

# Package: ANN2 (via r-universe)

August 28, 2024

**Type** Package

**Title** Artificial Neural Networks for Anomaly Detection

**Version** 2.3.4

**Date** 2020-11-29

**Author** Bart Lammers

**Maintainer** Bart Lammers <bart.f.lammers@gmail.com>

**Description** Training of neural networks for classification and regression tasks using mini-batch gradient descent. Special features include a function for training autoencoders, which can be used to detect anomalies, and some related plotting functions. Multiple activation functions are supported, including tanh, relu, step and ramp. For the use of the step and ramp activation functions in detecting anomalies using autoencoders, see Hawkins et al. (2002) <doi:10.1007/3-540-46145-0\_17>. Furthermore, several loss functions are supported, including robust ones such as Huber and pseudo-Huber loss, as well as L1 and L2 regularization. The possible options for optimization algorithms are RMSprop, Adam and SGD with momentum. The package contains a vectorized C++ implementation that facilitates fast training through mini-batch learning.

**License** GPL (>= 3) | file LICENSE

**URL** <https://github.com/bflammers/ANN2>

**Encoding** UTF-8

**LazyData** true

**SystemRequirements** C++11

**Imports** Rcpp (>= 0.12.18), reshape2 (>= 1.4.3), ggplot2 (>= 3.0.0), viridisLite (>= 0.3.0), methods

**LinkingTo** Rcpp, RcppArmadillo, testthat

**Suggests** testthat

**RoxygenNote** 7.1.1

**Repository** <https://bflammers.r-universe.dev>

**RemoteUrl** <https://github.com/bflammers/ann2>

**RemoteRef** HEAD

**RemoteSha** be9809f7deb4ca589cc4d7f068af650462b1b7d0

## Contents

ANN . . . . .	2
autoencoder . . . . .	2
compression_plot . . . . .	5
decode . . . . .	6
encode . . . . .	6
neuralnetwork . . . . .	7
plot.ANN . . . . .	10
predict.ANN . . . . .	11
print.ANN . . . . .	11
read_ANN . . . . .	12
reconstruct . . . . .	12
reconstruction_plot . . . . .	13
train . . . . .	13
write_ANN . . . . .	15
<b>Index</b>	<b>16</b>

---

ANN	<i>Rcpp module exposing C++ class ANN</i>
-----	---

---

### Description

C++ class ANN is the work horse of this package

---

autoencoder	<i>Train an Autoencoding Neural Network</i>
-------------	---

---

### Description

Construct and train an Autoencoder by setting the target variables equal to the input variables. The number of nodes in the middle layer should be smaller than the number of input variables in X in order to create a bottleneck layer.

**Usage**

```

autoencoder(
  X,
  hidden.layers,
  standardize = TRUE,
  loss.type = "squared",
  huber.delta = 1,
  activ.functions = "tanh",
  step.H = 5,
  step.k = 100,
  optim.type = "sgd",
  learn.rates = 1e-04,
  L1 = 0,
  L2 = 0,
  sgd.momentum = 0.9,
  rmsprop.decay = 0.9,
  adam.beta1 = 0.9,
  adam.beta2 = 0.999,
  n.epochs = 100,
  batch.size = 32,
  drop.last = TRUE,
  val.prop = 0.1,
  verbose = TRUE,
  random.seed = NULL
)

```

**Arguments**

X	matrix with explanatory variables
hidden.layers	vector specifying the number of nodes in each layer. The number of hidden layers in the network is implicitly defined by the length of this vector. Set hidden.layers to NA for a network with no hidden layers
standardize	logical indicating if X and Y should be standardized before training the network. Recommended to leave at TRUE for faster convergence.
loss.type	which loss function should be used. Options are "squared", "absolute", "huber" and "pseudo-huber"
huber.delta	used only in case of loss functions "huber" and "pseudo-huber". This parameter controls the cut-off point between quadratic and absolute loss.
activ.functions	character vector of activation functions to be used in each hidden layer. Possible options are 'tanh', 'sigmoid', 'relu', 'linear', 'ramp' and 'step'. Should be either the size of the number of hidden layers or equal to one. If a single activation type is specified, this type will be broadcasted across the hidden layers.
step.H	number of steps of the step activation function. Only applicable if activ.functions includes 'step'

<code>step.k</code>	parameter controlling the smoothness of the step activation function. Larger values lead to a less smooth step function. Only applicable if <code>activ.functions</code> includes 'step'.
<code>optim.type</code>	type of optimizer to use for updating the parameters. Options are 'sgd', 'rmsprop' and 'adam'. SGD is implemented with momentum.
<code>learn.rates</code>	the size of the steps to make in gradient descent. If set too large, the optimization might not converge to optimal values. If set too small, convergence will be slow. Should be either the size of the number of hidden layers plus one or equal to one. If a single learn rate is specified, this learn rate will be broadcasted across the layers.
L1	L1 regularization. Non-negative number. Set to zero for no regularization.
L2	L2 regularization. Non-negative number. Set to zero for no regularization.
<code>sgd.momentum</code>	numeric value specifying how much momentum should be used. Set to zero for no momentum, otherwise a value between zero and one.
<code>rmsprop.decay</code>	level of decay in the rms term. Controls the strength of the exponential decay of the squared gradients in the term that scales the gradient before the parameter update. Common values are 0.9, 0.99 and 0.999
<code>adam.beta1</code>	level of decay in the first moment estimate (the mean). The recommended value is 0.9
<code>adam.beta2</code>	level of decay in the second moment estimate (the uncentered variance). The recommended value is 0.999
<code>n.epochs</code>	the number of epochs to train. One epoch is a single iteration through the training data.
<code>batch.size</code>	the number of observations to use in each batch. Batch learning is computationally faster than stochastic gradient descent. However, large batches might not result in optimal learning, see Efficient Backprop by LeCun for details.
<code>drop.last</code>	logical. Only applicable if the size of the training set is not perfectly divisible by the batch size. Determines if the last chosen observations should be discarded (in the current epoch) or should constitute a smaller batch. Note that a smaller batch leads to a noisier approximation of the gradient.
<code>val.prop</code>	proportion of training data to use for tracking the loss on a validation set during training. Useful for assessing the training process and identifying possible overfitting. Set to zero for only tracking the loss on the training data.
<code>verbose</code>	logical indicating if additional information should be printed
<code>random.seed</code>	optional seed for the random number generator

### Details

A function for training Autoencoders. During training, the network will learn a generalised representation of the data (generalised since the middle layer acts as a bottleneck, resulting in reproduction of only the most important features of the data). As such, the network models the normal state of the data and therefore has a denoising property. This property can be exploited to detect anomalies by comparing input to reconstruction. If the difference (the reconstruction error) is large, the observation is a possible anomaly.

**Value**

An ANN object. Use function `plot(<object>)` to assess loss on training and optionally validation data during training process. Use function `predict(<object>, <newdata>)` for prediction.

**Examples**

```
# Autoencoder example
X <- USArrests
AE <- autoencoder(X, c(10,2,10), loss.type = 'pseudo-huber',
  activ.functions = c('tanh','linear','tanh'),
  batch.size = 8, optim.type = 'adam',
  n.epochs = 1000, val.prop = 0)

# Plot loss during training
plot(AE)

# Make reconstruction and compression plots
reconstruction_plot(AE, X)
compression_plot(AE, X)

# Reconstruct data and show states with highest anomaly scores
recX <- reconstruct(AE, X)
sort(recX$anomaly_scores, decreasing = TRUE)[1:5]
```

---

compression\_plot      *Compression plot*

---

**Description**

plot compressed observation in pairwise dimensions

**Usage**

```
compression_plot(object, ...)

## S3 method for class 'ANN'
compression_plot(object, X, colors = NULL, jitter = FALSE, ...)
```

**Arguments**

object	autoencoder object of class ANN
...	arguments to be passed to <code>jitter()</code>
X	data matrix with original values to be compressed and plotted
colors	optional vector of discrete colors
jitter	logical specifying whether to apply jitter to the compressed values. Especially useful with step activation function that clusters the compressions and reconstructions.

**Details**

Matrix plot of pairwise dimensions

**Value**

Plots

---

decode	<i>Decoding step</i>
--------	----------------------

---

**Description**

Decompress low-dimensional representation resulting from the nodes of the middle layer. Output are the reconstructed inputs to function encode()

**Usage**

```
decode(object, ...)

## S3 method for class 'ANN'
decode(object, compressed, compression.layer = NULL, ...)
```

**Arguments**

object	Object of class ANN
...	arguments to be passed down
compressed	Compressed data
compression.layer	Integer specifying which hidden layer is the compression layer. If NULL this parameter is inferred from the structure of the network (hidden layer with smallest number of nodes)

---

encode	<i>Encoding step</i>
--------	----------------------

---

**Description**

Compress data according to trained replicator or autoencoder. Outputs are the activations of the nodes in the middle layer for each observation in newdata

**Usage**

```
encode(object, ...)

## S3 method for class 'ANN'
encode(object, newdata, compression.layer = NULL, ...)
```

**Arguments**

object	Object of class ANN
...	arguments to be passed down
newdata	Data to compress
compression.layer	Integer specifying which hidden layer is the compression layer. If NULL this parameter is inferred from the structure of the network (hidden layer with smallest number of nodes)

---

neuralnetwork	<i>Train a Neural Network</i>
---------------	-------------------------------

---

**Description**

Construct and train a Multilayer Neural Network for regression or classification

**Usage**

```
neuralnetwork(  
  X,  
  y,  
  hidden.layers,  
  regression = FALSE,  
  standardize = TRUE,  
  loss.type = "log",  
  huber.delta = 1,  
  activ.functions = "tanh",  
  step.H = 5,  
  step.k = 100,  
  optim.type = "sgd",  
  learn.rates = 1e-04,  
  L1 = 0,  
  L2 = 0,  
  sgd.momentum = 0.9,  
  rmsprop.decay = 0.9,  
  adam.beta1 = 0.9,  
  adam.beta2 = 0.999,  
  n.epochs = 100,  
  batch.size = 32,  
  drop.last = TRUE,  
  val.prop = 0.1,  
  verbose = TRUE,  
  random.seed = NULL  
)
```

**Arguments**

X	matrix with explanatory variables
y	matrix with dependent variables. For classification this should be a one-column matrix containing the classes - classes will be one-hot encoded.
hidden.layers	vector specifying the number of nodes in each layer. The number of hidden layers in the network is implicitly defined by the length of this vector. Set hidden.layers to NA for a network with no hidden layers
regression	logical indicating regression or classification. In case of TRUE (regression), the activation function in the last hidden layer will be the linear activation function (identity function). In case of FALSE (classification), the activation function in the last hidden layer will be the softmax, and the log loss function should be used.
standardize	logical indicating if X and Y should be standardized before training the network. Recommended to leave at TRUE for faster convergence.
loss.type	which loss function should be used. Options are "log", "squared", "absolute", "huber" and "pseudo-huber". The log loss function should be used for classification (regression = FALSE), and ONLY for classification.
huber.delta	used only in case of loss functions "huber" and "pseudo-huber". This parameter controls the cut-off point between quadratic and absolute loss.
activ.functions	character vector of activation functions to be used in each hidden layer. Possible options are 'tanh', 'sigmoid', 'relu', 'linear', 'ramp' and 'step'. Should be either the size of the number of hidden layers or equal to one. If a single activation type is specified, this type will be broadcasted across the hidden layers.
step.H	number of steps of the step activation function. Only applicable if activ.functions includes 'step'.
step.k	parameter controlling the smoothness of the step activation function. Larger values lead to a less smooth step function. Only applicable if activ.functions includes 'step'.
optim.type	type of optimizer to use for updating the parameters. Options are 'sgd', 'rmsprop' and 'adam'. SGD is implemented with momentum.
learn.rates	the size of the steps to make in gradient descent. If set too large, the optimization might not converge to optimal values. If set too small, convergence will be slow. Should be either the size of the number of hidden layers plus one or equal to one. If a single learn rate is specified, this learn rate will be broadcasted across the layers.
L1	L1 regularization. Non-negative number. Set to zero for no regularization.
L2	L2 regularization. Non-negative number. Set to zero for no regularization.
sgd.momentum	numeric value specifying how much momentum should be used. Set to zero for no momentum, otherwise a value between zero and one.
rmsprop.decay	level of decay in the rms term. Controls the strength of the exponential decay of the squared gradients in the term that scales the gradient before the parameter update. Common values are 0.9, 0.99 and 0.999.



adam.beta1	level of decay in the first moment estimate (the mean). The recommended value is 0.9.
adam.beta2	level of decay in the second moment estimate (the uncentered variance). The recommended value is 0.999.
n.epochs	the number of epochs to train. One epoch is a single iteration through the training data.
batch.size	the number of observations to use in each batch. Batch learning is computationally faster than stochastic gradient descent. However, large batches might not result in optimal learning, see Efficient Backprop by LeCun for details.
drop.last	logical. Only applicable if the size of the training set is not perfectly divisible by the batch size. Determines if the last chosen observations should be discarded (in the current epoch) or should constitute a smaller batch. Note that a smaller batch leads to a noisier approximation of the gradient.
val.prop	proportion of training data to use for tracking the loss on a validation set during training. Useful for assessing the training process and identifying possible overfitting. Set to zero for only tracking the loss on the training data.
verbose	logical indicating if additional information should be printed
random.seed	optional seed for the random number generator

### Details

A generic function for training Neural Networks for classification and regression problems. Various types of activation and loss functions are supported, as well as L1 and L2 regularization. Possible optimizer include SGD (with or without momentum), RMSprop and Adam.

### Value

An ANN object. Use function `plot(<object>)` to assess loss on training and optionally validation data during training process. Use function `predict(<object>, <newdata>)` for prediction.

### References

LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

### Examples

```
# Example on iris dataset
# Prepare test and train sets
random_draw <- sample(1:nrow(iris), size = 100)
X_train <- iris[random_draw, 1:4]
y_train <- iris[random_draw, 5]
X_test <- iris[setdiff(1:nrow(iris), random_draw), 1:4]
y_test <- iris[setdiff(1:nrow(iris), random_draw), 5]

# Train neural network on classification task
NN <- neuralnetwork(X = X_train, y = y_train, hidden.layers = c(5, 5),
                    optim.type = 'adam', learn.rates = 0.01, val.prop = 0)
```

```
# Plot the loss during training
plot(NN)

# Make predictions
y_pred <- predict(NN, newdata = X_test)

# Plot predictions
correct <- (y_test == y_pred$predictions)
plot(X_test, pch = as.numeric(y_test), col = correct + 2)
```

---

plot.ANN

*Plot training and validation loss*

---

### Description

plot Generate plots of the loss against epochs

### Usage

```
## S3 method for class 'ANN'
plot(x, max.points = 1000, ...)
```

### Arguments

x	Object of class ANN
max.points	Maximum number of points to plot, set to NA, NULL or Inf to include all points in the plot
...	further arguments to be passed to plot

### Details

A generic function for plot loss of neural net

### Value

Plots

---

predict.ANN	<i>Make predictions for new data</i>
-------------	--------------------------------------

---

**Description**

predict Predict class or value for new data

**Usage**

```
## S3 method for class 'ANN'  
predict(object, newdata, ...)
```

**Arguments**

object	Object of class ANN
newdata	Data to make predictions on
...	further arguments (not in use)

**Details**

A generic function for training neural nets

**Value**

A list with predicted classes for classification and fitted probabilities

---

print.ANN	<i>Print ANN</i>
-----------	------------------

---

**Description**

Print info on trained Neural Network

**Usage**

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

**Arguments**

x	Object of class ANN
...	Further arguments

---

read_ANN	<i>Read ANN object from file</i>
----------	----------------------------------

---

**Description**

Deserialize ANN object from binary file

**Usage**

```
read_ANN(file)
```

**Arguments**

file                    character specifying file path

**Value**

Object of class ANN

---

reconstruct	<i>Reconstruct data using trained ANN object of type autoencoder</i>
-------------	--

---

**Description**

reconstruct takes new data as input and reconstructs the observations using a trained replicator or autoencoder object.

**Usage**

```
reconstruct(object, X)
```

**Arguments**

object                Object of class ANN created with autoencoder()  
X                      data matrix to reconstruct

**Details**

A generic function for training neural nets

**Value**

Reconstructed observations and anomaly scores (reconstruction errors)

---

reconstruction\_plot    *Reconstruction plot*


---

**Description**

plots original and reconstructed data points in a single plot with connecting lines between original value and corresponding reconstruction

**Usage**

```
reconstruction_plot(object, ...)

## S3 method for class 'ANN'
reconstruction_plot(object, X, colors = NULL, ...)
```

**Arguments**

object	autoencoder object of class ANN
...	arguments to be passed down
X	data matrix with original values to be reconstructed and plotted
colors	optional vector of discrete colors. The reconstruction errors are used as color if this argument is not specified

**Details**

Matrix plot of pairwise dimensions

**Value**

Plots

---

train                    *Continue training of a Neural Network*


---

**Description**

Continue training of a neural network object returned by neuralnetwork() or autoencoder()

**Usage**

```
train(
  object,
  X,
  y = NULL,
  n.epochs = 100,
  batch.size = 32,
  drop.last = TRUE,
  val.prop = 0.1,
  random.seed = NULL
)
```

**Arguments**

object	object of class ANN produced by neuralnetwork() or autoencoder()
X	matrix with explanatory variables
y	matrix with dependent variables. Not required if object is an autoencoder
n.epochs	the number of epochs to train. This parameter largely determines the training time (one epoch is a single iteration through the training data).
batch.size	the number of observations to use in each batch. Batch learning is computationally faster than stochastic gradient descent. However, large batches might not result in optimal learning, see Efficient Backprop by Le Cun for details.
drop.last	logical. Only applicable if the size of the training set is not perfectly divisible by the batch size. Determines if the last chosen observations should be discarded (in the current epoch) or should constitute a smaller batch. Note that a smaller batch leads to a noisier approximation of the gradient.
val.prop	proportion of training data to use for tracking the loss on a validation set during training. Useful for assessing the training process and identifying possible overfitting. Set to zero for only tracking the loss on the training data.
random.seed	optional seed for the random number generator

**Details**

A new validation set is randomly chosen. This can result in irregular jumps in the plot given by plot.ANN().

**Value**

An ANN object. Use function plot(<object>) to assess loss on training and optionally validation data during training process. Use function predict(<object>, <newdata>) for prediction.

**References**

LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

**Examples**

```
# Train a neural network on the iris dataset
X <- iris[,1:4]
y <- iris$Species
NN <- neuralnetwork(X, y, hidden.layers = 10, sgd.momentum = 0.9,
                    learn.rates = 0.01, val.prop = 0.3, n.epochs = 100)

# Plot training and validation loss during training
plot(NN)

# Continue training for 1000 epochs
train(NN, X, y, n.epochs = 200, val.prop = 0.3)

# Again plot the loss - note the jump in the validation loss at the 100th epoch
# This is due to the random selection of a new validation set
plot(NN)
```

---

write\_ANN

*Write ANN object to file*

---

**Description**

Serialize ANN object to binary file

**Usage**

```
write_ANN(object, file)
```

**Arguments**

object	Object of class ANN
file	character specifying file path

# Index

ANN, [2](#)  
autoencoder, [2](#)  
compression\_plot, [5](#)  
decode, [6](#)  
encode, [6](#)  
neuralnetwork, [7](#)  
plot.ANN, [10](#)  
predict.ANN, [11](#)  
print.ANN, [11](#)  
Rcpp\_ANN-class (ANN), [2](#)  
read\_ANN, [12](#)  
reconstruct, [12](#)  
reconstruction\_plot, [13](#)  
train, [13](#)  
write\_ANN, [15](#)