificial Bee Colony Algorithm on Pattern Classification. Neural Network World, 19(3), 279–292.

Mirjalili, S. (2015). How Effective is the Grey Wolf Optimizer in Training Multi-Layer Perceptrons. Applied Intelligence, 43, 150–161. doi:10.1007/s10489-014-0645-7

Dilber, B., and Ozdemir, A. F. (2026). A novel approach to training feed-forward multi-layer perceptrons with recently proposed secretary bird optimization algorithm. Neural Computing and Applications, 38(5). doi:10.1007/s00521-026-11874-x

## Examples

```
fit <- metann(
  formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length,
  data = iris,
  hidden_layers = c(5),
  optimizer = optimizer_pso(pop_size = 10, max_iter = 20),
  loss = "mse",
  metrics = c("rmse", "mae", "r2"),
  seed = 123,
  verbose = FALSE
)
fit

iris_bin <- iris
iris_bin$IsSetosa <- factor(
  ifelse(iris_bin$Species == "setosa", "setosa", "other")
)

fit_class <- metann(
  formula = IsSetosa ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = iris_bin,
  hidden_layers = c(5),
  optimizer = optimizer_pso(pop_size = 10, max_iter = 20),
  seed = 123,
  verbose = FALSE
)
fit_class
```

---

# metric_accuracy *Accuracy Metric*

---

## Description

Creates an accuracy metric object for classification tasks.

## Usage

```
metric_accuracy()
```

### Value

An object of class "met_metric".

### Examples

```
metric <- metric_accuracy()
metric$fn(c(0, 1, 1), c(0, 1, 0))
```

---

## metric_f1 *F1 Score Metric*

---

### Description

Creates an F1 score metric object for classification tasks.

### Usage

```
metric_f1(positive_class = 1)
```

### Arguments

positive_class The class label treated as the positive class. Defaults to 1.

### Value

An object of class "met_metric".

### Examples

```
metric <- metric_f1()
metric$fn(c(0, 1, 1, 0), c(0, 1, 0, 0))
```

---

## metric_mae *Mean Absolute Error Metric*

---

### Description

Creates a mean absolute error metric object.

### Usage

```
metric_mae()
```

### Value

An object of class "met_metric".

### Examples

```
metric <- metric_mae()
metric$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## metric_mse *Mean Squared Error Metric*

### Description

Creates a mean squared error metric object.

### Usage

```
metric_mse()
```

### Value

An object of class "met_metric".

### Examples

```
metric <- metric_mse()
metric$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## metric_precision *Precision Metric*

### Description

Creates a precision metric object for classification tasks.

### Usage

```
metric_precision(positive_class = 1)
```

### Arguments

positive_class The class label treated as the positive class. Defaults to 1.

### Value

An object of class "met_metric".

### Examples

```
metric <- metric_precision()
metric$fn(c(0, 1, 1, 0), c(0, 1, 0, 0))
```

## metric_r2 *Coefficient of Determination Metric*

### Description

Creates an R-squared metric object.

### Usage

```
metric_r2()
```

### Value

An object of class "met_metric".

### Examples

```
metric <- metric_r2()
metric$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## metric_recall *Recall Metric*

### Description

Creates a recall metric object for classification tasks.

### Usage

```
metric_recall(positive_class = 1)
```

### Arguments

`positive_class` The class label treated as the positive class. Defaults to 1.

### Value

An object of class "met_metric".

### Examples

```
metric <- metric_recall()
metric$fn(c(0, 1, 1, 0), c(0, 1, 0, 0))
```

## `metric_rmse` *Root Mean Squared Error Metric*

### Description

Creates a root mean squared error metric object.

### Usage

```
metric_rmse()
```

### Value

An object of class "`met_metric`".

### Examples

```
metric <- metric_rmse()
metric$fn(c(1, 2, 3), c(1.1, 1.9, 3.2))
```

## `mlp_architecture` *Create an MLP Architecture*

### Description

Creates a multilayer perceptron architecture object from a list of dense layers.

### Usage

```
mlp_architecture(layers, input_dim = NULL, name = "mlp")
```

### Arguments

| | |
|---|---|
| `layers` | A list of dense layer objects created by `dense_layer()`. |
| `input_dim` | Optional positive integer specifying the number of input features. This can be left as `NULL` and inferred later from data. |
| `name` | Optional character string specifying the architecture name. |

### Value

An object of class "`met_mlp_architecture`".

## Examples

```
architecture <- mlp_architecture(
  layers = list(
    dense_layer(10, activation = "relu"),
    dense_layer(1, activation = "linear")
  )
)
architecture
```

---

`optimizer_abc` *Artificial Bee Colony Optimizer*

---

## Description

Creates an Artificial Bee Colony optimizer object for continuous optimization problems.

## Usage

```
optimizer_abc(colony_size = 30, max_iter = 100, limit = NULL)
```

## Arguments

| | |
|---|---|
| `colony_size` | Total colony size. Half of the colony is used as employed bees and half as onlooker bees. |
| `max_iter` | Maximum number of iterations. |
| `limit` | Number of unsuccessful trials before a food source is abandoned. |

## Value

An object of class `"met_optimizer"`.

## References

Karaboga, D., and Basturk, B. (2007). A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm. Journal of Global Optimization, 39, 459–471. doi:10.1007/s10898-007-9149-x

Karaboga, D., and Ozturk, C. (2009). Neural Networks Training by Artificial Bee Colony Algorithm on Pattern Classification. Neural Network World, 19(3), 279–292.

## Examples

```
optimizer_abc()
```

optimizer_adam *Adam Optimizer*

## Description

Creates an Adam optimizer object.

## Usage

```
optimizer_adam(
  learning_rate = 0.001,
  beta1 = 0.9,
  beta2 = 0.999,
  epsilon = 1e-08,
  epochs = 100,
  batch_size = NULL
)
```

## Arguments

| | |
|---|---|
| `learning_rate` | Learning rate. |
| `beta1` | Exponential decay rate for the first moment estimates. |
| `beta2` | Exponential decay rate for the second moment estimates. |
| `epsilon` | Small positive constant for numerical stability. |
| `epochs` | Number of training epochs. |
| `batch_size` | Mini-batch size. If `NULL`, full-batch training is used. |

## Value

An object of class `"met_optimizer"`.

## References

Kingma, D. P., and Ba, J. (2015). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

## Examples

```
optimizer_adam()
```

optimizer_de *Differential Evolution Optimizer*

## Description

Creates a Differential Evolution optimizer object.

## Usage

```
optimizer_de(
  pop_size = 30,
  max_iter = 100,
  F = 0.5,
  CR = 0.9,
  strategy = "rand/1/bin"
)
```

## Arguments

| | |
|---|---|
| `pop_size` | Population size. |
| `max_iter` | Maximum number of iterations. |
| `F` | Differential weight. Common values are between 0.4 and 1. |
| `CR` | Crossover probability. Must be between 0 and 1. |
| `strategy` | Differential evolution strategy. Currently only `"rand/1/bin"` is supported. |

## Value

An object of class `"met_optimizer"`.

## References

Storn, R., and Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization, 11, 341–359. doi:10.1023/A:1008202821328

Ilonen, J., Kamarainen, J.-K., and Lampinen, J. (2003). Differential Evolution Training Algorithm for Feed-Forward Neural Networks. Neural Processing Letters, 17, 93–105. doi:10.1023/A:1022995128597

## Examples

```
optimizer_de()
```

optimizer_ga *Genetic Algorithm Optimizer*

## Description

Creates a real-coded Genetic Algorithm optimizer object.

## Usage

```
optimizer_ga(
  pop_size = 30,
  max_iter = 100,
  crossover_rate = 0.8,
  mutation_rate = 0.1,
  mutation_sd = 0.1,
  elitism = TRUE,
  selection = "tournament",
  tournament_size = 2
)
```

## Arguments

| | |
|---|---|
| pop_size | Population size. |
| max_iter | Maximum number of iterations. |
| crossover_rate | Probability of crossover. |
| mutation_rate | Probability of mutating each parameter. |
| mutation_sd | Standard deviation of Gaussian mutation noise. |
| elitism | Logical. Whether to preserve the best solution in each generation. |
| selection | Selection method. Currently only "tournament" is supported. |
| tournament_size | Number of individuals used in tournament selection. |

## Value

An object of class "met_optimizer".

## References

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.

Montana, D. J., and Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. Proceedings of the 11th International Joint Conference on Artificial Intelligence, 762–767.

## Examples

```
optimizer_ga()
```

`optimizer_gwo` *Grey Wolf Optimizer*

### Description

Creates a Grey Wolf Optimizer object for continuous optimization problems.

### Usage

```
optimizer_gwo(pop_size = 30, max_iter = 100, a_start = 2, a_end = 0)
```

### Arguments

| | |
|---|---|
| `pop_size` | Population size. |
| `max_iter` | Maximum number of iterations. |
| `a_start` | Initial value of the control parameter `a`. |
| `a_end` | Final value of the control parameter `a`. |

### Value

An object of class `"met_optimizer"`.

### References

Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey Wolf Optimizer. Advances in Engineering Software, 69, 46–61. doi:10.1016/j.advengsoft.2013.12.007

Mirjalili, S. (2015). How Effective is the Grey Wolf Optimizer in Training Multi-Layer Perceptrons. Applied Intelligence, 43, 150–161. doi:10.1007/s10489-014-0645-7

### Examples

```
optimizer_gwo()
```

`optimizer_hybrid` *Hybrid Optimizer*

### Description

Creates a hybrid optimizer object by combining a global optimizer and a local optimizer.

### Usage

```
optimizer_hybrid(
  global = optimizer_pso(),
  local = optimizer_adam(),
  strategy = "sequential"
)
```

### Arguments

| | |
|---|---|
| `global` | A metaheuristic optimizer object. |
| `local` | A gradient-based optimizer object. |
| `strategy` | Hybrid training strategy. Currently `"sequential"` is used as the default strategy. |

### Value

An object of class `"met_optimizer"`.

### Examples

```
optimizer_hybrid(
  global = optimizer_pso(max_iter = 10),
  local = optimizer_adam(epochs = 10)
)
```

---

## `optimizer_info` *Get Optimizer Information*

---

### Description

Returns basic information about an optimizer available in the metANN package.

### Usage

```
optimizer_info(optimizer)
```

### Arguments

| | |
|---|---|
| `optimizer` | Character name of an optimizer or an optimizer object created by functions such as `optimizer_pso()`, `optimizer_sboa()`, `optimizer_sgd()`, or `optimizer_adam()`. |

### Value

An object of class `"met_optimizer_info"`.

### Examples

```
optimizer_info("pso")
optimizer_info("sboa")
optimizer_info(optimizer_adam())
```

`optimizer_pso` *Particle Swarm Optimization Optimizer*

## Description

Creates a Particle Swarm Optimization optimizer object.

## Usage

```
optimizer_pso(
  pop_size = 30,
  max_iter = 100,
  w = 0.7,
  c1 = 1.5,
  c2 = 1.5,
  velocity_clamp = NULL
)
```

## Arguments

| | |
|---|---|
| `pop_size` | Population size. |
| `max_iter` | Maximum number of iterations. |
| `w` | Inertia weight. |
| `c1` | Cognitive acceleration coefficient. |
| `c2` | Social acceleration coefficient. |
| `velocity_clamp` | Optional maximum absolute velocity. If `NULL`, velocity is not clamped. |

## Value

An object of class `"met_optimizer"`.

## References

Kennedy, J., and Eberhart, R. (1995). Particle Swarm Optimization. Proceedings of ICNN'95 - International Conference on Neural Networks, 4, 1942–1948. doi:10.1109/ICNN.1995.488968

## Examples

```
optimizer_pso()
```

---

`optimizer_sboa` *Secretary Bird Optimization Algorithm Optimizer*

---

## Description

Creates a Secretary Bird Optimization Algorithm optimizer object for continuous optimization problems.

## Usage

```
optimizer_sboa(pop_size = 30, max_iter = 100)
```

## Arguments

| | |
|---|---|
| `pop_size` | Population size. |
| `max_iter` | Maximum number of iterations. |

## Value

An object of class `"met_optimizer"`.

## References

Fu, Y., Liu, D., Chen, J., and He, L. (2024). Secretary Bird Optimization Algorithm: A New Metaheuristic for Solving Global Optimization Problems. Artificial Intelligence Review, 57, 123. doi:10.1007/s10462-024-10729-y

Dilber, B., and Ozdemir, A. F. (2026). A novel approach to training feed-forward multi-layer perceptrons with recently proposed secretary bird optimization algorithm. Neural Computing and Applications, 38(5). doi:10.1007/s00521-026-11874-x

## Examples

```
optimizer_sboa()
```

---

`optimizer_sgd` *Stochastic Gradient Descent Optimizer*

---

## Description

Creates a stochastic gradient descent optimizer object.

## Usage

```
optimizer_sgd(learning_rate = 0.01, epochs = 100, batch_size = NULL)
```

### Arguments

| | |
|---|---|
| `learning_rate` | Learning rate. |
| `epochs` | Number of training epochs. |
| `batch_size` | Mini-batch size. If `NULL`, full-batch training is used. |

### Value

An object of class "met_optimizer".

### References

Robbins, H., and Monro, S. (1951). A Stochastic Approximation Method. The Annals of Mathematical Statistics, 22(3), 400–407. doi:10.1214/aoms/1177729586

### Examples

```
optimizer_sgd()
```

---

## `optimizer_tlbo` *Teaching-Learning-Based Optimization Optimizer*

---

### Description

Creates a Teaching-Learning-Based Optimization optimizer object for continuous optimization problems.

### Usage

```
optimizer_tlbo(pop_size = 30, max_iter = 100)
```

### Arguments

| | |
|---|---|
| `pop_size` | Population size. |
| `max_iter` | Maximum number of iterations. |

### Value

An object of class "met_optimizer".

### References

Rao, R. V., Savsani, V. J., and Vakharia, D. P. (2011). Teaching-Learning-Based Optimization: A Novel Method for Constrained Mechanical Design Optimization Problems. Computer-Aided Design, 43, 303–315. doi:10.1016/j.cad.2010.12.015

### Examples

```
optimizer_tlbo()
```

## optimizer_woa *Whale Optimization Algorithm Optimizer*

### Description

Creates a Whale Optimization Algorithm optimizer object for continuous optimization problems.

### Usage

```
optimizer_woa(pop_size = 30, max_iter = 100, a_start = 2, a_end = 0, b = 1)
```

### Arguments

| | |
|---|---|
| pop_size | Population size. |
| max_iter | Maximum number of iterations. |
| a_start | Initial value of the control parameter a. |
| a_end | Final value of the control parameter a. |
| b | Constant defining the spiral shape in the bubble-net mechanism. |

### Value

An object of class "met_optimizer".

### References

Mirjalili, S., and Lewis, A. (2016). The Whale Optimization Algorithm. Advances in Engineering Software, 95, 51–67. doi:10.1016/j.advengsoft.2016.01.008

### Examples

```
optimizer_woa()
```

## plot.met_optimize_result *Plot Optimization Convergence*

### Description

Plots the convergence curve of a metANN optimization result.

### Usage

```
## S3 method for class 'met_optimize_result'
plot(x, log = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `x` | An object of class "`met_optimize_result`". |
| `log` | Logical. If `TRUE`, the y-axis is shown on a logarithmic scale. Only positive convergence values can be displayed on a log scale. |
| `...` | Additional graphical arguments passed to `plot()`. |

### Value

The input object invisibly.

---

## `plot.metann` *Plot a metANN Model*

---

### Description

Plot a metANN Model

### Usage

```
## S3 method for class 'metann'
plot(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | A fitted metANN model. |
| `...` | Additional arguments passed to `plot()`. |

### Value

The input object invisibly.

---

## `plot_network` *Plot Neural Network Architecture*

---

### Description

Plots the architecture of a feed-forward multilayer perceptron, showing input, hidden, and output layers in a visually enhanced layout.

## Usage

```
plot_network(
  object,
  max_neurons = 20,
  show_connections = TRUE,
  neuron_cex = 2.2,
  label_cex = 0.9,
  main = "Neural Network Architecture",
  ...
)
```

## Arguments

| | |
|---|---|
| object | A fitted "metann" object or an MLP architecture object. |
| max_neurons | Maximum number of neurons to display per layer. If a layer has more neurons than this value, only a subset is displayed and the layer is annotated. |
| show_connections | Logical. If TRUE, connections between adjacent layers are drawn. |
| neuron_cex | Size of neuron circles. |
| label_cex | Size of text labels. |
| main | Main title of the plot. |
| ... | Additional graphical arguments. |

## Value

The input object invisibly.

## Examples

```
fit <- met_mlp(
  formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length,
  data = iris,
  hidden_layers = c(5),
  optimizer = optimizer_pso(pop_size = 10, max_iter = 10),
  seed = 123,
  verbose = FALSE
)

plot_network(fit)
```

## predict.metann *Predict with a metANN Model*

### Description

Generates predictions from a fitted metANN model.

### Usage

```
## S3 method for class 'metann'
predict(
  object,
  newdata,
  type = c("response", "prob", "class"),
  threshold = 0.5,
  ...
)
```

### Arguments

| | |
|---|---|
| `object` | A fitted object of class "metann". |
| `newdata` | New data used for prediction. For formula-based models, this should be a data frame. For x-y models, this should be a numeric matrix or numeric data frame. |
| `type` | Prediction type. For regression models, "response" returns numeric predictions. For classification models, "class" returns predicted class labels, "prob" returns predicted probabilities, and "response" returns the default response, which is class labels. |
| `threshold` | Classification threshold for binary classification. |
| `...` | Additional arguments. |

### Value

A numeric vector, matrix, or factor depending on the task and prediction type.

## print.met_dense_layer *Print a Dense Layer*

### Description

Print a Dense Layer

### Usage

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

### Arguments

| | |
|---|---|
| x | A dense layer object. |
| ... | Additional arguments, currently unused. |

### Value

The input object invisibly.

---

## print.met_mlp_architecture

*Print an MLP Architecture*

---

### Description

Print an MLP Architecture

### Usage

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

### Arguments

| | |
|---|---|
| x | An MLP architecture object. |
| ... | Additional arguments, currently unused. |

### Value

The input object invisibly.

---

## print.met_optimize_result

*Print a metANN Optimization Result*

---

### Description

Print a metANN Optimization Result

### Usage

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

### Arguments

| | |
|---|---|
| `x` | A metANN optimization result object. |
| `...` | Additional arguments, currently unused. |

### Value

The input object invisibly.

---

## `print.met_optimizer` *Print a metANN Optimizer*

---

### Description

Print a metANN Optimizer

### Usage

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

### Arguments

| | |
|---|---|
| `x` | A metANN optimizer object. |
| `...` | Additional arguments, currently unused. |

### Value

The input object invisibly.

---

## `print.met_optimizer_info` *Print Optimizer Information*

---

### Description

Print Optimizer Information

### Usage

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

### Arguments

| | |
|---|---|
| `x` | An object of class "`met_optimizer_info`". |
| `...` | Additional arguments. |

### Value

The input object invisibly.

---

## print.metann — *Print a metANN Model*

### Description

Print a metANN Model

### Usage

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

### Arguments

| | |
|---|---|
| x | A metANN model object. |
| ... | Additional arguments, currently unused. |

### Value

The input object invisibly.

---

## print.metann_evaluation — *Print metANN Evaluation Results*

### Description

Print metANN Evaluation Results

### Usage

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

### Arguments

| | |
|---|---|
| x | An object of class "metann_evaluation". |
| ... | Additional arguments. |

### Value

The input object invisibly.

---

`summary.met_optimize_result` *Summarize a metANN Optimization Result*

---

### Description

Summarize a metANN Optimization Result

### Usage

```
## S3 method for class 'met_optimize_result'
summary(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | A metANN optimization result object. |
| `...` | Additional arguments, currently unused. |

### Value

A list containing the main optimization results.

---

`summary.metann` *Summarize a metANN Model*

---

### Description

Summarize a metANN Model

### Usage

```
## S3 method for class 'metann'
summary(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | A metANN model object. |
| `...` | Additional arguments, currently unused. |

### Value

A list containing model summary information.

# Index