

nnlib2Rcpp is an R package containing a number of Neural Network (NN) implementations. The NNs are implemented in C++ (using **nnlib2** C++ class library) and are interfaced with R via **Rcpp** package (which is required). The package *currently* includes versions of Back-Propagation, Autoencoder, Learning Vector Quantization (unsupervised and supervised) and simple Matrix-Associative-Memory neural networks. Functions and modules for directly using these models from R are provided.

Furthermore, a new “NN” module (NN-class) has been added to version 0.1.4, that allows the creation of custom NNs using predefined components, and manipulation of the network and its components from R. It also provides a fixed procedure for defining new NN component types (layers, nodes, sets of connections etc) which can then be used in the module (some familiarity with C++ is required).

The “NN” (aka NN-class) provides methods for handling the NN, such as: `add_layer`, `add_connection_set`, `create_connections_in_sets`, `connect_layers_at`, `fully_connect_layers_at`, `add_single_connection`, `input_at`, `encode_at`, `encode_all`, `recall_at`, `recall_all`, `get_output_from`, `get_input_at`, `get_weights_at`, `print`, `outline`.

A (rather useless and silly) example of using the “NN” module follows:

```
# (1.A) create new 'NN' object:
n <- new("NN")

# (1.B) Add topology components:
# 1. add a layer of 4 generic nodes:
n$add_layer("generic",4)
# 2. add (empty) set for connections that pass data unmodified:
n$add_connection_set("pass-through")
# 3. add another layer of 2 generic nodes:
n$add_layer("generic",2)
# 4. add (empty) set for connections that pass data * weight:
n$add_connection_set("wpass-through")
# 5. add a layer of 1 generic node:
n$add_layer("generic",1)
# Create actual connections in sets,w/random initial weights in [0,1]:
n$create_connections_in_sets(0,1)
# Optionally, show an outline of the topology:
n$outline()

# (1.C) use the network.
# input some data, and run create output for it:
n$input_at(1,c(10,20,30,40))
n$recall_all(TRUE)
# the final output:
n$get_output_from(5)

# (1.D) optionally, examine the network:
# the input at first layer at position 1:
n$get_input_at(1)
# Data is passed unmodified through connections at position 2,
# and (by default) summed together at each node of layer at 3.
# Final output from layer in position 3:
n$get_output_from(3)
# Data is then passed multiplied by the random weights through
# connections at position 4. The weights of these connections:
n$get_weights_at(4)
# Data is finally summed together at the node of layer at position 5,
# producing the final output, which (again) is:
n$get_output_from(5)
```

The next example, creates a simple MAM using the “NN” (NN-class) module:

```
# (2.A) Create the NN and its components:

m <- new( "NN" )
m$add_layer( "generic" , 4 )
```

```

m$add_layer( "generic" , 3 )
m$fully_connect_layers_at(1, 2, "MAM", 0, 0)

# (2.B) Use it to store iris species:

iris_data    <- as.matrix( scale( iris[1:4] ) )
iris_species <- as.integer( iris$Species )
for(r in 1:nrow( iris_data ) )
{
  x <- iris_data[r,]
  z <- rep( -1, 3 )
  z [ iris_species[r] ] <- 1
  m$input_at( 1, x )
  m$input_at( 3, z )
  m$encode_all( TRUE )
}

# (2.C) Attempt to recall iris species:

recalled_ids <- NULL;
for(r in 1:nrow(iris_data))
{
  x <- iris_data[r,]
  m$input_at( 1, x )
  m$recall_all( TRUE )
  z <- m$get_output_from( 3 )
  recalled_ids <- c( recalled_ids, which.max ( z ) )
}

plot(iris_data, pch=recalled_ids)

```

Hopefully the collection of predefined components will expand (and any contribution of components is welcomed).