NeuraNet

P. Baillehache

March 16, 2019

Contents

Introduction

NeuraNet is a C library providing structures and functions to implement a neural network.

The neural network implemented in NeuraNet consists of a layer of input values, a layer of output values, a layer of hidden values, a set of generic base functions and a set of links. Each base function has 3 parameters (detailed below) and each links has 3 parameters: the base function index and the indices of input and output values. A NeuraNet is defined by the parameters' values of its generic base functions and links, and the number of input, output and hidden values.

The evaluation of the NeuraNet consists of taking each link, ordered on index of values, and apply the generic base function on the first value and store the result in the second value. If several links has the same second value index, the sum value of all these links is used. However if several links have same input and output values, the outputs of these links are multiplied instead of added (before being eventually added to other links having same output value but different input value).

The generic base functions is a linear function. However by using several links with same input and output values it is possible to simulate any polynomial function. Also, there is no concept of layer inside hidden values, but the input value index is constrained to be lower than the output one. So, the links can be arranged to form layers of subset of hidden values, while still allowing any other type of arrangement inside hidden values. Also, a link can be inactivated by setting its base function index to -1. Finally, the parameters of the base function and the hidden values are constrained to [-1.0,1.0].

NeuraNet provides functions to easily use the library GenAlg to search the values of base functions and links' parameters. An example is given in the unit tests (see below). It also provides functions to save and load the neural network (in JSON format).

NeuraNet has been validated on the Iris data set, the Abalone data set, the Arrhythmia data set, the Wisconsin Diagnostic Breast Cancer data set, the MNIST data set, the ORHD data set.

A utility tool allows to generate a file to be used as input of the CloudGraph tool to visualize the network of the NeuraNet.

It uses the `PBErr` library.

1 Definitions

The generic base function is defined as follow:

$$B(x) = [tan(1.57079 * b_0)(x + b_1) + b_2] \tag{1}$$

where $\{b_0, b_1, b_2\} \in [-1.0, 1.0]^3$ are the parameters of the base function and $x \in \mathbb{R}$ and $B(x) \in \mathbb{R}$.

2 Interface

```
// ============ NEURANET.H ================

#ifndef NEURANET_H
#define NEURANET_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"
#include "pbcextension.h"
#include "pbmath.h"
#include "gset.h"

// ----- NeuraNetBaseFun

// ================= Define ==================

#define NN_THETA 1.57079

// ================ Functions declaration ====================

// Generic base function for the NeuraNet
// 'param' is an array of 3 float all in [-1,1]
// 'x' is the input value
// NNBaseFun(param,x)=
// tan(param[0]*NN_THETA)*(x+param[1])+param[2]
#if BUILDMODE != 0
inline
#endif
float NNBaseFun(const float* const param, const float x);

// ----- NeuraNet

// ================= Define ==================

#define NN_NBPARAMBASE 3
#define NN_NBPARAMLINK 3

// ================= Data structure ==================

typedef struct NeuraNet {
  // Nb of input values
```

```
  const int _nbInputVal;
  // Nb of output values
  const int _nbOutputVal;
  // Nb max of hidden values
  const long _nbMaxHidVal;
  // Nb max of base functions
  const long _nbMaxBases;
  // Nb max of links
  const long _nbMaxLinks;
  // VecFloat describing the base functions
  // NN_NBPARAMBASE values per base function
  VecFloat* _bases;
  // VecShort describing the links
  // NN_NBPARAMLINK values per link (base id, input id, output id)
  // if (base id equals -1 the link is inactive)
  VecLong* _links;
  // Hidden values
  VecFloat* _hidVal;
  // Nb bases used for convolution
  const long _nbBasesConv;
  // Nb bases per cell used for convolution
  const long _nbBasesCellConv;
} NeuraNet;

// ================ Functions declaration ====================

// Create a new NeuraNet with 'nbInput' input values, 'nbOutput'
// output values, 'nbMaxHidden' hidden values, 'nbMaxBases' base
// functions, 'nbMaxLinks' links
NeuraNet* NeuraNetCreate(const int nbInput, const int nbOutput,
  const long nbMaxHidden, const long nbMaxBases, const long nbMaxLinks);

// Free the memory used by the NeuraNet 'that'
void NeuraNetFree(NeuraNet** that);

// Create a new NeuraNet with 'nbInput' input values, 'nbOutput'
// output values and a set of hidden layers described by
// 'hiddenLayers' as follow:
// The dimension of 'hiddenLayers' is the number of hidden layers
// and each component of 'hiddenLayers' is the number of hidden value
// in the corresponding hidden layer
// For example, <3,4> means 2 hidden layers, the first one with 3
// hidden values and the second one with 4 hidden values
// If 'hiddenValues' is null it means there is no hidden layers
// Then, links are automatically added between each input values
// toward each hidden values in the first hidden layer, then from each
// hidden values of the first hidden layer to each hidden value of the
// 2nd hidden layer and so on until each values of the output
NeuraNet* NeuraNetCreateFullyConnected(const int nbIn, const int nbOut,
  const VecLong* const hiddenLayers);

// Create a NeuraNet using convolution
// The input's dimension is equal to the dimension of 'dimIn', for
// example if dimIn==<2,3> the input is a 2D array of width 2 and
// height 3, input values are expected ordered by lines
// The NeuraNet has 'nbOutput' outputs
// The dimension of each convolution cells is 'dimCell'
// The maximum number of convolution (in depth) is 'depthConv'
// Each convolution layer has 'thickConv' convolutions in parallel
// The outputs are fully connected to the last layer of convolution cells
// For example, if the input is a 2D array of 4 cols and 3 rows, 2
// outputs, 2x2 convolution cell, convolution depth of 2, and
```

```
// convolution thickness of 2:
// index of values from input layer to ouput layer
// 00,01,02,03,
// 04,05,06,07,
// 08,09,10,11
//
// 12,13,14,  18,19,20,
// 15,16,17,  21,22,23,
//
// 24,25  26,27
//
// 28,29
//
// nbInput: 12
// nbOutput: 2
// nbHidden: 16
// nbMaxBases: 24
// nbMaxLinks: 72
// links:
//    0,0,12, 4,0,18, 1,1,12, 0,1,13, 5,1,18, 4,1,19, 1,2,13, 0,2,14,
//    5,2,19, 4,2,20, 1,3,14, 5,3,20, 2,4,12, 0,4,15, 6,4,18, 4,4,21,
//    3,5,12, 2,5,13, 1,5,15, 0,5,16, 7,5,18, 6,5,19, 5,5,21, 4,5,22,
//    3,6,13, 2,6,14, 1,6,16, 0,6,17, 7,6,19, 6,6,20, 5,6,22, 4,6,23,
//    3,7,14, 1,7,17, 7,7,20, 5,7,23, 2,8,15, 6,8,21, 3,9,15, 2,9,16,
//    7,9,21, 6,9,22, 3,10,16, 2,10,17, 7,10,22, 6,10,23, 3,11,17,
//    7,11,23, 8,12,24, 9,13,24, 8,13,25, 9,14,25, 10,15,24, 11,16,24,
//    10,16,25, 11,17,25, 12,18,26, 13,19,26, 12,19,27, 13,20,27,
//    14,21,26, 15,22,26, 14,22,27, 15,23,27, 16,24,28, 17,24,29,
//    18,25,28, 19,25,29, 20,26,28, 21,26,29, 22,27,28, 23,27,29
NeuraNet* NeuraNetCreateConvolution(const VecShort* const dimIn,
  const int nbOutput, const VecShort* const dimCell,
  const int depthConv, const int thickConv);

// Get the nb of input values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
int NNGetNbInput(const NeuraNet* const that);

// Get the nb of output values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
int NNGetNbOutput(const NeuraNet* const that);

// Get the nb max of hidden values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbMaxHidden(const NeuraNet* const that);

// Get the nb max of base functions of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbMaxBases(const NeuraNet* const that);

// Get the nb of base functions for convolution of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbBasesConv(const NeuraNet* const that);
```

```
// Get the nb of base functions per cell for convolution of
// the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbBasesCellConv(const NeuraNet* const that);

// Get the nb max of links of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbMaxLinks(const NeuraNet* const that);

// Get the parameters of the base functions of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* NNBases(const NeuraNet* const that);

// Get the links description of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
const VecLong* NNLinks(const NeuraNet* const that);

// Get the hidden values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* NNHiddenValues(const NeuraNet* const that);

// Get the 'iVal'-th hidden value of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
float NNGetHiddenValue(const NeuraNet* const that, const long iVal);

// Set the parameters of the base functions of the NeuraNet 'that' to
// a copy of 'bases'
// 'bases' must be of dimension that->nbMaxBases * NN_NBPARAMBASE
//  each base is defined as param[3] in [-1,1]
// tan(param[0]*NN_THETA)*(x+param[1])+param[2]
#if BUILDMODE != 0
inline
#endif
void NNSetBases(NeuraNet* const that, const VecFloat* const bases);

// Set the 'iBase'-th parameter of the base functions of the NeuraNet
// 'that' to 'base'
#if BUILDMODE != 0
inline
#endif
void NNBasesSet(NeuraNet* const that, const long iBase, const float base);

// Set the links description of the NeuraNet 'that' to a copy of 'links'
// Links with a base function equals to -1 are ignored
// If the input id is higher than the output id they are swap
// The links description in the NeuraNet are ordered in increasing
// value of input id and output id, but 'links' doesn't have to be
// sorted
// Each link is defined by (base index, input index, output index)
```

```
// If base index equals -1 it means the link is inactive
void NNSetLinks(NeuraNet* const that, VecLong* const links);

// Calculate the output values for the input values 'input' for the
// NeuraNet 'that' and memorize the result in 'output'
// input values in [-1,1] and output values in [-1,1]
// All values of 'output' are set to 0.0 before evaluating
// Links which refer to values out of bounds of 'input' or 'output'
// are ignored
void NNEval(const NeuraNet* const that, const VecFloat* const input, VecFloat* const output);

// Function which return the JSON encoding of 'that'
JSONNode* NNEncodeAsJSON(const NeuraNet* const that);

// Function which decode from JSON encoding 'json' to 'that'
bool NNDecodeAsJSON(NeuraNet** that, const JSONNode* const json);

// Save the NeuraNet 'that' to the stream 'stream'
// If 'compact' equals true it saves in compact form, else it saves in
// readable form
// Return true if the NeuraNet could be saved, false else
bool NNSave(const NeuraNet* const that, FILE* const stream, const bool compact);

// Load the NeuraNet 'that' from the stream 'stream'
// If 'that' is not null the memory is first freed
// Return true if the NeuraNet could be loaded, false else
bool NNLoad(NeuraNet** that, FILE* const stream);

// Print the NeuraNet 'that' to the stream 'stream'
void NNPrintln(const NeuraNet* const that, FILE* const stream);

// Get the number of active links in the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbActiveLinks(const NeuraNet* const that);

// Save the links of the NeuraNet 'that' into the file at 'url' in a
// format readable by CloudGraph
// Return true if we could save, else false
bool NNSaveLinkAsCloudGraph(const NeuraNet* const that, const char* url);

// Get the Simpson's diversity index of the hidden values of the
// NeuraNet 'that'
// Return value in [0.0, 1.0], 0.0 means no diversity, 1.0 means max
// diversity
float NNGetHiddenValSimpsonDiv(const NeuraNet* const that);

// Prune the NeuraNet 'that' by removing the useless links (those with
// no influence on outputs
void NNPrune(const NeuraNet* const that);

// Get the mutability vector for bases of the NeuraNet 'that' according
// to output's 'accuracy'
// accuracy is a VecFloat of dimension nbOutput, where accuracy[iOut]
// equals 1.0 if the NeuraNet always returns the perfect answer for the
// output iOut, and 0.0 if never
// Return a VecFloat of dimension nbBase * NB_PARAMBASE with values in
// [0.0, 1.0]
VecFloat* NNGetMutabilityBases(const NeuraNet* const that,
  const VecFloat* const accuracy);
```

```
// Get the mutability vector for links of the NeuraNet 'that' according
// to output's 'accuracy'
// accuracy is a VecFloat of dimension nbOutput, where accuracy[iOut]
// equals 1.0 if the NeuraNet always returns the perfect answer for the
// output iOut, and 0.0 if never
// Return a VecFloat of dimension nbLink * NB_PARAMLINK with values in
// [0.0, 1.0]
VecFloat* NNGetMutabilityLinks(const NeuraNet* const that,
  const VecFloat* const accuracy);

// ================= Interface with library GenAlg ==================
// To use the following functions the user must include the header
// 'genalg.h' before the header 'neuranet.h'

#ifdef GENALG_H

// Get the length of the adn of float values to be used in the GenAlg
// library for the NeuraNet 'that'
static long NNGetGAAdnFloatLength(const NeuraNet* const that)
  __attribute__((unused));
static long NNGetGAAdnFloatLength(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return NNGetNbMaxBases(that) * NN_NBPARAMBASE;
}

// Get the length of the adn of int values to be used in the GenAlg
// library for the NeuraNet 'that'
static long NNGetGAAdnIntLength(const NeuraNet* const that)
  __attribute__((unused));
static long NNGetGAAdnIntLength(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return NNGetNbMaxLinks(that) * NN_NBPARAMLINK;
}

// Set the bounds of the GenAlg 'ga' to be used for bases parameters of
// the NeuraNet 'that'
static void NNSetGABoundsBases(const NeuraNet* const that, GenAlg* const ga)
  __attribute__((unused));
static void NNSetGABoundsBases(const NeuraNet* const that, GenAlg* const ga) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (ga == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'ga' is null");
    PBErrCatch(NeuraNetErr);
  }
```

```
  if (GAGetLengthAdnFloat(ga) != NNGetGAAdnFloatLength(that)) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'ga' 's float genes dimension doesn't\
 matches 'that' 's max nb of bases (%ld==%ld)",
      GAGetLengthAdnFloat(ga), NNGetGAAdnFloatLength(that));
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a vector to memorize the bounds
  VecFloat2D bounds = VecFloatCreateStatic2D();
  // Init the bounds
  VecSet(&bounds, 0, -1.0); VecSet(&bounds, 1, 1.0);
  // For each gene
  for (long iGene = NNGetGAAdnFloatLength(that); iGene--;)
    // Set the bounds
    GASetBoundsAdnFloat(ga, iGene, &bounds);
}

// Set the bounds of the GenAlg 'ga' to be used for links description of
// the NeuraNet 'that'
static void NNSetGABoundsLinks(const NeuraNet* const that, GenAlg* const ga)
  __attribute__((unused));
static void NNSetGABoundsLinks(const NeuraNet* const that, GenAlg* const ga) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (ga == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'ga' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (GAGetLengthAdnInt(ga) != NNGetGAAdnIntLength(that)) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'ga' 's int genes dimension doesn't\
 matches 'that' 's max nb of links (%ld==%ld)",
      GAGetLengthAdnInt(ga), NNGetGAAdnIntLength(that));
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a vector to memorize the bounds
  VecLong2D bounds = VecLongCreateStatic2D();
  // For each gene
  for (long iGene = 0; iGene < NNGetGAAdnIntLength(that);
    iGene += NN_NBPARAMLINK) {
    // Set the bounds for base id
    VecSet(&bounds, 0, -1);
    VecSet(&bounds, 1, NNGetNbMaxBases(that) - 1);
    GASetBoundsAdnInt(ga, iGene, &bounds);
    // Set the bounds for input value
    VecSet(&bounds, 0, 0);
    VecSet(&bounds, 1, NNGetNbInput(that) + NNGetNbMaxHidden(that) - 1);
    GASetBoundsAdnInt(ga, iGene + 1, &bounds);
    // Set the bounds for input value
    VecSet(&bounds, 0, NNGetNbInput(that));
    VecSet(&bounds, 1, NNGetNbInput(that) + NNGetNbMaxHidden(that) +
      NNGetNbOutput(that) - 1);
    GASetBoundsAdnInt(ga, iGene + 2, &bounds);
  }
}
```

```
#endif

// ================ Inliner ====================

#if BUILDMODE != 0
#include "neuranet-inline.c"
#endif


#endif
```

3 Code

3.1 neuranet.c

```
// ============ NEURANET.C ================

// ================= Include =================

#include "neuranet.h"
#if BUILDMODE == 0
#include "neuranet-inline.c"
#endif

// ----- NeuraNet

// ================ Functions implementation ====================

// Create a new NeuraNet with 'nbInput' input values, 'nbOutput'
// output values, 'nbMaxHidden' hidden values, 'nbMaxBases' base
// functions, 'nbMaxLinks' links
NeuraNet* NeuraNetCreate(const int nbInput, const int nbOutput,
  const long nbMaxHidden, const long nbMaxBases, const long nbMaxLinks) {
#if BUILDMODE == 0
  if (nbInput <= 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbInput' is invalid (0<%d)", nbInput);
    PBErrCatch(NeuraNetErr);
  }
  if (nbOutput <= 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbOutput' is invalid (0<%d)", nbOutput);
    PBErrCatch(NeuraNetErr);
  }
  if (nbMaxHidden < 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbMaxHidden' is invalid (0<=%ld)",
      nbMaxHidden);
    PBErrCatch(NeuraNetErr);
  }
  if (nbMaxBases <= 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbMaxBases' is invalid (0<%ld)",
      nbMaxBases);
    PBErrCatch(NeuraNetErr);
  }
  if (nbMaxLinks <= 0) {
```

```
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbMaxLinks' is invalid (0<%ld)",
      nbMaxLinks);
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare the new NeuraNet
  NeuraNet* that = PBErrMalloc(NeuraNetErr, sizeof(NeuraNet));
  // Set properties
  *(int*)&(that->_nbInputVal) = nbInput;
  *(int*)&(that->_nbOutputVal) = nbOutput;
  *(long*)&(that->_nbMaxHidVal) = nbMaxHidden;
  *(long*)&(that->_nbMaxBases) = nbMaxBases;
  *(long*)&(that->_nbMaxLinks) = nbMaxLinks;
  *(long*)&(that->_nbBasesConv) = 0;
  *(long*)&(that->_nbBasesCellConv) = 0;
  that->_bases = VecFloatCreate(nbMaxBases * NN_NBPARAMBASE);
  that->_links = VecLongCreate(nbMaxLinks * NN_NBPARAMLINK);
  if (nbMaxHidden > 0)
    that->_hidVal = VecFloatCreate(nbMaxHidden);
  else
    that->_hidVal = NULL;
  // Return the new NeuraNet
  return that;
}

// Free the memory used by the NeuraNet 'that'
void NeuraNetFree(NeuraNet** that) {
  // Check argument
  if (that == NULL || *that == NULL)
    // Nothing to do
    return;
  // Free memory
  VecFree(&((*that)->_bases));
  VecFree(&((*that)->_links));
  VecFree(&((*that)->_hidVal));
  free(*that);
  *that = NULL;
}

// Create a new NeuraNet with 'nbIn' innput values, 'nbOut'
// output values and a set of hidden layers described by
// 'hiddenLayers' as follow:
// The dimension of 'hiddenLayers' is the number of hidden layers
// and each component of 'hiddenLayers' is the number of hidden value
// in the corresponding hidden layer
// For example, <3,4> means 2 hidden layers, the first one with 3
// hidden values and the second one with 4 hidden values
// If 'hiddenValues' is null it means there is no hidden layers
// Then, links are automatically added between each input values
// toward each hidden values in the first hidden layer, then from each
// hidden values of the first hidden layer to each hidden value of the
// 2nd hidden layer and so on until each values of the output
NeuraNet* NeuraNetCreateFullyConnected(const int nbIn, const int nbOut,
  const VecLong* const hiddenLayers) {
#if BUILDMODE == 0
  if (nbIn <= 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbInput' is invalid (0<%d)", nbIn);
    PBErrCatch(NeuraNetErr);
  }
  if (nbOut <= 0) {
```

```
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbOutput' is invalid (0<%d)", nbOut);
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare variable to memorize the number of links, bases
  // and hidden values
  long nbHiddenVal = 0;
  long nbBases = 0;
  long nbLinks = 0;
  long nbHiddenLayer = 0;
  // If there are hidden layers
  if (hiddenLayers != NULL) {
    // Get the number of hidden layers
    nbHiddenLayer = VecGetDim(hiddenLayers);
    // Declare two variables for computation
    long nIn = nbIn;
    long nOut = 0;
    // Calculate the nb of links and hidden values
    for (long iLayer = 0; iLayer < nbHiddenLayer; ++iLayer) {
      nOut = VecGet(hiddenLayers, iLayer);
      nbHiddenVal += nOut;
      nbLinks += nIn * nOut;
      nIn = nOut;
    }
    nbLinks += nIn * nbOut;
  // Else, there is no hidden layers
  } else {
    // Set the number of links
    nbLinks = nbIn * nbOut;
  }
  // There is one base function per link
  nbBases = nbLinks;
  // Create the NeuraNet
  NeuraNet* nn =
    NeuraNetCreate(nbIn, nbOut, nbHiddenVal, nbBases, nbLinks);
  // Declare a variable to memorize the index of the link
  long iLink = 0;
  // Declare variables for computation
  long shiftIn = 0;
  long shiftOut = nbIn;
  long nIn = nbIn;
  long nOut = 0;
  // Loop on hidden layers
  for (long iLayer = 0; iLayer <= nbHiddenLayer; ++iLayer) {
    // Init the links
    if (iLayer < nbHiddenLayer)
      nOut = VecGet(hiddenLayers, iLayer);
    else
      nOut = nbOut;
    for (long iIn = 0; iIn < nIn; ++iIn) {
      for (long iOut = 0; iOut < nOut; ++iOut) {
        long jLink = NN_NBPARAMLINK * iLink;
        VecSet(nn->_links, jLink, iLink);
        VecSet(nn->_links, jLink + 1, iIn + shiftIn);
        VecSet(nn->_links, jLink + 2, iOut + shiftOut);
        ++iLink;
      }
    }
    shiftIn = shiftOut;
    shiftOut += nOut;
    nIn = nOut;
```

```
  }
  // Return the new NeuraNet
  return nn;
}

// Create a NeuraNet using convolution
// The input's dimension is equal to the dimension of 'dimIn', for
// example if dimIn==<2,3> the input is a 2D array of width 2 and
// height 3, input values are expected ordered by lines
// The NeuraNet has 'nbOutput' outputs
// The dimension of each convolution cells is 'dimCell'
// The maximum number of convolution (in depth) is 'depthConv'
// Each convolution layer has 'thickConv' convolutions in parallel
// The outputs are fully connected to the last layer of convolution cells
// For example, if the input is a 2D array of 4 cols and 3 rows, 2
// outputs, 2x2 convolution cell, convolution depth of 2, and
// convolution thickness of 2:
// index of values from input layer to ouput layer
// 00,01,02,03,
// 04,05,06,07,
// 08,09,10,11
//
// 12,13,14,  18,19,20,
// 15,16,17,  21,22,23,
//
// 24,25  26,27
//
// 28,29
//
// nbInput: 12
// nbOutput: 2
// nbHidden: 16
// nbMaxBases: 24
// nbMaxLinks: 72
// links:
//    0,0,12, 4,0,18, 1,1,12, 0,1,13, 5,1,18, 4,1,19, 1,2,13, 0,2,14,
//    5,2,19, 4,2,20, 1,3,14, 5,3,20, 2,4,12, 0,4,15, 6,4,18, 4,4,21,
//    3,5,12, 2,5,13, 1,5,15, 0,5,16, 7,5,18, 6,5,19, 5,5,21, 4,5,22,
//    3,6,13, 2,6,14, 1,6,16, 0,6,17, 7,6,19, 6,6,20, 5,6,22, 4,6,23,
//    3,7,14, 1,7,17, 7,7,20, 5,7,23, 2,8,15, 6,8,21, 3,9,15, 2,9,16,
//    7,9,21, 6,9,22, 3,10,16, 2,10,17, 7,10,22, 6,10,23, 3,11,17,
//    7,11,23, 8,12,24, 9,13,24, 8,13,25, 9,14,25, 10,15,24, 11,16,24,
//    10,16,25, 11,17,25, 12,18,26, 13,19,26, 12,19,27, 13,20,27,
//    14,21,26, 15,22,26, 14,22,27, 15,23,27, 16,24,28, 17,24,29,
//    18,25,28, 19,25,29, 20,26,28, 21,26,29, 22,27,28, 23,27,29
NeuraNet* NeuraNetCreateConvolution(const VecShort* const dimIn,
  const int nbOutput, const VecShort* const dimCell,
  const int depthConv, const int thickConv) {
#if BUILDMODE == 0
  if (dimIn == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'dimIn' is null");
    PBErrCatch(NeuraNetErr);
  }
  for (long iDim = VecGetDim(dimIn); iDim--;)
    if (VecGet(dimIn, iDim) <= 0) {
      NeuraNetErr->_type = PBErrTypeInvalidArg;
      sprintf(NeuraNetErr->_msg, "'dimIn' %ldth dim is invalid (%d>0)",
        iDim, VecGet(dimIn, iDim));
      PBErrCatch(NeuraNetErr);
    }
  if (nbOutput <= 0) {
```

```
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbOutput' is invalid (0<%d)", nbOutput);
    PBErrCatch(NeuraNetErr);
  }
  if (dimCell == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'dimCell' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (VecGetDim(dimCell) != VecGetDim(dimIn)) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'dimCell' 's dim is invalid (%ld==%ld)",
      VecGetDim(dimCell), VecGetDim(dimIn));
    PBErrCatch(NeuraNetErr);
  }
  for (long iDim = VecGetDim(dimCell); iDim--;)
    if (VecGet(dimCell, iDim) <= 0) {
      NeuraNetErr->_type = PBErrTypeInvalidArg;
      sprintf(NeuraNetErr->_msg, "'dimCell' %ldth dim is invalid (%d>0)",
        iDim, VecGet(dimCell, iDim));
      PBErrCatch(NeuraNetErr);
    }
  if (depthConv < 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'depthConv' is invalid (0<=%d)",
      depthConv);
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a variable to memorize the nb of input, hidden values,
  // bases and links
  long nbIn = 0;
  long nbHiddenVal = 0;
  long nbBases = 0;
  long nbLinks = 0;
  // Calculate the number of inputs
  nbIn = 1;
  for (long iDim = VecGetDim(dimIn); iDim--;)
    nbIn *= VecGet(dimIn, iDim);
  // Calculate the number of bases, links and hidden values
  // Declare a variable to memorize the number of links per cell
  long nbLinkPerCell = 1;
  for (long iDim = VecGetDim(dimCell); iDim--;)
    nbLinkPerCell *= VecGet(dimCell, iDim);
  // Declare a variable to memorize the position of the convolution
  // cell in the current convolution layer
  VecShort* pos = VecShortCreate(VecGetDim(dimIn));
  // Declare a variable to memorize the position in the convolution cell
  VecShort* posCell = VecShortCreate(VecGetDim(dimIn));
  // Declare variables to memorize the dimension and size of the input
  // layer at current convolution level
  VecShort* curDimIn = VecClone(dimIn);
  long sizeLayerIn = 1;
  for (long iDim = VecGetDim(curDimIn); iDim--;)
    sizeLayerIn *= VecGet(curDimIn, iDim);
  // Declare variables to memorize the dimension and size of the
  // output layer at current convolution level
  VecShort* curDimOut = VecClone(curDimIn);
  long sizeLayerOut = 1;
  for (long iDim = VecGetDim(curDimOut); iDim--;) {
    VecSetAdd(curDimOut, iDim, -1 * VecGet(dimCell, iDim) + 1);
    sizeLayerOut *= VecGet(curDimOut, iDim);
```

```
}
// Loop on convolution levels
for (long iConv = 0; iConv < depthConv; ++iConv) {
  // Update the number of bases
  nbBases += nbLinkPerCell;
  // Update the number of hidden values
  nbHiddenVal += sizeLayerOut;
  // Update the number of links
  nbLinks += sizeLayerOut * nbLinkPerCell;
  // If we are not a the last convolution level
  if (iConv < depthConv - 1) {
    // Update input and output dimensions at next convolution level
    VecCopy(curDimIn, curDimOut);
    sizeLayerIn = sizeLayerOut;
    sizeLayerOut = 1;
    for (long iDim = VecGetDim(curDimOut); iDim--;) {
      VecSetAdd(curDimOut, iDim, -1 * VecGet(dimCell, iDim) + 1);
      sizeLayerOut *= VecGet(curDimOut, iDim);
    }
  }
}
// Multiply by the number of convolution in parallel
nbHiddenVal *= thickConv;
nbBases *= thickConv;
nbLinks *= thickConv;
long nbBasesConv = nbBases;
// Add the links and bases for the fully connected layer toward output
nbBases += sizeLayerOut * thickConv * nbOutput;
nbLinks += sizeLayerOut * thickConv * nbOutput;
// Create the NeuraNet
NeuraNet* nn =
  NeuraNetCreate(nbIn, nbOutput, nbHiddenVal, nbBases, nbLinks);
*(long*)&(nn->_nbBasesConv) = nbBasesConv;
*(long*)&(nn->_nbBasesCellConv) = nbLinkPerCell;
// Declare variables to create the links
VecLong* links = VecLongCreate(nbLinks * NN_NBPARAMLINK);
// Declare a variable to memorize the index of the currenlty
// created link
long iLink = 0;
// Reset the dimension and size of the input layer at current
// convolution level
VecCopy(curDimIn, dimIn);
sizeLayerIn = 1;
for (long iDim = VecGetDim(curDimIn); iDim--;) {
  sizeLayerIn *= VecGet(curDimIn, iDim);
}
// Reset the dimension and size of the output layer at current
// convolution level
VecCopy(curDimOut, dimIn);
sizeLayerOut = 1;
for (long iDim = VecGetDim(curDimOut); iDim--;) {
  VecSetAdd(curDimOut, iDim, -1 * VecGet(dimCell, iDim) + 1);
  sizeLayerOut *= VecGet(curDimOut, iDim);
}
// Declare variables to memorize the index of the beginning of the
// input and output layer and base functions at current convolution
// level
long* iStartBase = PBErrMalloc(NeuraNetErr, sizeof(long) * thickConv);
long* iStartLayerIn = PBErrMalloc(NeuraNetErr,
  sizeof(long) * thickConv);
long* iStartLayerOut = PBErrMalloc(NeuraNetErr,
  sizeof(long) * thickConv);
```

```
for (long iThick = 0; iThick < thickConv; ++iThick) {
  iStartLayerIn[iThick] = 0;
  iStartLayerOut[iThick] = sizeLayerIn + iThick * sizeLayerOut;
  iStartBase[iThick] = iThick * nbLinkPerCell;
}
// Loop on convolution levels
for (long iConv = 0; iConv < depthConv; ++iConv) {
  // Reset the position of the convolution cell in the input layer
  VecSetNull(pos);
  // Loop on position of the convolution cell at the current
  // convolution levels
  do {
    do {
      // Loop on convolution in parallel
      for (long iThick = 0; iThick < thickConv; ++iThick) {
        // Declare a variable to memorize the index of the input of the
        // current link
        long iInput = 0;
        for (long iDim = VecGetDim(curDimIn); iDim--;) {
          iInput *= VecGet(curDimIn, iDim);
          iInput += VecGet(posCell, iDim) + VecGet(pos, iDim);
        }
        iInput += iStartLayerIn[iThick];
        // Declare a variable to memorize the index of the output of
        // the current link
        long iOutput = 0;
        for (long iDim = VecGetDim(curDimOut); iDim--;) {
          iOutput *= VecGet(curDimOut, iDim);
          iOutput += VecGet(pos, iDim);
        }
        iOutput += iStartLayerOut[iThick];
        // Declare a variable to memorize the index of the base of the
        // current link
        long iBase = 0;
        for (long iDim = VecGetDim(posCell); iDim--;) {
          iBase *= VecGet(dimCell, iDim);
          iBase += VecGet(posCell, iDim);
        }
        iBase += iStartBase[iThick];
        // Set the current link's parameters
        VecSet(links, iLink * NN_NBPARAMLINK, iBase);
        VecSet(links, iLink * NN_NBPARAMLINK + 1, iInput);
        VecSet(links, iLink * NN_NBPARAMLINK + 2, iOutput);
        // Increment the index of the current link
        ++iLink;
      }
    } while (VecPStep(posCell, dimCell));
  } while (VecPStep(pos, curDimOut));
  // If we are not at the last convolution level
  if (iConv < depthConv - 1) {
    // Update input and output dimensions at next convolution level
    VecCopy(curDimIn, curDimOut);
    sizeLayerIn = sizeLayerOut;
    sizeLayerOut = 1;
    for (long iDim = VecGetDim(curDimOut); iDim--;) {
      VecSetAdd(curDimOut, iDim, -1 * VecGet(dimCell, iDim) + 1);
      sizeLayerOut *= VecGet(curDimOut, iDim);
    }
  }
  // Update the start index of input and output layers and bases
  // for each convolution in parallel
  for (long iThick = 0; iThick < thickConv; ++iThick) {
```

```
      iStartLayerIn[iThick] = iStartLayerOut[iThick];
      iStartLayerOut[iThick] = iStartLayerIn[0] +
        thickConv * sizeLayerIn + iThick * sizeLayerOut;
      iStartBase[iThick] =
        ((iConv + 1) * thickConv + iThick) * nbLinkPerCell;
    }
  }
  // Set the links of the last fully connected layer between last
  // convolution and NeuraNet output
  // Declare a variable to remember the index of the base
  long iBase = iStartBase[0];
  // Loop on the last output of convolution layer
  for (long iLayerOut = 0; iLayerOut < sizeLayerOut; ++iLayerOut) {
    // Loop on parallel convolution
    for (long iThick = 0; iThick < thickConv; ++iThick) {
      // Loop on output of the NeuraNet
      for (long iOut = 0; iOut < nbOutput; ++iOut) {
        // Declare a variable to memorize the index of the input of
        // the link
        long iInput = iStartLayerIn[0] +
          iLayerOut * thickConv + iThick;
        // Declare a variable to memorize the index of the output of
        // the link
        long iOutput = iOut + nbIn + nbHiddenVal;
        // Set the link's parameters
        VecSet(links, iLink * NN_NBPARAMLINK, iBase);
        VecSet(links, iLink * NN_NBPARAMLINK + 1, iInput);
        VecSet(links, iLink * NN_NBPARAMLINK + 2, iOutput);
        // Increment the link index
        ++iLink;
        // Increment the base function
        ++iBase;
      }
    }
  }
  // Set up the links
  NNSetLinks(nn, links);
  // Free memory
  VecFree(&links);
  VecFree(&pos);
  VecFree(&posCell);
  VecFree(&curDimIn);
  VecFree(&curDimOut);
  free(iStartBase);
  free(iStartLayerIn);
  free(iStartLayerOut);
  // Return the new NeuraNet
  return nn;
}

// Calculate the output values for the input values 'input' for the
// NeuraNet 'that' and memorize the result in 'output'
// input values in [-1,1] and output values in [-1,1]
// All values of 'output' are set to 0.0 before evaluating
// Links which refer to values out of bounds of 'input' or 'output'
// are ignored
void NNEval(const NeuraNet* const that, const VecFloat* const input, VecFloat* const output) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
```

```
  }
  if (input == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'input' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (output == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'output' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (VecGetDim(input) != that->_nbInputVal) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg,
      "'input' 's dimension is invalid (%ld!=%d)",
      VecGetDim(input), that->_nbInputVal);
    PBErrCatch(NeuraNetErr);
  }
  if (VecGetDim(output) != that->_nbOutputVal) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg,
      "'output' 's dimension is invalid (%ld!=%d)",
      VecGetDim(output), that->_nbOutputVal);
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Reset the hidden values and output
  if (NNGetNbMaxHidden(that) > 0)
    VecSetNull(that->_hidVal);
  VecSetNull(output);
  // If there are links in the network
  if (VecGet(that->_links, 0) != -1) {
    // Declare two variables to memorize the starting index of hidden
    // values and output values in the link definition
    long startHid = NNGetNbInput(that);
    long startOut = NNGetNbMaxHidden(that) + NNGetNbInput(that);
    // Declare a variable to memorize the previous link
    long prevLink[2] = {-1, -1};
    // Declare a variable to memorize the previous output value
    float prevOut = 1.0;
    // Loop on links
    long iLink = 0;
    while (iLink < NNGetNbMaxLinks(that) &&
      VecGet(that->_links, NN_NBPARAMLINK * iLink) != -1) {
      // Declare a variable for optimization
      long jLink = NN_NBPARAMLINK * iLink;
      // If this link has different input or output than previous link
      // and we are not on the first link
      if (iLink != 0 &&
        (VecGet(that->_links, jLink + 1) != prevLink[0] ||
        VecGet(that->_links, jLink + 2) != prevLink[1])) {
        // Add the previous output value to the output of the previous
        // link
        if (prevLink[1] < startOut) {
          long iVal = prevLink[1] - startHid;
          float nVal = MIN(1.0, MAX(-1.0, VecGet(that->_hidVal, iVal) + prevOut));
          VecSet(that->_hidVal, iVal, nVal);
        } else {
          long iVal = prevLink[1] - startOut;
          float nVal = VecGet(output, iVal) + prevOut;
          VecSet(output, iVal, nVal);
        }
```

```
        // Reset the previous output
        prevOut = 1.0;
      }
      // Update the previous link
      prevLink[0] = VecGet(that->_links, jLink + 1);
      prevLink[1] = VecGet(that->_links, jLink + 2);
      // Multiply the previous output by the evaluation of the current
      // link with the base function of the link and the normalised
      // input value
      float* param = that->_bases->_val +
        VecGet(that->_links, jLink) * NN_NBPARAMBASE;
      float x = 0.0;
      if (prevLink[0] < startHid)
        x = VecGet(input, prevLink[0]);
      else
        x = NNGetHiddenValue(that, prevLink[0] - startHid);
      prevOut *= NNBaseFun(param, x);
      // Move to the next link
      ++iLink;
    }
    // Update the output of the last link
    if (prevLink[1] < startOut) {
      long iVal = prevLink[1] - startHid;
      float nVal =
        MIN(1.0, MAX(-1.0, VecGet(that->_hidVal, iVal) + prevOut));
      VecSet(that->_hidVal, iVal, nVal);
    } else {
      long iVal = prevLink[1] - startOut;
      float nVal = VecGet(output, iVal) + prevOut;
      VecSet(output, iVal, nVal);
    }
  }
}

// Function which return the JSON encoding of 'that'
JSONNode* NNEncodeAsJSON(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'that' is null");
    PBErrCatch(PBMathErr);
  }
#endif
  // Create the JSON structure
  JSONNode* json = JSONCreate();
  // Declare a buffer to convert value into string
  char val[100];
  // Encode the nbInputVal
  sprintf(val, "%d", that->_nbInputVal);
  JSONAddProp(json, "_nbInputVal", val);
  // Encode the nbOutputVal
  sprintf(val, "%d", that->_nbOutputVal);
  JSONAddProp(json, "_nbOutputVal", val);
  // Encode the nbMaxHidVal
  sprintf(val, "%ld", that->_nbMaxHidVal);
  JSONAddProp(json, "_nbMaxHidVal", val);
  // Encode the nbMaxBases
  sprintf(val, "%ld", that->_nbMaxBases);
  JSONAddProp(json, "_nbMaxBases", val);
  // Encode the nbMaxLinks
  sprintf(val, "%ld", that->_nbMaxLinks);
  JSONAddProp(json, "_nbMaxLinks", val);
```

```
  // Encode the bases
  JSONAddProp(json, "_bases", VecEncodeAsJSON(that->_bases));
  // Encode the links
  JSONAddProp(json, "_links", VecEncodeAsJSON(that->_links));
  // Return the created JSON
  return json;
}

// Function which decode from JSON encoding 'json' to 'that'
bool NNDecodeAsJSON(NeuraNet** that, const JSONNode* const json) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'that' is null");
    PBErrCatch(PBMathErr);
  }
  if (json == NULL) {
    PBMathErr->_type = PBErrTypeNullPointer;
    sprintf(PBMathErr->_msg, "'json' is null");
    PBErrCatch(PBMathErr);
  }
#endif
  // If 'that' is already allocated
  if (*that != NULL)
    // Free memory
    NeuraNetFree(that);
  // Decode the nbInputVal
  JSONNode* prop = JSONProperty(json, "_nbInputVal");
  if (prop == NULL) {
    return false;
  }
  int nbInputVal = atoi(JSONLabel(JSONValue(prop, 0)));
  // Decode the nbOutputVal
  prop = JSONProperty(json, "_nbOutputVal");
  if (prop == NULL) {
    return false;
  }
  int nbOutputVal = atoi(JSONLabel(JSONValue(prop, 0)));
  // Decode the nbMaxHidVal
  prop = JSONProperty(json, "_nbMaxHidVal");
  if (prop == NULL) {
    return false;
  }
  long nbMaxHidVal = atol(JSONLabel(JSONValue(prop, 0)));
  // Decode the nbMaxBases
  prop = JSONProperty(json, "_nbMaxBases");
  if (prop == NULL) {
    return false;
  }
  long nbMaxBases = atol(JSONLabel(JSONValue(prop, 0)));
  // Decode the nbMaxLinks
  prop = JSONProperty(json, "_nbMaxLinks");
  if (prop == NULL) {
    return false;
  }
  long nbMaxLinks = atol(JSONLabel(JSONValue(prop, 0)));
  // Allocate memory
  *that = NeuraNetCreate(nbInputVal, nbOutputVal, nbMaxHidVal,
    nbMaxBases, nbMaxLinks);
  // Decode the bases
  prop = JSONProperty(json, "_bases");
  if (prop == NULL) {
```

```
    return false;
  }
  if (!VecDecodeAsJSON(&((*that)->_bases), prop)) {
    return false;
  }
  // Decode the links
  prop = JSONProperty(json, "_links");
  if (prop == NULL) {
    return false;
  }
  if (!VecDecodeAsJSON(&((*that)->_links), prop)) {
    return false;
  }
  // Return the success code
  return true;
}

// Save the NeuraNet 'that' to the stream 'stream'
// If 'compact' equals true it saves in compact form, else it saves in
// readable form
// Return true if the NeuraNet could be saved, false else
bool NNSave(const NeuraNet* const that, FILE* const stream, const bool compact) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (stream == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'stream' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Get the JSON encoding
  JSONNode* json = NNEncodeAsJSON(that);
  // Save the JSON
  if (!JSONSave(json, stream, compact)) {
    return false;
  }
  // Free memory
  JSONFree(&json);
  // Return success code
  return true;
}

// Load the NeuraNet 'that' from the stream 'stream'
// If 'that' is not null the memory is first freed
// Return true if the NeuraNet could be loaded, false else
bool NNLoad(NeuraNet** that, FILE* const stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (stream == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'stream' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
```

```
  // Declare a json to load the encoded data
  JSONNode* json = JSONCreate();
  // Load the whole encoded data
  if (!JSONLoad(json, stream)) {
    return false;
  }
  // Decode the data from the JSON
  if (!NNDecodeAsJSON(that, json)) {
    return false;
  }
  // Free the memory used by the JSON
  JSONFree(&json);
  // Return the success code
  return true;
}

// Print the NeuraNet 'that' to the stream 'stream'
void NNPrintln(const NeuraNet* const that, FILE* const stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (stream == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'stream' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  fprintf(stream, "nbInput: %d\n", that->_nbInputVal);
  fprintf(stream, "nbOutput: %d\n", that->_nbOutputVal);
  fprintf(stream, "nbHidden: %ld\n", that->_nbMaxHidVal);
  fprintf(stream, "nbMaxBases: %ld\n", that->_nbMaxBases);
  fprintf(stream, "nbMaxLinks: %ld\n", that->_nbMaxLinks);
  fprintf(stream, "bases: ");
  VecPrint(that->_bases, stream);
  fprintf(stream, "\n");
  fprintf(stream, "links: ");
  VecPrint(that->_links, stream);
  fprintf(stream, "\n");
  fprintf(stream, "hidden values: ");
  VecPrint(that->_hidVal, stream);
  fprintf(stream, "\n");
}

// Set the links description of the NeuraNet 'that' to a copy of 'links'
// Links with a base function equals to -1 are ignored
// If the input id is higher than the output id they are swap
// The links description in the NeuraNet are ordered in increasing
// value of input id and output id, but 'links' doesn't have to be
// sorted
// Each link is defined by (base index, input index, output index)
// If base index equals -1 it means the link is inactive
void NNSetLinks(NeuraNet* const that, VecLong* const links) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (links == NULL) {
```

```
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'links' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (VecGetDim(links) != that->_nbMaxLinks * NN_NBPARAMLINK) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg,
      "'links' 's dimension is invalid (%ld!=%ld)",
      VecGetDim(links), that->_nbMaxLinks);
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a GSet to sort the links
  GSet set = GSetCreateStatic();
  // Declare a variable to memorize the maximum id
  long maxId = NNGetNbInput(that) + NNGetNbMaxHidden(that) +
    NNGetNbOutput(that);
  // Loop on links
  for (long iLink = 0; iLink < NNGetNbMaxLinks(that) * NN_NBPARAMLINK;
    iLink += NN_NBPARAMLINK) {
    // If this link is active
    if (VecGet(links, iLink) != -1) {
      // Declare two variables to memorize the effective input and output
      long in = VecGet(links, iLink + 1);
      long out = VecGet(links, iLink + 2);
      // If the input is greater than the output
      if (in > out) {
        // Swap the input and output
        swap(in, out);
      }
      // Add the link to the set, sorting on input and ouput
      float sortVal = (float)(in * maxId + out);
      GSetAddSort(&set, links->_val + iLink, sortVal);
    }
  }
  // Declare a variable to memorize the number of active links
  long nbLink = GSetNbElem(&set);
  // If there are active links
  if (nbLink > 0) {
    // loop on active sorted links
    GSetIterForward iter = GSetIterForwardCreateStatic(&set);
    long iLink = 0;
    do {
      long *link = GSetIterGet(&iter);
      VecSet(that->_links, iLink * NN_NBPARAMLINK, link[0]);
      if (link[1] <= link[2]) {
        VecSet(that->_links, iLink * NN_NBPARAMLINK + 1, link[1]);
        VecSet(that->_links, iLink * NN_NBPARAMLINK + 2, link[2]);
      } else {
        VecSet(that->_links, iLink * NN_NBPARAMLINK + 1, link[2]);
        VecSet(that->_links, iLink * NN_NBPARAMLINK + 2, link[1]);
      }
      ++iLink;
    } while (GSetIterStep(&iter));
  }
  // Reset the inactive links
  for (long iLink = nbLink; iLink < NNGetNbMaxLinks(that); ++iLink)
    VecSet(that->_links, iLink * NN_NBPARAMLINK, -1);
  // Free the memory
  GSetFlush(&set);
}
```

```
// Save the links of the NeuraNet 'that' into the file at 'url' in a
// format readable by CloudGraph
// Return true if we could save, else false
bool NNSaveLinkAsCloudGraph(const NeuraNet* const that,
  const char* url) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (url == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'url' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a flag to memorize the returned value
  bool ret = true;
  // Open the file
  FILE* cloud = fopen(url, "w");
  // If we could open the file
  if (cloud != NULL) {
    // Save the categories
    if (!PBErrPrintf(NeuraNetErr, cloud, "%s", "3\n")) {
      fclose(cloud);
      return false;
    }
    if (!PBErrPrintf(NeuraNetErr, cloud, "%s", "0 255 0 0 Input\n")) {
      fclose(cloud);
      return false;
    }
    if (!PBErrPrintf(NeuraNetErr, cloud, "%s", "1 0 255 0 Hidden\n")) {
      fclose(cloud);
      return false;
    }
    if (!PBErrPrintf(NeuraNetErr, cloud, "%s", "2 0 0 255 Output\n")) {
      fclose(cloud);
      return false;
    }
    // Save the nb of nodes
    if (!PBErrPrintf(NeuraNetErr, cloud, "%ld\n",
      NNGetNbInput(that) + NNGetNbMaxHidden(that) +
      NNGetNbOutput(that))) {
      fclose(cloud);
      return false;
    }
    // Save the input nodes
    for (long iNode = 0; iNode < NNGetNbInput(that); ++iNode) {
      if (!PBErrPrintf(NeuraNetErr, cloud, "%ld 0 ", iNode)) {
        fclose(cloud);
        return false;
      }
      if (!PBErrPrintf(NeuraNetErr, cloud, "%ld\n", iNode)) {
        fclose(cloud);
        return false;
      }
    }
    // Save the hidden nodes
    for (long iNode = 0; iNode < NNGetNbMaxHidden(that); ++iNode) {
      long jNode = iNode + NNGetNbInput(that);
      if (!PBErrPrintf(NeuraNetErr, cloud, "%ld 1 ", jNode)) {
```

```
        fclose(cloud);
        return false;
      }
      if (!PBErrPrintf(NeuraNetErr, cloud, "%ld\n", jNode)) {
        fclose(cloud);
        return false;
      }
    }
    // Save the output nodes
    for (long iNode = 0; iNode < NNGetNbOutput(that); ++iNode) {
      long jNode = iNode + NNGetNbInput(that) + NNGetNbMaxHidden(that);
      if (!PBErrPrintf(NeuraNetErr, cloud, "%ld 2 ", jNode)) {
        fclose(cloud);
        return false;
      }
      if (!PBErrPrintf(NeuraNetErr, cloud, "%ld\n", jNode)) {
        fclose(cloud);
        return false;
      }
    }
    // Save the links
    PBErrPrintf(NeuraNetErr, cloud, "%ld\n", NNGetNbActiveLinks(that));
    for (long iLink = 0; iLink < NNGetNbMaxLinks(that); ++iLink) {
      // If this link is active
      if (VecGet(NNLinks(that), iLink * NN_NBPARAMLINK) != -1) {
        if (!PBErrPrintf(NeuraNetErr, cloud, "%ld ",
          VecGet(NNLinks(that), iLink * NN_NBPARAMLINK + 1))) {
          fclose(cloud);
          return false;
        }
        if (!PBErrPrintf(NeuraNetErr, cloud, "%ld\n",
          VecGet(NNLinks(that), iLink * NN_NBPARAMLINK + 2))) {
          fclose(cloud);
          return false;
        }
      }
    }
    // Close the file
    fclose(cloud);
  // Else, we coudln't open the file
  } else {
    // Set the flag
    ret = false;
  }
  // Return the flag
  return ret;
}

// Get the Simpson's diversity index of the hidden values of the
// NeuraNet 'that'
// Return value in [0.0, 1.0], 0.0 means no diversity, 1.0 means max
// diversity
float NNGetHiddenValSimpsonDiv(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a variable to emmorize the result
  float div = 0.0;
```

```
  // Declare a variable to check if a link output to a hidden value
  long iMaxHidden = NNGetNbInput(that) + NNGetNbMaxHidden(that);
  // Declare two variables for calculation
  VecFloat* nb = VecFloatCreate(iMaxHidden);
  float tot = 0.0;
  // Loop on links
  for (int iLink = NNGetNbMaxLinks(that); iLink--;) {
    // If this link is active and its ouput is a hidden value
    if (VecGet(NNLinks(that), iLink * NN_NBPARAMLINK) != -1 &&
      VecGet(NNLinks(that), iLink * NN_NBPARAMLINK + 2) < iMaxHidden) {
      // Calculate the diversity
      ++tot;
      VecSetAdd(nb, VecGet(NNLinks(that),
        iLink * NN_NBPARAMLINK + 1), 1.0);
    }
  }
  // Calculate the diversity
  if (tot >= 2.0) {
    for (int iLink = iMaxHidden; iLink--;) {
      div += VecGet(nb, iLink) * (VecGet(nb, iLink) - 1.0);
    }
    div = 1.0 - div / (tot * (tot - 1.0));
    div *= tot / (float)NNGetNbMaxLinks(that);
  }
  // Free memory
  VecFree(&nb);
  // Return the diversity
  return MIN(1.0, MAX(-1.0, div));
}

// Prune the NeuraNet 'that' by removing the useless links (those with
// no influence on outputs
void NNPrune(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a variable to memorise the start index of output
  long startOut = NNGetNbInput(that) + NNGetNbMaxHidden(that);
  // Loop on links
  for (int iLink = NNGetNbMaxLinks(that); iLink--;) {
    // If the output of this link is not an output and it's active
    if (VecGet(NNLinks(that), iLink * NN_NBPARAMLINK) != -1 &&
      VecGet(NNLinks(that), iLink * NN_NBPARAMLINK + 2) < startOut) {
      // Search in following links one that use the output of this link
      // as an input
      bool flag = false;
      for (int jLink = iLink + 1;
        !flag && jLink < NNGetNbMaxLinks(that); ++jLink) {
        if (VecGet(NNLinks(that), iLink * NN_NBPARAMLINK + 2) ==
          VecGet(NNLinks(that), jLink * NN_NBPARAMLINK + 1)) {
          flag = true;
        }
      }
      // If the output of this link is never used
      if (!flag)
        // Disactivate it
        VecSet(that->_links, iLink * NN_NBPARAMLINK, -1);
    }
```

```
  }
}

// Helper functions to propagate recursively the accuracy in the nework
void NNPropagateAccuracyLink(const NeuraNet* that, const long out,
  float accuracy, VecFloat* mutability) {
  // Loop on links
  for (int iLink = NNGetNbMaxLinks(that); iLink--;) {
    // If this link is active and its output is the current output
    if (VecGet(that->_links, iLink * NN_NBPARAMLINK) != -1 &&
      VecGet(that->_links, iLink * NN_NBPARAMLINK + 2) == out) {
      // Update the mutability
      for (int iParam = NN_NBPARAMLINK; iParam--;)
        VecSetAdd(mutability, iLink * NN_NBPARAMLINK + iParam,
          (1.0 - accuracy) * (1.0 -
          VecGet(mutability, iLink * NN_NBPARAMLINK + iParam)));
      // Keep on propagating from the input of this link
      NNPropagateAccuracyLink(that,
        VecGet(that->_links, iLink * NN_NBPARAMLINK + 1), out, mutability);
    }
  }
}
void NNPropagateAccuracyBase(const NeuraNet* that, const long out,
  float accuracy, VecFloat* mutability) {
  // Loop on links
  for (int iLink = NNGetNbMaxLinks(that); iLink--;) {
    // If this link is active and its output is the current output
    if (VecGet(that->_links, iLink * NN_NBPARAMLINK) != -1 &&
      VecGet(that->_links, iLink * NN_NBPARAMLINK + 2) == out) {
      // Update the mutability
      long iBase = VecGet(that->_links, iLink * NN_NBPARAMLINK);
      for (int iParam = NN_NBPARAMBASE; iParam--;)
        VecSetAdd(mutability, iBase * NN_NBPARAMBASE + iParam,
          (1.0 - accuracy) * (1.0 -
          VecGet(mutability, iLink * NN_NBPARAMBASE + iParam)));
      // Keep on propagating from the input of this link
      NNPropagateAccuracyBase(that,
        VecGet(that->_links, iLink * NN_NBPARAMLINK + 1), out, mutability);
    }
  }
}

// Get the mutability vector for bases of the NeuraNet 'that' according
// to output's 'accuracy'
// accuracy is a VecFloat of dimension nbOutput, where accuracy[iOut]
// equals 1.0 if the NeuraNet always returns the perfect answer for the
// output iOut, and 0.0 if never
// Return a VecFloat of dimension nbBase * NB_PARAMBASE with values in
// [0.0, 1.0]
VecFloat* NNGetMutabilityBases(const NeuraNet* const that,
  const VecFloat* const accuracy) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (accuracy == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'accuracy' is null");
    PBErrCatch(NeuraNetErr);
  }
```

```
  if (VecGetDim(accuracy) != NNGetNbOutput(that)) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'accuracy''s dim is invalid (%ld==%d)",
      VecGetDim(accuracy), NNGetNbOutput(that));
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Create the mutability vector
  VecFloat* ret = VecFloatCreate(NNGetNbMaxBases(that) * NN_NBPARAMBASE);
  // Loop on output
  for (int iOut = NNGetNbOutput(that); iOut--;) {
    // Propagate the accuracy in the nework from the output
    NNPropagateAccuracyBase(that, iOut, VecGet(accuracy, iOut), ret);
  }
  // Scale to have the highest mutability to one
  float maxVal = VecGetMaxVal(ret);
  if (maxVal > PBMATH_EPSILON)
    VecScale(ret, 1.0 / maxVal);
  // Return the mutability vector
  return ret;
}

// Get the mutability vector for links of the NeuraNet 'that' according
// to output's 'accuracy'
// accuracy is a VecFloat of dimension nbOutput, where accuracy[iOut]
// equals 1.0 if the NeuraNet always returns the perfect answer for the
// output iOut, and 0.0 if never
// Return a VecFloat of dimension nbLink * NB_PARAMLINK with values in
// [0.0, 1.0]
VecFloat* NNGetMutabilityLinks(const NeuraNet* const that,
  const VecFloat* const accuracy) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (accuracy == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'accuracy' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (VecGetDim(accuracy) != NNGetNbOutput(that)) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'accuracy''s dim is invalid (%ld==%d)",
      VecGetDim(accuracy), NNGetNbOutput(that));
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Create the mutability vector
  VecFloat* ret = VecFloatCreate(NNGetNbMaxLinks(that) * NN_NBPARAMLINK);
  // Loop on output
  for (int iOut = NNGetNbOutput(that); iOut--;) {
    // Propagate the accuracy in the nework from the output
    NNPropagateAccuracyLink(that, iOut, VecGet(accuracy, iOut), ret);
  }
  // Scale to have the highest mutability to one
  float maxVal = VecGetMaxVal(ret);
  if (maxVal > PBMATH_EPSILON)
    VecScale(ret, 1.0 / maxVal);
  // Return the mutability vector
  return ret;
```

```
}
```

3.2 neuranet-inline.c

```
// ============ NEURANET-INLINE.C ================

// ----- NeuraNetBaseFun

// ================ Functions implementation ====================

// Generic base function for the NeuraNet
// 'param' is an array of 3 float all in [-1,1]
// 'x' is the input value
// NNBaseFun(param,x)=
// tan(param[0]*NN_THETA)*(x+param[1])+param[2]
#if BUILDMODE != 0
inline
#endif
float NNBaseFun(const float* const param, const float x) {
#if BUILDMODE == 0
  if (param == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'param' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return tan(param[0] * NN_THETA) * (x + param[1]) + param[2];
}

// ----- NeuraNet

// ================ Functions implementation ====================

// Get the nb of input values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
int NNGetNbInput(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_nbInputVal;
}

// Get the nb of output values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
int NNGetNbOutput(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
```

```
#endif
  return that->_nbOutputVal;
}

// Get the nb max of hidden values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbMaxHidden(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_nbMaxHidVal;
}

// Get the nb max of base functions of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbMaxBases(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_nbMaxBases;
}

// Get the nb of base functions for convolution of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbBasesConv(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_nbBasesConv;
}

// Get the nb of base functions per cell for convolution of
// the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbBasesCellConv(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
```

```
  return that->_nbBasesCellConv;
}

// Get the nb max of links of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbMaxLinks(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_nbMaxLinks;
}

// Get the parameters of the base functions of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* NNBases(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_bases;
}

// Get the links description of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
const VecLong* NNLinks(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_links;
}

// Get the hidden values of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
const VecFloat* NNHiddenValues(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  return that->_hidVal;
}
```

```
// Get the 'iVal'-th hidden value of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
float NNGetHiddenValue(const NeuraNet* const that, const long iVal) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (iVal < 0 || iVal >= that->_nbMaxHidVal) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'iVal' is invalid (0<=%ld<%ld)",
      iVal, that->_nbMaxHidVal);
    PBErrCatch(NeuraNetErr);
  }
#endif
  return VecGet(that->_hidVal, iVal);
}

// Set the parameters of the base functions of the NeuraNet 'that' to
// a copy of 'bases'
// 'bases' must be of dimension that->nbMaxBases * NN_NBPARAMBASE
//  each base is defined as param[3] in [-1,1]
// tan(param[0]*NN_THETA)*(x+param[1])+param[2]
#if BUILDMODE != 0
inline
#endif
void NNSetBases(NeuraNet* const that, const VecFloat* const bases) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (bases == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'bases' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (VecGetDim(bases) != that->_nbMaxBases * NN_NBPARAMBASE) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg,
      "'bases' 's dimension is invalid (%ld!=%ld)",
      VecGetDim(bases), that->_nbMaxBases * NN_NBPARAMBASE);
    PBErrCatch(NeuraNetErr);
  }
#endif
  VecCopy(that->_bases, bases);
}
// Set the 'iBase'-th parameter of the base functions of the NeuraNet
// 'that' to 'base'
#if BUILDMODE != 0
inline
#endif
void NNBasesSet(NeuraNet* const that, const long iBase,
  const float base) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
```

```
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
  if (iBase < 0 || iBase >= that->_nbMaxBases * NN_NBPARAMBASE) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg,
      "'iBase' is invalid (0<=%ld<%ld)",
      iBase, that->_nbMaxBases * NN_NBPARAMBASE);
    PBErrCatch(NeuraNetErr);
  }
#endif
  VecSet(that->_bases, iBase, base);
}

// Get the number of active links in the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
long NNGetNbActiveLinks(const NeuraNet* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    NeuraNetErr->_type = PBErrTypeNullPointer;
    sprintf(NeuraNetErr->_msg, "'that' is null");
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a variable to memorize the result
  long nb = 0;
  // Loop on links
  for (long iLink = NNGetNbMaxLinks(that); iLink--;) {
    // If this link is active
    if (VecGet(NNLinks(that), iLink * NN_NBPARAMLINK) != -1)
      // Increment the number of active links
      ++nb;
  }
  // Return the result
  return nb;
}
```

4 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=1

all: pbmake_wget main nn2cloud

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)
```

```
# Rules to make the executable
repo=neuranet
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) ‘echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ’ ’ ’\n’ | sort -u‘ $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) ‘echo "$($(repo)_INC_DIR)" | tr ’ ’ ’\n’ | sort -u‘ -c $($(repo)_DIR)/$

nn2cloud: \
nn2cloud.o \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) nn2cloud.o ‘echo "$($(repo)_EXE_DEP)" | tr ’ ’ ’\n’ | sort -u‘ $(LINK_ARG) $($(repo)_LINK_ARG) -o nn2clou

nn2cloud.o: \
nn2cloud.c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) ‘echo "$($(repo)_INC_DIR)" | tr ’ ’ ’\n’ | sort -u‘ -c $($(repo)_DIR)/n

cloud: cloud.txt
../CloudGraph/cloudGraph -file ./cloud.txt -tga cloud.tga -familyLabel -circle -curved 0.5
```

5 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "genalg.h"
#include "neuranet.h"

#define RANDOMSEED 4

void UnitTestNNBaseFun() {
  srandom(RANDOMSEED);
  float param[4];
  float x = 0.0;
  float check[100] = {
    -4.664967,-3.920526,-3.176085,-2.431644,-1.687203,-0.942763,
    -0.198322,0.546119,1.290560,2.035000,-0.153181,-0.403978,
    -0.654776,-0.905573,-1.156371,-1.407168,-1.657966,-1.908763,
    -2.159561,-2.410358,0.586943,0.301165,0.015387,-0.270391,
    -0.556169,-0.841946,-1.127724,-1.413502,-1.699280,-1.985057,
    2.760699,2.805863,2.851027,2.896191,2.941355,2.986519,
    3.031683,3.076847,3.122011,3.167175,0.774302,0.903425,
    1.032548,1.161672,1.290795,1.419918,1.549042,1.678165,
    1.807288,1.936412,2.321817,2.100005,1.878192,1.656379,
```

```
    1.434567,1.212754,0.990941,0.769129,0.547316,0.325503,
    -1.349660,-1.452492,-1.555323,-1.658154,-1.760985,-1.863817,
    -1.966648,-2.069479,-2.172311,-2.275142,2.030713,1.867117,
    1.703522,1.539926,1.376330,1.212735,1.049139,0.885544,0.721949,
    0.558353,-1.439830,-1.174441,-0.909051,-0.643662,-0.378272,
    -0.112883,0.152507,0.417896,0.683286,0.948675,0.819425,0.765620,
    0.711816,0.658011,0.604206,0.550401,0.496596,0.442791,0.388987,
    0.335182
    };
  for (int iTest = 0; iTest < 10; ++iTest) {
    param[0] = 2.0 * (rnd() - 0.5);
    param[1] = 2.0 * rnd();
    param[2] = 2.0 * (rnd() - 0.5) * PBMATH_PI;
    param[3] = 2.0 * (rnd() - 0.5);
    for (int ix = 0; ix < 10; ++ix) {
      x = -1.0 + 2.0 * 0.1 * (float)ix;
      float y = NNBaseFun(param, x);
      if (ISEQUALF(y, check[iTest * 10 + ix]) == false) {
        NeuraNetErr->_type = PBErrTypeUnitTestFailed;
        sprintf(NeuraNetErr->_msg, "NNBaseFun failed");
        PBErrCatch(NeuraNetErr);
      }
    }
  }
  printf("UnitTestNNBaseFun OK\n");
}

void UnitTestNeuraNetCreateFree() {
  int nbIn = 1;
  int nbOut = 2;
  int nbHid = 3;
  int nbBase = 4;
  int nbLink = 5;
  NeuraNet* nn = NeuraNetCreate(nbIn, nbOut, nbHid, nbBase, nbLink);
  if (nn == NULL ||
    nn->_nbInputVal != nbIn ||
    nn->_nbOutputVal != nbOut ||
    nn->_nbMaxHidVal != nbHid ||
    nn->_nbMaxBases != nbBase ||
    nn->_nbBasesConv != 0 ||
    nn->_nbMaxLinks != nbLink ||
    nn->_bases == NULL ||
    nn->_links == NULL ||
    nn->_hidVal == NULL) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NeuraNetFree failed");
    PBErrCatch(NeuraNetErr);
  }
  NeuraNetFree(&nn);
  if (nn != NULL) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NeuraNetFree failed");
    PBErrCatch(NeuraNetErr);
  }
  printf("UnitTestNeuraNetCreateFree OK\n");
}

void UnitTestNeuraNetCreateFullyConnected() {
  int nbIn = 2;
  int nbOut = 3;
  VecLong* hiddenLayers = NULL;
  NeuraNet* nn = NeuraNetCreateFullyConnected(nbIn, nbOut, hiddenLayers);
```

```
if (nn == NULL ||
  nn->_nbInputVal != nbIn ||
  nn->_nbOutputVal != nbOut ||
  nn->_nbMaxHidVal != 0 ||
  nn->_nbMaxBases != 6 ||
  nn->_nbMaxLinks != 6 ||
  nn->_bases == NULL ||
  nn->_links == NULL ||
  nn->_hidVal != NULL) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NeuraNetCreateFullyConnected failed");
  PBErrCatch(NeuraNetErr);
}
int checka[18] = {
  0,0,2, 1,0,3, 2,0,4,
  3,1,2, 4,1,3, 5,1,4
};
for (int i = 18; i--;)
  if (VecGet(nn->_links, i) != checka[i]) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NeuraNetCreateFullyConnected failed");
    PBErrCatch(NeuraNetErr);
  }
NeuraNetFree(&nn);
nbIn = 5;
nbOut = 2;
hiddenLayers = VecLongCreate(2);
VecSet(hiddenLayers, 0, 4);
VecSet(hiddenLayers, 1, 3);
nn = NeuraNetCreateFullyConnected(nbIn, nbOut, hiddenLayers);
if (nn == NULL ||
  nn->_nbInputVal != nbIn ||
  nn->_nbOutputVal != nbOut ||
  nn->_nbMaxHidVal != 7 ||
  nn->_nbMaxBases != 38 ||
  nn->_nbMaxLinks != 38 ||
  nn->_bases == NULL ||
  nn->_links == NULL ||
  nn->_hidVal == NULL) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NeuraNetCreateFullyConnected failed");
  PBErrCatch(NeuraNetErr);
}
int checkb[114] = {
  0,0,5, 1,0,6, 2,0,7, 3,0,8,
  4,1,5, 5,1,6, 6,1,7, 7,1,8,
  8,2,5, 9,2,6, 10,2,7, 11,2,8,
  12,3,5, 13,3,6, 14,3,7, 15,3,8,
  16,4,5, 17,4,6, 18,4,7, 19,4,8,
  20,5,9, 21,5,10, 22,5,11,
  23,6,9, 24,6,10, 25,6,11,
  26,7,9, 27,7,10, 28,7,11,
  29,8,9, 30,8,10, 31,8,11,
  32,9,12, 33,9,13,
  34,10,12, 35,10,13,
  36,11,12, 37,11,13
};
for (int i = 114; i--;)
  if (VecGet(nn->_links, i) != checkb[i]) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NeuraNetCreateFullyConnected failed");
    PBErrCatch(NeuraNetErr);
```

```
    }
  NeuraNetFree(&nn);
  VecFree(&hiddenLayers);
  printf("UnitTestNeuraNetCreateFullyConnected OK\n");
}

void UnitTestNeuraNetCreateConvolution() {
  int nbOut = 2;
  int thickConv = 2;
  int depthConv = 2;
  VecShort* dimIn = VecShortCreate(2);
  VecSet(dimIn, 0, 4);
  VecSet(dimIn, 1, 3);
  VecShort* dimCell = VecShortCreate(2);
  VecSet(dimCell, 0, 2);
  VecSet(dimCell, 1, 2);
  NeuraNet* nn = NeuraNetCreateConvolution(dimIn, nbOut, dimCell,
    depthConv, thickConv);
  NNPrintln(nn, stdout);
  if (nn == NULL ||
    nn->_nbInputVal != 12 ||
    nn->_nbOutputVal != 2 ||
    nn->_nbMaxHidVal != 16 ||
    nn->_nbMaxBases != 24 ||
    nn->_nbMaxLinks != 72 ||
    nn->_bases == NULL ||
    nn->_links == NULL ||
    nn->_hidVal == NULL) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NeuraNetCreateConvolution failed");
    PBErrCatch(NeuraNetErr);
  }
  int check[216] = {
    0,0,12, 4,0,18, 1,1,12, 0,1,13, 5,1,18, 4,1,19, 1,2,13, 0,2,14,
    5,2,19, 4,2,20, 1,3,14, 5,3,20, 2,4,12, 0,4,15, 6,4,18, 4,4,21,
    3,5,12, 2,5,13, 1,5,15, 0,5,16, 7,5,18, 6,5,19, 5,5,21, 4,5,22,
    3,6,13, 2,6,14, 1,6,16, 0,6,17, 7,6,19, 6,6,20, 5,6,22, 4,6,23,
    3,7,14, 1,7,17, 7,7,20, 5,7,23, 2,8,15, 6,8,21, 3,9,15, 2,9,16,
    7,9,21, 6,9,22, 3,10,16, 2,10,17, 7,10,22, 6,10,23, 3,11,17,
    7,11,23, 8,12,24, 9,13,24, 8,13,25, 9,14,25, 10,15,24, 11,16,24,
    10,16,25, 11,17,25, 12,18,26, 13,19,26, 12,19,27, 13,20,27,
    14,21,26, 15,22,26, 14,22,27, 15,23,27, 16,24,28, 17,24,29,
    18,25,28, 19,25,29, 20,26,28, 21,26,29, 22,27,28, 23,27,29
    };
  for (int iCheck = 216; iCheck--;) {
    if (VecGet(nn->_links, iCheck) != check[iCheck]) {
      NeuraNetErr->_type = PBErrTypeUnitTestFailed;
      sprintf(NeuraNetErr->_msg, "NeuraNetCreateConvolution failed");
      PBErrCatch(NeuraNetErr);
    }
  }
  NNSaveLinkAsCloudGraph(nn, "./cloudConv.txt");
  NeuraNetFree(&nn);
  VecFree(&dimIn);
  VecFree(&dimCell);
  printf("UnitTestNeuraNetCreateConvolution OK\n");
}

void UnitTestNeuraNetGetSet() {
  int nbIn = 10;
  int nbOut = 20;
  int nbHid = 30;
```

```
int nbBase = 4;
int nbLink = 5;
NeuraNet* nn = NeuraNetCreate(nbIn, nbOut, nbHid, nbBase, nbLink);
if (NNGetNbInput(nn) != nbIn) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNGetNbInput failed");
  PBErrCatch(NeuraNetErr);
}
if (NNGetNbMaxBases(nn) != nbBase) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNGetNbMaxBases failed");
  PBErrCatch(NeuraNetErr);
}
if (NNGetNbBasesConv(nn) != 0) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNGetNbBasesConv failed");
  PBErrCatch(NeuraNetErr);
}
if (NNGetNbBasesCellConv(nn) != 0) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNGetNbBasesCellConv failed");
  PBErrCatch(NeuraNetErr);
}
if (NNGetNbMaxHidden(nn) != nbHid) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNGetNbMaxHidden failed");
  PBErrCatch(NeuraNetErr);
}
if (NNGetNbMaxLinks(nn) != nbLink) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNGetNbMaxLinks failed");
  PBErrCatch(NeuraNetErr);
}
if (NNGetNbOutput(nn) != nbOut) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNGetNbOutput failed");
  PBErrCatch(NeuraNetErr);
}
if (NNBases(nn) != nn->_bases) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNBases failed");
  PBErrCatch(NeuraNetErr);
}
if (NNLinks(nn) != nn->_links) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNLinks failed");
  PBErrCatch(NeuraNetErr);
}
if (NNHiddenValues(nn) != nn->_hidVal) {
  NeuraNetErr->_type = PBErrTypeUnitTestFailed;
  sprintf(NeuraNetErr->_msg, "NNHiddenValues failed");
  PBErrCatch(NeuraNetErr);
}
VecFloat* bases = VecFloatCreate(nbBase * NN_NBPARAMBASE);
for (int i = nbBase * NN_NBPARAMBASE; i--;)
  VecSet(bases, i, 0.01 * (float)i);
NNSetBases(nn, bases);
for (int i = nbBase * NN_NBPARAMBASE; i--;)
  if (ISEQUALF(VecGet(NNBases(nn), i), 0.01 * (float)i) == false) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NNSetBases failed");
    PBErrCatch(NeuraNetErr);
```

```
    }
  VecFree(&bases);
  VecLong* links = VecLongCreate(15);
  short data[15] = {2,2,35, 1,1,12, -1,0,0, 2,15,20, 3,20,15};
  for (int i = 15; i--;)
    VecSet(links, i, data[i]);
  NNSetLinks(nn, links);
  short check[15] = {1,1,12,2,2,35,2,15,20,3,15,20,-1,0,0};
  for (int i = 15; i--;)
    if (VecGet(NNLinks(nn), i) != check[i]) {
      NeuraNetErr->_type = PBErrTypeUnitTestFailed;
      sprintf(NeuraNetErr->_msg, "NNSetLinks failed");
      PBErrCatch(NeuraNetErr);
    }
  if (NNGetNbActiveLinks(nn) != 4) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NNGetNbActiveLinks failed");
    PBErrCatch(NeuraNetErr);
  }
  VecFree(&links);
  NeuraNetFree(&nn);
  printf("UnitTestNeuraNetGetSet OK\n");
}

void UnitTestNeuraNetSaveLoadPrune() {
  int nbIn = 10;
  int nbOut = 20;
  int nbHid = 30;
  int nbBase = 4;
  int nbLink = 5;
  NeuraNet* nn = NeuraNetCreate(nbIn, nbOut, nbHid, nbBase, nbLink);
  VecFloat* bases = VecFloatCreate(nbBase * NN_NBPARAMBASE);
  for (int i = nbBase * NN_NBPARAMBASE; i--;)
    VecSet(bases, i, 0.01 * (float)i);
  NNSetBases(nn, bases);
  VecFree(&bases);
  VecLong* links = VecLongCreate(15);
  short data[15] = {2,2,35, 1,1,12, -1,0,0, 2,15,20, 3,20,15};
  for (int i = 15; i--;)
    VecSet(links, i, data[i]);
  NNSetLinks(nn, links);
  VecFree(&links);
  FILE* fd = fopen("./neuranet.txt", "w");
  if (NNSave(nn, fd, false) == false) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NNSave failed");
    PBErrCatch(NeuraNetErr);
  }
  fclose(fd);
  fd = fopen("./neuranet.txt", "r");
  NeuraNet* loaded = NeuraNetCreate(1, 1, 1, 1, 1);
  if (NNLoad(&loaded, fd) == false) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NNLoad failed");
    PBErrCatch(NeuraNetErr);
  }
  if (NNGetNbInput(loaded) != nbIn ||
    NNGetNbMaxBases(loaded) != nbBase ||
    NNGetNbMaxHidden(loaded) != nbHid ||
    NNGetNbMaxLinks(loaded) != nbLink ||
    NNGetNbOutput(loaded) != nbOut) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
```

```
    sprintf(NeuraNetErr->_msg, "NNLoad failed");
    PBErrCatch(NeuraNetErr);
  }
  for (int i = nbBase * NN_NBPARAMBASE; i--;)
    if (ISEQUALF(VecGet(NNBases(loaded), i), 0.01 * (float)i) == false) {
      NeuraNetErr->_type = PBErrTypeUnitTestFailed;
      sprintf(NeuraNetErr->_msg, "NNLoad failed");
      PBErrCatch(NeuraNetErr);
    }
  short check[15] = {1,1,12,2,2,35,2,15,20,3,15,20,-1,0,0};
  for (int i = 15; i--;)
    if (VecGet(NNLinks(loaded), i) != check[i]) {
      NeuraNetErr->_type = PBErrTypeUnitTestFailed;
      sprintf(NeuraNetErr->_msg, "NNLoad failed");
      PBErrCatch(NeuraNetErr);
    }
  fclose(fd);
  NeuraNetFree(&loaded);
  NNPrune(nn);
  short checkprune[15] = {-1,1,12,-1,2,35,-1,15,20,-1,15,20,-1,0,0};
  for (int i = 15; i--;)
    if (VecGet(NNLinks(nn), i) != checkprune[i]) {
      NeuraNetErr->_type = PBErrTypeUnitTestFailed;
      sprintf(NeuraNetErr->_msg, "NNPrune failed");
      PBErrCatch(NeuraNetErr);
    }
  NeuraNetFree(&nn);
  printf("UnitTestNeuraNetSaveLoadPrune OK\n");
}

void UnitTestNeuraNetEvalPrintHiddenLinkSimpsonDiv() {
  int nbIn = 3;
  int nbOut = 3;
  int nbHid = 3;
  int nbBase = 3;
  int nbLink = 7;
  NeuraNet* nn = NeuraNetCreate(nbIn, nbOut, nbHid, nbBase, nbLink);
  // hidden[0] = tan(0.5*NN_THETA)*tan(-0.5*NN_THETA)*input[0]^2
  // hidden[1] = tan(0.5*NN_THETA)*input[1]
  // hidden[2] = 0
  // output[0] = tan(0.5*NN_THETA)*hidden[0]+tan(0.5*NN_THETA)*hidden[1]
  // output[1] = tan(0.5*NN_THETA)*hidden[1]
  // output[2] = 0
  NNBasesSet(nn, 0, 0.5);
  NNBasesSet(nn, 3, -0.5);
  NNBasesSet(nn, 8, -0.5);
  short data[21] = {0,0,3, 1,0,3, 0,1,4, 0,3,6, 0,4,6, 0,4,7, -1,0,0};
  VecLong *links = VecLongCreate(21);
  for (int i = 21; i--;)
    VecSet(links, i, data[i]);
  NNSetLinks(nn, links);
  VecFree(&links);
  VecFloat3D input = VecFloatCreateStatic3D();
  VecFloat3D output = VecFloatCreateStatic3D();
  VecFloat3D check = VecFloatCreateStatic3D();
  VecFloat3D checkhidden = VecFloatCreateStatic3D();
  NNPrintln(nn, stdout);
  for (int i = -10; i <= 10; ++i) {
    for (int j = -10; j <= 10; ++j) {
      for (int k = -10; k <= 10; ++k) {
        VecSet(&input, 0, 0.1 * (float)i);
        VecSet(&input, 1, 0.1 * (float)j);
```

```
        VecSet(&input, 2, 0.1 * (float)k);
        NNEval(nn, (VecFloat*)&input, (VecFloat*)&output);
        VecSet(&checkhidden, 0, tan(0.5 * NN_THETA) * tan(-0.5 * NN_THETA) * fsquare(VecGet(&input, 0)));
        VecSet(&checkhidden, 1, tan(0.5 * NN_THETA) * VecGet(&input, 1));
        VecSet(&check, 0,
          tan(0.5 * NN_THETA) * (VecGet(&checkhidden, 0) + VecGet(&checkhidden, 1)));
        VecSet(&check, 1, tan(0.5 * NN_THETA) * VecGet(&checkhidden, 1));
        if (VecIsEqual(&output, &check) == false ||
          VecIsEqual(NNHiddenValues(nn), &checkhidden) == false) {
          NeuraNetErr->_type = PBErrTypeUnitTestFailed;
          sprintf(NeuraNetErr->_msg, "NNEval failed");
          PBErrCatch(NeuraNetErr);
        }
      }
    }
  }
  char* cloudUrl = "./cloud.txt";
  if (NNSaveLinkAsCloudGraph(nn, cloudUrl) == false) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NNSaveLinkAsCloudGraph failed");
    PBErrCatch(NeuraNetErr);
  }
  if (ISEQUALF(NNGetHiddenValSimpsonDiv(nn), 0.285714) == false) {
    NeuraNetErr->_type = PBErrTypeUnitTestFailed;
    sprintf(NeuraNetErr->_msg, "NNGetHiddenValSimpsonDiv failed");
    PBErrCatch(NeuraNetErr);
  }
  NeuraNetFree(&nn);
  printf("UnitTestNeuraNetEvalPrintHiddenLinkSimpsonDiv OK\n");
}

#ifdef GENALG_H
float evaluate(const NeuraNet* const nn) {
  VecFloat3D input = VecFloatCreateStatic3D();
  VecFloat3D output = VecFloatCreateStatic3D();
  VecFloat3D check = VecFloatCreateStatic3D();
  float val = 0.0;
  int nb = 0;
  for (int i = -5; i <= 5; ++i) {
    for (int j = -5; j <= 5; ++j) {
      for (int k = -5; k <= 5; ++k) {
        VecSet(&input, 0, 0.2 * (float)i);
        VecSet(&input, 1, 0.2 * (float)j);
        VecSet(&input, 2, 0.2 * (float)k);
        NNEval(nn, (VecFloat*)&input, (VecFloat*)&output);
        VecSet(&check, 0,
          0.5 * (VecGet(&input, 1) - fsquare(VecGet(&input, 0))));
        VecSet(&check, 1, VecGet(&input, 1));
        val += VecDist(&output, &check);
        ++nb;
      }
    }
  }
  return -1.0 * val / (float)nb;
}

void UnitTestNeuraNetGA() {
  //srandom(RANDOMSEED);
  srandom(time(NULL));
  int nbIn = 3;
  int nbOut = 3;
  int nbHid = 3;
```

```
  int nbBase = 7;
  int nbLink = 7;
  NeuraNet* nn = NeuraNetCreate(nbIn, nbOut, nbHid, nbBase, nbLink);
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  // Must be declared as a GenAlg applied to a NeuraNet or links will
  // get corrupted
  GASetTypeNeuraNet(ga, nbIn, nbHid, nbOut);
  GAInit(ga);
  float best = -1000000.0;
  float ev = 0.0;
  do {
    for (int iEnt = GAGetNbAdns(ga); iEnt--;) {
      if (GAAdnIsNew(GAAdn(ga, iEnt))) {
        NNSetBases(nn, GAAdnAdnF(GAAdn(ga, iEnt)));
        NNSetLinks(nn, GAAdnAdnI(GAAdn(ga, iEnt)));
        float value = evaluate(nn);
        GASetAdnValue(ga, GAAdn(ga, iEnt), value);
      }
    }
    GAStep(ga);
    NNSetBases(nn, GABestAdnF(ga));
    NNSetLinks(nn, GABestAdnI(ga));
    ev = evaluate(nn);
    if (ev > best + PBMATH_EPSILON) {
      best = ev;
      printf("%lu %f\n", GAGetCurEpoch(ga), best);
      fflush(stdout);
    }
  } while (GAGetCurEpoch(ga) < 30000 && fabs(ev) > 0.001);
  //} while (GAGetCurEpoch(ga) < 10 && fabs(ev) > 0.001);
  printf("best after %lu epochs: %f \n", GAGetCurEpoch(ga), best);
  NNPrintln(nn, stdout);
  FILE* fd = fopen("./bestnn.txt", "w");
  NNSave(nn, fd, false);
  fclose(fd);
  NeuraNetFree(&nn);
  GenAlgFree(&ga);
  printf("UnitTestNeuraNetGA OK\n");
}
#endif

void UnitTestNeuraNet() {
  UnitTestNeuraNetCreateFree();
  UnitTestNeuraNetCreateFullyConnected();
  UnitTestNeuraNetCreateConvolution();
  UnitTestNeuraNetGetSet();
  UnitTestNeuraNetSaveLoadPrune();
  UnitTestNeuraNetEvalPrintHiddenLinkSimpsonDiv();
#ifdef GENALG_H
  UnitTestNeuraNetGA();
#endif

  printf("UnitTestNeuraNet OK\n");
}

void UnitTestAll() {
  UnitTestNNBaseFun();
  UnitTestNeuraNet();
  printf("UnitTestAll OK\n");
```

```
}

int main() {
  UnitTestAll();
  // Return success code
  return 0;
}
```

6 Unit tests output

```
UnitTestNNBaseFun OK
UnitTestNeuraNetCreateFree OK
UnitTestNeuraNetCreateFullyConnected OK
nbInput: 12
nbOutput: 2
nbHidden: 16
nbMaxBases: 24
nbMaxLinks: 72
bases: <0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000>
links: <0,0,12,4,0,18,1,1,12,0,1,13,5,1,18,4,1,19,1,2,13,0,2,14,5,2,19,4,2,20,1,3,14,5,3,20,2,4,12,0,4,15,6,4,18,4,4,
hidden values: <0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000>
UnitTestNeuraNetCreateConvolution OK
UnitTestNeuraNetGetSet OK
UnitTestNeuraNetSaveLoadPrune OK
nbInput: 3
nbOutput: 3
nbHidden: 3
nbMaxBases: 3
nbMaxLinks: 7
bases: <0.500,0.000,0.000,-0.500,0.000,0.000,0.000,0.000,-0.500>
links: <0,0,3,1,0,3,0,1,4,0,3,6,0,4,6,0,4,7,-1,0,0>
hidden values: <0.000,0.000,0.000>
UnitTestNeuraNetEvalPrintHiddenLinkSimpsonDiv OK
1 -1.147484
2 -0.503211
5 -0.459072
6 -0.372083
8 -0.357212
9 -0.328562
11 -0.319573
13 -0.319140
14 -0.297351
15 -0.283395
17 -0.267530
19 -0.265638
20 -0.252939
24 -0.227010
26 -0.226615
37 -0.176831
41 -0.170383
79 -0.162464
110 -0.146485
120 -0.138904
121 -0.106904
122 -0.096049
137 -0.091082
202 -0.046639
690 -0.018232
```

```
2692 -0.017945
5273 -0.013330
15291 -0.009036
26762 -0.007761
26959 -0.006984
best after 30000 epochs: -0.006984
nbInput: 3
nbOutput: 3
nbHidden: 3
nbMaxBases: 7
nbMaxLinks: 7
bases: <0.866,0.350,0.504,-0.629,0.755,0.517,0.954,0.711,0.606,0.291,-0.772,0.292,0.503,0.686,-0.691,0.204,0.840,-0.4
links: <1,0,6,5,0,6,3,1,6,4,1,7,-1,2,5,-1,2,7,-1,2,8>
hidden values: <0.000,0.000,0.000>
UnitTestNeuraNetGA OK
UnitTestNeuraNet OK
UnitTestAll OK
```

neuranet.txt:


```
{
  "_nbInputVal":"10",
  "_nbOutputVal":"20",
  "_nbMaxHidVal":"30",
  "_nbMaxBases":"4",
  "_nbMaxLinks":"5",
  "_bases":{
    "_dim":"12",
    "_val":["0.000000","0.010000","0.020000","0.030000","0.040000","0.050000","0.060000","0.070000","0.080000","0.090
  },
  "_links":{
    "_dim":"15",
    "_val":["1","1","12","2","2","35","2","15","20","3","15","20","-1","0","0"]
  }
}
```

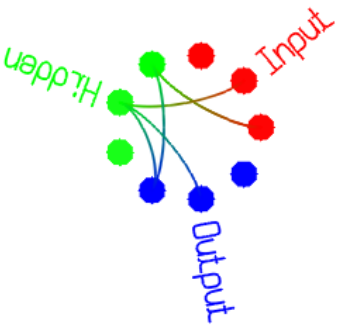

bestnn.txt:


```
{
  "_nbInputVal":"3",
  "_nbOutputVal":"3",
  "_nbMaxHidVal":"3",
  "_nbMaxBases":"7",
  "_nbMaxLinks":"7",
  "_bases":{
    "_dim":"21",
    "_val":["0.866184","0.349639","0.504356","-0.628966","0.754808","0.517012","0.953597","0.711039","0.605664","0.29
  },
  "_links":{
    "_dim":"21",
    "_val":["1","0","6","5","0","6","3","1","6","4","1","7","-1","2","5","-1","2","7","-1","2","8"]
  }
}
```


cloud.txt:

```
3
0 255 0 0 Input
1 0 255 0 Hidden
```

```
2 0 0 255 Output
9
0 0 0
1 0 1
2 0 2
3 1 3
4 1 4
5 1 5
6 2 6
7 2 7
8 2 8
6
0 3
0 3
1 4
3 6
4 6
4 7
```



7 Validation

7.1 Iris data set

Source: https://archive.ics.uci.edu/ml/datasets/iris

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "genalg.h"
```

```
#include "neuranet.h"

// https://archive.ics.uci.edu/ml/datasets/iris

// Nb of step between each save of the GenAlg
// Saving it allows to restart a stop learning process but is
// very time consuming if there are many input/hidden/output
// If 0 never save
#define SAVE_GA_EVERY 0
// Nb input and output of the NeuraNet
#define NB_INPUT 4
#define NB_OUTPUT 3
// Nb max of hidden values, links and base functions
#define NB_MAXHIDDEN 1
#define NB_MAXLINK 50
#define NB_MAXBASE NB_MAXLINK
// Size of the gene pool and elite pool
#define ADN_SIZE_POOL 500
#define ADN_SIZE_ELITE 20
// Initial best value during learning, must be lower than any
// possible value returned by Evaluate()
#define INIT_BEST_VAL 0.0
// Value of the NeuraNet above which the learning process stops
#define STOP_LEARNING_AT_VAL 0.999
// Number of epoch above which the learning process stops
#define STOP_LEARNING_AT_EPOCH 100
// Save NeuraNet in compact format
#define COMPACT true

// Categories of data sets

typedef enum DataSetCat {
  unknownDataSet,
  datalearn,
  datatest,
  dataall
} DataSetCat;
#define NB_DATASET 4
const char* dataSetNames[NB_DATASET] = {
  "unknownDataSet", "datalearn", "datatest", "dataall"
  };

// Structure for the data set
typedef enum IrisCat {
  setosa, versicolor, virginica
} IrisCat;
const char* irisCatNames[3] = {
  "setosa", "versicolor", "virginica"
  };

typedef struct Iris {
  float _props[4];
  IrisCat _cat;
} Iris;

typedef struct DataSet {
  // Category of the data set
  DataSetCat _cat;
  // Number of sample
  int _nbSample;
  // Samples
  Iris* _samples;
```

```
} DataSet;

// Get the DataSetCat from its 'name'
DataSetCat GetCategoryFromName(const char* const name) {
  // Declare a variable to memorize the DataSetCat
  DataSetCat cat = unknownDataSet;
  // Search the dataset
  for (int iSet = NB_DATASET; iSet--;)
    if (strcmp(name, dataSetNames[iSet]) == 0)
      cat = iSet;
  // Return the category
  return cat;
}

// Load the data set of category 'cat' in the DataSet 'that'
// Return true on success, else false
bool DataSetLoad(DataSet* const that, const DataSetCat cat) {
  // Set the category
  that->_cat = cat;

  // Load the data according to 'cat'
  FILE* f = fopen("./bezdekIris.data", "r");
  if (f == NULL) {
    printf("Couldn't open the data set file\n");
    return false;
  }
  char buffer[500];
  int ret = 0;
  if (cat == datalearn) {
    that->_nbSample = 75;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Iris) * that->_nbSample);
    for (int iCat = 0; iCat < 3; ++iCat) {
      for (int iSample = 0; iSample < 25; ++iSample) {
        ret = fscanf(f, "%f,%f,%f,%f,%s",
          that->_samples[25 * iCat + iSample]._props,
          that->_samples[25 * iCat + iSample]._props + 1,
          that->_samples[25 * iCat + iSample]._props + 2,
          that->_samples[25 * iCat + iSample]._props + 3,
          buffer);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
        that->_samples[25 * iCat + iSample]._cat = (IrisCat)iCat;
      }
      for (int iSample = 0; iSample < 25; ++iSample) {
        ret = fscanf(f, "%s\n", buffer);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
      }
    }
  } else if (cat == datatest) {
    that->_nbSample = 75;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Iris) * that->_nbSample);
    for (int iCat = 0; iCat < 3; ++iCat) {
      for (int iSample = 0; iSample < 25; ++iSample) {
```

```
        ret = fscanf(f, "%s\n", buffer);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
      }
      for (int iSample = 0; iSample < 25; ++iSample) {
        ret = fscanf(f, "%f,%f,%f,%f,%s",
          that->_samples[25 * iCat + iSample]._props,
          that->_samples[25 * iCat + iSample]._props + 1,
          that->_samples[25 * iCat + iSample]._props + 2,
          that->_samples[25 * iCat + iSample]._props + 3,
          buffer);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
        that->_samples[25 * iCat + iSample]._cat = (IrisCat)iCat;
      }
    }
  } else if (cat == dataall) {
    that->_nbSample = 150;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Iris) * that->_nbSample);
    for (int iCat = 0; iCat < 3; ++iCat) {
      for (int iSample = 0; iSample < 50; ++iSample) {
        ret = fscanf(f, "%f,%f,%f,%f,%s",
          that->_samples[50 * iCat + iSample]._props,
          that->_samples[50 * iCat + iSample]._props + 1,
          that->_samples[50 * iCat + iSample]._props + 2,
          that->_samples[50 * iCat + iSample]._props + 3,
          buffer);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
        that->_samples[50 * iCat + iSample]._cat = (IrisCat)iCat;
      }
    }
  } else {
    printf("Invalid dataset\n");
    fclose(f);
    return false;
  }
  fclose(f);

  // Return success code
  return true;
}

// Free memory for the DataSet 'that'
void DataSetFree(DataSet** that) {
  if (*that == NULL) return;
  // Free the memory
  free((*that)->_samples);
  free(*that);
  *that = NULL;
}
```

```
// Evalutation function for the NeuraNet 'that' on the DataSet 'dataset'
// Return the value of the NeuraNet, the bigger the better
float Evaluate(const NeuraNet* const that,
  const DataSet* const dataset) {
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Declare a variable to memorize the value
  float val = 0.0;

  // Evaluate

  for (int iSample = dataset->_nbSample; iSample--;) {
    for (int iInp = 0; iInp < NNGetNbInput(that); ++iInp) {
      VecSet(input, iInp,
        dataset->_samples[iSample]._props[iInp]);
    }
    NNEval(that, input, output);
    int pred = VecGetIMaxVal(output);
    if (dataset->_cat == datatest) {
      printf("#%d pred%d real%d ", iSample, pred,
        dataset->_samples[iSample]._cat);
      VecPrint(output, stdout);
    }
    if ((IrisCat)pred == dataset->_samples[iSample]._cat) {
      if (dataset->_cat == datatest)
        printf(" OK\n");
      val += 1.0;
    } else {
      if (dataset->_cat == datatest)
        printf(" NG\n");
    }

  }
  val /= (float)(dataset->_nbSample);

  // Free memory
  VecFree(&input);
  VecFree(&output);
  // Return the result of the evaluation
  return val;
}

// Create the NeuraNet
NeuraNet* createNN(void) {
  // Create the NeuraNet
  int nbIn = NB_INPUT;
  int nbOut = NB_OUTPUT;
  int nbMaxHid = NB_MAXHIDDEN;
  int nbMaxLink = NB_MAXLINK;
  int nbMaxBase = NB_MAXBASE;
  NeuraNet* nn =
    NeuraNetCreate(nbIn, nbOut, nbMaxHid, nbMaxBase, nbMaxLink);

  // Return the NeuraNet
  return nn;
}

// Learn based on the DataSetCat 'cat'
void Learn(DataSetCat cat) {
  // Init the random generator
  srandom(time(NULL));
```

```
// Declare variables to measure time
struct timespec start, stop;
// Start measuring time
clock_gettime(CLOCK_REALTIME, &start);
// Load the DataSet
DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
bool ret = DataSetLoad(dataset, cat);
if (!ret) {
  printf("Couldn't load the data\n");
  return;
}
// Create the NeuraNet
NeuraNet* nn = createNN();
// Declare a variable to memorize the best value
float bestVal = INIT_BEST_VAL;
// Declare a variable to memorize the limit in term of epoch
unsigned long int limitEpoch = STOP_LEARNING_AT_EPOCH;
// Create the GenAlg used for learning
// If previous weights are available in "./bestga.txt" reload them
GenAlg* ga = NULL;
FILE* fd = fopen("./bestga.txt", "r");
if (fd) {
  printf("Reloading previous GenAlg...\n");
  if (!GALoad(&ga, fd)) {
    printf("Failed to reload the GenAlg.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous GenAlg reloaded.\n");
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GABestAdnF(ga));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GABestAdnI(ga));
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    limitEpoch += GAGetCurEpoch(ga);
  }
  fclose(fd);
} else {
  printf("Creating new GenAlg...\n");
  fflush(stdout);
  ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
    NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  // Must be declared as a GenAlg applied to a NeuraNet with
  // convolution
  GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
  GAInit(ga);
}
// If there is a NeuraNet available, reload it into the GenAlg
fd = fopen("./bestnn.txt", "r");
if (fd) {
  printf("Reloading previous NeuraNet...\n");
  if (!NNLoad(&nn, fd)) {
    printf("Failed to reload the NeuraNet.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous NeuraNet reloaded.\n");
```

```
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    GenAlgAdn* adn = GAAdn(ga, 0);
    if (adn->_adnF)
      VecCopy(adn->_adnF, nn->_bases);
    if (adn->_adnI)
      VecCopy(adn->_adnI, nn->_links);
  }
  fclose(fd);
}
// Start learning process
printf("Learning...\n");
printf("Will stop when curEpoch >= %lu or bestVal >= %f\n",
  limitEpoch, STOP_LEARNING_AT_VAL);
printf("Will save the best NeuraNet in ./bestnn.txt at each improvement\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = 0.0;
float curWorst = 0.0;
// Declare a variable to manage the save of GenAlg
int delaySave = 0;
// Learning loop
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  curWorst = curBest;
  curBest = INIT_BEST_VAL;
  int curBestI = 0;
  unsigned long int ageBest = 0;
  // For each adn in the GenAlg
  //for (int iEnt = GAGetNbAdns(ga); iEnt--;) {
  for (int iEnt = 0; iEnt < GAGetNbAdns(ga); ++iEnt) {
    // Get the adn
    GenAlgAdn* adn = GAAdn(ga, iEnt);
    // Set the links and base functions of the NeuraNet according
    // to this adn
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GAAdnAdnF(adn));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GAAdnAdnI(adn));
    // Evaluate the NeuraNet
    float value = Evaluate(nn, dataset);
    // Update the value of this adn
    GASetAdnValue(ga, adn, value);
    // Update the best value in the current epoch
    if (value > curBest) {
      curBest = value;
      curBestI = iEnt;
      ageBest = GAAdnGetAge(adn);
    }
    if (value < curWorst)
      curWorst = value;
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
```

```
  // If there has been improvement during this epoch
  if (curBest > bestVal) {
    bestVal = curBest;
    // Display info about the improvment
    printf("Improvement at epoch %05lu: %f(%03d) (in %02d:%02d:%02d:%02ds)       \n",
      GAGetCurEpoch(ga), bestVal, curBestI, day, hour, min, sec);
    fflush(stdout);
    // Set the links and base functions of the NeuraNet according
    // to the best adn
    GenAlgAdn* bestAdn = GAAdn(ga, curBestI);
    if (GAAdnAdnF(bestAdn) != NULL)
      NNSetBases(nn, GAAdnAdnF(bestAdn));
    if (GAAdnAdnI(bestAdn) != NULL)
      NNSetLinks(nn, GAAdnAdnI(bestAdn));
    // Save the best NeuraNet
    fd = fopen("./bestnn.txt", "w");
    if (!NNSave(nn, fd, COMPACT)) {
      printf("Couldn't save the NeuraNet\n");
      NeuraNetFree(&nn);
      GenAlgFree(&ga);
      DataSetFree(&dataset);
      return;
    }
    fclose(fd);
  } else {
    fprintf(stderr,
      "Epoch %05lu: v%f a%03lu(%03d) kt%03lu ",
      GAGetCurEpoch(ga), curBest, ageBest, curBestI,
      GAGetNbKTEvent(ga));
    fprintf(stderr, "(in %02d:%02d:%02d:%02ds)  \r",
      day, hour, min, sec);
    fflush(stderr);
  }
  ++delaySave;
  if (SAVE_GA_EVERY != 0 && delaySave >= SAVE_GA_EVERY) {
    delaySave = 0;
    // Save the adns of the GenAlg, use a temporary file to avoid
    // loosing the previous one if something goes wrong during
    // writing, then replace the previous file with the temporary one
    fd = fopen("./bestga.tmp", "w");
    if (!GASave(ga, fd, COMPACT)) {
      printf("Couldn't save the GenAlg\n");
      NeuraNetFree(&nn);
      GenAlgFree(&ga);
      DataSetFree(&dataset);
      return;
    }
    fclose(fd);
    int ret = system("mv ./bestga.tmp ./bestga.txt");
    (void)ret;
  }
  // Step the GenAlg
  GAStep(ga);
}
// Measure time
clock_gettime(CLOCK_REALTIME, &stop);
float elapsed = stop.tv_sec - start.tv_sec;
int day = (int)floor(elapsed / 86400);
elapsed -= (float)(day * 86400);
int hour = (int)floor(elapsed / 3600);
elapsed -= (float)(hour * 3600);
int min = (int)floor(elapsed / 60);
```

```
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  printf("\nLearning complete (in %d:%d:%d:%ds)\n",
    day, hour, min, sec);
  fflush(stdout);
  // Free memory
  NeuraNetFree(&nn);
  GenAlgFree(&ga);
  DataSetFree(&dataset);
}

// Check the NeuraNet 'that' on the DataSetCat 'cat'
void Validate(const NeuraNet* const that, const DataSetCat cat) {
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Evaluate the NeuraNet
  float value = Evaluate(that, dataset);
  // Display the result
  printf("Value: %.6f\n", value);
  // Free memory
  DataSetFree(&dataset);
}

// Predict using the NeuraNet 'that' on 'inputs' (given as an array of
// 'nbInp' char*)
void Predict(const NeuraNet* const that, const int nbInp,
  char** const inputs) {
  // Start measuring time
  clock_t clockStart = clock();
  // Check the number of inputs
  if (nbInp != NNGetNbInput(that)) {
    printf("Wrong number of inputs, there should %d, there was %d\n",
      NNGetNbInput(that), nbInp);
    return;
  }
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Set the input
  for (int iInp = 0; iInp < nbInp; ++iInp) {
    float v = 0.0;
    sscanf(inputs[iInp], "%f", &v);
    VecSet(input, iInp, v);
  }
  // Predict
  NNEval(that, input, output);
  int pred = -1;
  if (VecGet(output, 0) > VecGet(output, 1) &&
    VecGet(output, 0) > VecGet(output, 2))
    pred = 0;
  else if (VecGet(output, 1) > VecGet(output, 0) &&
    VecGet(output, 1) > VecGet(output, 2))
    pred = 1;
  else if (VecGet(output, 2) > VecGet(output, 1) &&
    VecGet(output, 2) > VecGet(output, 0))
    pred = 2;
  // End measuring time
```

```
  clock_t clockEnd = clock();
  double timeUsed =
    ((double)(clockEnd - clockStart)) / (CLOCKS_PER_SEC * 0.001) ;
  // If the clock has been reset meanwhile
  if (timeUsed < 0.0)
    timeUsed = 0.0;
  printf("Prediction: %s (in %fms)\n", irisCatNames[pred], timeUsed);

  // Free memory
  VecFree(&input);
  VecFree(&output);
}

int main(int argc, char** argv) {
  // Declare a variable to memorize the mode (learning/checking)
  int mode = -1;
  // Declare a variable to memorize the dataset used
  DataSetCat cat = unknownDataSet;
  // Decode mode from arguments
  if (argc >= 3) {
    if (strcmp(argv[1], "-learn") == 0) {
      mode = 0;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-check") == 0) {
      mode = 1;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-predict") == 0) {
      mode = 2;
    }
  }
  // If the mode is invalid print some help
  if (mode == -1) {
    printf("Select a mode from:\n");
    printf("-learn <dataset name>\n");
    printf("-check <dataset name>\n");
    printf("-predict <input values>\n");
    return 0;
  }
  if (mode == 0) {
    Learn(cat);
  } else if (mode == 1) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Validate(nn, cat);
    NeuraNetFree(&nn);
  } else if (mode == 2) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Predict(nn, argc - 2, argv + 2);
    NeuraNetFree(&nn);
  }
  // Return success code
```

```
  return 0;
}
```

learn.txt:

```
Creating new GenAlg...
Learning...
Will stop when curEpoch >= 100 or bestVal >= 0.999000
Will save the best NeuraNet in ./bestnn.txt at each improvement
Improvement at epoch 00000: 0.666667(014) (in 00:00:00:00s)
Improvement at epoch 00001: 0.866667(078) (in 00:00:00:00s)
Improvement at epoch 00002: 0.946667(277) (in 00:00:00:00s)
Improvement at epoch 00004: 0.960000(077) (in 00:00:00:00s)
Improvement at epoch 00008: 0.973333(394) (in 00:00:00:01s)
Improvement at epoch 00030: 0.986667(421) (in 00:00:00:01s)

Learning complete (in 0:0:0:2s)
```

checktest.txt:

```
#74 pred2 real2 <-1.460,0.982,1.071> OK
#73 pred2 real2 <-2.548,1.028,1.268> OK
#72 pred2 real2 <-1.863,1.001,1.142> OK
#71 pred2 real2 <-1.618,0.991,1.047> OK
#70 pred2 real2 <-2.495,1.028,1.189> OK
#69 pred2 real2 <-3.092,1.046,1.418> OK
#68 pred2 real2 <-2.679,1.028,1.465> OK
#67 pred2 real2 <-1.644,0.991,1.086> OK
#66 pred2 real2 <-2.469,1.028,1.149> OK
#65 pred2 real2 <-2.828,1.037,1.363> OK
#64 pred2 real2 <-2.118,1.010,1.236> OK
#63 pred1 real2 <-1.381,0.982,0.952> NG
#62 pred2 real2 <-1.565,0.982,1.229> OK
#61 pred2 real2 <-2.828,1.037,1.363> OK
#60 pred2 real2 <-2.732,1.028,1.545> OK
#59 pred2 real2 <-0.943,0.946,1.205> OK
#58 pred2 real2 <-0.961,0.955,1.023> OK
#57 pred2 real2 <-2.381,1.019,1.331> OK
#56 pred2 real2 <-2.179,1.001,1.616> OK
#55 pred2 real2 <-1.907,0.991,1.482> OK
#54 pred2 real2 <-1.303,0.964,1.316> OK
#53 pred2 real2 <-2.170,1.010,1.315> OK
#52 pred2 real2 <-1.407,0.982,0.992> OK
#51 pred1 real2 <-1.381,0.982,0.952> NG
#50 pred2 real2 <-1.697,0.982,1.426> OK
#49 pred1 real1 <-0.408,0.936,0.597> OK
#48 pred1 real1 <0.135,0.918,0.131> OK
#47 pred1 real1 <-0.461,0.936,0.676> OK
#46 pred1 real1 <-0.435,0.936,0.636> OK
#45 pred1 real1 <-0.303,0.927,0.621> OK
#44 pred1 real1 <-0.435,0.936,0.636> OK
#43 pred1 real1 <0.169,0.909,0.233> OK
#42 pred1 real1 <-0.251,0.927,0.541> OK
#41 pred1 real1 <-0.680,0.946,0.810> OK
#40 pred1 real1 <-0.356,0.927,0.700> OK
#39 pred1 real1 <-0.382,0.936,0.557> OK
#38 pred1 real1 <-0.408,0.936,0.597> OK
#37 pred1 real1 <-0.487,0.936,0.715> OK
#36 pred1 real1 <-0.855,0.955,0.865> OK
#35 pred1 real1 <-0.960,0.964,0.802> OK
```

```
#34 pred1 real1 <-0.802,0.955,0.786> OK
#33 pred2 real1 <-1.118,0.964,1.039> NG
#32 pred1 real1 <-0.224,0.927,0.502> OK
#31 pred1 real1 <0.064,0.909,0.391> OK
#30 pred1 real1 <-0.076,0.918,0.447> OK
#29 pred1 real1 <0.117,0.909,0.312> OK
#28 pred1 real1 <-0.802,0.955,0.786> OK
#27 pred2 real1 <-1.258,0.973,1.015> NG
#26 pred1 real1 <-0.733,0.946,0.889> OK
#25 pred1 real1 <-0.627,0.946,0.731> OK
#24 pred0 real0 <1.260,0.836,-0.643> OK
#23 pred0 real0 <1.234,0.836,-0.604> OK
#22 pred0 real0 <1.260,0.836,-0.643> OK
#21 pred0 real0 <1.208,0.836,-0.564> OK
#20 pred0 real0 <1.217,0.845,-0.628> OK
#19 pred0 real0 <1.034,0.854,-0.414> OK
#18 pred0 real0 <0.983,0.872,-0.501> OK
#17 pred0 real0 <1.287,0.836,-0.683> OK
#16 pred0 real0 <1.244,0.845,-0.667> OK
#15 pred0 real0 <1.244,0.845,-0.667> OK
#14 pred0 real0 <1.234,0.836,-0.604> OK
#13 pred0 real0 <1.287,0.836,-0.683> OK
#12 pred0 real0 <1.294,0.827,-0.659> OK
#11 pred0 real0 <1.287,0.836,-0.683> OK
#10 pred0 real0 <1.313,0.836,-0.722> OK
#9 pred0 real0 <1.234,0.836,-0.604> OK
#8 pred0 real0 <1.260,0.836,-0.643> OK
#7 pred0 real0 <1.268,0.827,-0.620> OK
#6 pred0 real0 <1.139,0.854,-0.572> OK
#5 pred0 real0 <1.208,0.836,-0.564> OK
#4 pred0 real0 <1.208,0.836,-0.564> OK
#3 pred0 real0 <1.260,0.836,-0.643> OK
#2 pred0 real0 <1.234,0.836,-0.604> OK
#1 pred0 real0 <1.113,0.854,-0.533> OK
#0 pred0 real0 <1.208,0.836,-0.564> OK
Value: 0.946667
```

checkall.txt:

```
Value: 0.966667
```

7.2 Abalone data set

Source: http://www.cs.toronto.edu/ delve/data/abalone/desc.html

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "genalg.h"
#include "neuranet.h"
```

```
// http://www.cs.toronto.edu/~delve/data/abalone/desc.html

// Nb of step between each save of the GenAlg
// Saving it allows to restart a stop learning process but is
// very time consuming if there are many input/hidden/output
// If 0 never save
#define SAVE_GA_EVERY 0
// Nb input and output of the NeuraNet
#define NB_INPUT 10
#define NB_OUTPUT 1
// Nb max of hidden values, links and base functions
#define NB_MAXHIDDEN 1
#define NB_MAXLINK 100
#define NB_MAXBASE NB_MAXLINK
// Size of the gene pool and elite pool
#define ADN_SIZE_POOL 500
#define ADN_SIZE_ELITE 20
// Initial best value during learning, must be lower than any
// possible value returned by Evaluate()
#define INIT_BEST_VAL -10000.0
// Value of the NeuraNet above which the learning process stops
#define STOP_LEARNING_AT_VAL -0.01
// Number of epoch above which the learning process stops
#define STOP_LEARNING_AT_EPOCH 1000
// Save NeuraNet in compact format
#define COMPACT true

// Categories of data sets

typedef enum DataSetCat {
  unknownDataSet,
  datalearn,
  datatest,
  dataall
} DataSetCat;
#define NB_DATASET 4
const char* dataSetNames[NB_DATASET] = {
  "unknownDataSet", "datalearn", "datatest", "dataall"
  };

// Structure for the data set

typedef struct Abalone {
  float _props[10];
  float _age;
} Abalone;

typedef struct DataSet {
  // Category of the data set
  DataSetCat _cat;
  // Number of sample
  int _nbSample;
  // Samples
  Abalone* _samples;
  float _weights[29];
} DataSet;

// Get the DataSetCat from its 'name'
DataSetCat GetCategoryFromName(const char* const name) {
  // Declare a variable to memorize the DataSetCat
  DataSetCat cat = unknownDataSet;
```

```
  // Search the dataset
  for (int iSet = NB_DATASET; iSet--;)
    if (strcmp(name, dataSetNames[iSet]) == 0)
      cat = iSet;
  // Return the category
  return cat;
}

// Load the data set of category 'cat' in the DataSet 'that'
// Return true on success, else false
bool DataSetLoad(DataSet* const that, const DataSetCat cat) {
  // Set the category
  that->_cat = cat;

  // Load the data according to 'cat'
  FILE* f = fopen("./Prototask.data", "r");
  if (f == NULL) {
    printf("Couldn't open the data set file\n");
    return false;
  }
  char sex;
  int age;
  int ret = 0;
  if (cat == datalearn) {
    that->_nbSample = 3000;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Abalone) * that->_nbSample);
    for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
      ret = fscanf(f, "%c %f %f %f %f %f %f %f %d\n",
        &sex,
        that->_samples[iSample]._props + 3,
        that->_samples[iSample]._props + 4,
        that->_samples[iSample]._props + 5,
        that->_samples[iSample]._props + 6,
        that->_samples[iSample]._props + 7,
        that->_samples[iSample]._props + 8,
        that->_samples[iSample]._props + 9,
        &age);
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
      that->_samples[iSample]._age = (float)age;
      if (sex == 'M') {
        that->_samples[iSample]._props[0] = 1.0;
        that->_samples[iSample]._props[1] = -1.0;
        that->_samples[iSample]._props[2] = -1.0;
      } else if (sex == 'F') {
        that->_samples[iSample]._props[0] = -1.0;
        that->_samples[iSample]._props[1] = 1.0;
        that->_samples[iSample]._props[2] = -1.0;
      } else if (sex == 'I') {
        that->_samples[iSample]._props[0] = -1.0;
        that->_samples[iSample]._props[1] = -1.0;
        that->_samples[iSample]._props[2] = 1.0;
      }
    }
  } else if (cat == datatest) {
    for (int iSample = 0; iSample < 3000; ++iSample) {
      float dummy;
      ret = fscanf(f, "%c %f %f %f %f %f %f %f %d\n",
```

```
      &sex,
      &dummy,
      &dummy,
      &dummy,
      &dummy,
      &dummy,
      &dummy,
      &dummy,
      &age);
    (void)dummy;
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
  }
  that->_nbSample = 1177;
  that->_samples =
    PBErrMalloc(NeuraNetErr, sizeof(Abalone) * that->_nbSample);
  for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
    ret = fscanf(f, "%c %f %f %f %f %f %f %f %d\n",
      &sex,
      that->_samples[iSample]._props + 3,
      that->_samples[iSample]._props + 4,
      that->_samples[iSample]._props + 5,
      that->_samples[iSample]._props + 6,
      that->_samples[iSample]._props + 7,
      that->_samples[iSample]._props + 8,
      that->_samples[iSample]._props + 9,
      &age);
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
    that->_samples[iSample]._age = (float)age;
    if (sex == 'M') {
      that->_samples[iSample]._props[0] = 1.0;
      that->_samples[iSample]._props[1] = -1.0;
      that->_samples[iSample]._props[2] = -1.0;
    } else if (sex == 'F') {
      that->_samples[iSample]._props[0] = -1.0;
      that->_samples[iSample]._props[1] = 1.0;
      that->_samples[iSample]._props[2] = -1.0;
    } else if (sex == 'I') {
      that->_samples[iSample]._props[0] = -1.0;
      that->_samples[iSample]._props[1] = -1.0;
      that->_samples[iSample]._props[2] = 1.0;
    }
  }
} else if (cat == dataall) {
  that->_nbSample = 4177;
  that->_samples =
    PBErrMalloc(NeuraNetErr, sizeof(Abalone) * that->_nbSample);
  for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
    ret = fscanf(f, "%c %f %f %f %f %f %f %f %d\n",
      &sex,
      that->_samples[iSample]._props + 3,
      that->_samples[iSample]._props + 4,
      that->_samples[iSample]._props + 5,
      that->_samples[iSample]._props + 6,
      that->_samples[iSample]._props + 7,
```

```
        that->_samples[iSample]._props + 8,
        that->_samples[iSample]._props + 9,
        &age);
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
      that->_samples[iSample]._age = (float)age;
      if (sex == 'M') {
        that->_samples[iSample]._props[0] = 1.0;
        that->_samples[iSample]._props[1] = -1.0;
        that->_samples[iSample]._props[2] = -1.0;
      } else if (sex == 'F') {
        that->_samples[iSample]._props[0] = -1.0;
        that->_samples[iSample]._props[1] = 1.0;
        that->_samples[iSample]._props[2] = -1.0;
      } else if (sex == 'I') {
        that->_samples[iSample]._props[0] = -1.0;
        that->_samples[iSample]._props[1] = -1.0;
        that->_samples[iSample]._props[2] = 1.0;
      }
    }
  } else {
    printf("Invalid dataset\n");
    fclose(f);
    return false;
  }
  fclose(f);

  for (int iCat = 29; iCat--;)
    that->_weights[iCat] = 0.0;
  for (int iSample = that->_nbSample; iSample--;) {
    int cat = (int)round(that->_samples[iSample]._age) - 1;
    if (cat < 0 || cat >= 29) {
      printf("Invalid age #%d %f\n", iSample,
        that->_samples[iSample]._age);
      return false;
    }
    that->_weights[cat] += 1.0;
  }
  for (int iCat = 29; iCat--;)
    that->_weights[iCat] =
      ((float)(that->_nbSample) - that->_weights[iCat]) /
      (float)(that->_nbSample);

  // Return success code
  return true;
}

// Free memory for the DataSet 'that'
void DataSetFree(DataSet** that) {
  if (*that == NULL) return;
  // Free the memory
  free((*that)->_samples);
  free(*that);
  *that = NULL;
}

// Evalutation function for the NeuraNet 'that' on the DataSet 'dataset'
// Return the value of the NeuraNet, the bigger the better
float Evaluate(const NeuraNet* const that,
```

```
  const DataSet* const dataset) {
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Declare a variable to memorize the value
  float val = 0.0;

  // Evaluate

  int count[29] = {0};
  for (int iSample = dataset->_nbSample; iSample--;) {
    for (int iInp = 0; iInp < NNGetNbInput(that); ++iInp) {
      VecSet(input, iInp,
        dataset->_samples[iSample]._props[iInp]);
    }
    NNEval(that, input, output);

    float pred = VecGet(output, 0);
    float age = dataset->_samples[iSample]._age + 0.5;
    float v = fabs(pred - age);
    val -= v;
    if (dataset->_cat != datalearn) {
      int iErr = (int)round(v);
      ++(count[iErr]);
    }

  }
  val /= (float)(dataset->_nbSample);
  if (dataset->_cat != datalearn) {
    float perc = 0.0;
    printf("age_err count cumul_perc\n");
    for (int iErr = 0; iErr < 29; ++ iErr) {
      perc += (float)(count[iErr]) / (float)(dataset->_nbSample);
      printf("%2d %4d %f\n", iErr, count[iErr], perc);
    }
  }

  // Free memory
  VecFree(&input);
  VecFree(&output);
  // Return the result of the evaluation
  return val;
}

// Create the NeuraNet
NeuraNet* createNN(void) {
  // Create the NeuraNet
  int nbIn = NB_INPUT;
  int nbOut = NB_OUTPUT;
  int nbMaxHid = NB_MAXHIDDEN;
  int nbMaxLink = NB_MAXLINK;
  int nbMaxBase = NB_MAXBASE;
  NeuraNet* nn =
    NeuraNetCreate(nbIn, nbOut, nbMaxHid, nbMaxBase, nbMaxLink);

  // Return the NeuraNet
  return nn;
}

// Learn based on the DataSetCat 'cat'
void Learn(DataSetCat cat) {
  // Init the random generator
```

```
srandom(time(NULL));
// Declare variables to measure time
struct timespec start, stop;
// Start measuring time
clock_gettime(CLOCK_REALTIME, &start);
// Load the DataSet
DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
bool ret = DataSetLoad(dataset, cat);
if (!ret) {
  printf("Couldn't load the data\n");
  return;
}
// Create the NeuraNet
NeuraNet* nn = createNN();
// Declare a variable to memorize the best value
float bestVal = INIT_BEST_VAL;
// Declare a variable to memorize the limit in term of epoch
unsigned long int limitEpoch = STOP_LEARNING_AT_EPOCH;
// Create the GenAlg used for learning
// If previous weights are available in "./bestga.txt" reload them
GenAlg* ga = NULL;
FILE* fd = fopen("./bestga.txt", "r");
if (fd) {
  printf("Reloading previous GenAlg...\n");
  if (!GALoad(&ga, fd)) {
    printf("Failed to reload the GenAlg.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous GenAlg reloaded.\n");
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GABestAdnF(ga));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GABestAdnI(ga));
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    limitEpoch += GAGetCurEpoch(ga);
  }
  fclose(fd);
} else {
  printf("Creating new GenAlg...\n");
  fflush(stdout);
  ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
    NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  // Must be declared as a GenAlg applied to a NeuraNet with
  // convolution
  GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
  GAInit(ga);
}
// If there is a NeuraNet available, reload it into the GenAlg
fd = fopen("./bestnn.txt", "r");
if (fd) {
  printf("Reloading previous NeuraNet...\n");
  if (!NNLoad(&nn, fd)) {
    printf("Failed to reload the NeuraNet.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
```

```
    printf("Previous NeuraNet reloaded.\n");
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    GenAlgAdn* adn = GAAdn(ga, 0);
    if (adn->_adnF)
      VecCopy(adn->_adnF, nn->_bases);
    if (adn->_adnI)
      VecCopy(adn->_adnI, nn->_links);
  }
  fclose(fd);
}
// Start learning process
printf("Learning...\n");
printf("Will stop when curEpoch >= %lu or bestVal >= %f\n",
  limitEpoch, STOP_LEARNING_AT_VAL);
printf("Will save the best NeuraNet in ./bestnn.txt at each improvement\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = 0.0;
float curWorst = 0.0;
// Declare a variable to manage the save of GenAlg
int delaySave = 0;
// Learning loop
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  curWorst = curBest;
  curBest = INIT_BEST_VAL;
  int curBestI = 0;
  unsigned long int ageBest = 0;
  // For each adn in the GenAlg
  //for (int iEnt = GAGetNbAdns(ga); iEnt--;) {
  for (int iEnt = 0; iEnt < GAGetNbAdns(ga); ++iEnt) {
    // Get the adn
    GenAlgAdn* adn = GAAdn(ga, iEnt);
    // Set the links and base functions of the NeuraNet according
    // to this adn
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GAAdnAdnF(adn));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GAAdnAdnI(adn));
    // Evaluate the NeuraNet
    float value = Evaluate(nn, dataset);
    // Update the value of this adn
    GASetAdnValue(ga, adn, value);
    // Update the best value in the current epoch
    if (value > curBest) {
      curBest = value;
      curBestI = iEnt;
      ageBest = GAAdnGetAge(adn);
    }
    if (value < curWorst)
      curWorst = value;
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
```

```
  int sec = (int)floor(elapsed);
  // If there has been improvement during this epoch
  if (curBest > bestVal) {
    bestVal = curBest;
    // Display info about the improvment
    printf("Improvement at epoch %05lu: %f(%03d) (in %02d:%02d:%02d:%02ds)       \n",
      GAGetCurEpoch(ga), bestVal, curBestI, day, hour, min, sec);
    fflush(stdout);
    // Set the links and base functions of the NeuraNet according
    // to the best adn
    GenAlgAdn* bestAdn = GAAdn(ga, curBestI);
    if (GAAdnAdnF(bestAdn) != NULL)
      NNSetBases(nn, GAAdnAdnF(bestAdn));
    if (GAAdnAdnI(bestAdn) != NULL)
      NNSetLinks(nn, GAAdnAdnI(bestAdn));
    // Save the best NeuraNet
    fd = fopen("./bestnn.txt", "w");
    if (!NNSave(nn, fd, COMPACT)) {
      printf("Couldn't save the NeuraNet\n");
      NeuraNetFree(&nn);
      GenAlgFree(&ga);
      DataSetFree(&dataset);
      return;
    }
    fclose(fd);
  } else {
    fprintf(stderr,
      "Epoch %05lu: v%f a%03lu(%03d) kt%03lu ",
      GAGetCurEpoch(ga), curBest, ageBest, curBestI,
      GAGetNbKTEvent(ga));
    fprintf(stderr, "(in %02d:%02d:%02d:%02ds)  \r",
      day, hour, min, sec);
    fflush(stderr);
  }
  ++delaySave;
  if (SAVE_GA_EVERY != 0 && delaySave >= SAVE_GA_EVERY) {
    delaySave = 0;
    // Save the adns of the GenAlg, use a temporary file to avoid
    // loosing the previous one if something goes wrong during
    // writing, then replace the previous file with the temporary one
    fd = fopen("./bestga.tmp", "w");
    if (!GASave(ga, fd, COMPACT)) {
      printf("Couldn't save the GenAlg\n");
      NeuraNetFree(&nn);
      GenAlgFree(&ga);
      DataSetFree(&dataset);
      return;
    }
    fclose(fd);
    int ret = system("mv ./bestga.tmp ./bestga.txt");
    (void)ret;
  }
  // Step the GenAlg
  GAStep(ga);
}
// Measure time
clock_gettime(CLOCK_REALTIME, &stop);
float elapsed = stop.tv_sec - start.tv_sec;
int day = (int)floor(elapsed / 86400);
elapsed -= (float)(day * 86400);
int hour = (int)floor(elapsed / 3600);
elapsed -= (float)(hour * 3600);
```

```
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  printf("\nLearning complete (in %d:%d:%d:%ds)\n",
    day, hour, min, sec);
  fflush(stdout);
  // Free memory
  NeuraNetFree(&nn);
  GenAlgFree(&ga);
  DataSetFree(&dataset);
}

// Check the NeuraNet 'that' on the DataSetCat 'cat'
void Validate(const NeuraNet* const that, const DataSetCat cat) {
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Evaluate the NeuraNet
  float value = Evaluate(that, dataset);
  // Display the result
  printf("Value: %.6f\n", value);
  // Free memory
  DataSetFree(&dataset);
}

// Predict using the NeuraNet 'that' on 'inputs' (given as an array of
// 'nbInp' char*)
void Predict(const NeuraNet* const that, const int nbInp,
  char** const inputs) {
  // Check the number of inputs
  if (nbInp != NNGetNbInput(that)) {
    printf("Wrong number of inputs, there should %d, there was %d\n",
      NNGetNbInput(that), nbInp);
    return;
  }
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Set the input
  for (int iInp = 0; iInp < nbInp; ++iInp) {
    float v = 0.0;
    sscanf(inputs[iInp], "%f", &v);
    VecSet(input, iInp, v);
  }
  // Predict
  NNEval(that, input, output);
  printf("Prediction: %f rings\n", VecGet(output, 0));
  // Free memory
  VecFree(&input);
  VecFree(&output);
}

int main(int argc, char** argv) {
  // Declare a variable to memorize the mode (learning/checking)
  int mode = -1;
  // Declare a variable to memorize the dataset used
  DataSetCat cat = unknownDataSet;
  // Decode mode from arguments
```

```
  if (argc >= 3) {
    if (strcmp(argv[1], "-learn") == 0) {
      mode = 0;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-check") == 0) {
      mode = 1;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-predict") == 0) {
      mode = 2;
    }
  }
  // If the mode is invalid print some help
  if (mode == -1) {
    printf("Select a mode from:\n");
    printf("-learn <dataset name>\n");
    printf("-check <dataset name>\n");
    printf("-predict <input values>\n");
    return 0;
  }
  if (mode == 0) {
    Learn(cat);
  } else if (mode == 1) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Validate(nn, cat);
    NeuraNetFree(&nn);
  } else if (mode == 2) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Predict(nn, argc - 2, argv + 2);
    NeuraNetFree(&nn);
  }
  // Return success code
  return 0;
}
```

learn.txt:

```
Creating new GenAlg...
Learning...
Will stop when curEpoch >= 1000 or bestVal >= -0.010000
Will save the best NeuraNet in ./bestnn.txt at each improvement
Improvement at epoch 00000: -1.967101(073) (in 00:00:00:00s)
Improvement at epoch 00001: -1.947549(196) (in 00:00:00:01s)
Improvement at epoch 00002: -1.822080(022) (in 00:00:00:01s)
Improvement at epoch 00003: -1.821908(381) (in 00:00:00:01s)
Improvement at epoch 00004: -1.797778(290) (in 00:00:00:02s)
Improvement at epoch 00007: -1.784985(342) (in 00:00:00:03s)
Improvement at epoch 00008: -1.774693(022) (in 00:00:00:04s)
Improvement at epoch 00010: -1.751626(442) (in 00:00:00:05s)
```

```
Improvement at epoch 00013: -1.740393(389) (in 00:00:00:07s)
Improvement at epoch 00015: -1.731586(269) (in 00:00:00:09s)
Improvement at epoch 00017: -1.729946(041) (in 00:00:00:12s)
Improvement at epoch 00018: -1.727182(223) (in 00:00:00:14s)
Improvement at epoch 00019: -1.723659(238) (in 00:00:00:15s)
Improvement at epoch 00021: -1.716766(438) (in 00:00:00:18s)
Improvement at epoch 00022: -1.713578(431) (in 00:00:00:20s)
Improvement at epoch 00023: -1.702757(227) (in 00:00:00:22s)
Improvement at epoch 00024: -1.702167(215) (in 00:00:00:24s)
Improvement at epoch 00025: -1.701438(281) (in 00:00:00:25s)
Improvement at epoch 00026: -1.699764(405) (in 00:00:00:27s)
Improvement at epoch 00027: -1.698579(297) (in 00:00:00:29s)
Improvement at epoch 00028: -1.695291(305) (in 00:00:00:31s)
Improvement at epoch 00029: -1.692761(071) (in 00:00:00:33s)
Improvement at epoch 00030: -1.691165(318) (in 00:00:00:35s)
Improvement at epoch 00031: -1.686244(361) (in 00:00:00:37s)
Improvement at epoch 00032: -1.683863(439) (in 00:00:00:39s)
Improvement at epoch 00033: -1.683309(273) (in 00:00:00:41s)
Improvement at epoch 00034: -1.675182(121) (in 00:00:00:43s)
Improvement at epoch 00036: -1.673951(222) (in 00:00:00:47s)
Improvement at epoch 00063: -1.671398(313) (in 00:00:01:12s)
Improvement at epoch 00064: -1.663314(172) (in 00:00:01:13s)
Improvement at epoch 00065: -1.656416(033) (in 00:00:01:15s)
Improvement at epoch 00066: -1.655980(389) (in 00:00:01:17s)
Improvement at epoch 00068: -1.654141(420) (in 00:00:01:20s)
Improvement at epoch 00069: -1.653538(352) (in 00:00:01:22s)
Improvement at epoch 00070: -1.653316(101) (in 00:00:01:24s)
Improvement at epoch 00071: -1.652912(026) (in 00:00:01:26s)
Improvement at epoch 00072: -1.652640(473) (in 00:00:01:29s)
Improvement at epoch 00073: -1.651053(299) (in 00:00:01:31s)
Improvement at epoch 00074: -1.650940(241) (in 00:00:01:33s)
Improvement at epoch 00075: -1.650549(131) (in 00:00:01:35s)
Improvement at epoch 00076: -1.650093(218) (in 00:00:01:37s)
Improvement at epoch 00077: -1.649934(104) (in 00:00:01:39s)
Improvement at epoch 00118: -1.649762(268) (in 00:00:02:51s)
Improvement at epoch 00126: -1.649136(048) (in 00:00:03:11s)
Improvement at epoch 00185: -1.648163(125) (in 00:00:04:28s)
Improvement at epoch 00186: -1.647570(298) (in 00:00:04:31s)
Improvement at epoch 00187: -1.646153(254) (in 00:00:04:33s)
Improvement at epoch 00188: -1.645568(227) (in 00:00:04:35s)
Improvement at epoch 00189: -1.644680(051) (in 00:00:04:37s)
Improvement at epoch 00190: -1.641614(473) (in 00:00:04:40s)
Improvement at epoch 00191: -1.640594(121) (in 00:00:04:43s)
Improvement at epoch 00192: -1.637252(391) (in 00:00:04:45s)
Improvement at epoch 00193: -1.636787(196) (in 00:00:04:48s)
Improvement at epoch 00195: -1.636021(376) (in 00:00:04:53s)
Improvement at epoch 00196: -1.635918(273) (in 00:00:04:56s)
Improvement at epoch 00197: -1.634898(094) (in 00:00:04:58s)
Improvement at epoch 00198: -1.634716(330) (in 00:00:05:01s)
Improvement at epoch 00199: -1.629333(337) (in 00:00:05:04s)
Improvement at epoch 00200: -1.627779(053) (in 00:00:05:06s)
Improvement at epoch 00201: -1.626536(233) (in 00:00:05:09s)
Improvement at epoch 00202: -1.622901(206) (in 00:00:05:12s)
Improvement at epoch 00203: -1.621486(400) (in 00:00:05:14s)
Improvement at epoch 00204: -1.617865(387) (in 00:00:05:17s)
Improvement at epoch 00206: -1.617449(458) (in 00:00:05:23s)
Improvement at epoch 00207: -1.617200(116) (in 00:00:05:26s)
Improvement at epoch 00208: -1.616985(124) (in 00:00:05:29s)
Improvement at epoch 00209: -1.616505(170) (in 00:00:05:32s)
Improvement at epoch 00210: -1.616277(262) (in 00:00:05:34s)
Improvement at epoch 00211: -1.616060(088) (in 00:00:05:37s)
Improvement at epoch 00212: -1.615896(422) (in 00:00:05:40s)
```

```
Improvement at epoch 00213: -1.615847(035) (in 00:00:05:43s)
Improvement at epoch 00214: -1.615556(167) (in 00:00:05:46s)
Improvement at epoch 00215: -1.614975(054) (in 00:00:05:49s)
Improvement at epoch 00216: -1.614802(259) (in 00:00:05:52s)
Improvement at epoch 00217: -1.614675(022) (in 00:00:05:55s)
Improvement at epoch 00218: -1.613659(143) (in 00:00:05:58s)
Improvement at epoch 00219: -1.612360(217) (in 00:00:06:01s)
Improvement at epoch 00241: -1.612096(096) (in 00:00:06:45s)
Improvement at epoch 00242: -1.611673(131) (in 00:00:06:48s)
Improvement at epoch 00243: -1.611223(111) (in 00:00:06:51s)
Improvement at epoch 00244: -1.610752(481) (in 00:00:06:54s)
Improvement at epoch 00245: -1.608308(124) (in 00:00:06:57s)
Improvement at epoch 00246: -1.607643(233) (in 00:00:07:00s)
Improvement at epoch 00248: -1.606705(227) (in 00:00:07:06s)
Improvement at epoch 00250: -1.606588(197) (in 00:00:07:12s)
Improvement at epoch 00251: -1.606457(322) (in 00:00:07:16s)
Improvement at epoch 00253: -1.606405(218) (in 00:00:07:22s)
Improvement at epoch 00254: -1.606150(263) (in 00:00:07:25s)
Improvement at epoch 00255: -1.605659(488) (in 00:00:07:28s)
Improvement at epoch 00256: -1.605186(247) (in 00:00:07:31s)
Improvement at epoch 00257: -1.604736(034) (in 00:00:07:34s)
Improvement at epoch 00258: -1.604302(438) (in 00:00:07:38s)
Improvement at epoch 00259: -1.603701(259) (in 00:00:07:41s)
Improvement at epoch 00260: -1.603629(121) (in 00:00:07:44s)
Improvement at epoch 00261: -1.603504(047) (in 00:00:07:48s)
Improvement at epoch 00262: -1.602790(426) (in 00:00:07:51s)
Improvement at epoch 00264: -1.602417(226) (in 00:00:07:58s)
Improvement at epoch 00265: -1.602391(352) (in 00:00:08:02s)
Improvement at epoch 00267: -1.602278(291) (in 00:00:08:09s)
Improvement at epoch 00268: -1.602251(107) (in 00:00:08:12s)
Improvement at epoch 00269: -1.602077(059) (in 00:00:08:15s)
Improvement at epoch 00279: -1.602073(202) (in 00:00:08:33s)
Improvement at epoch 00313: -1.601810(456) (in 00:00:09:50s)
Improvement at epoch 00316: -1.601802(092) (in 00:00:10:00s)
Improvement at epoch 00371: -1.591448(444) (in 00:00:11:42s)
Improvement at epoch 00372: -1.586445(068) (in 00:00:11:44s)
Improvement at epoch 00373: -1.582647(248) (in 00:00:11:47s)
Improvement at epoch 00374: -1.581346(043) (in 00:00:11:49s)
Improvement at epoch 00375: -1.581313(254) (in 00:00:11:51s)
Improvement at epoch 00377: -1.581220(286) (in 00:00:11:56s)
Improvement at epoch 00379: -1.581208(177) (in 00:00:12:01s)
Improvement at epoch 00381: -1.580844(187) (in 00:00:12:07s)
Improvement at epoch 00420: -1.580417(247) (in 00:00:13:01s)
Improvement at epoch 00423: -1.580329(436) (in 00:00:13:08s)
Improvement at epoch 00424: -1.579368(115) (in 00:00:13:10s)
Improvement at epoch 00426: -1.578666(458) (in 00:00:13:15s)
Improvement at epoch 00427: -1.578647(065) (in 00:00:13:18s)
Improvement at epoch 00428: -1.578347(325) (in 00:00:13:21s)
Improvement at epoch 00429: -1.577873(487) (in 00:00:13:23s)
Improvement at epoch 00430: -1.577741(055) (in 00:00:13:26s)
Improvement at epoch 00431: -1.577668(118) (in 00:00:13:29s)
Improvement at epoch 00432: -1.577201(235) (in 00:00:13:32s)
Improvement at epoch 00434: -1.576851(168) (in 00:00:13:37s)
Improvement at epoch 00435: -1.576692(499) (in 00:00:13:40s)
Improvement at epoch 00436: -1.576662(221) (in 00:00:13:43s)
Improvement at epoch 00437: -1.575794(217) (in 00:00:13:46s)
Improvement at epoch 00438: -1.574517(335) (in 00:00:13:50s)
Improvement at epoch 00439: -1.571976(457) (in 00:00:13:53s)
Improvement at epoch 00440: -1.571706(276) (in 00:00:13:56s)
Improvement at epoch 00441: -1.570741(373) (in 00:00:13:59s)
Improvement at epoch 00442: -1.570452(225) (in 00:00:14:02s)
Improvement at epoch 00444: -1.570429(328) (in 00:00:14:09s)
```

```
Improvement at epoch 00445: -1.570336(082) (in 00:00:14:13s)
Improvement at epoch 00446: -1.570177(152) (in 00:00:14:16s)
Improvement at epoch 00448: -1.569848(143) (in 00:00:14:23s)
Improvement at epoch 00449: -1.569413(291) (in 00:00:14:26s)
Improvement at epoch 00451: -1.569366(407) (in 00:00:14:33s)
Improvement at epoch 00775: -1.569297(000) (in 00:00:28:58s)
Improvement at epoch 00877: -1.568829(117) (in 00:00:33:51s)
Improvement at epoch 00878: -1.568331(121) (in 00:00:33:55s)
Improvement at epoch 00880: -1.568228(215) (in 00:00:34:03s)
Improvement at epoch 00881: -1.568123(121) (in 00:00:34:08s)
Improvement at epoch 00882: -1.568029(430) (in 00:00:34:12s)
Improvement at epoch 00883: -1.567954(447) (in 00:00:34:16s)
Improvement at epoch 00884: -1.567876(035) (in 00:00:34:20s)
Improvement at epoch 00885: -1.567472(118) (in 00:00:34:25s)
Improvement at epoch 00887: -1.564456(177) (in 00:00:34:33s)
Improvement at epoch 00888: -1.563824(207) (in 00:00:34:38s)
Improvement at epoch 00889: -1.563218(353) (in 00:00:34:42s)
Improvement at epoch 00890: -1.563053(391) (in 00:00:34:46s)
Improvement at epoch 00891: -1.562357(469) (in 00:00:34:51s)
Improvement at epoch 00892: -1.558799(049) (in 00:00:34:55s)
Improvement at epoch 00901: -1.558687(089) (in 00:00:35:35s)
Improvement at epoch 00906: -1.558620(073) (in 00:00:35:56s)
Improvement at epoch 00908: -1.558544(137) (in 00:00:36:05s)
Improvement at epoch 00910: -1.558535(216) (in 00:00:36:14s)
Improvement at epoch 00911: -1.558253(160) (in 00:00:36:18s)
Improvement at epoch 00913: -1.558193(308) (in 00:00:36:27s)
Improvement at epoch 00914: -1.558163(205) (in 00:00:36:31s)
Improvement at epoch 00916: -1.557722(363) (in 00:00:36:41s)
Improvement at epoch 00917: -1.557698(149) (in 00:00:36:45s)
Improvement at epoch 00918: -1.557533(260) (in 00:00:36:50s)
Improvement at epoch 00923: -1.556852(286) (in 00:00:37:12s)
Improvement at epoch 00925: -1.556196(085) (in 00:00:37:22s)
Improvement at epoch 00928: -1.555755(355) (in 00:00:37:36s)
Improvement at epoch 00929: -1.555347(208) (in 00:00:37:40s)
Improvement at epoch 00930: -1.554444(322) (in 00:00:37:45s)
Improvement at epoch 00931: -1.554244(077) (in 00:00:37:49s)
Improvement at epoch 00933: -1.554148(220) (in 00:00:37:58s)
Improvement at epoch 00934: -1.553339(412) (in 00:00:38:03s)
Improvement at epoch 00938: -1.553334(322) (in 00:00:38:21s)
Improvement at epoch 00940: -1.552841(182) (in 00:00:38:30s)
Improvement at epoch 00945: -1.552840(463) (in 00:00:38:53s)
Improvement at epoch 00946: -1.552750(041) (in 00:00:38:57s)
Improvement at epoch 00947: -1.552714(235) (in 00:00:39:02s)
Improvement at epoch 00949: -1.552525(438) (in 00:00:39:11s)
Improvement at epoch 00953: -1.552398(142) (in 00:00:39:29s)
Improvement at epoch 00954: -1.546104(330) (in 00:00:39:34s)
Improvement at epoch 00957: -1.546008(181) (in 00:00:39:47s)
Improvement at epoch 00958: -1.545687(222) (in 00:00:39:52s)
Improvement at epoch 00959: -1.544402(285) (in 00:00:39:56s)
Improvement at epoch 00960: -1.544266(041) (in 00:00:40:01s)
Improvement at epoch 00961: -1.543246(086) (in 00:00:40:05s)
Improvement at epoch 00966: -1.543232(031) (in 00:00:40:28s)
Improvement at epoch 00968: -1.542711(373) (in 00:00:40:38s)
Improvement at epoch 00971: -1.542218(180) (in 00:00:40:52s)
Improvement at epoch 00977: -1.542141(193) (in 00:00:41:20s)
Improvement at epoch 00979: -1.542127(023) (in 00:00:41:30s)
Improvement at epoch 00982: -1.542078(463) (in 00:00:41:45s)
Improvement at epoch 00986: -1.542073(446) (in 00:00:42:04s)
Improvement at epoch 00988: -1.542016(335) (in 00:00:42:14s)
Improvement at epoch 00989: -1.541980(346) (in 00:00:42:19s)
Improvement at epoch 00990: -1.541965(259) (in 00:00:42:24s)
Improvement at epoch 00991: -1.541934(057) (in 00:00:42:29s)
```

```
Improvement at epoch 00992: -1.541708(170) (in 00:00:42:34s)
Improvement at epoch 00994: -1.541696(335) (in 00:00:42:43s)
Improvement at epoch 00995: -1.541660(127) (in 00:00:42:48s)
Improvement at epoch 00996: -1.541592(172) (in 00:00:42:53s)

Learning complete (in 0:0:43:8s)
```

checktest.txt:

```
age_err count cumul_perc
 0  305 0.259133
 1  467 0.655905
 2  213 0.836873
 3   86 0.909941
 4   41 0.944775
 5   29 0.969414
 6   16 0.983008
 7   12 0.993203
 8    3 0.995752
 9    4 0.999150
10    1 1.000000
11    0 1.000000
12    0 1.000000
13    0 1.000000
14    0 1.000000
15    0 1.000000
16    0 1.000000
17    0 1.000000
18    0 1.000000
19    0 1.000000
20    0 1.000000
21    0 1.000000
22    0 1.000000
23    0 1.000000
24    0 1.000000
25    0 1.000000
26    0 1.000000
27    0 1.000000
28    0 1.000000
Value: -1.480951
```

checkall.txt:

```
age_err count cumul_perc
 0 1089 0.260713
 1 1645 0.654537
 2  757 0.835767
 3  294 0.906153
 4  138 0.939191
 5   94 0.961695
 6   53 0.974384
 7   52 0.986833
 8   20 0.991621
 9   15 0.995212
10    8 0.997127
11    7 0.998803
12    3 0.999521
13    1 0.999761
14    0 0.999761
15    0 0.999761
```

```
16    0 0.999761
17    1 1.000000
18    0 1.000000
19    0 1.000000
20    0 1.000000
21    0 1.000000
22    0 1.000000
23    0 1.000000
24    0 1.000000
25    0 1.000000
26    0 1.000000
27    0 1.000000
28    0 1.000000
Value: -1.524505
```

7.3 Arrhythmia data set

Source: https://archive.ics.uci.edu/ml/datasets/arrhythmia

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "genalg.h"
#include "neuranet.h"

// https://archive.ics.uci.edu/ml/datasets/arrhythmia

// Nb of step between each save of the GenAlg
// Saving it allows to restart a stop learning process but is
// very time consuming if there are many input/hidden/output
// If 0 never save
#define SAVE_GA_EVERY 100
// Nb input and output of the NeuraNet
#define NB_INPUT 279
#define NB_OUTPUT 16
// Nb max of hidden values, links and base functions
#define NB_MAXHIDDEN 1
#define NB_MAXLINK 1000
#define NB_MAXBASE NB_MAXLINK
// Size of the gene pool and elite pool
#define ADN_SIZE_POOL 500
#define ADN_SIZE_ELITE 20
// Initial best value during learning, must be lower than any
// possible value returned by Evaluate()
#define INIT_BEST_VAL -100000.0
// Value of the NeuraNet above which the learning process stops
#define STOP_LEARNING_AT_VAL 0.999
// Number of epoch above which the learning process stops
#define STOP_LEARNING_AT_EPOCH 3000
// Save NeuraNet in compact format
#define COMPACT true
```

```
// Categories of data sets

typedef enum DataSetCat {
  unknownDataSet,
  datalearn,
  datatest,
  dataall
} DataSetCat;
#define NB_DATASET 4
const char* dataSetNames[NB_DATASET] = {
  "unknownDataSet", "datalearn", "datatest", "dataall"
  };

const char* catNames[NB_OUTPUT] = {
  "Normal",
  "Ischemic changes (Coronary Artery Disease)",
  "Old Anterior Myocardial Infarction",
  "Old Inferior Myocardial Infarction",
  "Sinus tachycardy",
  "Sinus bradycardy",
  "Ventricular Premature Contraction (PVC)",
  "Supraventricular Premature Contraction",
  "Left bundle branch block",
  "Right bundle branch block",
  "1. degree AtrioVentricular block",
  "2. degree AV block",
  "3. degree AV block",
  "Left ventricule hypertrophy",
  "Atrial Fibrillation or Flutter",
  "Others"
};



typedef struct Arrhytmia {
  float _props[NB_INPUT];
  int _cat;
} Arrhytmia;

typedef struct DataSet {
  // Category of the data set
  DataSetCat _cat;
  // Number of sample
  int _nbSample;
  // Samples
  Arrhytmia* _samples;
} DataSet;

// Get the DataSetCat from its 'name'
DataSetCat GetCategoryFromName(const char* const name) {
  // Declare a variable to memorize the DataSetCat
  DataSetCat cat = unknownDataSet;
  // Search the dataset
  for (int iSet = NB_DATASET; iSet--;)
    if (strcmp(name, dataSetNames[iSet]) == 0)
      cat = iSet;
  // Return the category
  return cat;
}

// Load the data set of category 'cat' in the DataSet 'that'
```

```
// Return true on success, else false
bool DataSetLoad(DataSet* const that, const DataSetCat cat) {
  // Set the category
  that->_cat = cat;

  // Load the data according to 'cat'
  FILE* f = fopen("./arrhythmia.data", "r");
  if (f == NULL) {
    printf("Couldn't open the data set file\n");
    return false;
  }
  int ret = 0;
  if (cat == datalearn) {
    that->_nbSample = 300;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Arrhytmia) * that->_nbSample);
    for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
      for (int iProp = 0; iProp < NB_INPUT; ++iProp) {
        ret = fscanf(f, "%f,",
          that->_samples[iSample]._props + iProp);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
      }
      ret = fscanf(f, "%d", &(that->_samples[iSample]._cat));
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
    }
  } else if (cat == datatest) {
    char buffer[1000];
    for (int iSample = 0; iSample < 300; ++iSample) {
      ret = fscanf(f, "%s", buffer);
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
    }
    that->_nbSample = 152;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Arrhytmia) * that->_nbSample);
    for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
      for (int iProp = 0; iProp < NB_INPUT; ++iProp) {
        ret = fscanf(f, "%f,",
          that->_samples[iSample]._props + iProp);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
      }
      ret = fscanf(f, "%d", &(that->_samples[iSample]._cat));
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
```

```
    }
  } else if (cat == dataall) {
    that->_nbSample = 452;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Arrhytmia) * that->_nbSample);
    for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
      for (int iProp = 0; iProp < NB_INPUT; ++iProp) {
        ret = fscanf(f, "%f,",
          that->_samples[iSample]._props + iProp);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
      }
      ret = fscanf(f, "%d", &(that->_samples[iSample]._cat));
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
    }
  } else {
    printf("Invalid dataset\n");
    fclose(f);
    return false;
  }
  fclose(f);

  // Return success code
  return true;
}

// Free memory for the DataSet 'that'
void DataSetFree(DataSet** that) {
  if (*that == NULL) return;
  // Free the memory
  free((*that)->_samples);
  free(*that);
  *that = NULL;
}

// Evalutation function for the NeuraNet 'that' on the DataSet 'dataset'
// Return the value of the NeuraNet, the bigger the better
float Evaluate(const NeuraNet* const that,
  const DataSet* const dataset) {
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Declare a variable to memorize the value
  float val = 0.0;

  // Evaluate

  int countCat[NB_OUTPUT] = {0};
  int countOk[NB_OUTPUT] = {0};
  int countNg[NB_OUTPUT] = {0};
  for (int iSample = dataset->_nbSample; iSample--;) {
    for (int iInp = 0; iInp < NNGetNbInput(that); ++iInp) {
      VecSet(input, iInp,
        dataset->_samples[iSample]._props[iInp]);
    }
```

```
    NNEval(that, input, output);
    int pred = VecGetIMaxVal(output) + 1;
    ++(countCat[dataset->_samples[iSample]._cat - 1]);
    if (pred == dataset->_samples[iSample]._cat) {
      ++(countOk[dataset->_samples[iSample]._cat - 1]);
    } else if (dataset->_cat == datalearn) {
      ++(countNg[dataset->_samples[iSample]._cat - 1]);
    }

  }
  int nbCat = 0;
  for (int iCat = 0; iCat < NB_OUTPUT; ++iCat) {
    if (countCat[iCat] > 0) {
      ++nbCat;
      float perc = 0.0;
      if (dataset->_cat != datalearn) {
        perc = (float)(countOk[iCat]) / (float)(countCat[iCat]);
        printf("%43s (%3d): %f\n", catNames[iCat], countCat[iCat], perc);
        val += countOk[iCat];
      } else {
        perc = (float)(countOk[iCat] - countNg[iCat]) /
          (float)(countCat[iCat]);
        val += perc;
      }
    }
  }
  if (dataset->_cat != datalearn)
    val /= (float)(dataset->_nbSample);
  else
    val /= (float)nbCat;

  // Free memory
  VecFree(&input);
  VecFree(&output);
  // Return the result of the evaluation
  return val;
}

// Create the NeuraNet
NeuraNet* createNN(void) {
  // Create the NeuraNet
  int nbIn = NB_INPUT;
  int nbOut = NB_OUTPUT;
  int nbMaxHid = NB_MAXHIDDEN;
  int nbMaxLink = NB_MAXLINK;
  int nbMaxBase = NB_MAXBASE;
  NeuraNet* nn =
    NeuraNetCreate(nbIn, nbOut, nbMaxHid, nbMaxBase, nbMaxLink);

  // Return the NeuraNet
  return nn;
}

// Learn based on the DataSetCat 'cat'
void Learn(DataSetCat cat) {
  // Init the random generator
  srandom(time(NULL));
  // Declare variables to measure time
  struct timespec start, stop;
  // Start measuring time
  clock_gettime(CLOCK_REALTIME, &start);
  // Load the DataSet
```

```
DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
bool ret = DataSetLoad(dataset, cat);
if (!ret) {
  printf("Couldn't load the data\n");
  return;
}
// Create the NeuraNet
NeuraNet* nn = createNN();
// Declare a variable to memorize the best value
float bestVal = INIT_BEST_VAL;
// Declare a variable to memorize the limit in term of epoch
unsigned long int limitEpoch = STOP_LEARNING_AT_EPOCH;
// Create the GenAlg used for learning
// If previous weights are available in "./bestga.txt" reload them
GenAlg* ga = NULL;
FILE* fd = fopen("./bestga.txt", "r");
if (fd) {
  printf("Reloading previous GenAlg...\n");
  if (!GALoad(&ga, fd)) {
    printf("Failed to reload the GenAlg.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous GenAlg reloaded.\n");
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GABestAdnF(ga));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GABestAdnI(ga));
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    limitEpoch += GAGetCurEpoch(ga);
  }
  fclose(fd);
} else {
  printf("Creating new GenAlg...\n");
  fflush(stdout);
  ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
    NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  // Must be declared as a GenAlg applied to a NeuraNet with
  // convolution
  GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
  GAInit(ga);
}
// If there is a NeuraNet available, reload it into the GenAlg
fd = fopen("./bestnn.txt", "r");
if (fd) {
  printf("Reloading previous NeuraNet...\n");
  if (!NNLoad(&nn, fd)) {
    printf("Failed to reload the NeuraNet.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous NeuraNet reloaded.\n");
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    GenAlgAdn* adn = GAAdn(ga, 0);
    if (adn->_adnF)
      VecCopy(adn->_adnF, nn->_bases);
```

```
    if (adn->_adnI)
      VecCopy(adn->_adnI, nn->_links);
  }
  fclose(fd);
}
// Start learning process
printf("Learning...\n");
printf("Will stop when curEpoch >= %lu or bestVal >= %f\n",
  limitEpoch, STOP_LEARNING_AT_VAL);
printf("Will save the best NeuraNet in ./bestnn.txt at each improvement\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = 0.0;
float curWorst = 0.0;
// Declare a variable to manage the save of GenAlg
int delaySave = 0;
// Learning loop
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  curWorst = curBest;
  curBest = INIT_BEST_VAL;
  int curBestI = 0;
  unsigned long int ageBest = 0;
  // For each adn in the GenAlg
  //for (int iEnt = GAGetNbAdns(ga); iEnt--;) {
  for (int iEnt = 0; iEnt < GAGetNbAdns(ga); ++iEnt) {
    // Get the adn
    GenAlgAdn* adn = GAAdn(ga, iEnt);
    // Set the links and base functions of the NeuraNet according
    // to this adn
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GAAdnAdnF(adn));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GAAdnAdnI(adn));
    // Evaluate the NeuraNet
    float value = Evaluate(nn, dataset);
    // Update the value of this adn
    GASetAdnValue(ga, adn, value);
    // Update the best value in the current epoch
    if (value > curBest) {
      curBest = value;
      curBestI = iEnt;
      ageBest = GAAdnGetAge(adn);
    }
    if (value < curWorst)
      curWorst = value;
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  // If there has been improvement during this epoch
  if (curBest > bestVal) {
    bestVal = curBest;
    // Display info about the improvment
    printf("Improvement at epoch %05lu: %f(%03d) (in %02d:%02d:%02d:%02ds)       \n",
```

```
      GAGetCurEpoch(ga), bestVal, curBestI, day, hour, min, sec);
    fflush(stdout);
    // Set the links and base functions of the NeuraNet according
    // to the best adn
    GenAlgAdn* bestAdn = GAAdn(ga, curBestI);
    if (GAAdnAdnF(bestAdn) != NULL)
      NNSetBases(nn, GAAdnAdnF(bestAdn));
    if (GAAdnAdnI(bestAdn) != NULL)
      NNSetLinks(nn, GAAdnAdnI(bestAdn));
    // Save the best NeuraNet
    fd = fopen("./bestnn.txt", "w");
    if (!NNSave(nn, fd, COMPACT)) {
      printf("Couldn't save the NeuraNet\n");
      NeuraNetFree(&nn);
      GenAlgFree(&ga);
      DataSetFree(&dataset);
      return;
    }
    fclose(fd);
  } else {
    fprintf(stderr,
      "Epoch %05lu: v%f a%03lu(%03d) kt%03lu ",
      GAGetCurEpoch(ga), curBest, ageBest, curBestI,
      GAGetNbKTEvent(ga));
    fprintf(stderr, "(in %02d:%02d:%02d:%02ds)  \r",
      day, hour, min, sec);
    fflush(stderr);
  }
  ++delaySave;
  if (SAVE_GA_EVERY != 0 && delaySave >= SAVE_GA_EVERY) {
    delaySave = 0;
    // Save the adns of the GenAlg, use a temporary file to avoid
    // loosing the previous one if something goes wrong during
    // writing, then replace the previous file with the temporary one
    fd = fopen("./bestga.tmp", "w");
    if (!GASave(ga, fd, COMPACT)) {
      printf("Couldn't save the GenAlg\n");
      NeuraNetFree(&nn);
      GenAlgFree(&ga);
      DataSetFree(&dataset);
      return;
    }
    fclose(fd);
    int ret = system("mv ./bestga.tmp ./bestga.txt");
    (void)ret;
  }
  // Step the GenAlg
  GAStep(ga);
}
// Measure time
clock_gettime(CLOCK_REALTIME, &stop);
float elapsed = stop.tv_sec - start.tv_sec;
int day = (int)floor(elapsed / 86400);
elapsed -= (float)(day * 86400);
int hour = (int)floor(elapsed / 3600);
elapsed -= (float)(hour * 3600);
int min = (int)floor(elapsed / 60);
elapsed -= (float)(min * 60);
int sec = (int)floor(elapsed);
printf("\nLearning complete (in %d:%d:%d:%ds)\n",
  day, hour, min, sec);
fflush(stdout);
```

```
  // Free memory
  NeuraNetFree(&nn);
  GenAlgFree(&ga);
  DataSetFree(&dataset);
}

// Check the NeuraNet 'that' on the DataSetCat 'cat'
void Validate(const NeuraNet* const that, const DataSetCat cat) {
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Evaluate the NeuraNet
  float value = Evaluate(that, dataset);
  // Display the result
  printf("Value: %.6f\n", value);
  // Free memory
  DataSetFree(&dataset);
}

// Predict using the NeuraNet 'that' on 'inputs' (given as an array of
// 'nbInp' char*)
void Predict(const NeuraNet* const that, const int nbInp,
  char** const inputs) {
  // Start measuring time
  clock_t clockStart = clock();
  // Check the number of inputs
  if (nbInp != NNGetNbInput(that)) {
    printf("Wrong number of inputs, there should %d, there was %d\n",
      NNGetNbInput(that), nbInp);
    return;
  }
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Set the input
  for (int iInp = 0; iInp < nbInp; ++iInp) {
    float v = 0.0;
    sscanf(inputs[iInp], "%f", &v);
    VecSet(input, iInp, v);
  }
  // Predict
  NNEval(that, input, output);
  int pred = VecGetIMaxVal(output);
  // End measuring time
  clock_t clockEnd = clock();
  double timeUsed =
    ((double)(clockEnd - clockStart)) / (CLOCKS_PER_SEC * 0.001) ;
  // If the clock has been reset meanwhile
  if (timeUsed < 0.0)
    timeUsed = 0.0;
  printf("Prediction: %s (in %fms)\n", catNames[pred], timeUsed);

  // Free memory
  VecFree(&input);
  VecFree(&output);
}

int main(int argc, char** argv) {
```

```
  // Declare a variable to memorize the mode (learning/checking)
  int mode = -1;
  // Declare a variable to memorize the dataset used
  DataSetCat cat = unknownDataSet;
  // Decode mode from arguments
  if (argc >= 3) {
    if (strcmp(argv[1], "-learn") == 0) {
      mode = 0;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-check") == 0) {
      mode = 1;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-predict") == 0) {
      mode = 2;
    }
  }
  // If the mode is invalid print some help
  if (mode == -1) {
    printf("Select a mode from:\n");
    printf("-learn <dataset name>\n");
    printf("-check <dataset name>\n");
    printf("-predict <input values>\n");
    return 0;
  }
  if (mode == 0) {
    Learn(cat);
  } else if (mode == 1) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Validate(nn, cat);
    NeuraNetFree(&nn);
  } else if (mode == 2) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Predict(nn, argc - 2, argv + 2);
    NeuraNetFree(&nn);
  }
  // Return success code
  return 0;
}
```

learn.txt:

```
Creating new GenAlg...
Learning...
Will stop when curEpoch >= 20000 or bestVal >= 0.999000
Will save the best NeuraNet in ./bestnn.txt at each improvement
Improvement at epoch 00000: -0.592796(181) (in 00:00:00:00s)
Improvement at epoch 00001: -0.538462(300) (in 00:00:00:00s)
Improvement at epoch 00003: -0.449206(446) (in 00:00:00:00s)
```

```
Improvement at epoch 00004: -0.444007(090) (in 00:00:00:01s)
Improvement at epoch 00006: -0.388047(376) (in 00:00:00:01s)
Improvement at epoch 00008: -0.365458(185) (in 00:00:00:02s)
Improvement at epoch 00010: -0.334996(379) (in 00:00:00:02s)
Improvement at epoch 00012: -0.200050(382) (in 00:00:00:03s)
Improvement at epoch 00013: -0.181649(149) (in 00:00:00:03s)
Improvement at epoch 00020: -0.150722(460) (in 00:00:00:05s)
Improvement at epoch 00022: -0.149817(023) (in 00:00:00:05s)
Improvement at epoch 00028: -0.147353(314) (in 00:00:00:07s)
Improvement at epoch 00029: -0.141270(246) (in 00:00:00:07s)
Improvement at epoch 00030: -0.098535(276) (in 00:00:00:07s)
Improvement at epoch 00031: -0.082346(397) (in 00:00:00:08s)
Improvement at epoch 00032: -0.074073(458) (in 00:00:00:08s)
Improvement at epoch 00033: -0.066431(150) (in 00:00:00:09s)
Improvement at epoch 00034: -0.065526(026) (in 00:00:00:09s)
Improvement at epoch 00053: -0.049758(489) (in 00:00:00:14s)
Improvement at epoch 00054: -0.048853(023) (in 00:00:00:15s)
Improvement at epoch 00055: -0.045358(489) (in 00:00:00:15s)
Improvement at epoch 00056: -0.044453(204) (in 00:00:00:15s)
Improvement at epoch 00071: -0.041084(217) (in 00:00:00:20s)
Improvement at epoch 00073: -0.038958(352) (in 00:00:00:21s)
Improvement at epoch 00074: -0.035590(033) (in 00:00:00:22s)
Improvement at epoch 00075: -0.031065(131) (in 00:00:00:22s)
Improvement at epoch 00076: -0.030285(422) (in 00:00:00:23s)
Improvement at epoch 00077: -0.009087(055) (in 00:00:00:23s)
Improvement at epoch 00078: -0.003782(024) (in 00:00:00:24s)
Improvement at epoch 00079: 0.012290(109) (in 00:00:00:24s)
Improvement at epoch 00080: 0.043964(097) (in 00:00:00:25s)
Improvement at epoch 00081: 0.068045(289) (in 00:00:00:25s)
Improvement at epoch 00082: 0.080255(446) (in 00:00:00:26s)
Improvement at epoch 00083: 0.106633(300) (in 00:00:00:26s)
Improvement at epoch 00084: 0.119708(314) (in 00:00:00:27s)
Improvement at epoch 00085: 0.125013(053) (in 00:00:00:27s)
Improvement at epoch 00086: 0.138738(205) (in 00:00:00:28s)
Improvement at epoch 00087: 0.141476(368) (in 00:00:00:28s)
Improvement at epoch 00089: 0.144043(179) (in 00:00:00:29s)
Improvement at epoch 00105: 0.144948(037) (in 00:00:00:40s)
Improvement at epoch 00113: 0.150253(066) (in 00:00:00:43s)
Improvement at epoch 00114: 0.165461(433) (in 00:00:00:44s)
Improvement at epoch 00116: 0.170766(185) (in 00:00:00:45s)
Improvement at epoch 00148: 0.174261(214) (in 00:00:01:00s)
Improvement at epoch 00149: 0.193516(209) (in 00:00:01:01s)
Improvement at epoch 00150: 0.199726(123) (in 00:00:01:01s)
Improvement at epoch 00151: 0.205031(099) (in 00:00:01:02s)
Improvement at epoch 00159: 0.205936(054) (in 00:00:01:06s)
Improvement at epoch 00166: 0.212146(341) (in 00:00:01:09s)
Improvement at epoch 00167: 0.216546(423) (in 00:00:01:10s)
Improvement at epoch 00168: 0.217451(162) (in 00:00:01:11s)
Improvement at epoch 00212: 0.218356(441) (in 00:00:01:36s)
Improvement at epoch 00214: 0.219261(256) (in 00:00:01:37s)
Improvement at epoch 00258: 0.220166(099) (in 00:00:01:59s)
Improvement at epoch 00265: 0.227808(099) (in 00:00:02:03s)
Improvement at epoch 00266: 0.235450(189) (in 00:00:02:03s)
Improvement at epoch 00267: 0.243092(296) (in 00:00:02:04s)
Improvement at epoch 00268: 0.250734(496) (in 00:00:02:05s)
Improvement at epoch 00294: 0.251639(246) (in 00:00:02:19s)
Improvement at epoch 00296: 0.252544(355) (in 00:00:02:20s)
Improvement at epoch 00329: 0.261450(027) (in 00:00:02:42s)
Improvement at epoch 00330: 0.269997(191) (in 00:00:02:43s)
Improvement at epoch 00331: 0.273681(338) (in 00:00:02:44s)
Improvement at epoch 00332: 0.275491(401) (in 00:00:02:45s)
Improvement at epoch 00333: 0.278271(040) (in 00:00:02:46s)
```

```
Improvement at epoch 00334: 0.280986(232) (in 00:00:02:47s)
Improvement at epoch 00442: 0.281891(112) (in 00:00:03:58s)
Improvement at epoch 00523: 0.282796(027) (in 00:00:04:50s)
Improvement at epoch 00525: 0.283701(246) (in 00:00:04:52s)
Improvement at epoch 00553: 0.303933(284) (in 00:00:05:09s)
Improvement at epoch 00554: 0.310143(110) (in 00:00:05:10s)
Improvement at epoch 00608: 0.316353(068) (in 00:00:05:47s)
Improvement at epoch 00626: 0.317258(322) (in 00:00:05:59s)
Improvement at epoch 00637: 0.318163(396) (in 00:00:06:07s)
Improvement at epoch 00646: 0.323468(414) (in 00:00:06:14s)
Improvement at epoch 00647: 0.327868(492) (in 00:00:06:15s)
Improvement at epoch 00649: 0.328773(023) (in 00:00:06:17s)
Improvement at epoch 00679: 0.332268(359) (in 00:00:06:36s)
Improvement at epoch 00680: 0.334983(037) (in 00:00:06:37s)
Improvement at epoch 00973: 0.335888(224) (in 00:00:10:07s)
Improvement at epoch 01004: 0.338603(481) (in 00:00:10:34s)
Improvement at epoch 01070: 0.339508(115) (in 00:00:11:32s)
Improvement at epoch 01143: 0.341382(221) (in 00:00:12:43s)
Improvement at epoch 01244: 0.342287(234) (in 00:00:14:27s)
Improvement at epoch 01269: 0.343192(127) (in 00:00:14:50s)
Improvement at epoch 01567: 0.344097(073) (in 00:00:20:03s)
Improvement at epoch 01590: 0.350518(488) (in 00:00:20:27s)
Improvement at epoch 01591: 0.351423(027) (in 00:00:20:29s)
Improvement at epoch 01601: 0.352328(058) (in 00:00:20:42s)
Improvement at epoch 01623: 0.353233(141) (in 00:00:21:05s)
Improvement at epoch 01659: 0.354203(250) (in 00:00:21:38s)
Improvement at epoch 01661: 0.356982(081) (in 00:00:21:41s)
Improvement at epoch 01662: 0.360602(383) (in 00:00:21:42s)
Improvement at epoch 01664: 0.361507(355) (in 00:00:21:45s)
Improvement at epoch 01965: 0.363317(116) (in 00:00:26:53s)
Improvement at epoch 02044: 0.364222(368) (in 00:00:28:15s)
Improvement at epoch 02143: 0.375056(421) (in 00:00:30:02s)
Improvement at epoch 02153: 0.375961(173) (in 00:00:30:13s)
Improvement at epoch 02154: 0.382171(205) (in 00:00:30:14s)
Improvement at epoch 02156: 0.394991(306) (in 00:00:30:17s)
Improvement at epoch 02228: 0.398676(132) (in 00:00:31:39s)
Improvement at epoch 02239: 0.399581(272) (in 00:00:31:52s)
Improvement at epoch 02389: 0.405980(069) (in 00:00:34:12s)
Improvement at epoch 02390: 0.407790(194) (in 00:00:34:13s)
Improvement at epoch 02391: 0.412380(020) (in 00:00:34:15s)
Improvement at epoch 02392: 0.413285(201) (in 00:00:34:16s)
Improvement at epoch 02393: 0.417874(036) (in 00:00:34:17s)
Improvement at epoch 02400: 0.422464(231) (in 00:00:34:26s)
Improvement at epoch 02401: 0.431643(210) (in 00:00:34:28s)
Improvement at epoch 02402: 0.438947(456) (in 00:00:34:29s)
Improvement at epoch 02404: 0.441727(058) (in 00:00:34:32s)
Improvement at epoch 02406: 0.453621(107) (in 00:00:34:34s)
Improvement at epoch 02720: 0.454526(221) (in 00:00:40:00s)
Improvement at epoch 02802: 0.458926(419) (in 00:00:41:32s)
Improvement at epoch 02896: 0.478821(367) (in 00:00:43:03s)
Improvement at epoch 02898: 0.487031(030) (in 00:00:43:06s)
Improvement at epoch 02899: 0.487936(020) (in 00:00:43:07s)
Improvement at epoch 02900: 0.496483(023) (in 00:00:43:12s)
Improvement at epoch 02931: 0.497388(269) (in 00:00:43:44s)
Improvement at epoch 02932: 0.498293(118) (in 00:00:43:45s)
Improvement at epoch 02958: 0.499198(497) (in 00:00:44:10s)
```

checktest.txt:

```
                                  Normal ( 75): 0.400000
Ischemic changes (Coronary Artery Disease) ( 16): 0.500000
```

```
        Old Anterior Myocardial Infarction (  6): 0.166667
        Old Inferior Myocardial Infarction (  1): 1.000000
                           Sinus tachycardy (  6): 0.333333
                           Sinus bradycardy (  7): 0.000000
    Ventricular Premature Contraction (PVC) (  2): 0.000000
     Supraventricular Premature Contraction (  1): 0.000000
                  Left bundle branch block (  4): 0.500000
                 Right bundle branch block ( 21): 0.476190
               Left ventricule hypertrophy (  1): 1.000000
            Atrial Fibrillation or Flutter (  2): 0.500000
                                       Others ( 10): 0.000000
Value: 0.368421
```

checkall.txt:

```
                                    Normal (245): 0.510204
 Ischemic changes (Coronary Artery Disease) ( 44): 0.590909
        Old Anterior Myocardial Infarction ( 15): 0.666667
        Old Inferior Myocardial Infarction ( 15): 0.733333
                           Sinus tachycardy ( 13): 0.307692
                           Sinus bradycardy ( 25): 0.240000
    Ventricular Premature Contraction (PVC) (  3): 0.333333
     Supraventricular Premature Contraction (  2): 0.500000
                  Left bundle branch block (  9): 0.777778
                 Right bundle branch block ( 50): 0.660000
               Left ventricule hypertrophy (  4): 0.750000
            Atrial Fibrillation or Flutter (  5): 0.800000
                                       Others ( 22): 0.409091
Value: 0.530973
```

7.4 Wisconsin Diagnostic Breast Cancer

Source: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+Diagnostic

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "genalg.h"
#include "neuranet.h"

// https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

// Nb of step between each save of the GenAlg
// Saving it allows to restart a stop learning process but is
// very time consuming if there are many input/hidden/output
// If 0 never save
#define SAVE_GA_EVERY 100
// Nb input and output of the NeuraNet
#define NB_INPUT 30
#define NB_OUTPUT 2
```

```
// Nb max of hidden values, links and base functions
#define NB_MAXHIDDEN 0
#define NB_MAXLINK 300
#define NB_MAXBASE NB_MAXLINK
// Size of the gene pool and elite pool
#define ADN_SIZE_POOL 100
#define ADN_SIZE_ELITE 20
// Initial best value during learning, must be lower than any
// possible value returned by Evaluate()
#define INIT_BEST_VAL -1000.0
// Value of the NeuraNet above which the learning process stops
#define STOP_LEARNING_AT_VAL 0.999
// Number of epoch above which the learning process stops
#define STOP_LEARNING_AT_EPOCH 2000
// Save NeuraNet in compact format
#define COMPACT true

// Categories of data sets

typedef enum DataSetCat {
  unknownDataSet,
  datalearn,
  datatest,
  dataall
} DataSetCat;
#define NB_DATASET 4
const char* dataSetNames[NB_DATASET] = {
  "unknownDataSet", "datalearn", "datatest", "dataall"
  };

const char* catNames[NB_OUTPUT] = {
  "Malignant",
  "Benign"
};

// Structure for the data set

typedef struct Sample {
  float _props[NB_INPUT];
  int _cat;
  int _id;
} Sample;

typedef struct DataSet {
  // Category of the data set
  DataSetCat _cat;
  // Number of sample
  int _nbSample;
  // Samples
  Sample* _samples;
} DataSet;

// Get the DataSetCat from its 'name'
DataSetCat GetCategoryFromName(const char* const name) {
  // Declare a variable to memorize the DataSetCat
  DataSetCat cat = unknownDataSet;
  // Search the dataset
  for (int iSet = NB_DATASET; iSet--;)
    if (strcmp(name, dataSetNames[iSet]) == 0)
      cat = iSet;
  // Return the category
  return cat;
```

```
}

// Load the data set of category 'cat' in the DataSet 'that'
// Return true on success, else false
bool DataSetLoad(DataSet* const that, const DataSetCat cat) {
  // Set the category
  that->_cat = cat;

  // Load the data according to 'cat'
  FILE* f = fopen("./wdbc.data", "r");
  if (f == NULL) {
    printf("Couldn't open the data set file\n");
    return false;
  }
  int ret = 0;
  if (cat == datalearn) {
    that->_nbSample = 400;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(Sample) * that->_nbSample);
    for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
      ret = fscanf(f, "%d,", &(that->_samples[iSample]._id));
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
      char cat;
      ret = fscanf(f, "%c,", &cat);
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
      if (cat == 'M')
        that->_samples[iSample]._cat = 0;
      else if (cat == 'B')
        that->_samples[iSample]._cat = 1;
      else {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
      for (int iProp = 0; iProp < NB_INPUT; ++iProp) {
        ret = fscanf(f, "%f,",
          that->_samples[iSample]._props + iProp);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
      }
    }
  } else if (cat == datatest) {
    char buffer[1000];
    for (int iSample = 0; iSample < 400; ++iSample) {
      ret = fscanf(f, "%s", buffer);
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
    }
```

```
  that->_nbSample = 169;
  that->_samples =
    PBErrMalloc(NeuraNetErr, sizeof(Sample) * that->_nbSample);
  for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
    ret = fscanf(f, "%d,", &(that->_samples[iSample]._id));
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
    char cat;
    ret = fscanf(f, "%c,", &cat);
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
    if (cat == 'M')
      that->_samples[iSample]._cat = 0;
    else if (cat == 'B')
      that->_samples[iSample]._cat = 1;
    else {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
    for (int iProp = 0; iProp < NB_INPUT; ++iProp) {
      ret = fscanf(f, "%f,",
        that->_samples[iSample]._props + iProp);
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
    }
  }
} else if (cat == dataall) {
  that->_nbSample = 569;
  that->_samples =
    PBErrMalloc(NeuraNetErr, sizeof(Sample) * that->_nbSample);
  for (int iSample = 0; iSample < that->_nbSample; ++iSample) {
    ret = fscanf(f, "%d,", &(that->_samples[iSample]._id));
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
    char cat;
    ret = fscanf(f, "%c,", &cat);
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
    if (cat == 'M')
      that->_samples[iSample]._cat = 0;
    else if (cat == 'B')
      that->_samples[iSample]._cat = 1;
    else {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
```

```
      }
      for (int iProp = 0; iProp < NB_INPUT; ++iProp) {
        ret = fscanf(f, "%f,",
          that->_samples[iSample]._props + iProp);
        if (ret == EOF) {
          printf("Couldn't read the dataset\n");
          fclose(f);
          return false;
        }
      }
    }
  } else {
    printf("Invalid dataset\n");
    fclose(f);
    return false;
  }
  fclose(f);

  // Return success code
  return true;
}

// Free memory for the DataSet 'that'
void DataSetFree(DataSet** that) {
  if (*that == NULL) return;
  // Free the memory

  free(*that);
  *that = NULL;
}

// Evalutation function for the NeuraNet 'that' on the DataSet 'dataset'
// Return the value of the NeuraNet, the bigger the better
float Evaluate(const NeuraNet* const that,
  const DataSet* const dataset) {
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Declare a variable to memorize the value
  float val = 0.0;

  // Evaluate

  int countCat[NB_OUTPUT] = {0};
  int countOk[NB_OUTPUT] = {0};
  int countNg[NB_OUTPUT] = {0};
  for (int iSample = dataset->_nbSample; iSample--;) {
    for (int iInp = 0; iInp < NNGetNbInput(that); ++iInp) {
      VecSet(input, iInp,
        dataset->_samples[iSample]._props[iInp]);
    }
    NNEval(that, input, output);
    int pred = VecGetIMaxVal(output);
    ++(countCat[dataset->_samples[iSample]._cat]);
    if (pred == dataset->_samples[iSample]._cat) {
      ++(countOk[dataset->_samples[iSample]._cat]);
    } else if (dataset->_cat == datalearn) {
      ++(countNg[dataset->_samples[iSample]._cat]);
    }
    /*if (dataset->_cat != datalearn) {
      printf("%010d %10s ", dataset->_samples[iSample]._id,
        catNames[pred]);
```

```
      if (pred == dataset->_samples[iSample]._cat)
        printf("OK");
      else
        printf("NG");
      printf("\n");
    }*/
  }
  int nbCat = 0;
  for (int iCat = 0; iCat < NB_OUTPUT; ++iCat) {
    if (countCat[iCat] > 0) {
      ++nbCat;
      float perc = 0.0;
      if (dataset->_cat != datalearn) {
        perc = (float)(countOk[iCat]) / (float)(countCat[iCat]);
        printf("%10s (%3d): %f\n", catNames[iCat], countCat[iCat], perc);
        val += countOk[iCat];
      } else {
        perc = (float)(countOk[iCat] - countNg[iCat]) /
          (float)(countCat[iCat]);
        val += perc;
      }
    }
  }
  if (dataset->_cat != datalearn)
    val /= (float)(dataset->_nbSample);
  else
    val /= (float)nbCat;

  // Free memory
  VecFree(&input);
  VecFree(&output);
  // Return the result of the evaluation
  return val;
}

// Create the NeuraNet
NeuraNet* createNN(void) {
  // Create the NeuraNet
  int nbIn = NB_INPUT;
  int nbOut = NB_OUTPUT;
  int nbMaxHid = NB_MAXHIDDEN;
  int nbMaxLink = NB_MAXLINK;
  int nbMaxBase = NB_MAXBASE;
  NeuraNet* nn =
    NeuraNetCreate(nbIn, nbOut, nbMaxHid, nbMaxBase, nbMaxLink);

  // Set the links

  //VecShort* links = VecShortCreate(nbMaxLink * NN_NBPARAMLINK);
  //for (int iLink = VecGetDim(links); iLink--;) {
  //  VecSet(links, iLink * NN_NBPARAMLINK, base index
  //    -1 means inactive );
  //  VecSet(links, iLink * NN_NBPARAMLINK + 1, input index );
  //  VecSet(links, iLink * NN_NBPARAMLINK + 2, output index );
  //}
  //NNSetLinks(nn, links);
  //VecFree(&links);

  // Set the bases

  //VecFloat* bases = VecFloatCreate(nbMaxBase * NN_NBPARAMBASE);
  //for (int iBase = VecGetDim(bases); iBases--;) {
```

```
  //  tan(param[0]*NN_THETA)*(x+param[1])+param[2]
  //  param[] in [-1,1]
  //  VecSet(bases, iBase * NN_NBPARAMBASE,  );
  //  VecSet(bases, iBase * NN_NBPARAMBASE + 1,  );
  //  VecSet(bases, iBase * NN_NBPARAMBASE + 2,  );
  //}
  //NNSetBases(nn, bases);
  //VecFree(&bases);

  // Return the NeuraNet
  return nn;
}

// Learn based on the DataSetCat 'cat'
void Learn(DataSetCat cat) {
  // Init the random generator
  srandom(time(NULL));
  // Declare variables to measure time
  struct timespec start, stop;
  // Start measuring time
  clock_gettime(CLOCK_REALTIME, &start);
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Create the NeuraNet
  NeuraNet* nn = createNN();
  // Declare a variable to memorize the best value
  float bestVal = INIT_BEST_VAL;
  // Declare a variable to memorize the limit in term of epoch
  unsigned long int limitEpoch = STOP_LEARNING_AT_EPOCH;
  // Create the GenAlg used for learning
  // If previous weights are available in "./bestga.txt" reload them
  GenAlg* ga = NULL;
  FILE* fd = fopen("./bestga.txt", "r");
  if (fd) {
    printf("Reloading previous GenAlg...\n");
    if (!GALoad(&ga, fd)) {
      printf("Failed to reload the GenAlg.\n");
      NeuraNetFree(&nn);
      DataSetFree(&dataset);
      return;
    } else {
      printf("Previous GenAlg reloaded.\n");
      if (GABestAdnF(ga) != NULL)
        NNSetBases(nn, GABestAdnF(ga));
      if (GABestAdnI(ga) != NULL)
        NNSetLinks(nn, GABestAdnI(ga));
      bestVal = Evaluate(nn, dataset);
      printf("Starting with best at %f.\n", bestVal);
      limitEpoch += GAGetCurEpoch(ga);
    }
    fclose(fd);
  } else {
    printf("Creating new GenAlg...\n");
    fflush(stdout);
    ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
      NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
    NNSetGABoundsBases(nn, ga);
```

```
  NNSetGABoundsLinks(nn, ga);
  // Must be declared as a GenAlg applied to a NeuraNet with
  // convolution
  GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
  GAInit(ga);
}
// If there is a NeuraNet available, reload it into the GenAlg
fd = fopen("./bestnn.txt", "r");
if (fd) {
  printf("Reloading previous NeuraNet...\n");
  if (!NNLoad(&nn, fd)) {
    printf("Failed to reload the NeuraNet.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous NeuraNet reloaded.\n");
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    GenAlgAdn* adn = GAAdn(ga, 0);
    if (adn->_adnF)
      VecCopy(adn->_adnF, nn->_bases);
    if (adn->_adnI)
      VecCopy(adn->_adnI, nn->_links);
  }
  fclose(fd);
}
// Start learning process
printf("Learning...\n");
printf("Will stop when curEpoch >= %lu or bestVal >= %f\n",
  limitEpoch, STOP_LEARNING_AT_VAL);
printf("Will save the best NeuraNet in ./bestnn.txt at each improvement\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = 0.0;
float curWorst = 0.0;
// Declare a variable to manage the save of GenAlg
int delaySave = 0;
// Learning loop
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  curWorst = curBest;
  curBest = INIT_BEST_VAL;
  int curBestI = 0;
  unsigned long int ageBest = 0;
  // For each adn in the GenAlg
  //for (int iEnt = GAGetNbAdns(ga); iEnt--;) {
  for (int iEnt = 0; iEnt < GAGetNbAdns(ga); ++iEnt) {
    // Get the adn
    GenAlgAdn* adn = GAAdn(ga, iEnt);
    // Set the links and base functions of the NeuraNet according
    // to this adn
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GAAdnAdnF(adn));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GAAdnAdnI(adn));
    // Evaluate the NeuraNet
    float value = Evaluate(nn, dataset);
    // Update the value of this adn
    GASetAdnValue(ga, adn, value);
    // Update the best value in the current epoch
    if (value > curBest) {
```

```
    curBest = value;
    curBestI = iEnt;
    ageBest = GAAdnGetAge(adn);
  }
  if (value < curWorst)
    curWorst = value;
}
// Measure time
clock_gettime(CLOCK_REALTIME, &stop);
float elapsed = stop.tv_sec - start.tv_sec;
int day = (int)floor(elapsed / 86400);
elapsed -= (float)(day * 86400);
int hour = (int)floor(elapsed / 3600);
elapsed -= (float)(hour * 3600);
int min = (int)floor(elapsed / 60);
elapsed -= (float)(min * 60);
int sec = (int)floor(elapsed);
// If there has been improvement during this epoch
if (curBest > bestVal) {
  bestVal = curBest;
  // Display info about the improvment
  printf("Improvement at epoch %05lu: %f(%03d) (in %02d:%02d:%02d:%02ds)       \n",
    GAGetCurEpoch(ga), bestVal, curBestI, day, hour, min, sec);
  fflush(stdout);
  // Set the links and base functions of the NeuraNet according
  // to the best adn
  GenAlgAdn* bestAdn = GAAdn(ga, curBestI);
  if (GAAdnAdnF(bestAdn) != NULL)
    NNSetBases(nn, GAAdnAdnF(bestAdn));
  if (GAAdnAdnI(bestAdn) != NULL)
    NNSetLinks(nn, GAAdnAdnI(bestAdn));
  // Save the best NeuraNet
  fd = fopen("./bestnn.txt", "w");
  if (!NNSave(nn, fd, COMPACT)) {
    printf("Couldn't save the NeuraNet\n");
    NeuraNetFree(&nn);
    GenAlgFree(&ga);
    DataSetFree(&dataset);
    return;
  }
  fclose(fd);
} else {
  fprintf(stderr,
    "Epoch %05lu: v%f a%03lu(%03d) kt%03lu ",
    GAGetCurEpoch(ga), curBest, ageBest, curBestI,
    GAGetNbKTEvent(ga));
  fprintf(stderr, "(in %02d:%02d:%02d:%02ds)  \r",
    day, hour, min, sec);
  fflush(stderr);
}
++delaySave;
if (SAVE_GA_EVERY != 0 && delaySave >= SAVE_GA_EVERY) {
  delaySave = 0;
  // Save the adns of the GenAlg, use a temporary file to avoid
  // loosing the previous one if something goes wrong during
  // writing, then replace the previous file with the temporary one
  fd = fopen("./bestga.tmp", "w");
  if (!GASave(ga, fd, COMPACT)) {
    printf("Couldn't save the GenAlg\n");
    NeuraNetFree(&nn);
    GenAlgFree(&ga);
    DataSetFree(&dataset);
```

```
        return;
      }
      fclose(fd);
      int ret = system("mv ./bestga.tmp ./bestga.txt");
      (void)ret;
    }
    // Step the GenAlg
    GAStep(ga);
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  printf("\nLearning complete (in %d:%d:%d:%ds)\n",
    day, hour, min, sec);
  fflush(stdout);
  // Free memory
  NeuraNetFree(&nn);
  GenAlgFree(&ga);
  DataSetFree(&dataset);
}

// Check the NeuraNet 'that' on the DataSetCat 'cat'
void Check(const NeuraNet* const that, const DataSetCat cat) {
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Evaluate the NeuraNet
  float value = Evaluate(that, dataset);
  // Display the result
  printf("Value: %.6f\n", value);
  // Free memory
  DataSetFree(&dataset);
}

// Predict using the NeuraNet 'that' on 'inputs' (given as an array of
// 'nbInp' char*)
void Predict(const NeuraNet* const that, const int nbInp,
  char** const inputs) {
  // Start measuring time
  clock_t clockStart = clock();
  // Check the number of inputs
  if (nbInp != NNGetNbInput(that)) {
    printf("Wrong number of inputs, there should %d, there was %d\n",
      NNGetNbInput(that), nbInp);
    return;
  }
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Set the input
  for (int iInp = 0; iInp < nbInp; ++iInp) {
```

```
    float v = 0.0;
    sscanf(inputs[iInp], "%f", &v);
    VecSet(input, iInp, v);
  }
  // Predict
  NNEval(that, input, output);

  // End measuring time
  clock_t clockEnd = clock();
  double timeUsed =
    ((double)(clockEnd - clockStart)) / (CLOCKS_PER_SEC * 0.001) ;
  // If the clock has been reset meanwhile
  if (timeUsed < 0.0)
    timeUsed = 0.0;
  //if (VecGet(output, 0) == ...)
  //  printf("...(in %fms)", timeUsed);

  // Free memory
  VecFree(&input);
  VecFree(&output);
}

int main(int argc, char** argv) {
  // Declare a variable to memorize the mode (learning/checking)
  int mode = -1;
  // Declare a variable to memorize the dataset used
  DataSetCat cat = unknownDataSet;
  // Decode mode from arguments
  if (argc >= 3) {
    if (strcmp(argv[1], "-learn") == 0) {
      mode = 0;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-check") == 0) {
      mode = 1;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-predict") == 0) {
      mode = 2;
    }
  }
  // If the mode is invalid print some help
  if (mode == -1) {
    printf("Select a mode from:\n");
    printf("-learn <dataset name>\n");
    printf("-check <dataset name>\n");
    printf("-predict <input values>\n");
    return 0;
  }
  if (mode == 0) {
    Learn(cat);
  } else if (mode == 1) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Check(nn, cat);
    NeuraNetFree(&nn);
  } else if (mode == 2) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
```

```
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Predict(nn, argc - 2, argv + 2);
    NeuraNetFree(&nn);
  }
  // Return success code
  return 0;
}
```

learn.txt:

```
Creating new GenAlg...
Learning...
Will stop when curEpoch >= 2000 or bestVal >= 0.999000
Will save the best NeuraNet in ./bestnn.txt at each improvement
Improvement at epoch 00000: 0.535561(074) (in 00:00:00:00s)
Improvement at epoch 00002: 0.562833(090) (in 00:00:00:00s)
Improvement at epoch 00003: 0.569988(060) (in 00:00:00:00s)
Improvement at epoch 00004: 0.635507(026) (in 00:00:00:00s)
Improvement at epoch 00009: 0.680910(087) (in 00:00:00:00s)
Improvement at epoch 00014: 0.685340(052) (in 00:00:00:00s)
Improvement at epoch 00015: 0.695246(072) (in 00:00:00:01s)
Improvement at epoch 00016: 0.696621(048) (in 00:00:00:01s)
Improvement at epoch 00017: 0.705431(023) (in 00:00:00:01s)
Improvement at epoch 00018: 0.715617(034) (in 00:00:00:01s)
Improvement at epoch 00019: 0.757455(099) (in 00:00:00:01s)
Improvement at epoch 00021: 0.798477(082) (in 00:00:00:01s)
Improvement at epoch 00027: 0.826284(022) (in 00:00:00:01s)
Improvement at epoch 00028: 0.832064(040) (in 00:00:00:01s)
Improvement at epoch 00029: 0.843625(079) (in 00:00:00:01s)
Improvement at epoch 00030: 0.858216(038) (in 00:00:00:01s)
Improvement at epoch 00032: 0.883553(091) (in 00:00:00:01s)
Improvement at epoch 00033: 0.901174(030) (in 00:00:00:01s)
Improvement at epoch 00037: 0.906674(080) (in 00:00:00:02s)
Improvement at epoch 00039: 0.911079(096) (in 00:00:00:02s)
Improvement at epoch 00040: 0.915485(026) (in 00:00:00:02s)
Improvement at epoch 00042: 0.918515(033) (in 00:00:00:02s)
Improvement at epoch 00047: 0.919890(073) (in 00:00:00:02s)
Improvement at epoch 00053: 0.921545(043) (in 00:00:00:03s)
Improvement at epoch 00058: 0.922920(059) (in 00:00:00:03s)
Improvement at epoch 00069: 0.928701(071) (in 00:00:00:04s)
Improvement at epoch 00070: 0.935856(030) (in 00:00:00:04s)
Improvement at epoch 00086: 0.937511(058) (in 00:00:00:05s)
Improvement at epoch 00239: 0.940541(043) (in 00:00:00:11s)
Improvement at epoch 00244: 0.946322(098) (in 00:00:00:12s)
Improvement at epoch 00449: 0.948792(021) (in 00:00:00:20s)
Improvement at epoch 00453: 0.951822(038) (in 00:00:00:20s)
Improvement at epoch 00464: 0.953197(037) (in 00:00:00:21s)
Improvement at epoch 00468: 0.956227(036) (in 00:00:00:21s)
Improvement at epoch 00470: 0.957602(096) (in 00:00:00:21s)
Improvement at epoch 00753: 0.959538(071) (in 00:00:00:32s)
Improvement at epoch 01936: 0.960633(087) (in 00:00:01:29s)
Improvement at epoch 01972: 0.962008(023) (in 00:00:01:31s)
Improvement at epoch 01983: 0.966413(092) (in 00:00:01:31s)

Learning complete (in 0:0:1:32s)
```

checktest.txt:

```
 Malignant ( 39): 1.000000
    Benign (130): 0.938462
Value: 0.952663
```

checkall.txt:

```
 Malignant (212): 0.990566
    Benign (357): 0.963585
Value: 0.973638
```

7.5 MNIST

Source: http://yann.lecun.com/exdb/mnist/

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "genalg.h"
#include "neuranet.h"

// http://yann.lecun.com/exdb/mnist/

// Nb of step between each save of the GenAlg
// Saving it allows to restart a stop learning process but is
// very time consuming if there are many input/hidden/output
// If 0 never save
#define SAVE_GA_EVERY 100
// Nb input and output of the NeuraNet
#define MNIST_IMGSIZE 28
#define NB_INPUT MNIST_IMGSIZE * MNIST_IMGSIZE
#define NB_OUTPUT 10
// Nb max of hidden values, links and base functions
#define NB_MAXHIDDEN 1
#define NB_MAXLINK 500
#define NB_MAXBASE NB_MAXLINK
// Size of the gene pool and elite pool
#define ADN_SIZE_POOL 100
#define ADN_SIZE_ELITE 20
// Initial best value during learning, must be lower than any
// possible value returned by Evaluate()
#define INIT_BEST_VAL -1000.0
// Value of the NeuraNet above which the learning process stops
#define STOP_LEARNING_AT_VAL 0.999
// Number of epoch above which the learning process stops
#define STOP_LEARNING_AT_EPOCH 20000
// Save NeuraNet in compact format
#define COMPACT true
```

```
// Categories of data sets

typedef enum DataSetCat {
  unknownDataSet,
  datalearn,
  datatest,
  dataall
} DataSetCat;
#define NB_DATASET 4
const char* dataSetNames[NB_DATASET] = {
  "unknownDataSet", "datalearn", "datatest", "dataall"
  };

const char* catNames[NB_OUTPUT] = {
  "0",
  "1",
  "2",
  "3",
  "4",
  "5",
  "6",
  "7",
  "8",
  "9",
};

// Structure for the data set

typedef struct MNISTImg {
  unsigned char _cat;
  unsigned char _pixels[MNIST_IMGSIZE * MNIST_IMGSIZE];
} MNISTImg;

void MNISTImgPrintln(MNISTImg* img) {
  for (int i = 0; i < MNIST_IMGSIZE; ++i) {
    for (int j = 0; j < MNIST_IMGSIZE; ++j) {
      if (img->_pixels[i * MNIST_IMGSIZE + j] > 127)
        printf("#");
      else
        printf(" ");
    }
    printf("\n");
  }
}

typedef struct MNIST {
  int _nbImg;
  MNISTImg* _imgs;
} MNIST;

void MNISTFree(MNIST** that) {
  free((*that)->_imgs);
  free(*that);
  *that = NULL;
}

typedef struct DataSet {
  // Category of the data set
  DataSetCat _cat;
  // Number of sample
  unsigned int _nbMNISTImg;
```

```
  // MNISTImgs
  MNISTImg* _samples;
} DataSet;

// Get the DataSetCat from its 'name'
DataSetCat GetCategoryFromName(const char* const name) {
  // Declare a variable to memorize the DataSetCat
  DataSetCat cat = unknownDataSet;
  // Search the dataset
  for (int iSet = NB_DATASET; iSet--;)
    if (strcmp(name, dataSetNames[iSet]) == 0)
      cat = iSet;
  // Return the category
  return cat;
}

// Load the data set of category 'cat' in the DataSet 'that'
// Return true on success, else false

MNIST* MNISTLoad(char* fnLbl, char* fnImg) {
  FILE* fLbl = fopen(fnLbl, "rb");
  if (!fLbl) {
    printf("Couldn't open %s\n", fnLbl);
    return NULL;
  }
  FILE* fImg = fopen(fnImg, "rb");
  if (!fImg) {
    printf("Couldn't open %s\n", fnImg);
    fclose(fLbl);
    return NULL;
  }
  MNIST* mnist = PBErrMalloc(&thePBErr, sizeof(MNIST));
  int buff;
  int ret;
  // Magic number
  for (int i = 4; i--;)
    ret = fread((char*)(&buff) + i, 1, 1, fLbl);
  if (buff != 2049) {
    printf("Magic number for %s is invalid (%d==2049)\n", fnLbl, buff);
    fclose(fLbl);
    fclose(fImg);
    return NULL;
  }
  for (int i = 4; i--;)
    ret = fread((char*)(&buff) + i, 1, 1, fImg);
  if (buff != 2051) {
    printf("Magic number for %s is invalid (%d==2051)\n", fnLbl, buff);
    fclose(fLbl);
    fclose(fImg);
    return NULL;
  }
  // Number of items
  for (int i = 4; i--;)
    ret = fread((char*)(&(mnist->_nbImg)) + i, 1, 1, fLbl);
  for (int i = 4; i--;)
    ret = fread((char*)(&buff) + i, 1, 1, fImg);
  if (buff != mnist->_nbImg) {
    printf("Nb of items doesn't match (%d==%d)\n", buff, mnist->_nbImg);
    fclose(fLbl);
    fclose(fImg);
    return NULL;
  }
```

```
  // Number of rows and columns
  for (int i = 4; i--;)
    ret = fread((char*)(&buff) + i, 1, 1, fImg);
  if (buff != 28) {
    printf("Unexpected image size (rows) (%d==%d)\n",
      buff, 28);
    fclose(fLbl);
    fclose(fImg);
    return NULL;
  }
  for (int i = 4; i--;)
    ret = fread((char*)(&buff) + i, 1, 1, fImg);
  if (buff != 28) {
    printf("Unexpected image size (columns) (%d==%d)\n",
      buff, 28);
    fclose(fLbl);
    fclose(fImg);
    return NULL;
  }
  // Images
  printf("Loading %d images...\n", mnist->_nbImg);
  mnist->_imgs =
    PBErrMalloc(&thePBErr, sizeof(MNISTImg) * mnist->_nbImg);
  for (int iImg = 0; iImg < mnist->_nbImg; ++iImg) {
    MNISTImg* img = mnist->_imgs + iImg;
    // Label
    ret = fread(&(img->_cat), 1, 1, fLbl);
    // Pixels
    for (int iPixel = 0; iPixel < MNIST_IMGSIZE * MNIST_IMGSIZE;
      ++iPixel) {
      ret = fread(img->_pixels + iPixel, 1, 1, fImg);
    }
  }
  printf("Loaded MNIST successfully.\n");
  fflush(stdout);
  fclose(fImg);
  fclose(fLbl);
  (void)ret;
  return mnist;
}

bool DataSetLoad(DataSet* const that, const DataSetCat cat) {
  // Set the category
  that->_cat = cat;
  // Load the data according to 'cat'
  MNIST* mnist =
    MNISTLoad("train-labels.idx1-ubyte", "train-images.idx3-ubyte");
  if (!mnist) {
    printf("Couldn't load the MNIST data\n");
    return false;
  }
  if (cat == datalearn) {
    that->_nbMNISTImg = 50000;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(MNISTImg) * that->_nbMNISTImg);
    memcpy(that->_samples, mnist->_imgs,
      sizeof(MNISTImg) * that->_nbMNISTImg);
  } else if (cat == datatest) {
    that->_nbMNISTImg = 10000;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(MNISTImg) * that->_nbMNISTImg);
    memcpy(that->_samples, mnist->_imgs + 50000,
```

```
      sizeof(MNISTImg) * that->_nbMNISTImg);
  } else if (cat == dataall) {
    that->_nbMNISTImg = 60000;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(MNISTImg) * that->_nbMNISTImg);
    memcpy(that->_samples, mnist->_imgs,
      sizeof(MNISTImg) * that->_nbMNISTImg);
  } else {
    printf("Invalid dataset\n");
    MNISTFree(&mnist);
    return false;
  }
  MNISTFree(&mnist);
  printf("Created dataset with %u samples\n", that->_nbMNISTImg);
  fflush(stdout);
  // Return success code
  return true;
}

// Free memory for the DataSet 'that'
void DataSetFree(DataSet** that) {
  if (*that == NULL) return;
  // Free the memory

  free(*that);
  *that = NULL;
}

// Evalutation function for the NeuraNet 'that' on the DataSet 'dataset'
// Return the value of the NeuraNet, the bigger the better
float Evaluate(const NeuraNet* const that,
  const DataSet* const dataset) {
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Declare a variable to memorize the value
  float val = 0.0;

  // Evaluate

  int countCat[NB_OUTPUT] = {0};
  int countOk[NB_OUTPUT] = {0};
  int countNg[NB_OUTPUT] = {0};
  for (unsigned int iMNISTImg = dataset->_nbMNISTImg; iMNISTImg--;) {
    for (int iInp = 0; iInp < NNGetNbInput(that); ++iInp) {
      VecSet(input, iInp,
        dataset->_samples[iMNISTImg]._pixels[iInp]);
    }
    NNEval(that, input, output);
    int pred = VecGetIMaxVal(output);
    ++(countCat[dataset->_samples[iMNISTImg]._cat]);
    if (pred == dataset->_samples[iMNISTImg]._cat) {
      ++(countOk[dataset->_samples[iMNISTImg]._cat]);
    } else if (dataset->_cat == datalearn) {
      ++(countNg[dataset->_samples[iMNISTImg]._cat]);
    }
  }
  int nbCat = 0;
  for (int iCat = 0; iCat < NB_OUTPUT; ++iCat) {
    if (countCat[iCat] > 0) {
      ++nbCat;
      float perc = 0.0;
```

```
      if (dataset->_cat != datalearn) {
        perc = (float)(countOk[iCat]) / (float)(countCat[iCat]);
        printf("%10s (%4d): %f\n",
          catNames[iCat], countCat[iCat], perc);
        val += countOk[iCat];
      } else {
        perc = (float)(countOk[iCat] - countNg[iCat]) /
          (float)(countCat[iCat]);
        val += perc;
      }
    }
  }
  if (dataset->_cat != datalearn)
    val /= (float)(dataset->_nbMNISTImg);
  else
    val /= (float)nbCat;

  // Free memory
  VecFree(&input);
  VecFree(&output);
  // Return the result of the evaluation
  return val;
}

// Create the NeuraNet
NeuraNet* createNN(void) {
  // Create the NeuraNet
  int nbIn = NB_INPUT;
  int nbOut = NB_OUTPUT;
  int nbMaxHid = NB_MAXHIDDEN;
  int nbMaxLink = NB_MAXLINK;
  int nbMaxBase = NB_MAXBASE;
  NeuraNet* nn =
    NeuraNetCreate(nbIn, nbOut, nbMaxHid, nbMaxBase, nbMaxLink);
  // Return the NeuraNet
  return nn;
}

// Learn based on the DataSetCat 'cat'
void Learn(DataSetCat cat) {
  // Init the random generator
  srandom(time(NULL));
  // Declare variables to measure time
  struct timespec start, stop;
  // Start measuring time
  clock_gettime(CLOCK_REALTIME, &start);
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Create the NeuraNet
  NeuraNet* nn = createNN();
  // Declare a variable to memorize the best value
  float bestVal = INIT_BEST_VAL;
  // Declare a variable to memorize the limit in term of epoch
  unsigned long int limitEpoch = STOP_LEARNING_AT_EPOCH;
  // Create the GenAlg used for learning
  // If previous weights are available in "./bestga.txt" reload them
  GenAlg* ga = NULL;
```

```
FILE* fd = fopen("./bestga.txt", "r");
if (fd) {
  printf("Reloading previous GenAlg...\n");
  if (!GALoad(&ga, fd)) {
    printf("Failed to reload the GenAlg.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous GenAlg reloaded.\n");
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GABestAdnF(ga));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GABestAdnI(ga));
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    limitEpoch += GAGetCurEpoch(ga);
  }
  fclose(fd);
} else {
  printf("Creating new GenAlg...\n");
  fflush(stdout);
  ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
    NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  // Must be declared as a GenAlg applied to a NeuraNet with
  // convolution
  GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
  GAInit(ga);
}
// If there is a NeuraNet available, reload it into the GenAlg
fd = fopen("./bestnn.txt", "r");
if (fd) {
  printf("Reloading previous NeuraNet...\n");
  if (!NNLoad(&nn, fd)) {
    printf("Failed to reload the NeuraNet.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous NeuraNet reloaded.\n");
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    GenAlgAdn* adn = GAAdn(ga, 0);
    if (adn->_adnF)
      VecCopy(adn->_adnF, nn->_bases);
    if (adn->_adnI)
      VecCopy(adn->_adnI, nn->_links);
  }
  fclose(fd);
}
// Start learning process
printf("Learning...\n");
printf("Will stop when curEpoch >= %lu or bestVal >= %f\n",
  limitEpoch, STOP_LEARNING_AT_VAL);
printf("Will save the best NeuraNet in ./bestnn.txt at each improvement\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = 0.0;
float curWorst = 0.0;
// Declare a variable to manage the save of GenAlg
```

```
int delaySave = 0;
// Learning loop
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  curWorst = curBest;
  curBest = INIT_BEST_VAL;
  int curBestI = 0;
  unsigned long int ageBest = 0;
  // For each adn in the GenAlg
  //for (int iEnt = GAGetNbAdns(ga); iEnt--;) {
  for (int iEnt = 0; iEnt < GAGetNbAdns(ga); ++iEnt) {
    // Get the adn
    GenAlgAdn* adn = GAAdn(ga, iEnt);
    // Set the links and base functions of the NeuraNet according
    // to this adn
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GAAdnAdnF(adn));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GAAdnAdnI(adn));
    // Evaluate the NeuraNet
    float value = Evaluate(nn, dataset);
    // Update the value of this adn
    GASetAdnValue(ga, adn, value);
    // Update the best value in the current epoch
    if (value > curBest) {
      curBest = value;
      curBestI = iEnt;
      ageBest = GAAdnGetAge(adn);
    }
    if (value < curWorst)
      curWorst = value;
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  // If there has been improvement during this epoch
  if (curBest > bestVal) {
    bestVal = curBest;
    // Display info about the improvment
    printf("Improvement at epoch %05lu: %f(%03d) (in %02d:%02d:%02d:%02ds)       \n",
      GAGetCurEpoch(ga), bestVal, curBestI, day, hour, min, sec);
    fflush(stdout);
    // Set the links and base functions of the NeuraNet according
    // to the best adn
    GenAlgAdn* bestAdn = GAAdn(ga, curBestI);
    if (GAAdnAdnF(bestAdn) != NULL)
      NNSetBases(nn, GAAdnAdnF(bestAdn));
    if (GAAdnAdnI(bestAdn) != NULL)
      NNSetLinks(nn, GAAdnAdnI(bestAdn));
    // Save the best NeuraNet
    fd = fopen("./bestnn.txt", "w");
    if (!NNSave(nn, fd, COMPACT)) {
      printf("Couldn't save the NeuraNet\n");
      NeuraNetFree(&nn);
      GenAlgFree(&ga);
```

```
        DataSetFree(&dataset);
        return;
      }
      fclose(fd);
    } else {
      fprintf(stderr,
        "Epoch %05lu: v%f a%03lu(%03d) kt%03lu ",
        GAGetCurEpoch(ga), curBest, ageBest, curBestI,
        GAGetNbKTEvent(ga));
      fprintf(stderr, "(in %02d:%02d:%02d:%02ds)  \r",
        day, hour, min, sec);
      fflush(stderr);
    }
    ++delaySave;
    if (SAVE_GA_EVERY != 0 && delaySave >= SAVE_GA_EVERY) {
      delaySave = 0;
      // Save the adns of the GenAlg, use a temporary file to avoid
      // loosing the previous one if something goes wrong during
      // writing, then replace the previous file with the temporary one
      fd = fopen("./bestga.tmp", "w");
      if (!GASave(ga, fd, COMPACT)) {
        printf("Couldn't save the GenAlg\n");
        NeuraNetFree(&nn);
        GenAlgFree(&ga);
        DataSetFree(&dataset);
        return;
      }
      fclose(fd);
      int ret = system("mv ./bestga.tmp ./bestga.txt");
      (void)ret;
    }
    // Step the GenAlg
    GAStep(ga);
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  printf("\nLearning complete (in %d:%d:%d:%ds)\n",
    day, hour, min, sec);
  fflush(stdout);
  // Free memory
  NeuraNetFree(&nn);
  GenAlgFree(&ga);
  DataSetFree(&dataset);
}

// Check the NeuraNet 'that' on the DataSetCat 'cat'
void Check(const NeuraNet* const that, const DataSetCat cat) {
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
```

```
  // Evaluate the NeuraNet
  float value = Evaluate(that, dataset);
  // Display the result
  printf("Value: %.6f\n", value);
  // Free memory
  DataSetFree(&dataset);
}

// Predict using the NeuraNet 'that' on 'inputs' (given as an array of
// 'nbInp' char*)
void Predict(const NeuraNet* const that, const int nbInp,
  char** const inputs) {
  // Start measuring time
  clock_t clockStart = clock();
  // Check the number of inputs
  if (nbInp != NNGetNbInput(that)) {
    printf("Wrong number of inputs, there should %d, there was %d\n",
      NNGetNbInput(that), nbInp);
    return;
  }
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Set the input
  for (int iInp = 0; iInp < nbInp; ++iInp) {
    float v = 0.0;
    sscanf(inputs[iInp], "%f", &v);
    VecSet(input, iInp, v);
  }
  // Predict
  NNEval(that, input, output);

  // End measuring time
  clock_t clockEnd = clock();
  double timeUsed =
    ((double)(clockEnd - clockStart)) / (CLOCKS_PER_SEC * 0.001) ;
  // If the clock has been reset meanwhile
  if (timeUsed < 0.0)
    timeUsed = 0.0;
  //if (VecGet(output, 0) == ...)
  //  printf("...(in %fms)", timeUsed);

  // Free memory
  VecFree(&input);
  VecFree(&output);
}

int main(int argc, char** argv) {
  // Declare a variable to memorize the mode (learning/checking)
  int mode = -1;
  // Declare a variable to memorize the dataset used
  DataSetCat cat = unknownDataSet;
  // Decode mode from arguments
  if (argc >= 3) {
    if (strcmp(argv[1], "-learn") == 0) {
      mode = 0;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-check") == 0) {
      mode = 1;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-predict") == 0) {
      mode = 2;
```

```
    }
  }
  // If the mode is invalid print some help
  if (mode == -1) {
    printf("Select a mode from:\n");
    printf("-learn <dataset name>\n");
    printf("-check <dataset name>\n");
    printf("-predict <input values>\n");
    return 0;
  }
  if (mode == 0) {
    Learn(cat);
  } else if (mode == 1) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Check(nn, cat);
    NeuraNetFree(&nn);
  } else if (mode == 2) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Predict(nn, argc - 2, argv + 2);
    NeuraNetFree(&nn);
  }
  // Return success code
  return 0;
}
```

learn.txt:

```
Loading 60000 images...
Loaded MNIST successfully.
Created dataset with 50000 samples
Creating new GenAlg...
Learning...
Will stop when curEpoch >= 20000 or bestVal >= 0.999000
Will save the best NeuraNet in ./bestnn.txt at each improvement
Improvement at epoch 00000: -0.608666(082) (in 00:00:00:07s)
Improvement at epoch 00001: -0.544056(049) (in 00:00:00:13s)
Improvement at epoch 00003: -0.523350(080) (in 00:00:00:25s)
Improvement at epoch 00005: -0.515525(049) (in 00:00:00:38s)
Improvement at epoch 00006: -0.502999(030) (in 00:00:00:45s)
Improvement at epoch 00007: -0.475557(025) (in 00:00:00:51s)
Improvement at epoch 00008: -0.448818(074) (in 00:00:00:58s)
Improvement at epoch 00010: -0.423885(090) (in 00:00:01:13s)
Improvement at epoch 00012: -0.411657(024) (in 00:00:01:28s)
Improvement at epoch 00013: -0.411631(080) (in 00:00:01:37s)
Improvement at epoch 00014: -0.407876(057) (in 00:00:01:45s)
Improvement at epoch 00015: -0.380923(052) (in 00:00:01:54s)
Improvement at epoch 00016: -0.373167(083) (in 00:00:02:02s)
Improvement at epoch 00017: -0.357684(042) (in 00:00:02:11s)
```

```
Improvement at epoch 00018: -0.326765(062) (in 00:00:02:21s)
Improvement at epoch 00020: -0.326557(022) (in 00:00:02:40s)
Improvement at epoch 00021: -0.320867(088) (in 00:00:02:50s)
Improvement at epoch 00022: -0.303322(028) (in 00:00:03:01s)
Improvement at epoch 00023: -0.291340(064) (in 00:00:03:11s)
Improvement at epoch 00024: -0.273263(077) (in 00:00:03:23s)
Improvement at epoch 00025: -0.264895(052) (in 00:00:03:34s)
Improvement at epoch 00026: -0.262397(093) (in 00:00:03:46s)
Improvement at epoch 00027: -0.257429(025) (in 00:00:03:58s)
Improvement at epoch 00028: -0.245429(037) (in 00:00:04:10s)
Improvement at epoch 00029: -0.217701(029) (in 00:00:04:22s)
Improvement at epoch 00030: -0.199667(021) (in 00:00:04:35s)
Improvement at epoch 00031: -0.181229(096) (in 00:00:04:48s)
Improvement at epoch 00032: -0.163509(027) (in 00:00:05:01s)
Improvement at epoch 00033: -0.161470(091) (in 00:00:05:15s)
Improvement at epoch 00034: -0.151782(067) (in 00:00:05:29s)
Improvement at epoch 00035: -0.136996(035) (in 00:00:05:44s)
Improvement at epoch 00036: -0.129815(051) (in 00:00:05:59s)
Improvement at epoch 00037: -0.121845(097) (in 00:00:06:14s)
Improvement at epoch 00038: -0.112080(032) (in 00:00:06:30s)
Improvement at epoch 00039: -0.101136(052) (in 00:00:06:46s)
Improvement at epoch 00040: -0.087390(056) (in 00:00:07:03s)
Improvement at epoch 00041: -0.075563(054) (in 00:00:07:20s)
Improvement at epoch 00042: -0.069845(065) (in 00:00:07:37s)
Improvement at epoch 00043: -0.069136(053) (in 00:00:07:55s)
Improvement at epoch 00044: -0.060098(055) (in 00:00:08:13s)
Improvement at epoch 00045: -0.057333(093) (in 00:00:08:31s)
Improvement at epoch 00046: -0.044841(093) (in 00:00:08:50s)
Improvement at epoch 00047: -0.043775(040) (in 00:00:09:08s)
Improvement at epoch 00048: -0.030239(027) (in 00:00:09:27s)
Improvement at epoch 00049: -0.018006(091) (in 00:00:09:46s)
Improvement at epoch 00051: -0.008708(087) (in 00:00:10:25s)
Improvement at epoch 00052: -0.008033(078) (in 00:00:10:45s)
Improvement at epoch 00053: -0.000635(042) (in 00:00:11:05s)
Improvement at epoch 00054: 0.010078(056) (in 00:00:11:25s)
Improvement at epoch 00055: 0.027336(040) (in 00:00:11:46s)
Improvement at epoch 00056: 0.035067(095) (in 00:00:12:06s)
Improvement at epoch 00057: 0.042816(036) (in 00:00:12:27s)
Improvement at epoch 00058: 0.049444(045) (in 00:00:12:48s)
Improvement at epoch 00059: 0.067844(089) (in 00:00:13:10s)
Improvement at epoch 00060: 0.068337(046) (in 00:00:13:31s)
Improvement at epoch 00061: 0.080618(090) (in 00:00:13:54s)
Improvement at epoch 00062: 0.081638(097) (in 00:00:14:16s)
Improvement at epoch 00063: 0.083829(053) (in 00:00:14:39s)
Improvement at epoch 00064: 0.091928(073) (in 00:00:15:02s)
Improvement at epoch 00065: 0.097822(053) (in 00:00:15:26s)
Improvement at epoch 00066: 0.098309(065) (in 00:00:15:50s)
Improvement at epoch 00067: 0.105036(032) (in 00:00:16:14s)
Improvement at epoch 00068: 0.109656(068) (in 00:00:16:39s)
Improvement at epoch 00069: 0.114019(095) (in 00:00:17:03s)
Improvement at epoch 00070: 0.121109(078) (in 00:00:17:28s)
Improvement at epoch 00071: 0.129308(038) (in 00:00:17:54s)
Improvement at epoch 00072: 0.134164(034) (in 00:00:18:20s)
Improvement at epoch 00074: 0.140662(069) (in 00:00:19:14s)
Improvement at epoch 00075: 0.148934(028) (in 00:00:19:42s)
Improvement at epoch 00076: 0.151229(053) (in 00:00:20:10s)
Improvement at epoch 00077: 0.161809(024) (in 00:00:20:39s)
Improvement at epoch 00078: 0.162863(051) (in 00:00:21:07s)
Improvement at epoch 00079: 0.167377(033) (in 00:00:21:36s)
Improvement at epoch 00080: 0.168306(041) (in 00:00:22:05s)
Improvement at epoch 00081: 0.174791(064) (in 00:00:22:34s)
Improvement at epoch 00082: 0.186295(061) (in 00:00:23:04s)
```

```
Improvement at epoch 00083: 0.189690(080) (in 00:00:23:33s)
Improvement at epoch 00084: 0.194989(067) (in 00:00:24:03s)
Improvement at epoch 00085: 0.196500(077) (in 00:00:24:33s)
Improvement at epoch 00086: 0.212841(089) (in 00:00:25:04s)
Improvement at epoch 00087: 0.224296(077) (in 00:00:25:34s)
Improvement at epoch 00088: 0.227294(053) (in 00:00:26:05s)
Improvement at epoch 00089: 0.234354(035) (in 00:00:26:36s)
Improvement at epoch 00090: 0.234547(041) (in 00:00:27:08s)
Improvement at epoch 00091: 0.242559(049) (in 00:00:27:40s)
Improvement at epoch 00092: 0.250100(090) (in 00:00:28:12s)
Improvement at epoch 00093: 0.256330(045) (in 00:00:28:45s)
Improvement at epoch 00094: 0.256856(030) (in 00:00:29:18s)
Improvement at epoch 00095: 0.260790(043) (in 00:00:29:50s)
Improvement at epoch 00096: 0.264759(058) (in 00:00:30:23s)
Improvement at epoch 00097: 0.269616(040) (in 00:00:30:56s)
Improvement at epoch 00098: 0.270042(021) (in 00:00:31:30s)
Improvement at epoch 00099: 0.275091(041) (in 00:00:32:03s)
Improvement at epoch 00101: 0.275837(029) (in 00:00:33:11s)
Improvement at epoch 00102: 0.277709(062) (in 00:00:33:45s)
Improvement at epoch 00103: 0.280189(089) (in 00:00:34:19s)
Improvement at epoch 00104: 0.282687(094) (in 00:00:34:54s)
Improvement at epoch 00105: 0.284003(098) (in 00:00:35:29s)
Improvement at epoch 00106: 0.285191(031) (in 00:00:36:04s)
Improvement at epoch 00107: 0.287580(036) (in 00:00:36:40s)
Improvement at epoch 00108: 0.287873(055) (in 00:00:37:16s)
Improvement at epoch 00109: 0.289551(020) (in 00:00:37:52s)
Improvement at epoch 00110: 0.294111(052) (in 00:00:38:28s)
Improvement at epoch 00111: 0.294233(090) (in 00:00:39:04s)
Improvement at epoch 00112: 0.296588(039) (in 00:00:39:40s)
Improvement at epoch 00113: 0.300935(034) (in 00:00:40:17s)
Improvement at epoch 00114: 0.302006(093) (in 00:00:40:54s)
Improvement at epoch 00115: 0.307437(025) (in 00:00:41:31s)
Improvement at epoch 00117: 0.311836(076) (in 00:00:42:47s)
Improvement at epoch 00118: 0.314000(030) (in 00:00:43:24s)
Improvement at epoch 00119: 0.315254(035) (in 00:00:44:02s)
Improvement at epoch 00120: 0.318958(033) (in 00:00:44:40s)
Improvement at epoch 00121: 0.322099(094) (in 00:00:45:19s)
Improvement at epoch 00122: 0.324820(039) (in 00:00:46:06s)
Improvement at epoch 00123: 0.327015(030) (in 00:00:46:51s)
Improvement at epoch 00124: 0.332162(068) (in 00:00:47:35s)
Improvement at epoch 00125: 0.333039(030) (in 00:00:48:21s)
Improvement at epoch 00126: 0.334848(066) (in 00:00:49:10s)
Improvement at epoch 00127: 0.338198(022) (in 00:00:49:58s)
Improvement at epoch 00128: 0.340665(068) (in 00:00:50:46s)
Improvement at epoch 00129: 0.344046(027) (in 00:00:51:34s)
Improvement at epoch 00130: 0.344738(097) (in 00:00:52:20s)
Improvement at epoch 00131: 0.348724(043) (in 00:00:53:10s)
Improvement at epoch 00132: 0.351180(053) (in 00:00:54:01s)
Improvement at epoch 00133: 0.353264(028) (in 00:00:54:51s)
Improvement at epoch 00135: 0.356438(045) (in 00:00:56:36s)
Improvement at epoch 00136: 0.357223(024) (in 00:00:57:29s)
Improvement at epoch 00137: 0.362035(072) (in 00:00:58:17s)
Improvement at epoch 00138: 0.365262(085) (in 00:00:59:10s)
Improvement at epoch 00139: 0.368949(043) (in 00:01:00:00s)
Improvement at epoch 00140: 0.371727(069) (in 00:01:00:52s)
Improvement at epoch 00141: 0.373339(095) (in 00:01:01:44s)
Improvement at epoch 00142: 0.373648(068) (in 00:01:02:37s)
Improvement at epoch 00143: 0.375506(038) (in 00:01:03:31s)
Improvement at epoch 00144: 0.375593(050) (in 00:01:04:23s)
Improvement at epoch 00145: 0.379589(064) (in 00:01:05:14s)
Improvement at epoch 00147: 0.380017(078) (in 00:01:07:01s)
Improvement at epoch 00148: 0.380293(089) (in 00:01:07:52s)
```

```
Improvement at epoch 00149: 0.382799(057) (in 00:01:08:46s)
Improvement at epoch 00150: 0.383687(080) (in 00:01:09:41s)
Improvement at epoch 00151: 0.385293(020) (in 00:01:10:36s)
Improvement at epoch 00153: 0.386568(063) (in 00:01:12:21s)
Improvement at epoch 00154: 0.387461(071) (in 00:01:13:15s)
Improvement at epoch 00155: 0.388815(031) (in 00:01:14:09s)
Improvement at epoch 00156: 0.389875(087) (in 00:01:15:02s)
Improvement at epoch 00157: 0.390722(030) (in 00:01:15:57s)
Improvement at epoch 00158: 0.390803(043) (in 00:01:16:51s)
Improvement at epoch 00159: 0.392175(061) (in 00:01:17:46s)
Improvement at epoch 00160: 0.393177(033) (in 00:01:18:40s)
Improvement at epoch 00161: 0.394670(021) (in 00:01:19:31s)
Improvement at epoch 00162: 0.396827(098) (in 00:01:20:27s)
Improvement at epoch 00163: 0.397461(079) (in 00:01:21:23s)
Improvement at epoch 00164: 0.397711(064) (in 00:01:22:16s)
Improvement at epoch 00165: 0.398956(088) (in 00:01:23:12s)
Improvement at epoch 00166: 0.400852(041) (in 00:01:24:08s)
Improvement at epoch 00167: 0.401572(069) (in 00:01:25:02s)
Improvement at epoch 00168: 0.403837(077) (in 00:01:25:59s)
Improvement at epoch 00169: 0.404597(078) (in 00:01:26:56s)
Improvement at epoch 00170: 0.406961(031) (in 00:01:27:52s)
Improvement at epoch 00171: 0.407794(091) (in 00:01:28:50s)
Improvement at epoch 00172: 0.408105(086) (in 00:01:29:46s)
Improvement at epoch 00173: 0.410232(056) (in 00:01:30:41s)
Improvement at epoch 00174: 0.411070(038) (in 00:01:31:38s)
Improvement at epoch 00175: 0.411694(030) (in 00:01:32:39s)
Improvement at epoch 00176: 0.412659(020) (in 00:01:33:37s)
Improvement at epoch 00177: 0.412713(056) (in 00:01:34:34s)
Improvement at epoch 00178: 0.415186(098) (in 00:01:35:31s)
Improvement at epoch 00180: 0.415481(055) (in 00:01:37:28s)
Improvement at epoch 00181: 0.415954(027) (in 00:01:38:24s)
Improvement at epoch 00182: 0.416441(038) (in 00:01:39:24s)
Improvement at epoch 00183: 0.418146(066) (in 00:01:40:24s)
Improvement at epoch 00185: 0.419413(045) (in 00:01:42:23s)
Improvement at epoch 00187: 0.420123(087) (in 00:01:44:16s)
Improvement at epoch 00188: 0.420499(025) (in 00:01:45:17s)
Improvement at epoch 00189: 0.421603(064) (in 00:01:46:19s)
Improvement at epoch 00190: 0.425482(022) (in 00:01:47:20s)
Improvement at epoch 00192: 0.426509(087) (in 00:01:49:25s)
Improvement at epoch 00193: 0.427167(070) (in 00:01:50:27s)
Improvement at epoch 00194: 0.427842(061) (in 00:01:51:32s)
Improvement at epoch 00195: 0.430375(039) (in 00:01:52:35s)
Improvement at epoch 00196: 0.431395(094) (in 00:01:53:41s)
Improvement at epoch 00197: 0.431927(089) (in 00:01:54:44s)
Improvement at epoch 00198: 0.432528(092) (in 00:01:55:51s)
Improvement at epoch 00199: 0.434463(088) (in 00:01:56:55s)
Improvement at epoch 00200: 0.435063(088) (in 00:01:58:01s)
Improvement at epoch 00201: 0.437808(022) (in 00:01:59:04s)
Improvement at epoch 00203: 0.437818(057) (in 00:02:01:10s)
Improvement at epoch 00204: 0.440473(078) (in 00:02:02:15s)
Improvement at epoch 00207: 0.443024(038) (in 00:02:05:25s)
Improvement at epoch 00208: 0.443495(099) (in 00:02:06:27s)
Improvement at epoch 00210: 0.444615(078) (in 00:02:08:37s)
Improvement at epoch 00211: 0.444894(054) (in 00:02:09:43s)
Improvement at epoch 00212: 0.448340(075) (in 00:02:10:52s)
Improvement at epoch 00215: 0.449462(069) (in 00:02:14:14s)
Improvement at epoch 00216: 0.451520(092) (in 00:02:15:21s)
Improvement at epoch 00217: 0.452153(024) (in 00:02:16:32s)
Improvement at epoch 00218: 0.452735(080) (in 00:02:17:42s)
Improvement at epoch 00219: 0.453519(052) (in 00:02:18:51s)
Improvement at epoch 00220: 0.454314(032) (in 00:02:20:01s)
Improvement at epoch 00221: 0.455258(065) (in 00:02:21:10s)
```

```
Improvement at epoch 00222: 0.455599(062) (in 00:02:22:18s)
Improvement at epoch 00223: 0.456985(034) (in 00:02:23:28s)
Improvement at epoch 00225: 0.457472(024) (in 00:02:25:54s)
Improvement at epoch 00226: 0.458102(047) (in 00:02:27:06s)
Improvement at epoch 00227: 0.458965(025) (in 00:02:28:17s)
Improvement at epoch 00228: 0.459588(068) (in 00:02:29:32s)
Improvement at epoch 00229: 0.459727(039) (in 00:02:30:45s)
Improvement at epoch 00230: 0.460248(058) (in 00:02:31:58s)
Improvement at epoch 00231: 0.460877(039) (in 00:02:33:10s)
Improvement at epoch 00232: 0.462676(069) (in 00:02:34:21s)
Improvement at epoch 00233: 0.463588(048) (in 00:02:35:36s)
Improvement at epoch 00234: 0.464872(039) (in 00:02:36:50s)
Improvement at epoch 00235: 0.466333(070) (in 00:02:38:07s)
Improvement at epoch 00236: 0.466456(050) (in 00:02:39:23s)
Improvement at epoch 00237: 0.466980(066) (in 00:02:40:38s)
Improvement at epoch 00238: 0.469124(082) (in 00:02:41:55s)
Improvement at epoch 00239: 0.470949(058) (in 00:02:43:11s)
Improvement at epoch 00240: 0.471441(039) (in 00:02:44:27s)
Improvement at epoch 00241: 0.471993(096) (in 00:02:45:44s)
Improvement at epoch 00242: 0.473884(054) (in 00:02:46:57s)
Improvement at epoch 00243: 0.474657(062) (in 00:02:48:12s)
Improvement at epoch 00245: 0.475678(043) (in 00:02:50:43s)
Improvement at epoch 00246: 0.476839(047) (in 00:02:51:59s)
Improvement at epoch 00247: 0.478036(080) (in 00:02:53:15s)
Improvement at epoch 00250: 0.478213(042) (in 00:02:56:54s)
Improvement at epoch 00251: 0.478361(097) (in 00:02:58:10s)
Improvement at epoch 00252: 0.478988(032) (in 00:02:59:27s)
Improvement at epoch 00253: 0.479167(060) (in 00:03:00:44s)
Improvement at epoch 00254: 0.481566(084) (in 00:03:02:00s)
Improvement at epoch 00256: 0.485249(043) (in 00:03:04:31s)
Improvement at epoch 00257: 0.486818(075) (in 00:03:05:43s)
Improvement at epoch 00258: 0.487116(057) (in 00:03:06:59s)
Improvement at epoch 00260: 0.487698(023) (in 00:03:09:28s)
Improvement at epoch 00261: 0.488206(077) (in 00:03:10:42s)
Improvement at epoch 00262: 0.488269(098) (in 00:03:11:59s)
Improvement at epoch 00264: 0.490235(069) (in 00:03:14:33s)
Improvement at epoch 00266: 0.491073(075) (in 00:03:17:01s)
Improvement at epoch 00267: 0.491783(053) (in 00:03:18:14s)
Improvement at epoch 00268: 0.493721(036) (in 00:03:19:32s)
Improvement at epoch 00269: 0.493793(023) (in 00:03:20:50s)
Improvement at epoch 00270: 0.494085(034) (in 00:03:22:09s)
Improvement at epoch 00271: 0.494952(068) (in 00:03:23:27s)
Improvement at epoch 00273: 0.495971(089) (in 00:03:25:55s)
Improvement at epoch 00275: 0.496112(094) (in 00:03:28:05s)
Improvement at epoch 00276: 0.496407(046) (in 00:03:29:09s)
Improvement at epoch 00277: 0.497006(054) (in 00:03:30:14s)
Improvement at epoch 00278: 0.497125(039) (in 00:03:31:18s)
Improvement at epoch 00279: 0.497550(092) (in 00:03:32:23s)
Improvement at epoch 00280: 0.498044(044) (in 00:03:33:28s)
Improvement at epoch 00281: 0.498868(095) (in 00:03:34:33s)
Improvement at epoch 00282: 0.499283(048) (in 00:03:35:38s)
Improvement at epoch 00283: 0.499459(075) (in 00:03:36:42s)
Improvement at epoch 00284: 0.500013(039) (in 00:03:37:47s)
Improvement at epoch 00285: 0.500862(069) (in 00:03:39:10s)
Improvement at epoch 00287: 0.501091(043) (in 00:03:41:45s)
Improvement at epoch 00288: 0.501358(098) (in 00:03:43:05s)
Improvement at epoch 00289: 0.501760(039) (in 00:03:44:26s)
Improvement at epoch 00291: 0.503415(022) (in 00:03:47:06s)
Improvement at epoch 00293: 0.504177(038) (in 00:03:49:44s)
Improvement at epoch 00294: 0.504837(036) (in 00:03:51:03s)
Improvement at epoch 00295: 0.505532(097) (in 00:03:52:23s)
Improvement at epoch 00296: 0.507778(085) (in 00:03:53:44s)
```

```
Improvement at epoch 00297: 0.508143(086) (in 00:03:55:04s)
Improvement at epoch 00299: 0.509263(043) (in 00:03:57:43s)
Improvement at epoch 00301: 0.509651(057) (in 00:04:00:20s)
Improvement at epoch 00302: 0.510612(038) (in 00:04:01:44s)
Improvement at epoch 00303: 0.510920(081) (in 00:04:03:01s)
Improvement at epoch 00304: 0.510962(040) (in 00:04:04:19s)
Improvement at epoch 00305: 0.511663(087) (in 00:04:05:26s)
Improvement at epoch 00306: 0.512339(075) (in 00:04:06:33s)
Improvement at epoch 00307: 0.513001(061) (in 00:04:07:40s)
Improvement at epoch 00308: 0.513266(046) (in 00:04:08:47s)
Improvement at epoch 00309: 0.513650(067) (in 00:04:09:55s)
Improvement at epoch 00311: 0.513739(059) (in 00:04:12:11s)
Improvement at epoch 00312: 0.513947(041) (in 00:04:13:20s)
Improvement at epoch 00313: 0.525417(066) (in 00:04:14:28s)
Improvement at epoch 00314: 0.526242(078) (in 00:04:15:36s)
Improvement at epoch 00315: 0.527665(024) (in 00:04:16:45s)
Improvement at epoch 00316: 0.528169(050) (in 00:04:17:54s)
Improvement at epoch 00317: 0.528298(065) (in 00:04:19:02s)
Improvement at epoch 00318: 0.528603(036) (in 00:04:20:11s)
Improvement at epoch 00319: 0.528651(041) (in 00:04:21:19s)
Improvement at epoch 00320: 0.529022(038) (in 00:04:22:27s)
Improvement at epoch 00321: 0.529564(069) (in 00:04:23:36s)
Improvement at epoch 00323: 0.536717(060) (in 00:04:25:53s)
Improvement at epoch 00324: 0.538513(037) (in 00:04:27:02s)
Improvement at epoch 00325: 0.539013(024) (in 00:04:28:12s)
Improvement at epoch 00327: 0.539453(067) (in 00:04:30:31s)
Improvement at epoch 00328: 0.541877(063) (in 00:04:31:40s)
Improvement at epoch 00330: 0.543015(023) (in 00:04:34:00s)
Improvement at epoch 00332: 0.543708(062) (in 00:04:36:19s)
Improvement at epoch 00333: 0.543744(024) (in 00:04:37:34s)
Improvement at epoch 00334: 0.544462(073) (in 00:04:38:45s)
Improvement at epoch 00336: 0.545349(069) (in 00:04:41:02s)
Improvement at epoch 00339: 0.546303(025) (in 00:04:44:27s)
Improvement at epoch 00340: 0.546544(099) (in 00:04:45:36s)
Improvement at epoch 00341: 0.547130(028) (in 00:04:46:46s)
Improvement at epoch 00342: 0.547238(096) (in 00:04:47:55s)
Improvement at epoch 00343: 0.547643(051) (in 00:04:49:05s)
Improvement at epoch 00344: 0.548182(060) (in 00:04:50:14s)
Improvement at epoch 00345: 0.550333(072) (in 00:04:51:24s)
Improvement at epoch 00346: 0.551074(078) (in 00:04:52:34s)
Improvement at epoch 00348: 0.551959(042) (in 00:04:54:54s)
Improvement at epoch 00349: 0.553005(058) (in 00:04:56:04s)
Improvement at epoch 00350: 0.554039(064) (in 00:04:57:14s)
Improvement at epoch 00351: 0.554189(030) (in 00:04:58:24s)
Improvement at epoch 00352: 0.555215(087) (in 00:04:59:35s)
Improvement at epoch 00353: 0.556805(033) (in 00:05:00:45s)
Improvement at epoch 00355: 0.558967(044) (in 00:05:03:06s)
Improvement at epoch 00356: 0.569161(053) (in 00:05:04:17s)
Improvement at epoch 00358: 0.569983(021) (in 00:05:06:46s)
Improvement at epoch 00359: 0.573367(056) (in 00:05:08:00s)
Improvement at epoch 00360: 0.573466(020) (in 00:05:09:13s)
Improvement at epoch 00362: 0.580546(062) (in 00:05:11:39s)
Improvement at epoch 00366: 0.580812(056) (in 00:05:16:37s)
Improvement at epoch 00368: 0.583427(096) (in 00:05:19:05s)
Improvement at epoch 00371: 0.583541(054) (in 00:05:22:47s)
Improvement at epoch 00372: 0.585646(073) (in 00:05:24:01s)
Improvement at epoch 00373: 0.587477(039) (in 00:05:25:15s)
Improvement at epoch 00375: 0.589024(054) (in 00:05:27:39s)
Improvement at epoch 00376: 0.590809(068) (in 00:05:28:51s)
Improvement at epoch 00377: 0.593637(035) (in 00:05:30:04s)
Improvement at epoch 00379: 0.596539(087) (in 00:05:32:29s)
Improvement at epoch 00380: 0.596672(059) (in 00:05:33:41s)
```

```
Improvement at epoch 00381: 0.597874(039) (in 00:05:34:58s)
Improvement at epoch 00382: 0.598211(077) (in 00:05:36:22s)
Improvement at epoch 00383: 0.599502(038) (in 00:05:37:50s)
Improvement at epoch 00384: 0.600732(071) (in 00:05:39:15s)
Improvement at epoch 00385: 0.603107(063) (in 00:05:40:42s)
Improvement at epoch 00386: 0.603672(096) (in 00:05:42:09s)
Improvement at epoch 00387: 0.604423(082) (in 00:05:43:35s)
Improvement at epoch 00388: 0.605008(079) (in 00:05:45:00s)
Improvement at epoch 00389: 0.605396(063) (in 00:05:46:26s)
Improvement at epoch 00390: 0.606136(032) (in 00:05:47:45s)
Improvement at epoch 00391: 0.606299(053) (in 00:05:48:59s)
Improvement at epoch 00392: 0.606407(022) (in 00:05:50:13s)
Improvement at epoch 00393: 0.606988(091) (in 00:05:51:27s)
Improvement at epoch 00394: 0.608204(040) (in 00:05:52:41s)
Improvement at epoch 00396: 0.609543(038) (in 00:05:55:10s)
Improvement at epoch 00398: 0.610559(035) (in 00:05:57:42s)
Improvement at epoch 00399: 0.611327(082) (in 00:05:59:13s)
Improvement at epoch 00401: 0.612065(031) (in 00:06:02:08s)
Improvement at epoch 00403: 0.612211(038) (in 00:06:05:10s)
Improvement at epoch 00404: 0.612940(041) (in 00:06:06:39s)
Improvement at epoch 00405: 0.613562(025) (in 00:06:08:06s)
Improvement at epoch 00407: 0.614261(051) (in 00:06:11:01s)
Improvement at epoch 00409: 0.614474(066) (in 00:06:13:54s)
Improvement at epoch 00411: 0.614595(037) (in 00:06:16:58s)
Improvement at epoch 00412: 0.615366(029) (in 00:06:18:24s)
Improvement at epoch 00413: 0.615630(058) (in 00:06:19:53s)
Improvement at epoch 00414: 0.616989(053) (in 00:06:21:26s)
Improvement at epoch 00415: 0.617104(041) (in 00:06:22:56s)
Improvement at epoch 00416: 0.617198(071) (in 00:06:24:26s)
Improvement at epoch 00417: 0.618970(098) (in 00:06:25:58s)
Improvement at epoch 00418: 0.619750(030) (in 00:06:27:28s)
Improvement at epoch 00419: 0.620445(062) (in 00:06:28:58s)
Improvement at epoch 00420: 0.621811(083) (in 00:06:30:31s)
Improvement at epoch 00421: 0.622187(037) (in 00:06:31:48s)
Improvement at epoch 00423: 0.623431(091) (in 00:06:34:22s)
Improvement at epoch 00425: 0.624229(034) (in 00:06:36:56s)
Improvement at epoch 00426: 0.624842(051) (in 00:06:38:13s)
Improvement at epoch 00427: 0.625041(029) (in 00:06:39:31s)
Improvement at epoch 00428: 0.625526(055) (in 00:06:40:48s)
Improvement at epoch 00429: 0.626536(081) (in 00:06:42:05s)
Improvement at epoch 00431: 0.626754(048) (in 00:06:44:53s)
Improvement at epoch 00432: 0.627111(071) (in 00:06:46:22s)
Improvement at epoch 00433: 0.627765(026) (in 00:06:47:54s)
Improvement at epoch 00434: 0.628675(029) (in 00:06:49:26s)
Improvement at epoch 00437: 0.629850(080) (in 00:06:54:04s)
Improvement at epoch 00438: 0.629914(062) (in 00:06:55:36s)
Improvement at epoch 00439: 0.631476(060) (in 00:06:57:08s)
Improvement at epoch 00440: 0.631504(031) (in 00:06:58:38s)
Improvement at epoch 00441: 0.632018(025) (in 00:07:00:08s)
Improvement at epoch 00442: 0.632331(039) (in 00:07:01:41s)
Improvement at epoch 00444: 0.632638(038) (in 00:07:04:47s)
Improvement at epoch 00445: 0.632962(092) (in 00:07:06:22s)
Improvement at epoch 00446: 0.634229(037) (in 00:07:07:54s)
Improvement at epoch 00448: 0.635416(048) (in 00:07:10:57s)
Improvement at epoch 00450: 0.635879(022) (in 00:07:14:04s)
Improvement at epoch 00451: 0.637367(050) (in 00:07:15:34s)
Improvement at epoch 00452: 0.637652(048) (in 00:07:16:56s)
Improvement at epoch 00453: 0.638186(043) (in 00:07:18:17s)
Improvement at epoch 00454: 0.638241(096) (in 00:07:19:37s)
Improvement at epoch 00455: 0.638490(094) (in 00:07:20:59s)
Improvement at epoch 00456: 0.638997(058) (in 00:07:22:20s)
Improvement at epoch 00457: 0.639430(028) (in 00:07:23:41s)
```

```
Improvement at epoch 00459: 0.640014(023) (in 00:07:26:27s)
Improvement at epoch 00460: 0.640245(054) (in 00:07:27:50s)
Improvement at epoch 00462: 0.640301(087) (in 00:07:30:37s)
Improvement at epoch 00463: 0.640354(058) (in 00:07:31:59s)
Improvement at epoch 00464: 0.640584(079) (in 00:07:33:21s)
Improvement at epoch 00465: 0.641383(038) (in 00:07:34:45s)
Improvement at epoch 00467: 0.641412(061) (in 00:07:37:30s)
Improvement at epoch 00468: 0.642061(092) (in 00:07:38:54s)
Improvement at epoch 00469: 0.642132(079) (in 00:07:40:14s)
Improvement at epoch 00470: 0.642523(093) (in 00:07:41:35s)
Improvement at epoch 00471: 0.642580(050) (in 00:07:42:55s)
Improvement at epoch 00472: 0.642969(033) (in 00:07:44:16s)
Improvement at epoch 00473: 0.643169(066) (in 00:07:45:36s)
Improvement at epoch 00474: 0.643862(035) (in 00:07:46:57s)
Improvement at epoch 00475: 0.644986(051) (in 00:07:48:17s)
Improvement at epoch 00476: 0.645022(038) (in 00:07:49:38s)
Improvement at epoch 00477: 0.645060(032) (in 00:07:50:59s)
Improvement at epoch 00478: 0.645822(046) (in 00:07:52:20s)
Improvement at epoch 00479: 0.646207(061) (in 00:07:53:48s)
Improvement at epoch 00481: 0.647046(033) (in 00:07:57:04s)
Improvement at epoch 00483: 0.648475(064) (in 00:08:00:20s)
Improvement at epoch 00484: 0.649867(066) (in 00:08:01:59s)
Improvement at epoch 00486: 0.650171(089) (in 00:08:05:14s)
Improvement at epoch 00487: 0.650312(082) (in 00:08:06:54s)
Improvement at epoch 00488: 0.650454(025) (in 00:08:08:32s)
Improvement at epoch 00489: 0.650887(089) (in 00:08:10:12s)
Improvement at epoch 00490: 0.651297(072) (in 00:08:11:53s)
Improvement at epoch 00491: 0.651371(089) (in 00:08:13:29s)
Improvement at epoch 00492: 0.651674(080) (in 00:08:15:06s)
Improvement at epoch 00493: 0.651830(089) (in 00:08:16:45s)
Improvement at epoch 00494: 0.652292(058) (in 00:08:18:24s)
Improvement at epoch 00495: 0.652766(073) (in 00:08:20:02s)
Improvement at epoch 00496: 0.653869(042) (in 00:08:21:41s)
Improvement at epoch 00497: 0.654834(045) (in 00:08:23:21s)
Improvement at epoch 00498: 0.656160(060) (in 00:08:25:02s)
Improvement at epoch 00501: 0.656265(062) (in 00:08:29:55s)
Improvement at epoch 00502: 0.657120(097) (in 00:08:31:33s)
Improvement at epoch 00503: 0.657256(082) (in 00:08:33:13s)
Improvement at epoch 00504: 0.657340(038) (in 00:08:34:49s)
Improvement at epoch 00505: 0.657971(039) (in 00:08:36:31s)
Improvement at epoch 00507: 0.658149(028) (in 00:08:39:53s)
Improvement at epoch 00508: 0.658204(083) (in 00:08:41:34s)
Improvement at epoch 00509: 0.658265(086) (in 00:08:43:14s)
Improvement at epoch 00510: 0.658307(064) (in 00:08:44:54s)
Improvement at epoch 00511: 0.658459(085) (in 00:08:46:34s)
Improvement at epoch 00512: 0.658910(069) (in 00:08:48:18s)
Improvement at epoch 00514: 0.659257(091) (in 00:08:51:36s)
Improvement at epoch 00516: 0.659821(087) (in 00:08:54:56s)
Improvement at epoch 00518: 0.659844(061) (in 00:08:58:13s)
Improvement at epoch 00519: 0.660769(096) (in 00:08:59:54s)
Improvement at epoch 00521: 0.661009(023) (in 00:09:03:09s)
Improvement at epoch 00522: 0.661243(036) (in 00:09:04:45s)
Improvement at epoch 00523: 0.661274(070) (in 00:09:06:24s)
Improvement at epoch 00524: 0.661417(068) (in 00:09:08:07s)
Improvement at epoch 00525: 0.661763(077) (in 00:09:09:46s)
Improvement at epoch 00526: 0.662165(096) (in 00:09:11:26s)
Improvement at epoch 00528: 0.662281(099) (in 00:09:14:47s)
Improvement at epoch 00529: 0.662804(031) (in 00:09:16:25s)
Improvement at epoch 00530: 0.663049(027) (in 00:09:18:04s)
Improvement at epoch 00531: 0.663612(055) (in 00:09:19:44s)
Improvement at epoch 00533: 0.664334(082) (in 00:09:23:13s)
Improvement at epoch 00534: 0.665210(091) (in 00:09:24:56s)
```

```
Improvement at epoch 00536: 0.665304(071) (in 00:09:28:19s)
Improvement at epoch 00537: 0.665778(061) (in 00:09:30:03s)
Improvement at epoch 00538: 0.666253(045) (in 00:09:31:47s)
Improvement at epoch 00539: 0.666423(045) (in 00:09:33:32s)
Improvement at epoch 00540: 0.668076(093) (in 00:09:35:15s)
Improvement at epoch 00541: 0.668490(040) (in 00:09:37:00s)
Improvement at epoch 00543: 0.668604(054) (in 00:09:40:22s)
Improvement at epoch 00544: 0.669892(065) (in 00:09:42:05s)
Improvement at epoch 00545: 0.671635(092) (in 00:09:43:50s)
Improvement at epoch 00546: 0.672571(032) (in 00:09:45:33s)
Improvement at epoch 00547: 0.672959(020) (in 00:09:47:17s)
Improvement at epoch 00549: 0.673483(080) (in 00:09:50:45s)
Improvement at epoch 00550: 0.673716(026) (in 00:09:52:29s)
Improvement at epoch 00552: 0.673916(080) (in 00:09:55:57s)
Improvement at epoch 00555: 0.674482(074) (in 00:10:01:09s)
Improvement at epoch 00556: 0.675220(065) (in 00:10:02:52s)
Improvement at epoch 00559: 0.675280(085) (in 00:10:08:10s)
Improvement at epoch 00560: 0.675782(095) (in 00:10:09:55s)
Improvement at epoch 00561: 0.676008(063) (in 00:10:11:44s)
Improvement at epoch 00562: 0.676128(021) (in 00:10:13:30s)
Improvement at epoch 00563: 0.676268(056) (in 00:10:15:10s)
Improvement at epoch 00564: 0.676593(065) (in 00:10:16:54s)
Improvement at epoch 00565: 0.676670(067) (in 00:10:18:40s)
Improvement at epoch 00567: 0.677080(045) (in 00:10:22:11s)
Improvement at epoch 00568: 0.677252(039) (in 00:10:23:55s)
Improvement at epoch 00569: 0.677426(062) (in 00:10:25:44s)
Improvement at epoch 00570: 0.677600(043) (in 00:10:27:28s)
Improvement at epoch 00571: 0.677612(050) (in 00:10:29:15s)
Improvement at epoch 00572: 0.677788(071) (in 00:10:31:02s)
Improvement at epoch 00573: 0.677940(047) (in 00:10:32:46s)
Improvement at epoch 00574: 0.678285(083) (in 00:10:34:32s)
Improvement at epoch 00575: 0.678554(093) (in 00:10:36:17s)
Improvement at epoch 00576: 0.678699(034) (in 00:10:38:03s)
Improvement at epoch 00578: 0.678922(037) (in 00:10:41:35s)
Improvement at epoch 00579: 0.679228(056) (in 00:10:43:20s)
Improvement at epoch 00580: 0.679412(080) (in 00:10:45:03s)
Improvement at epoch 00581: 0.679670(070) (in 00:10:46:47s)
Improvement at epoch 00582: 0.679880(077) (in 00:10:48:32s)
Improvement at epoch 00584: 0.680168(023) (in 00:10:52:00s)
Improvement at epoch 00585: 0.680489(052) (in 00:10:53:50s)
Improvement at epoch 00588: 0.680528(061) (in 00:10:59:07s)
Improvement at epoch 00589: 0.680723(055) (in 00:11:00:53s)
Improvement at epoch 00590: 0.680963(095) (in 00:11:02:37s)
Improvement at epoch 00591: 0.681038(099) (in 00:11:04:23s)
Improvement at epoch 00592: 0.681457(034) (in 00:11:06:04s)
Improvement at epoch 00595: 0.681713(031) (in 00:11:11:12s)
Improvement at epoch 00597: 0.682310(041) (in 00:11:14:40s)
Improvement at epoch 00599: 0.682319(085) (in 00:11:18:10s)
Improvement at epoch 00600: 0.682360(050) (in 00:11:19:59s)
Improvement at epoch 00601: 0.682543(028) (in 00:11:21:45s)
Improvement at epoch 00602: 0.683489(050) (in 00:11:23:35s)
Improvement at epoch 00606: 0.683528(085) (in 00:11:30:52s)
Improvement at epoch 00607: 0.683771(091) (in 00:11:32:39s)
Improvement at epoch 00608: 0.684121(091) (in 00:11:34:27s)
Improvement at epoch 00611: 0.684524(078) (in 00:11:39:50s)
Improvement at epoch 00612: 0.684921(033) (in 00:11:41:41s)
Improvement at epoch 00613: 0.685204(032) (in 00:11:43:29s)
Improvement at epoch 00614: 0.685549(036) (in 00:11:45:19s)
Improvement at epoch 00615: 0.685560(077) (in 00:11:47:15s)
Improvement at epoch 00617: 0.685682(061) (in 00:11:50:51s)
Improvement at epoch 00619: 0.685849(050) (in 00:11:54:23s)
Improvement at epoch 00621: 0.686063(075) (in 00:11:57:54s)
```

```
Improvement at epoch 00624: 0.686096(070) (in 00:12:03:05s)
Improvement at epoch 00625: 0.686490(062) (in 00:12:04:50s)
Improvement at epoch 00627: 0.686563(082) (in 00:12:08:18s)
Improvement at epoch 00628: 0.686625(056) (in 00:12:10:04s)
Improvement at epoch 00630: 0.686721(095) (in 00:12:13:31s)
Improvement at epoch 00631: 0.686747(026) (in 00:12:15:14s)
Improvement at epoch 00632: 0.687360(070) (in 00:12:17:02s)
Improvement at epoch 00636: 0.687568(037) (in 00:12:24:05s)
Improvement at epoch 00637: 0.687655(037) (in 00:12:25:49s)
Improvement at epoch 00638: 0.687720(024) (in 00:12:27:38s)
Improvement at epoch 00639: 0.687821(064) (in 00:12:29:18s)
Improvement at epoch 00640: 0.687908(051) (in 00:12:31:01s)
Improvement at epoch 00641: 0.688098(055) (in 00:12:32:49s)
Improvement at epoch 00643: 0.688143(076) (in 00:12:36:22s)
Improvement at epoch 00645: 0.688336(020) (in 00:12:39:57s)
Improvement at epoch 00648: 0.688625(089) (in 00:12:45:12s)
Improvement at epoch 00650: 0.688768(057) (in 00:12:48:33s)
Improvement at epoch 00651: 0.688979(067) (in 00:12:50:14s)
Improvement at epoch 00653: 0.689239(071) (in 00:12:53:39s)
Improvement at epoch 00654: 0.689427(020) (in 00:12:55:20s)
Improvement at epoch 00655: 0.689675(039) (in 00:12:57:06s)
Improvement at epoch 00657: 0.690004(059) (in 00:13:00:35s)
Improvement at epoch 00659: 0.690667(047) (in 00:13:04:07s)
Improvement at epoch 00660: 0.690706(046) (in 00:13:05:50s)
Improvement at epoch 00661: 0.690711(030) (in 00:13:07:36s)
Improvement at epoch 00662: 0.690874(047) (in 00:13:09:21s)
Improvement at epoch 00663: 0.691219(033) (in 00:13:11:08s)
Improvement at epoch 00664: 0.691302(062) (in 00:13:12:50s)
Improvement at epoch 00665: 0.691632(050) (in 00:13:14:33s)
Improvement at epoch 00666: 0.691673(049) (in 00:13:16:17s)
Improvement at epoch 00667: 0.691701(020) (in 00:13:18:05s)
Improvement at epoch 00668: 0.692102(076) (in 00:13:19:46s)
Improvement at epoch 00670: 0.692488(064) (in 00:13:23:20s)
Improvement at epoch 00671: 0.692589(082) (in 00:13:25:03s)
Improvement at epoch 00673: 0.692870(066) (in 00:13:28:35s)
Improvement at epoch 00674: 0.692942(030) (in 00:13:30:22s)
Improvement at epoch 00675: 0.693263(021) (in 00:13:32:09s)
Improvement at epoch 00678: 0.693590(035) (in 00:13:37:27s)
Improvement at epoch 00683: 0.694029(054) (in 00:13:46:21s)
Improvement at epoch 00684: 0.694360(075) (in 00:13:48:07s)
Improvement at epoch 00685: 0.694366(062) (in 00:13:49:52s)
Improvement at epoch 00686: 0.694499(098) (in 00:13:51:38s)
Improvement at epoch 00687: 0.694723(075) (in 00:13:53:21s)
Improvement at epoch 00688: 0.694744(080) (in 00:13:55:08s)
Improvement at epoch 00692: 0.694858(065) (in 00:14:02:10s)
Improvement at epoch 00693: 0.695038(025) (in 00:14:03:54s)
Improvement at epoch 00696: 0.695184(022) (in 00:14:09:06s)
Improvement at epoch 00697: 0.695199(099) (in 00:14:10:54s)
Improvement at epoch 00698: 0.695218(096) (in 00:14:12:38s)
Improvement at epoch 00699: 0.695716(027) (in 00:14:14:25s)
Improvement at epoch 00701: 0.695753(066) (in 00:14:17:59s)
Improvement at epoch 00702: 0.695768(055) (in 00:14:19:44s)
Improvement at epoch 00703: 0.695797(042) (in 00:14:21:32s)
Improvement at epoch 00704: 0.695934(050) (in 00:14:23:21s)
Improvement at epoch 00705: 0.695989(030) (in 00:14:25:07s)
Improvement at epoch 00707: 0.696016(075) (in 00:14:28:41s)
Improvement at epoch 00709: 0.696443(091) (in 00:14:32:11s)
Improvement at epoch 00710: 0.696642(075) (in 00:14:33:56s)
Improvement at epoch 00714: 0.696717(076) (in 00:14:41:09s)
Improvement at epoch 00715: 0.696807(081) (in 00:14:42:59s)
Improvement at epoch 00716: 0.696820(068) (in 00:14:44:48s)
Improvement at epoch 00717: 0.696939(025) (in 00:14:46:31s)
```

```
Improvement at epoch 00718: 0.697014(091) (in 00:14:48:23s)
Improvement at epoch 00719: 0.697253(090) (in 00:14:50:12s)
Improvement at epoch 00720: 0.697258(026) (in 00:14:52:02s)
Improvement at epoch 00721: 0.697422(044) (in 00:14:53:46s)
Improvement at epoch 00722: 0.697547(066) (in 00:14:55:34s)
Improvement at epoch 00725: 0.697702(060) (in 00:15:01:01s)
Improvement at epoch 00726: 0.697729(098) (in 00:15:02:50s)
Improvement at epoch 00727: 0.697942(027) (in 00:15:04:37s)
Improvement at epoch 00728: 0.698296(055) (in 00:15:06:26s)
Improvement at epoch 00729: 0.698607(078) (in 00:15:08:13s)
Improvement at epoch 00733: 0.698993(036) (in 00:15:15:29s)
Improvement at epoch 00734: 0.699081(023) (in 00:15:17:20s)
Improvement at epoch 00736: 0.699411(077) (in 00:15:21:03s)
Improvement at epoch 00738: 0.699555(055) (in 00:15:24:45s)
Improvement at epoch 00739: 0.699664(061) (in 00:15:26:36s)
Improvement at epoch 00740: 0.699778(057) (in 00:15:28:28s)
Improvement at epoch 00741: 0.700104(033) (in 00:15:30:19s)
Improvement at epoch 00742: 0.700449(037) (in 00:15:32:10s)
Improvement at epoch 00743: 0.700478(099) (in 00:15:34:02s)
Improvement at epoch 00744: 0.700960(088) (in 00:15:35:52s)
Improvement at epoch 00749: 0.701120(045) (in 00:15:45:12s)
Improvement at epoch 00751: 0.701314(039) (in 00:15:48:54s)
Improvement at epoch 00752: 0.701332(046) (in 00:15:50:50s)
Improvement at epoch 00753: 0.701382(049) (in 00:15:52:44s)
Improvement at epoch 00754: 0.701652(034) (in 00:15:54:40s)
Improvement at epoch 00758: 0.701678(076) (in 00:16:02:10s)
Improvement at epoch 00759: 0.703051(047) (in 00:16:03:59s)
Improvement at epoch 00760: 0.703341(066) (in 00:16:05:49s)
Improvement at epoch 00764: 0.703351(046) (in 00:16:13:08s)
Improvement at epoch 00767: 0.703638(045) (in 00:16:18:44s)
Improvement at epoch 00770: 0.704641(031) (in 00:16:24:15s)
Improvement at epoch 00773: 0.704749(062) (in 00:16:29:52s)
Improvement at epoch 00777: 0.704889(054) (in 00:16:37:14s)
Improvement at epoch 00781: 0.705162(047) (in 00:16:44:41s)
Improvement at epoch 00785: 0.705245(055) (in 00:16:52:08s)
Improvement at epoch 00786: 0.705422(077) (in 00:16:54:00s)
Improvement at epoch 00787: 0.705455(038) (in 00:16:55:52s)
Improvement at epoch 00788: 0.705631(061) (in 00:16:57:46s)
Improvement at epoch 00789: 0.705860(090) (in 00:16:59:34s)
Improvement at epoch 00790: 0.706353(099) (in 00:17:01:25s)
Improvement at epoch 00792: 0.706571(078) (in 00:17:05:03s)
Improvement at epoch 00793: 0.706615(041) (in 00:17:06:53s)
Improvement at epoch 00797: 0.706820(089) (in 00:17:14:23s)
Improvement at epoch 00800: 0.706947(030) (in 00:17:20:10s)
Improvement at epoch 00801: 0.707226(087) (in 00:17:22:02s)
Improvement at epoch 00802: 0.707330(066) (in 00:17:23:52s)
Improvement at epoch 00805: 0.707487(067) (in 00:17:29:23s)
Improvement at epoch 00808: 0.707906(093) (in 00:17:35:05s)
Improvement at epoch 00812: 0.707951(050) (in 00:17:42:46s)
Improvement at epoch 00813: 0.707959(057) (in 00:17:44:42s)
Improvement at epoch 00814: 0.708173(093) (in 00:17:46:36s)
Improvement at epoch 00815: 0.708216(026) (in 00:17:48:30s)
Improvement at epoch 00816: 0.708575(094) (in 00:17:50:24s)
Improvement at epoch 00817: 0.708577(054) (in 00:17:52:14s)
Improvement at epoch 00819: 0.708593(095) (in 00:17:56:02s)
Improvement at epoch 00820: 0.708638(056) (in 00:17:57:55s)
Improvement at epoch 00821: 0.708667(025) (in 00:17:59:51s)
Improvement at epoch 00822: 0.708676(037) (in 00:18:01:48s)
Improvement at epoch 00823: 0.708690(087) (in 00:18:03:44s)
Improvement at epoch 00824: 0.708788(029) (in 00:18:05:39s)
Improvement at epoch 00826: 0.708843(074) (in 00:18:09:30s)
Improvement at epoch 00828: 0.709059(058) (in 00:18:13:23s)
```

```
Improvement at epoch 00831: 0.709147(034) (in 00:18:19:01s)
Improvement at epoch 00833: 0.709484(043) (in 00:18:22:51s)
Improvement at epoch 00836: 0.709496(023) (in 00:18:28:33s)
Improvement at epoch 00837: 0.709759(031) (in 00:18:30:30s)
Improvement at epoch 00838: 0.709774(093) (in 00:18:32:29s)
Improvement at epoch 00839: 0.710171(070) (in 00:18:34:20s)
Improvement at epoch 00843: 0.710436(086) (in 00:18:42:02s)
Improvement at epoch 00844: 0.710714(068) (in 00:18:43:58s)
Improvement at epoch 00845: 0.710903(097) (in 00:18:46:00s)
Improvement at epoch 00848: 0.711208(039) (in 00:18:51:37s)
Improvement at epoch 00849: 0.711282(037) (in 00:18:53:12s)
Improvement at epoch 00850: 0.711358(062) (in 00:18:54:48s)
Improvement at epoch 00851: 0.711533(064) (in 00:18:56:23s)
Improvement at epoch 00852: 0.711572(038) (in 00:18:57:59s)
Improvement at epoch 00854: 0.711772(027) (in 00:19:01:11s)
Improvement at epoch 00860: 0.711817(066) (in 00:19:11:58s)
Improvement at epoch 00861: 0.711905(080) (in 00:19:13:34s)
Improvement at epoch 00862: 0.711990(099) (in 00:19:15:10s)
Improvement at epoch 00863: 0.712005(095) (in 00:19:16:46s)
Improvement at epoch 00864: 0.712038(044) (in 00:19:18:21s)
Improvement at epoch 00865: 0.712261(089) (in 00:19:19:58s)
Improvement at epoch 00869: 0.712323(060) (in 00:19:27:35s)
Improvement at epoch 00870: 0.712371(082) (in 00:19:29:31s)
Improvement at epoch 00871: 0.712634(084) (in 00:19:31:28s)
Improvement at epoch 00874: 0.712915(059) (in 00:19:37:17s)
Improvement at epoch 00876: 0.713287(021) (in 00:19:41:05s)
Improvement at epoch 00878: 0.713439(086) (in 00:19:44:49s)
Improvement at epoch 00879: 0.713499(024) (in 00:19:46:44s)
Improvement at epoch 00881: 0.713558(065) (in 00:19:50:15s)
Improvement at epoch 00882: 0.713618(060) (in 00:19:51:53s)
Improvement at epoch 00883: 0.713632(044) (in 00:19:53:30s)
Improvement at epoch 00884: 0.713921(053) (in 00:19:55:06s)
Improvement at epoch 00885: 0.714475(069) (in 00:19:56:41s)
Improvement at epoch 00889: 0.714489(073) (in 00:20:03:03s)
Improvement at epoch 00890: 0.714562(070) (in 00:20:04:38s)
Improvement at epoch 00895: 0.714638(026) (in 00:20:12:36s)
Improvement at epoch 00896: 0.714738(037) (in 00:20:14:12s)
Improvement at epoch 00897: 0.714961(092) (in 00:20:15:48s)
Improvement at epoch 00901: 0.715127(029) (in 00:20:22:15s)
Improvement at epoch 00905: 0.715281(028) (in 00:20:28:43s)
Improvement at epoch 00906: 0.715488(028) (in 00:20:30:21s)
Improvement at epoch 00910: 0.715567(094) (in 00:20:36:48s)
Improvement at epoch 00911: 0.715578(046) (in 00:20:38:26s)
Improvement at epoch 00912: 0.715589(047) (in 00:20:40:03s)
Improvement at epoch 00914: 0.715775(086) (in 00:20:43:18s)
Improvement at epoch 00915: 0.715860(071) (in 00:20:44:56s)
Improvement at epoch 00916: 0.716009(091) (in 00:20:46:34s)
Improvement at epoch 00919: 0.716022(031) (in 00:20:51:27s)
Improvement at epoch 00921: 0.716036(095) (in 00:20:54:42s)
Improvement at epoch 00922: 0.716119(059) (in 00:20:56:20s)
Improvement at epoch 00924: 0.716559(020) (in 00:20:59:37s)
Improvement at epoch 00926: 0.716692(091) (in 00:21:02:54s)
Improvement at epoch 00929: 0.716694(092) (in 00:21:07:49s)
Improvement at epoch 00930: 0.716775(091) (in 00:21:09:28s)
Improvement at epoch 00932: 0.716792(053) (in 00:21:12:45s)
Improvement at epoch 00933: 0.716951(038) (in 00:21:14:23s)
Improvement at epoch 00937: 0.717129(084) (in 00:21:20:58s)
Improvement at epoch 00938: 0.717165(034) (in 00:21:22:37s)
Improvement at epoch 00940: 0.717390(025) (in 00:21:25:56s)
Improvement at epoch 00942: 0.717681(056) (in 00:21:29:16s)
Improvement at epoch 00944: 0.717728(037) (in 00:21:32:34s)
Improvement at epoch 00945: 0.717789(033) (in 00:21:34:13s)
```

```
Improvement at epoch 00948: 0.717828(020) (in 00:21:39:10s)
Improvement at epoch 00949: 0.717913(050) (in 00:21:40:49s)
Improvement at epoch 00950: 0.718379(054) (in 00:21:42:27s)
Improvement at epoch 00955: 0.718415(073) (in 00:21:51:36s)
Improvement at epoch 00957: 0.718590(083) (in 00:21:55:33s)
Improvement at epoch 00959: 0.718713(081) (in 00:21:59:26s)
Improvement at epoch 00962: 0.718758(051) (in 00:22:05:18s)
Improvement at epoch 00964: 0.718837(062) (in 00:22:08:51s)
Improvement at epoch 00965: 0.718923(056) (in 00:22:10:30s)
Improvement at epoch 00966: 0.718935(081) (in 00:22:12:10s)
Improvement at epoch 00968: 0.719040(070) (in 00:22:15:30s)
Improvement at epoch 00969: 0.719087(062) (in 00:22:17:09s)
Improvement at epoch 00973: 0.719099(024) (in 00:22:23:43s)
Improvement at epoch 00974: 0.719124(090) (in 00:22:25:21s)
Improvement at epoch 00975: 0.719174(058) (in 00:22:27:09s)
Improvement at epoch 00976: 0.719335(080) (in 00:22:29:05s)
Improvement at epoch 00980: 0.719342(065) (in 00:22:35:54s)
Improvement at epoch 00984: 0.719450(021) (in 00:22:42:38s)
Improvement at epoch 00986: 0.719535(022) (in 00:22:45:58s)
Improvement at epoch 00987: 0.719721(099) (in 00:22:47:39s)
Improvement at epoch 00990: 0.719765(081) (in 00:22:52:44s)
Improvement at epoch 00991: 0.719891(086) (in 00:22:54:23s)
Improvement at epoch 00993: 0.720129(032) (in 00:22:57:40s)
Improvement at epoch 00999: 0.720323(036) (in 00:23:07:32s)
Improvement at epoch 01001: 0.720418(086) (in 00:23:10:51s)
Improvement at epoch 01003: 0.720442(024) (in 00:23:14:11s)
Improvement at epoch 01005: 0.720588(098) (in 00:23:17:30s)
Improvement at epoch 01010: 0.720606(072) (in 00:23:25:48s)
Improvement at epoch 01011: 0.720638(044) (in 00:23:27:29s)
Improvement at epoch 01012: 0.720678(038) (in 00:23:29:26s)
Improvement at epoch 01014: 0.720994(022) (in 00:23:32:50s)
Improvement at epoch 01016: 0.721013(095) (in 00:23:36:11s)
Improvement at epoch 01017: 0.721063(069) (in 00:23:37:51s)
Improvement at epoch 01020: 0.721075(064) (in 00:23:42:58s)
Improvement at epoch 01021: 0.721255(096) (in 00:23:44:38s)
Improvement at epoch 01022: 0.721273(051) (in 00:23:46:19s)
Improvement at epoch 01023: 0.721435(045) (in 00:23:47:59s)
Improvement at epoch 01024: 0.721542(033) (in 00:23:49:39s)
Improvement at epoch 01028: 0.721771(042) (in 00:23:56:51s)
Improvement at epoch 01029: 0.721937(085) (in 00:23:58:53s)
Improvement at epoch 01031: 0.722156(071) (in 01:00:02:57s)
Improvement at epoch 01033: 0.722298(067) (in 01:00:07:08s)
Improvement at epoch 01034: 0.722352(086) (in 01:00:09:10s)
Improvement at epoch 01035: 0.722787(070) (in 01:00:11:12s)
Improvement at epoch 01036: 0.723195(083) (in 01:00:13:14s)
Improvement at epoch 01038: 0.723315(099) (in 01:00:17:17s)
Improvement at epoch 01040: 0.723537(054) (in 01:00:21:23s)
Improvement at epoch 01041: 0.723558(049) (in 01:00:23:29s)
Improvement at epoch 01042: 0.723686(026) (in 01:00:25:31s)
Improvement at epoch 01045: 0.723745(094) (in 01:00:31:39s)
Improvement at epoch 01046: 0.723888(051) (in 01:00:33:42s)
Improvement at epoch 01047: 0.723966(063) (in 01:00:35:45s)
Improvement at epoch 01052: 0.724000(033) (in 01:00:45:52s)
Improvement at epoch 01054: 0.724599(070) (in 01:00:49:57s)
Improvement at epoch 01057: 0.724791(099) (in 01:00:55:59s)
Improvement at epoch 01059: 0.725040(056) (in 01:00:59:59s)
Improvement at epoch 01061: 0.725625(041) (in 01:01:04:08s)
Improvement at epoch 01062: 0.726368(065) (in 01:01:06:04s)
Improvement at epoch 01066: 0.726425(035) (in 01:01:14:06s)
Improvement at epoch 01070: 0.726493(055) (in 01:01:22:17s)
Improvement at epoch 01071: 0.726672(060) (in 01:01:24:24s)
Improvement at epoch 01072: 0.726725(059) (in 01:01:26:25s)
```

```
Improvement at epoch 01073: 0.726800(071) (in 01:01:28:25s)
Improvement at epoch 01074: 0.727003(078) (in 01:01:30:23s)
Improvement at epoch 01075: 0.727060(099) (in 01:01:32:26s)
Improvement at epoch 01076: 0.727562(058) (in 01:01:34:29s)
Improvement at epoch 01077: 0.727768(099) (in 01:01:36:33s)
Improvement at epoch 01078: 0.727860(060) (in 01:01:38:35s)
Improvement at epoch 01082: 0.728093(099) (in 01:01:46:56s)
Improvement at epoch 01084: 0.728313(085) (in 01:01:51:05s)
Improvement at epoch 01087: 0.728354(035) (in 01:01:57:20s)
Improvement at epoch 01089: 0.728550(028) (in 01:02:01:34s)
Improvement at epoch 01090: 0.728682(050) (in 01:02:03:41s)
Improvement at epoch 01092: 0.728997(031) (in 01:02:07:53s)
Improvement at epoch 01093: 0.729244(094) (in 01:02:09:58s)
Improvement at epoch 01094: 0.729291(042) (in 01:02:12:03s)
Improvement at epoch 01096: 0.729346(037) (in 01:02:16:19s)
Improvement at epoch 01097: 0.729368(030) (in 01:02:18:22s)
Improvement at epoch 01098: 0.729634(037) (in 01:02:20:31s)
Improvement at epoch 01101: 0.729750(096) (in 01:02:26:43s)
Improvement at epoch 01102: 0.730129(029) (in 01:02:28:46s)
Improvement at epoch 01103: 0.730203(067) (in 01:02:30:51s)
Improvement at epoch 01105: 0.730254(023) (in 01:02:35:09s)
Improvement at epoch 01107: 0.730336(058) (in 01:02:39:18s)
Improvement at epoch 01108: 0.730496(052) (in 01:02:41:27s)
Improvement at epoch 01109: 0.730622(025) (in 01:02:43:33s)
Improvement at epoch 01111: 0.730635(077) (in 01:02:47:44s)
Improvement at epoch 01112: 0.731278(037) (in 01:02:49:46s)
Improvement at epoch 01115: 0.731828(030) (in 01:02:56:02s)
Improvement at epoch 01117: 0.731896(028) (in 01:03:00:18s)
Improvement at epoch 01120: 0.732099(045) (in 01:03:06:33s)
Improvement at epoch 01122: 0.732522(073) (in 01:03:10:44s)
Improvement at epoch 01126: 0.732556(081) (in 01:03:19:10s)
Improvement at epoch 01127: 0.732647(090) (in 01:03:21:10s)
Improvement at epoch 01128: 0.732703(097) (in 01:03:23:17s)
Improvement at epoch 01129: 0.732831(093) (in 01:03:25:17s)
Improvement at epoch 01131: 0.733039(049) (in 01:03:29:29s)
Improvement at epoch 01132: 0.733177(098) (in 01:03:31:35s)
Improvement at epoch 01133: 0.733386(085) (in 01:03:33:45s)
Improvement at epoch 01134: 0.733582(055) (in 01:03:35:51s)
Improvement at epoch 01135: 0.733766(054) (in 01:03:37:56s)
Improvement at epoch 01137: 0.733800(062) (in 01:03:42:05s)
Improvement at epoch 01138: 0.733874(045) (in 01:03:44:11s)
Improvement at epoch 01139: 0.733931(068) (in 01:03:46:20s)
Improvement at epoch 01141: 0.734143(046) (in 01:03:50:32s)
Improvement at epoch 01145: 0.734264(037) (in 01:03:58:54s)
Improvement at epoch 01147: 0.734285(078) (in 01:04:03:10s)
Improvement at epoch 01148: 0.734434(027) (in 01:04:05:15s)
Improvement at epoch 01150: 0.734477(029) (in 01:04:09:28s)
Improvement at epoch 01152: 0.734515(037) (in 01:04:13:43s)
Improvement at epoch 01153: 0.734551(046) (in 01:04:15:47s)
Improvement at epoch 01154: 0.734607(080) (in 01:04:17:55s)
Improvement at epoch 01155: 0.734610(074) (in 01:04:20:04s)
Improvement at epoch 01157: 0.734621(090) (in 01:04:24:13s)
Improvement at epoch 01160: 0.734689(075) (in 01:04:30:30s)
Improvement at epoch 01164: 0.734701(060) (in 01:04:38:47s)
Improvement at epoch 01165: 0.734704(073) (in 01:04:40:50s)
Improvement at epoch 01166: 0.734730(071) (in 01:04:42:55s)
Improvement at epoch 01167: 0.734759(081) (in 01:04:44:57s)
Improvement at epoch 01170: 0.734889(046) (in 01:04:51:13s)
Improvement at epoch 01171: 0.734942(055) (in 01:04:53:16s)
Improvement at epoch 01172: 0.735009(057) (in 01:04:55:25s)
Improvement at epoch 01174: 0.735110(032) (in 01:04:59:35s)
Improvement at epoch 01177: 0.735228(078) (in 01:05:05:50s)
```

```
Improvement at epoch 01181: 0.735339(036) (in 01:05:14:11s)
Improvement at epoch 01182: 0.735386(071) (in 01:05:16:18s)
Improvement at epoch 01184: 0.735670(086) (in 01:05:20:28s)
Improvement at epoch 01192: 0.735711(048) (in 01:05:37:09s)
Improvement at epoch 01194: 0.735712(066) (in 01:05:41:14s)
Improvement at epoch 01195: 0.735839(075) (in 01:05:43:15s)
Improvement at epoch 01196: 0.735930(096) (in 01:05:45:22s)
Improvement at epoch 01198: 0.736005(049) (in 01:05:49:32s)
Improvement at epoch 01200: 0.736097(065) (in 01:05:53:49s)
Improvement at epoch 01202: 0.736176(047) (in 01:05:57:55s)
Improvement at epoch 01204: 0.736208(037) (in 01:06:02:08s)
Improvement at epoch 01205: 0.736412(039) (in 01:06:04:13s)
Improvement at epoch 01208: 0.736459(041) (in 01:06:10:30s)
Improvement at epoch 01213: 0.736654(048) (in 01:06:20:01s)
Improvement at epoch 01214: 0.736703(099) (in 01:06:21:46s)
Improvement at epoch 01220: 0.736742(076) (in 01:06:32:13s)
Improvement at epoch 01223: 0.736764(040) (in 01:06:38:40s)
Improvement at epoch 01225: 0.736843(028) (in 01:06:43:04s)
Improvement at epoch 01227: 0.736896(041) (in 01:06:47:36s)
Improvement at epoch 01228: 0.736896(074) (in 01:06:49:50s)
Improvement at epoch 01229: 0.737111(028) (in 01:06:52:06s)
Improvement at epoch 01234: 0.737348(073) (in 01:07:03:16s)
Improvement at epoch 01242: 0.737407(079) (in 01:07:17:36s)
Improvement at epoch 01243: 0.737467(037) (in 01:07:19:51s)
Improvement at epoch 01245: 0.737500(071) (in 01:07:24:12s)
Improvement at epoch 01247: 0.737567(048) (in 01:07:28:35s)
Improvement at epoch 01250: 0.737610(075) (in 01:07:35:19s)
Improvement at epoch 01253: 0.737704(066) (in 01:07:41:50s)
Improvement at epoch 01257: 0.737746(071) (in 01:07:50:35s)
Improvement at epoch 01258: 0.737817(048) (in 01:07:52:45s)
Improvement at epoch 01260: 0.737826(092) (in 01:07:57:04s)
Improvement at epoch 01262: 0.737934(099) (in 01:08:01:20s)
Improvement at epoch 01269: 0.737969(026) (in 01:08:15:56s)
Improvement at epoch 01270: 0.738260(070) (in 01:08:17:56s)
Improvement at epoch 01273: 0.738372(031) (in 01:08:24:12s)
Improvement at epoch 01275: 0.739052(032) (in 01:08:28:20s)
Improvement at epoch 01277: 0.739098(055) (in 01:08:32:28s)
Improvement at epoch 01278: 0.739286(062) (in 01:08:34:37s)
Improvement at epoch 01279: 0.739614(078) (in 01:08:36:39s)
Improvement at epoch 01281: 0.739685(036) (in 01:08:40:46s)
Improvement at epoch 01282: 0.739688(057) (in 01:08:42:48s)
Improvement at epoch 01283: 0.740172(056) (in 01:08:44:53s)
Improvement at epoch 01286: 0.740218(043) (in 01:08:51:03s)
Improvement at epoch 01287: 0.740423(079) (in 01:08:53:07s)
Improvement at epoch 01288: 0.740424(071) (in 01:08:55:11s)
Improvement at epoch 01289: 0.740655(062) (in 01:08:57:13s)
Improvement at epoch 01291: 0.740732(083) (in 01:09:01:21s)
Improvement at epoch 01292: 0.740978(032) (in 01:09:03:27s)
Improvement at epoch 01297: 0.741216(043) (in 01:09:14:01s)
Improvement at epoch 01301: 0.741256(075) (in 01:09:22:35s)
Improvement at epoch 01307: 0.741600(093) (in 01:09:35:27s)
Improvement at epoch 01314: 0.741640(070) (in 01:09:50:26s)
Improvement at epoch 01318: 0.741866(059) (in 01:09:59:03s)
Improvement at epoch 01322: 0.741880(036) (in 01:10:07:44s)
Improvement at epoch 01327: 0.741908(071) (in 01:10:18:34s)
Improvement at epoch 01328: 0.741987(095) (in 01:10:20:47s)
Improvement at epoch 01329: 0.741993(046) (in 01:10:22:56s)
Improvement at epoch 01331: 0.742075(072) (in 01:10:27:16s)
Improvement at epoch 01334: 0.742085(047) (in 01:10:33:52s)
Improvement at epoch 01336: 0.742105(090) (in 01:10:38:21s)
Improvement at epoch 01337: 0.742172(051) (in 01:10:40:32s)
Improvement at epoch 01340: 0.742206(028) (in 01:10:47:08s)
```

```
Improvement at epoch 01341: 0.742255(028) (in 01:10:49:18s)
Improvement at epoch 01344: 0.742367(099) (in 01:10:55:41s)
Improvement at epoch 01355: 0.742422(034) (in 01:11:19:24s)
Improvement at epoch 01357: 0.742453(049) (in 01:11:23:43s)
Improvement at epoch 01359: 0.742598(041) (in 01:11:27:57s)
Improvement at epoch 01362: 0.742636(075) (in 01:11:34:12s)
Improvement at epoch 01364: 0.742727(029) (in 01:11:38:34s)
Improvement at epoch 01366: 0.742811(059) (in 01:11:42:50s)
Improvement at epoch 01368: 0.742991(076) (in 01:11:47:06s)
Improvement at epoch 01371: 0.743229(022) (in 01:11:53:29s)
Improvement at epoch 01381: 0.743311(090) (in 01:12:14:42s)
Improvement at epoch 01383: 0.743315(055) (in 01:12:18:59s)
Improvement at epoch 01387: 0.743398(097) (in 01:12:27:31s)
Improvement at epoch 01390: 0.743468(056) (in 01:12:34:03s)
Improvement at epoch 01394: 0.743597(033) (in 01:12:42:43s)
Improvement at epoch 01400: 0.743706(075) (in 01:12:55:34s)
Improvement at epoch 01402: 0.743711(078) (in 01:12:59:52s)
Improvement at epoch 01404: 0.743716(023) (in 01:13:04:07s)
Improvement at epoch 01407: 0.743745(027) (in 01:13:10:32s)
Improvement at epoch 01410: 0.743863(093) (in 01:13:16:56s)
Improvement at epoch 01411: 0.744128(039) (in 01:13:19:03s)
Improvement at epoch 01417: 0.744129(081) (in 01:13:31:57s)
Improvement at epoch 01418: 0.744248(096) (in 01:13:34:06s)
Improvement at epoch 01420: 0.744285(045) (in 01:13:38:26s)
Improvement at epoch 01423: 0.744332(045) (in 01:13:44:49s)
Improvement at epoch 01426: 0.744402(028) (in 01:13:51:16s)
Improvement at epoch 01429: 0.744633(097) (in 01:13:57:44s)
Improvement at epoch 01437: 0.744697(083) (in 01:14:15:09s)
Improvement at epoch 01439: 0.744726(035) (in 01:14:19:29s)
Improvement at epoch 01440: 0.744861(081) (in 01:14:21:39s)
Improvement at epoch 01443: 0.744895(093) (in 01:14:28:12s)
Improvement at epoch 01450: 0.744982(053) (in 01:14:43:13s)
Improvement at epoch 01453: 0.744984(027) (in 01:14:48:46s)
Improvement at epoch 01456: 0.745053(029) (in 01:14:54:02s)
```

checktest.txt:

checkall.txt:

```
Loading 60000 images...
Loaded MNIST successfully.
Created dataset with 60000 samples
         0 (5923): 0.952558
         1 (6742): 0.940522
         2 (5958): 0.861866
         3 (6131): 0.853042
         4 (5842): 0.879493
         5 (5421): 0.807785
         6 (5918): 0.929537
         7 (6265): 0.861453
         8 (5851): 0.832507
         9 (5949): 0.814254
Value: 0.874633
```

7.6 ORHD

Source: https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "genalg.h"
#include "neuranet.h"

// https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

// Nb of step between each save of the GenAlg
// Saving it allows to restart a stop learning process but is
// very time consuming if there are many input/hidden/output
// If 0 never save
#define SAVE_GA_EVERY 100
// Nb input and output of the NeuraNet
#define NB_INPUT 64
#define NB_OUTPUT 10
// Nb max of hidden values, links and base functions
#define NB_MAXHIDDEN 1
#define NB_MAXLINK 100
#define NB_MAXBASE NB_MAXLINK
// Size of the gene pool and elite pool
#define ADN_SIZE_POOL 100
#define ADN_SIZE_ELITE 20
// Initial best value during learning, must be lower than any
// possible value returned by Evaluate()
#define INIT_BEST_VAL -1000.0
// Value of the NeuraNet above which the learning process stops
#define STOP_LEARNING_AT_VAL 0.999
// Number of epoch above which the learning process stops
#define STOP_LEARNING_AT_EPOCH 5000
// Save NeuraNet in compact format
#define COMPACT true

// Categories of data sets

typedef enum DataSetCat {
  unknownDataSet,
  datalearn,
  datatest,
  dataall
} DataSetCat;
#define NB_DATASET 4
const char* dataSetNames[NB_DATASET] = {
  "unknownDataSet", "datalearn", "datatest", "dataall"
  };

const char* catNames[NB_OUTPUT] = {
  "0",
  "1",
  "2",
  "3",
  "4",
  "5",
  "6",
```

```
  "7",
  "8",
  "9",
};

// Structure for the data set

typedef struct ORHD {
  float _props[NB_INPUT];
  int _cat;
} ORHD;

typedef struct DataSet {
  // Category of the data set
  DataSetCat _cat;
  // Number of sample
  unsigned int _nbSample;
  // Samples
  ORHD* _samples;
} DataSet;

// Get the DataSetCat from its 'name'
DataSetCat GetCategoryFromName(const char* const name) {
  // Declare a variable to memorize the DataSetCat
  DataSetCat cat = unknownDataSet;
  // Search the dataset
  for (int iSet = NB_DATASET; iSet--;)
    if (strcmp(name, dataSetNames[iSet]) == 0)
      cat = iSet;
  // Return the category
  return cat;
}

// Load the data set of category 'cat' in the DataSet 'that'
// Return true on success, else false
// Load the data set of category 'cat' in the DataSet 'that'
// Return true on success, else false

bool DataSetLoad(DataSet* const that, const DataSetCat cat) {
  // Set the category
  that->_cat = cat;

  // Load the data according to 'cat'
  int ret = 0;
  if (cat == datalearn) {
    FILE* f = fopen("./optdigits.tra", "r");
    if (f == NULL) {
      printf("Couldn't open the data set file\n");
      return false;
    }
    that->_nbSample = 3823;
    that->_samples =
      PBErrMalloc(NeuraNetErr, sizeof(ORHD) * that->_nbSample);
    for (unsigned int iSample = 0; iSample < that->_nbSample; ++iSample) {
      ret = fscanf(f, "%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%
        that->_samples[iSample]._props,
        that->_samples[iSample]._props + 1,
        that->_samples[iSample]._props + 2,
        that->_samples[iSample]._props + 3,
        that->_samples[iSample]._props + 4,
        that->_samples[iSample]._props + 5,
        that->_samples[iSample]._props + 6,
```

```
  that->_samples[iSample]._props + 7,
  that->_samples[iSample]._props + 8,
  that->_samples[iSample]._props + 9,
  that->_samples[iSample]._props + 10,
  that->_samples[iSample]._props + 11,
  that->_samples[iSample]._props + 12,
  that->_samples[iSample]._props + 13,
  that->_samples[iSample]._props + 14,
  that->_samples[iSample]._props + 15,
  that->_samples[iSample]._props + 16,
  that->_samples[iSample]._props + 17,
  that->_samples[iSample]._props + 18,
  that->_samples[iSample]._props + 19,
  that->_samples[iSample]._props + 20,
  that->_samples[iSample]._props + 21,
  that->_samples[iSample]._props + 22,
  that->_samples[iSample]._props + 23,
  that->_samples[iSample]._props + 24,
  that->_samples[iSample]._props + 25,
  that->_samples[iSample]._props + 26,
  that->_samples[iSample]._props + 27,
  that->_samples[iSample]._props + 28,
  that->_samples[iSample]._props + 29,
  that->_samples[iSample]._props + 30,
  that->_samples[iSample]._props + 31,
  that->_samples[iSample]._props + 32,
  that->_samples[iSample]._props + 33,
  that->_samples[iSample]._props + 34,
  that->_samples[iSample]._props + 35,
  that->_samples[iSample]._props + 36,
  that->_samples[iSample]._props + 37,
  that->_samples[iSample]._props + 38,
  that->_samples[iSample]._props + 39,
  that->_samples[iSample]._props + 40,
  that->_samples[iSample]._props + 41,
  that->_samples[iSample]._props + 42,
  that->_samples[iSample]._props + 43,
  that->_samples[iSample]._props + 44,
  that->_samples[iSample]._props + 45,
  that->_samples[iSample]._props + 46,
  that->_samples[iSample]._props + 47,
  that->_samples[iSample]._props + 48,
  that->_samples[iSample]._props + 49,
  that->_samples[iSample]._props + 50,
  that->_samples[iSample]._props + 51,
  that->_samples[iSample]._props + 52,
  that->_samples[iSample]._props + 53,
  that->_samples[iSample]._props + 54,
  that->_samples[iSample]._props + 55,
  that->_samples[iSample]._props + 56,
  that->_samples[iSample]._props + 57,
  that->_samples[iSample]._props + 58,
  that->_samples[iSample]._props + 59,
  that->_samples[iSample]._props + 60,
  that->_samples[iSample]._props + 61,
  that->_samples[iSample]._props + 62,
  that->_samples[iSample]._props + 63,
  &(that->_samples[iSample]._cat));
for (int iProp = 64; iProp--;) {
  that->_samples[iSample]._props[iProp] /= 8.0;
  that->_samples[iSample]._props[iProp] -= 1.0;
}
```

```
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
  }
  fclose(f);
} else if (cat == datatest) {
  FILE* f = fopen("./optdigits.tes", "r");
  if (f == NULL) {
    printf("Couldn't open the data set file\n");
    return false;
  }
  that->_nbSample = 1797;
  that->_samples =
    PBErrMalloc(NeuraNetErr, sizeof(ORHD) * that->_nbSample);
  for (unsigned int iSample = 0; iSample < that->_nbSample; ++iSample) {
    ret = fscanf(f, "%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%
      that->_samples[iSample]._props,
      that->_samples[iSample]._props + 1,
      that->_samples[iSample]._props + 2,
      that->_samples[iSample]._props + 3,
      that->_samples[iSample]._props + 4,
      that->_samples[iSample]._props + 5,
      that->_samples[iSample]._props + 6,
      that->_samples[iSample]._props + 7,
      that->_samples[iSample]._props + 8,
      that->_samples[iSample]._props + 9,
      that->_samples[iSample]._props + 10,
      that->_samples[iSample]._props + 11,
      that->_samples[iSample]._props + 12,
      that->_samples[iSample]._props + 13,
      that->_samples[iSample]._props + 14,
      that->_samples[iSample]._props + 15,
      that->_samples[iSample]._props + 16,
      that->_samples[iSample]._props + 17,
      that->_samples[iSample]._props + 18,
      that->_samples[iSample]._props + 19,
      that->_samples[iSample]._props + 20,
      that->_samples[iSample]._props + 21,
      that->_samples[iSample]._props + 22,
      that->_samples[iSample]._props + 23,
      that->_samples[iSample]._props + 24,
      that->_samples[iSample]._props + 25,
      that->_samples[iSample]._props + 26,
      that->_samples[iSample]._props + 27,
      that->_samples[iSample]._props + 28,
      that->_samples[iSample]._props + 29,
      that->_samples[iSample]._props + 30,
      that->_samples[iSample]._props + 31,
      that->_samples[iSample]._props + 32,
      that->_samples[iSample]._props + 33,
      that->_samples[iSample]._props + 34,
      that->_samples[iSample]._props + 35,
      that->_samples[iSample]._props + 36,
      that->_samples[iSample]._props + 37,
      that->_samples[iSample]._props + 38,
      that->_samples[iSample]._props + 39,
      that->_samples[iSample]._props + 40,
      that->_samples[iSample]._props + 41,
      that->_samples[iSample]._props + 42,
      that->_samples[iSample]._props + 43,
```

```
      that->_samples[iSample]._props + 44,
      that->_samples[iSample]._props + 45,
      that->_samples[iSample]._props + 46,
      that->_samples[iSample]._props + 47,
      that->_samples[iSample]._props + 48,
      that->_samples[iSample]._props + 49,
      that->_samples[iSample]._props + 50,
      that->_samples[iSample]._props + 51,
      that->_samples[iSample]._props + 52,
      that->_samples[iSample]._props + 53,
      that->_samples[iSample]._props + 54,
      that->_samples[iSample]._props + 55,
      that->_samples[iSample]._props + 56,
      that->_samples[iSample]._props + 57,
      that->_samples[iSample]._props + 58,
      that->_samples[iSample]._props + 59,
      that->_samples[iSample]._props + 60,
      that->_samples[iSample]._props + 61,
      that->_samples[iSample]._props + 62,
      that->_samples[iSample]._props + 63,
      &(that->_samples[iSample]._cat));
    for (int iProp = 64; iProp--;) {
      that->_samples[iSample]._props[iProp] /= 8.0;
      that->_samples[iSample]._props[iProp] -= 1.0;
    }
    if (ret == EOF) {
      printf("Couldn't read the dataset\n");
      fclose(f);
      return false;
    }
  }
  fclose(f);
} else if (cat == dataall) {
  that->_nbSample = 3823 + 1797;
  that->_samples =
    PBErrMalloc(NeuraNetErr, sizeof(ORHD) * that->_nbSample);
  FILE* f = fopen("./optdigits.tra", "r");
  if (f == NULL) {
    printf("Couldn't open the data set file\n");
    return false;
  }
  for (unsigned int iSample = 0; iSample < 3823; ++iSample) {
    ret = fscanf(f, "%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%
      that->_samples[iSample]._props,
      that->_samples[iSample]._props + 1,
      that->_samples[iSample]._props + 2,
      that->_samples[iSample]._props + 3,
      that->_samples[iSample]._props + 4,
      that->_samples[iSample]._props + 5,
      that->_samples[iSample]._props + 6,
      that->_samples[iSample]._props + 7,
      that->_samples[iSample]._props + 8,
      that->_samples[iSample]._props + 9,
      that->_samples[iSample]._props + 10,
      that->_samples[iSample]._props + 11,
      that->_samples[iSample]._props + 12,
      that->_samples[iSample]._props + 13,
      that->_samples[iSample]._props + 14,
      that->_samples[iSample]._props + 15,
      that->_samples[iSample]._props + 16,
      that->_samples[iSample]._props + 17,
      that->_samples[iSample]._props + 18,
```

```
    that->_samples[iSample]._props + 19,
    that->_samples[iSample]._props + 20,
    that->_samples[iSample]._props + 21,
    that->_samples[iSample]._props + 22,
    that->_samples[iSample]._props + 23,
    that->_samples[iSample]._props + 24,
    that->_samples[iSample]._props + 25,
    that->_samples[iSample]._props + 26,
    that->_samples[iSample]._props + 27,
    that->_samples[iSample]._props + 28,
    that->_samples[iSample]._props + 29,
    that->_samples[iSample]._props + 30,
    that->_samples[iSample]._props + 31,
    that->_samples[iSample]._props + 32,
    that->_samples[iSample]._props + 33,
    that->_samples[iSample]._props + 34,
    that->_samples[iSample]._props + 35,
    that->_samples[iSample]._props + 36,
    that->_samples[iSample]._props + 37,
    that->_samples[iSample]._props + 38,
    that->_samples[iSample]._props + 39,
    that->_samples[iSample]._props + 40,
    that->_samples[iSample]._props + 41,
    that->_samples[iSample]._props + 42,
    that->_samples[iSample]._props + 43,
    that->_samples[iSample]._props + 44,
    that->_samples[iSample]._props + 45,
    that->_samples[iSample]._props + 46,
    that->_samples[iSample]._props + 47,
    that->_samples[iSample]._props + 48,
    that->_samples[iSample]._props + 49,
    that->_samples[iSample]._props + 50,
    that->_samples[iSample]._props + 51,
    that->_samples[iSample]._props + 52,
    that->_samples[iSample]._props + 53,
    that->_samples[iSample]._props + 54,
    that->_samples[iSample]._props + 55,
    that->_samples[iSample]._props + 56,
    that->_samples[iSample]._props + 57,
    that->_samples[iSample]._props + 58,
    that->_samples[iSample]._props + 59,
    that->_samples[iSample]._props + 60,
    that->_samples[iSample]._props + 61,
    that->_samples[iSample]._props + 62,
    that->_samples[iSample]._props + 63,
    &(that->_samples[iSample]._cat));
  for (int iProp = 64; iProp--;) {
    that->_samples[iSample]._props[iProp] /= 8.0;
    that->_samples[iSample]._props[iProp] -= 1.0;
  }
  if (ret == EOF) {
    printf("Couldn't read the dataset\n");
    fclose(f);
    return false;
  }
}
fclose(f);
f = fopen("./optdigits.tes", "r");
if (f == NULL) {
  printf("Couldn't open the data set file\n");
  return false;
}
```

```
for (unsigned int iSample = 3824; iSample < 3823 + 1797; ++iSample) {
  ret = fscanf(f, "%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%
    that->_samples[iSample]._props,
    that->_samples[iSample]._props + 1,
    that->_samples[iSample]._props + 2,
    that->_samples[iSample]._props + 3,
    that->_samples[iSample]._props + 4,
    that->_samples[iSample]._props + 5,
    that->_samples[iSample]._props + 6,
    that->_samples[iSample]._props + 7,
    that->_samples[iSample]._props + 8,
    that->_samples[iSample]._props + 9,
    that->_samples[iSample]._props + 10,
    that->_samples[iSample]._props + 11,
    that->_samples[iSample]._props + 12,
    that->_samples[iSample]._props + 13,
    that->_samples[iSample]._props + 14,
    that->_samples[iSample]._props + 15,
    that->_samples[iSample]._props + 16,
    that->_samples[iSample]._props + 17,
    that->_samples[iSample]._props + 18,
    that->_samples[iSample]._props + 19,
    that->_samples[iSample]._props + 20,
    that->_samples[iSample]._props + 21,
    that->_samples[iSample]._props + 22,
    that->_samples[iSample]._props + 23,
    that->_samples[iSample]._props + 24,
    that->_samples[iSample]._props + 25,
    that->_samples[iSample]._props + 26,
    that->_samples[iSample]._props + 27,
    that->_samples[iSample]._props + 28,
    that->_samples[iSample]._props + 29,
    that->_samples[iSample]._props + 30,
    that->_samples[iSample]._props + 31,
    that->_samples[iSample]._props + 32,
    that->_samples[iSample]._props + 33,
    that->_samples[iSample]._props + 34,
    that->_samples[iSample]._props + 35,
    that->_samples[iSample]._props + 36,
    that->_samples[iSample]._props + 37,
    that->_samples[iSample]._props + 38,
    that->_samples[iSample]._props + 39,
    that->_samples[iSample]._props + 40,
    that->_samples[iSample]._props + 41,
    that->_samples[iSample]._props + 42,
    that->_samples[iSample]._props + 43,
    that->_samples[iSample]._props + 44,
    that->_samples[iSample]._props + 45,
    that->_samples[iSample]._props + 46,
    that->_samples[iSample]._props + 47,
    that->_samples[iSample]._props + 48,
    that->_samples[iSample]._props + 49,
    that->_samples[iSample]._props + 50,
    that->_samples[iSample]._props + 51,
    that->_samples[iSample]._props + 52,
    that->_samples[iSample]._props + 53,
    that->_samples[iSample]._props + 54,
    that->_samples[iSample]._props + 55,
    that->_samples[iSample]._props + 56,
    that->_samples[iSample]._props + 57,
    that->_samples[iSample]._props + 58,
    that->_samples[iSample]._props + 59,
```

```
        that->_samples[iSample]._props + 60,
        that->_samples[iSample]._props + 61,
        that->_samples[iSample]._props + 62,
        that->_samples[iSample]._props + 63,
        &(that->_samples[iSample]._cat));
      for (int iProp = 64; iProp--;) {
        that->_samples[iSample]._props[iProp] /= 8.0;
        that->_samples[iSample]._props[iProp] -= 1.0;
      }
      if (ret == EOF) {
        printf("Couldn't read the dataset\n");
        fclose(f);
        return false;
      }
    }
    fclose(f);
  } else {
    printf("Invalid dataset\n");
    return false;
  }

  // Return success code
  return true;
}

// Free memory for the DataSet 'that'
void DataSetFree(DataSet** that) {
  if (*that == NULL) return;
  // Free the memory

  free(*that);
  *that = NULL;
}

// Evalutation function for the NeuraNet 'that' on the DataSet 'dataset'
// Return the value of the NeuraNet, the bigger the better
float Evaluate(const NeuraNet* const that,
  const DataSet* const dataset) {
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Declare a variable to memorize the value
  float val = 0.0;

  // Evaluate

  int countCat[NB_OUTPUT] = {0};
  int countOk[NB_OUTPUT] = {0};
  int countNg[NB_OUTPUT] = {0};
  for (unsigned int iSample = dataset->_nbSample; iSample--;) {
    for (int iInp = 0; iInp < NNGetNbInput(that); ++iInp) {
      VecSet(input, iInp,
        dataset->_samples[iSample]._props[iInp]);
    }
    NNEval(that, input, output);
    int pred = VecGetIMaxVal(output);
    ++(countCat[dataset->_samples[iSample]._cat]);
    if (pred == dataset->_samples[iSample]._cat) {
      ++(countOk[dataset->_samples[iSample]._cat]);
    } else if (dataset->_cat == datalearn) {
      ++(countNg[dataset->_samples[iSample]._cat]);
    }
```

```
  }

  int nbCat = 0;
  for (int iCat = 0; iCat < NB_OUTPUT; ++iCat) {
    if (countCat[iCat] > 0) {
      ++nbCat;
      float perc = 0.0;
      if (dataset->_cat != datalearn) {
        perc = (float)(countOk[iCat]) / (float)(countCat[iCat]);
        printf("%10s (%4d): %f\n",
          catNames[iCat], countCat[iCat], perc);
        val += countOk[iCat];
      } else {
        perc = (float)(countOk[iCat] - countNg[iCat]) /
          (float)(countCat[iCat]);
        val += perc;
      }
    }
  }

  if (dataset->_cat != datalearn)
    val /= (float)(dataset->_nbSample);
  else
    val /= (float)nbCat;

  // Free memory
  VecFree(&input);
  VecFree(&output);
  // Return the result of the evaluation
  return val;
}

// Create the NeuraNet
NeuraNet* createNN(void) {
  // Create the NeuraNet
  int nbIn = NB_INPUT;
  int nbOut = NB_OUTPUT;
  int nbMaxHid = NB_MAXHIDDEN;
  int nbMaxLink = NB_MAXLINK;
  int nbMaxBase = NB_MAXBASE;
  NeuraNet* nn =
    NeuraNetCreate(nbIn, nbOut, nbMaxHid, nbMaxBase, nbMaxLink);
  // Return the NeuraNet
  return nn;
}

// Learn based on the DataSetCat 'cat'
void Learn(DataSetCat cat) {
  // Init the random generator
  srandom(time(NULL));
  // Declare variables to measure time
  struct timespec start, stop;
  // Start measuring time
  clock_gettime(CLOCK_REALTIME, &start);
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Create the NeuraNet
```

```
NeuraNet* nn = createNN();
// Declare a variable to memorize the best value
float bestVal = INIT_BEST_VAL;
// Declare a variable to memorize the limit in term of epoch
unsigned long int limitEpoch = STOP_LEARNING_AT_EPOCH;
// Create the GenAlg used for learning
// If previous weights are available in "./bestga.txt" reload them
GenAlg* ga = NULL;
FILE* fd = fopen("./bestga.txt", "r");
if (fd) {
  printf("Reloading previous GenAlg...\n");
  if (!GALoad(&ga, fd)) {
    printf("Failed to reload the GenAlg.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous GenAlg reloaded.\n");
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GABestAdnF(ga));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GABestAdnI(ga));
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    limitEpoch += GAGetCurEpoch(ga);
  }
  fclose(fd);
} else {
  printf("Creating new GenAlg...\n");
  fflush(stdout);
  ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
    NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  // Must be declared as a GenAlg applied to a NeuraNet with
  // convolution
  GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
  GAInit(ga);
}
// If there is a NeuraNet available, reload it into the GenAlg
fd = fopen("./bestnn.txt", "r");
if (fd) {
  printf("Reloading previous NeuraNet...\n");
  if (!NNLoad(&nn, fd)) {
    printf("Failed to reload the NeuraNet.\n");
    NeuraNetFree(&nn);
    DataSetFree(&dataset);
    return;
  } else {
    printf("Previous NeuraNet reloaded.\n");
    bestVal = Evaluate(nn, dataset);
    printf("Starting with best at %f.\n", bestVal);
    GenAlgAdn* adn = GAAdn(ga, 0);
    if (adn->_adnF)
      VecCopy(adn->_adnF, nn->_bases);
    if (adn->_adnI)
      VecCopy(adn->_adnI, nn->_links);
  }
  fclose(fd);
}
// Start learning process
printf("Learning...\n");
```

```
printf("Will stop when curEpoch >= %lu or bestVal >= %f\n",
  limitEpoch, STOP_LEARNING_AT_VAL);
printf("Will save the best NeuraNet in ./bestnn.txt at each improvement\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = 0.0;
float curWorst = 0.0;
// Declare a variable to manage the save of GenAlg
int delaySave = 0;
// Learning loop
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  curWorst = curBest;
  curBest = INIT_BEST_VAL;
  int curBestI = 0;
  unsigned long int ageBest = 0;
  // For each adn in the GenAlg
  //for (int iEnt = GAGetNbAdns(ga); iEnt--;) {
  for (int iEnt = 0; iEnt < GAGetNbAdns(ga); ++iEnt) {
    // Get the adn
    GenAlgAdn* adn = GAAdn(ga, iEnt);
    // Set the links and base functions of the NeuraNet according
    // to this adn
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GAAdnAdnF(adn));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GAAdnAdnI(adn));
    // Evaluate the NeuraNet
    float value = Evaluate(nn, dataset);
    // Update the value of this adn
    GASetAdnValue(ga, adn, value);
    // Update the best value in the current epoch
    if (value > curBest) {
      curBest = value;
      curBestI = iEnt;
      ageBest = GAAdnGetAge(adn);
    }
    if (value < curWorst)
      curWorst = value;
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  // If there has been improvement during this epoch
  if (curBest > bestVal) {
    bestVal = curBest;
    // Display info about the improvment
    printf("Improvement at epoch %05lu: %f(%03d) (in %02d:%02d:%02d:%02ds)       \n",
      GAGetCurEpoch(ga), bestVal, curBestI, day, hour, min, sec);
    fflush(stdout);
    // Set the links and base functions of the NeuraNet according
    // to the best adn
    GenAlgAdn* bestAdn = GAAdn(ga, curBestI);
    if (GAAdnAdnF(bestAdn) != NULL)
      NNSetBases(nn, GAAdnAdnF(bestAdn));
```

```
      if (GAAdnAdnI(bestAdn) != NULL)
        NNSetLinks(nn, GAAdnAdnI(bestAdn));
      // Save the best NeuraNet
      fd = fopen("./bestnn.txt", "w");
      if (!NNSave(nn, fd, COMPACT)) {
        printf("Couldn't save the NeuraNet\n");
        NeuraNetFree(&nn);
        GenAlgFree(&ga);
        DataSetFree(&dataset);
        return;
      }
      fclose(fd);
    } else {
      fprintf(stderr,
        "Epoch %05lu: v%f a%03lu(%03d) kt%03lu ",
        GAGetCurEpoch(ga), curBest, ageBest, curBestI,
        GAGetNbKTEvent(ga));
      fprintf(stderr, "(in %02d:%02d:%02d:%02ds)  \r",
        day, hour, min, sec);
      fflush(stderr);
    }
    ++delaySave;
    if (SAVE_GA_EVERY != 0 && delaySave >= SAVE_GA_EVERY) {
      delaySave = 0;
      // Save the adns of the GenAlg, use a temporary file to avoid
      // loosing the previous one if something goes wrong during
      // writing, then replace the previous file with the temporary one
      fd = fopen("./bestga.tmp", "w");
      if (!GASave(ga, fd, COMPACT)) {
        printf("Couldn't save the GenAlg\n");
        NeuraNetFree(&nn);
        GenAlgFree(&ga);
        DataSetFree(&dataset);
        return;
      }
      fclose(fd);
      int ret = system("mv ./bestga.tmp ./bestga.txt");
      (void)ret;
    }
    // Step the GenAlg
    GAStep(ga);
  }
  // Measure time
  clock_gettime(CLOCK_REALTIME, &stop);
  float elapsed = stop.tv_sec - start.tv_sec;
  int day = (int)floor(elapsed / 86400);
  elapsed -= (float)(day * 86400);
  int hour = (int)floor(elapsed / 3600);
  elapsed -= (float)(hour * 3600);
  int min = (int)floor(elapsed / 60);
  elapsed -= (float)(min * 60);
  int sec = (int)floor(elapsed);
  printf("\nLearning complete (in %d:%d:%d:%ds)\n",
    day, hour, min, sec);
  fflush(stdout);
  // Free memory
  NeuraNetFree(&nn);
  GenAlgFree(&ga);
  DataSetFree(&dataset);
}

// Check the NeuraNet 'that' on the DataSetCat 'cat'
```

```
void Check(const NeuraNet* const that, const DataSetCat cat) {
  // Load the DataSet
  DataSet* dataset = PBErrMalloc(NeuraNetErr, sizeof(DataSet));
  bool ret = DataSetLoad(dataset, cat);
  if (!ret) {
    printf("Couldn't load the data\n");
    return;
  }
  // Evaluate the NeuraNet
  float value = Evaluate(that, dataset);
  // Display the result
  printf("Value: %.6f\n", value);
  // Free memory
  DataSetFree(&dataset);
}

// Predict using the NeuraNet 'that' on 'inputs' (given as an array of
// 'nbInp' char*)
void Predict(const NeuraNet* const that, const int nbInp,
  char** const inputs) {
  // Start measuring time
  clock_t clockStart = clock();
  // Check the number of inputs
  if (nbInp != NNGetNbInput(that)) {
    printf("Wrong number of inputs, there should %d, there was %d\n",
      NNGetNbInput(that), nbInp);
    return;
  }
  // Declare 2 vectors to memorize the input and output values
  VecFloat* input = VecFloatCreate(NNGetNbInput(that));
  VecFloat* output = VecFloatCreate(NNGetNbOutput(that));
  // Set the input
  for (int iInp = 0; iInp < nbInp; ++iInp) {
    float v = 0.0;
    sscanf(inputs[iInp], "%f", &v);
    VecSet(input, iInp, v);
  }
  // Predict
  NNEval(that, input, output);

  // End measuring time
  clock_t clockEnd = clock();
  double timeUsed =
    ((double)(clockEnd - clockStart)) / (CLOCKS_PER_SEC * 0.001) ;
  // If the clock has been reset meanwhile
  if (timeUsed < 0.0)
    timeUsed = 0.0;
  //if (VecGet(output, 0) == ...)
  //  printf("...(in %fms)", timeUsed);

  // Free memory
  VecFree(&input);
  VecFree(&output);
}

int main(int argc, char** argv) {
  // Declare a variable to memorize the mode (learning/checking)
  int mode = -1;
  // Declare a variable to memorize the dataset used
  DataSetCat cat = unknownDataSet;
  // Decode mode from arguments
  if (argc >= 3) {
```

```
    if (strcmp(argv[1], "-learn") == 0) {
      mode = 0;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-check") == 0) {
      mode = 1;
      cat = GetCategoryFromName(argv[2]);
    } else if (strcmp(argv[1], "-predict") == 0) {
      mode = 2;
    }
  }
  // If the mode is invalid print some help
  if (mode == -1) {
    printf("Select a mode from:\n");
    printf("-learn <dataset name>\n");
    printf("-check <dataset name>\n");
    printf("-predict <input values>\n");
    return 0;
  }
  if (mode == 0) {
    Learn(cat);
  } else if (mode == 1) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Check(nn, cat);
    NeuraNetFree(&nn);
  } else if (mode == 2) {
    NeuraNet* nn = NULL;
    FILE* fd = fopen("./bestnn.txt", "r");
    if (!NNLoad(&nn, fd)) {
      printf("Couldn't load the best NeuraNet\n");
      return 0;
    }
    fclose(fd);
    Predict(nn, argc - 2, argv + 2);
    NeuraNetFree(&nn);
  }
  // Return success code
  return 0;
}
```

learn.txt:

```
Creating new GenAlg...
Learning...
Will stop when curEpoch >= 5000 or bestVal >= 0.999000
Will save the best NeuraNet in ./bestnn.txt at each improvement
Improvement at epoch 00000: -0.622246(026) (in 00:00:00:00s)
Improvement at epoch 00001: -0.615265(037) (in 00:00:00:01s)
Improvement at epoch 00002: -0.548653(039) (in 00:00:00:01s)
Improvement at epoch 00003: -0.541905(039) (in 00:00:00:01s)
Improvement at epoch 00004: -0.506553(092) (in 00:00:00:01s)
Improvement at epoch 00005: -0.500762(035) (in 00:00:00:02s)
Improvement at epoch 00006: -0.444660(081) (in 00:00:00:02s)
Improvement at epoch 00007: -0.435504(037) (in 00:00:00:02s)
Improvement at epoch 00009: -0.418578(023) (in 00:00:00:03s)
```

```
Improvement at epoch 00011: -0.410427(076) (in 00:00:00:03s)
Improvement at epoch 00012: -0.398392(081) (in 00:00:00:04s)
Improvement at epoch 00013: -0.350431(072) (in 00:00:00:04s)
Improvement at epoch 00016: -0.345228(073) (in 00:00:00:05s)
Improvement at epoch 00018: -0.283322(064) (in 00:00:00:06s)
Improvement at epoch 00022: -0.279639(084) (in 00:00:00:08s)
Improvement at epoch 00024: -0.262701(098) (in 00:00:00:08s)
Improvement at epoch 00027: -0.214676(025) (in 00:00:00:10s)
Improvement at epoch 00028: -0.199805(022) (in 00:00:00:10s)
Improvement at epoch 00029: -0.186064(032) (in 00:00:00:11s)
Improvement at epoch 00030: -0.174704(087) (in 00:00:00:11s)
Improvement at epoch 00032: -0.158760(084) (in 00:00:00:12s)
Improvement at epoch 00033: -0.131149(083) (in 00:00:00:13s)
Improvement at epoch 00036: -0.126423(065) (in 00:00:00:14s)
Improvement at epoch 00037: -0.124503(027) (in 00:00:00:15s)
Improvement at epoch 00038: -0.124324(096) (in 00:00:00:16s)
Improvement at epoch 00039: -0.113647(070) (in 00:00:00:16s)
Improvement at epoch 00041: -0.097338(033) (in 00:00:00:17s)
Improvement at epoch 00042: -0.088334(045) (in 00:00:00:18s)
Improvement at epoch 00043: -0.069421(057) (in 00:00:00:19s)
Improvement at epoch 00045: -0.063014(076) (in 00:00:00:20s)
Improvement at epoch 00046: -0.053072(075) (in 00:00:00:21s)
Improvement at epoch 00047: -0.051377(020) (in 00:00:00:22s)
Improvement at epoch 00048: -0.048745(081) (in 00:00:00:22s)
Improvement at epoch 00049: -0.044572(070) (in 00:00:00:23s)
Improvement at epoch 00050: -0.043255(097) (in 00:00:00:24s)
Improvement at epoch 00051: -0.037652(085) (in 00:00:00:25s)
Improvement at epoch 00052: -0.036569(096) (in 00:00:00:26s)
Improvement at epoch 00053: -0.033961(035) (in 00:00:00:26s)
Improvement at epoch 00054: -0.031395(041) (in 00:00:00:27s)
Improvement at epoch 00055: -0.030929(021) (in 00:00:00:28s)
Improvement at epoch 00056: -0.026707(060) (in 00:00:00:29s)
Improvement at epoch 00058: -0.026128(098) (in 00:00:00:31s)
Improvement at epoch 00059: -0.025057(033) (in 00:00:00:32s)
Improvement at epoch 00060: -0.022378(078) (in 00:00:00:32s)
Improvement at epoch 00061: -0.022097(098) (in 00:00:00:33s)
Improvement at epoch 00062: -0.014547(074) (in 00:00:00:34s)
Improvement at epoch 00063: -0.008274(071) (in 00:00:00:35s)
Improvement at epoch 00065: -0.004555(069) (in 00:00:00:37s)
Improvement at epoch 00066: 0.008738(093) (in 00:00:00:38s)
Improvement at epoch 00067: 0.020369(053) (in 00:00:00:39s)
Improvement at epoch 00068: 0.027454(070) (in 00:00:00:40s)
Improvement at epoch 00069: 0.031884(064) (in 00:00:00:41s)
Improvement at epoch 00071: 0.041755(047) (in 00:00:00:42s)
Improvement at epoch 00072: 0.074635(047) (in 00:00:00:43s)
Improvement at epoch 00073: 0.094615(088) (in 00:00:00:44s)
Improvement at epoch 00074: 0.098502(056) (in 00:00:00:45s)
Improvement at epoch 00075: 0.107191(048) (in 00:00:00:46s)
Improvement at epoch 00076: 0.111987(093) (in 00:00:00:47s)
Improvement at epoch 00077: 0.118331(051) (in 00:00:00:48s)
Improvement at epoch 00078: 0.124759(070) (in 00:00:00:49s)
Improvement at epoch 00079: 0.131508(050) (in 00:00:00:50s)
Improvement at epoch 00081: 0.149045(037) (in 00:00:00:52s)
Improvement at epoch 00084: 0.153066(020) (in 00:00:00:55s)
Improvement at epoch 00085: 0.157080(090) (in 00:00:00:56s)
Improvement at epoch 00086: 0.179199(093) (in 00:00:00:57s)
Improvement at epoch 00087: 0.197811(072) (in 00:00:00:58s)
Improvement at epoch 00088: 0.207947(088) (in 00:00:00:59s)
Improvement at epoch 00090: 0.209078(098) (in 00:00:01:01s)
Improvement at epoch 00091: 0.228630(036) (in 00:00:01:02s)
Improvement at epoch 00092: 0.237772(099) (in 00:00:01:03s)
Improvement at epoch 00094: 0.241909(044) (in 00:00:01:06s)
```

```
Improvement at epoch 00095: 0.246291(029) (in 00:00:01:07s)
Improvement at epoch 00097: 0.247899(031) (in 00:00:01:09s)
Improvement at epoch 00098: 0.249008(078) (in 00:00:01:10s)
Improvement at epoch 00099: 0.252476(066) (in 00:00:01:11s)
Improvement at epoch 00101: 0.254373(064) (in 00:00:01:13s)
Improvement at epoch 00102: 0.260812(080) (in 00:00:01:14s)
Improvement at epoch 00103: 0.263753(076) (in 00:00:01:16s)
Improvement at epoch 00105: 0.275831(088) (in 00:00:01:18s)
Improvement at epoch 00107: 0.276822(053) (in 00:00:01:20s)
Improvement at epoch 00109: 0.278476(022) (in 00:00:01:22s)
Improvement at epoch 00110: 0.280222(057) (in 00:00:01:23s)
Improvement at epoch 00111: 0.281737(068) (in 00:00:01:24s)
Improvement at epoch 00113: 0.283463(096) (in 00:00:01:27s)
Improvement at epoch 00114: 0.291894(031) (in 00:00:01:28s)
Improvement at epoch 00115: 0.292875(097) (in 00:00:01:29s)
Improvement at epoch 00116: 0.299647(022) (in 00:00:01:30s)
Improvement at epoch 00118: 0.300977(039) (in 00:00:01:33s)
Improvement at epoch 00119: 0.303022(027) (in 00:00:01:34s)
Improvement at epoch 00120: 0.303314(090) (in 00:00:01:35s)
Improvement at epoch 00121: 0.308742(044) (in 00:00:01:36s)
Improvement at epoch 00123: 0.312426(052) (in 00:00:01:38s)
Improvement at epoch 00124: 0.312732(027) (in 00:00:01:39s)
Improvement at epoch 00125: 0.316393(095) (in 00:00:01:41s)
Improvement at epoch 00128: 0.317577(049) (in 00:00:01:44s)
Improvement at epoch 00129: 0.318116(027) (in 00:00:01:45s)
Improvement at epoch 00130: 0.329030(099) (in 00:00:01:47s)
Improvement at epoch 00131: 0.329270(022) (in 00:00:01:48s)
Improvement at epoch 00133: 0.331126(030) (in 00:00:01:50s)
Improvement at epoch 00134: 0.342726(086) (in 00:00:01:51s)
Improvement at epoch 00135: 0.343335(090) (in 00:00:01:53s)
Improvement at epoch 00136: 0.349863(038) (in 00:00:01:54s)
Improvement at epoch 00138: 0.351683(074) (in 00:00:01:56s)
Improvement at epoch 00139: 0.355326(032) (in 00:00:01:57s)
Improvement at epoch 00140: 0.359733(028) (in 00:00:01:59s)
Improvement at epoch 00141: 0.361369(094) (in 00:00:02:00s)
Improvement at epoch 00142: 0.362252(034) (in 00:00:02:01s)
Improvement at epoch 00143: 0.405269(084) (in 00:00:02:02s)
Improvement at epoch 00146: 0.421818(064) (in 00:00:02:06s)
Improvement at epoch 00147: 0.424388(068) (in 00:00:02:07s)
Improvement at epoch 00149: 0.444830(058) (in 00:00:02:09s)
Improvement at epoch 00152: 0.446907(030) (in 00:00:02:13s)
Improvement at epoch 00156: 0.448016(061) (in 00:00:02:18s)
Improvement at epoch 00157: 0.453738(054) (in 00:00:02:19s)
Improvement at epoch 00158: 0.454601(079) (in 00:00:02:21s)
Improvement at epoch 00159: 0.455880(029) (in 00:00:02:22s)
Improvement at epoch 00160: 0.456450(078) (in 00:00:02:23s)
Improvement at epoch 00161: 0.464766(057) (in 00:00:02:24s)
Improvement at epoch 00164: 0.467756(058) (in 00:00:02:28s)
Improvement at epoch 00165: 0.470275(038) (in 00:00:02:29s)
Improvement at epoch 00166: 0.476219(038) (in 00:00:02:31s)
Improvement at epoch 00167: 0.479738(027) (in 00:00:02:32s)
Improvement at epoch 00169: 0.487178(054) (in 00:00:02:34s)
Improvement at epoch 00170: 0.488326(045) (in 00:00:02:36s)
Improvement at epoch 00171: 0.491487(041) (in 00:00:02:37s)
Improvement at epoch 00172: 0.500015(051) (in 00:00:02:38s)
Improvement at epoch 00176: 0.502567(051) (in 00:00:02:43s)
Improvement at epoch 00177: 0.503659(047) (in 00:00:02:45s)
Improvement at epoch 00178: 0.519449(036) (in 00:00:02:46s)
Improvement at epoch 00179: 0.524779(021) (in 00:00:02:47s)
Improvement at epoch 00181: 0.528903(080) (in 00:00:02:50s)
Improvement at epoch 00182: 0.528981(039) (in 00:00:02:51s)
Improvement at epoch 00183: 0.530038(092) (in 00:00:02:53s)
```

```
Improvement at epoch 00185: 0.534008(034) (in 00:00:02:55s)
Improvement at epoch 00188: 0.534617(067) (in 00:00:02:59s)
Improvement at epoch 00189: 0.535070(071) (in 00:00:03:01s)
Improvement at epoch 00191: 0.535586(073) (in 00:00:03:03s)
Improvement at epoch 00192: 0.536586(056) (in 00:00:03:05s)
Improvement at epoch 00194: 0.539708(028) (in 00:00:03:07s)
Improvement at epoch 00196: 0.541737(062) (in 00:00:03:10s)
Improvement at epoch 00197: 0.543351(086) (in 00:00:03:11s)
Improvement at epoch 00198: 0.544028(057) (in 00:00:03:13s)
Improvement at epoch 00200: 0.546999(078) (in 00:00:03:16s)
Improvement at epoch 00202: 0.548090(036) (in 00:00:03:18s)
Improvement at epoch 00203: 0.549174(029) (in 00:00:03:20s)
Improvement at epoch 00205: 0.550188(043) (in 00:00:03:22s)
Improvement at epoch 00207: 0.551747(033) (in 00:00:03:25s)
Improvement at epoch 00209: 0.552742(057) (in 00:00:03:28s)
Improvement at epoch 00212: 0.554864(027) (in 00:00:03:32s)
Improvement at epoch 00215: 0.556043(020) (in 00:00:03:36s)
Improvement at epoch 00218: 0.557512(089) (in 00:00:03:40s)
Improvement at epoch 00219: 0.559776(038) (in 00:00:03:42s)
Improvement at epoch 00221: 0.566133(065) (in 00:00:03:44s)
Improvement at epoch 00232: 0.567601(023) (in 00:00:04:00s)
Improvement at epoch 00235: 0.570806(030) (in 00:00:04:04s)
Improvement at epoch 00238: 0.572954(060) (in 00:00:04:08s)
Improvement at epoch 00243: 0.574060(031) (in 00:00:04:16s)
Improvement at epoch 00246: 0.575639(056) (in 00:00:04:20s)
Improvement at epoch 00247: 0.576188(083) (in 00:00:04:21s)
Improvement at epoch 00248: 0.576704(077) (in 00:00:04:23s)
Improvement at epoch 00251: 0.577751(056) (in 00:00:04:27s)
Improvement at epoch 00252: 0.578265(071) (in 00:00:04:29s)
Improvement at epoch 00253: 0.580194(095) (in 00:00:04:30s)
Improvement at epoch 00259: 0.580241(073) (in 00:00:04:39s)
Improvement at epoch 00260: 0.582939(024) (in 00:00:04:41s)
Improvement at epoch 00262: 0.584628(020) (in 00:00:04:44s)
Improvement at epoch 00264: 0.586559(067) (in 00:00:04:46s)
Improvement at epoch 00265: 0.592349(037) (in 00:00:04:48s)
Improvement at epoch 00272: 0.592880(067) (in 00:00:04:58s)
Improvement at epoch 00276: 0.594431(074) (in 00:00:05:04s)
Improvement at epoch 00278: 0.596533(031) (in 00:00:05:07s)
Improvement at epoch 00284: 0.608877(095) (in 00:00:05:16s)
Improvement at epoch 00285: 0.610055(052) (in 00:00:05:18s)
Improvement at epoch 00291: 0.612440(052) (in 00:00:05:27s)
Improvement at epoch 00293: 0.616725(035) (in 00:00:05:30s)
Improvement at epoch 00299: 0.619434(096) (in 00:00:05:39s)
Improvement at epoch 00305: 0.621513(088) (in 00:00:05:48s)
Improvement at epoch 00310: 0.622133(033) (in 00:00:05:56s)
Improvement at epoch 00318: 0.622582(066) (in 00:00:06:08s)
Improvement at epoch 00321: 0.624207(071) (in 00:00:06:12s)
Improvement at epoch 00329: 0.625165(058) (in 00:00:06:25s)
Improvement at epoch 00330: 0.625651(044) (in 00:00:06:26s)
Improvement at epoch 00331: 0.627767(086) (in 00:00:06:28s)
Improvement at epoch 00334: 0.627773(041) (in 00:00:06:32s)
Improvement at epoch 00336: 0.627822(063) (in 00:00:06:35s)
Improvement at epoch 00338: 0.627834(026) (in 00:00:06:38s)
Improvement at epoch 00342: 0.628323(053) (in 00:00:06:44s)
Improvement at epoch 00343: 0.629859(031) (in 00:00:06:46s)
Improvement at epoch 00345: 0.630395(070) (in 00:00:06:49s)
Improvement at epoch 00352: 0.631992(064) (in 00:00:07:00s)
Improvement at epoch 00353: 0.633024(040) (in 00:00:07:01s)
Improvement at epoch 00354: 0.633051(092) (in 00:00:07:03s)
Improvement at epoch 00370: 0.633577(022) (in 00:00:07:28s)
Improvement at epoch 00371: 0.634067(042) (in 00:00:07:29s)
Improvement at epoch 00374: 0.634070(084) (in 00:00:07:34s)
```

```
Improvement at epoch 00383: 0.634553(096) (in 00:00:07:48s)
Improvement at epoch 00385: 0.634564(078) (in 00:00:07:52s)
Improvement at epoch 00386: 0.635083(058) (in 00:00:07:53s)
Improvement at epoch 00389: 0.635147(092) (in 00:00:07:58s)
Improvement at epoch 00398: 0.635612(067) (in 00:00:08:12s)
Improvement at epoch 00399: 0.635629(059) (in 00:00:08:14s)
Improvement at epoch 00408: 0.636120(036) (in 00:00:08:28s)
Improvement at epoch 00417: 0.636637(055) (in 00:00:08:43s)
Improvement at epoch 00421: 0.636653(092) (in 00:00:08:49s)
Improvement at epoch 00430: 0.636668(087) (in 00:00:09:04s)
Improvement at epoch 00446: 0.637170(046) (in 00:00:09:29s)
Improvement at epoch 00449: 0.637175(028) (in 00:00:09:34s)
Improvement at epoch 00457: 0.637179(020) (in 00:00:09:47s)
Improvement at epoch 00460: 0.637706(055) (in 00:00:09:52s)
Improvement at epoch 00468: 0.638773(072) (in 00:00:10:05s)
Improvement at epoch 00476: 0.640316(035) (in 00:00:10:18s)
Improvement at epoch 00477: 0.640833(078) (in 00:00:10:20s)
Improvement at epoch 00486: 0.641332(083) (in 00:00:10:34s)
Improvement at epoch 00496: 0.641347(068) (in 00:00:10:50s)
Improvement at epoch 00513: 0.641863(055) (in 00:00:11:17s)
Improvement at epoch 00532: 0.641872(039) (in 00:00:11:47s)
Improvement at epoch 00540: 0.658319(063) (in 00:00:12:00s)
Improvement at epoch 00541: 0.663452(094) (in 00:00:12:02s)
Improvement at epoch 00542: 0.676526(025) (in 00:00:12:03s)
Improvement at epoch 00543: 0.688216(047) (in 00:00:12:05s)
Improvement at epoch 00544: 0.691553(028) (in 00:00:12:07s)
Improvement at epoch 00545: 0.703180(094) (in 00:00:12:08s)
Improvement at epoch 00546: 0.713855(040) (in 00:00:12:10s)
Improvement at epoch 00551: 0.714702(064) (in 00:00:12:18s)
Improvement at epoch 00552: 0.718452(044) (in 00:00:12:19s)
Improvement at epoch 00555: 0.726324(086) (in 00:00:12:24s)
Improvement at epoch 00558: 0.727935(081) (in 00:00:12:29s)
Improvement at epoch 00562: 0.731176(040) (in 00:00:12:35s)
Improvement at epoch 00566: 0.731220(099) (in 00:00:12:41s)
Improvement at epoch 00567: 0.734822(060) (in 00:00:12:43s)
Improvement at epoch 00575: 0.734933(085) (in 00:00:12:55s)
Improvement at epoch 00577: 0.738571(090) (in 00:00:12:58s)
Improvement at epoch 00582: 0.739588(083) (in 00:00:13:06s)
Improvement at epoch 00583: 0.739992(059) (in 00:00:13:08s)
Improvement at epoch 00585: 0.742706(031) (in 00:00:13:11s)
Improvement at epoch 00587: 0.746875(088) (in 00:00:13:14s)
Improvement at epoch 00591: 0.749419(023) (in 00:00:13:20s)
Improvement at epoch 00593: 0.749425(055) (in 00:00:13:23s)
Improvement at epoch 00596: 0.751587(023) (in 00:00:13:28s)
Improvement at epoch 00597: 0.752071(081) (in 00:00:13:29s)
Improvement at epoch 00601: 0.752439(092) (in 00:00:13:36s)
Improvement at epoch 00607: 0.754105(062) (in 00:00:13:45s)
Improvement at epoch 00608: 0.754563(039) (in 00:00:13:47s)
Improvement at epoch 00610: 0.756684(092) (in 00:00:13:50s)
Improvement at epoch 00619: 0.758821(089) (in 00:00:14:04s)
Improvement at epoch 00630: 0.758829(028) (in 00:00:14:22s)
Improvement at epoch 00634: 0.758873(072) (in 00:00:14:28s)
Improvement at epoch 00642: 0.759969(078) (in 00:00:14:41s)
Improvement at epoch 00649: 0.760325(024) (in 00:00:14:52s)
Improvement at epoch 00650: 0.761455(058) (in 00:00:14:53s)
Improvement at epoch 00652: 0.761921(072) (in 00:00:14:56s)
Improvement at epoch 00654: 0.762011(093) (in 00:00:15:00s)
Improvement at epoch 00655: 0.763465(089) (in 00:00:15:01s)
Improvement at epoch 00663: 0.766169(023) (in 00:00:15:14s)
Improvement at epoch 00665: 0.767091(081) (in 00:00:15:17s)
Improvement at epoch 00674: 0.768275(066) (in 00:00:15:31s)
Improvement at epoch 00682: 0.769284(020) (in 00:00:15:44s)
```

```
Improvement at epoch 00685: 0.769339(031) (in 00:00:15:49s)
Improvement at epoch 00688: 0.769670(059) (in 00:00:15:53s)
Improvement at epoch 00689: 0.770800(060) (in 00:00:15:55s)
Improvement at epoch 00695: 0.772424(047) (in 00:00:16:04s)
Improvement at epoch 00696: 0.772469(045) (in 00:00:16:06s)
Improvement at epoch 00702: 0.775540(096) (in 00:00:16:15s)
Improvement at epoch 00705: 0.777608(043) (in 00:00:16:20s)
Improvement at epoch 00709: 0.780756(073) (in 00:00:16:26s)
Improvement at epoch 00712: 0.780811(023) (in 00:00:16:31s)
Improvement at epoch 00714: 0.781831(040) (in 00:00:16:34s)
Improvement at epoch 00717: 0.782909(063) (in 00:00:16:38s)
Improvement at epoch 00724: 0.785541(059) (in 00:00:16:49s)
Improvement at epoch 00725: 0.787057(086) (in 00:00:16:51s)
Improvement at epoch 00726: 0.787579(080) (in 00:00:16:52s)
Improvement at epoch 00734: 0.788097(067) (in 00:00:17:05s)
Improvement at epoch 00735: 0.788152(058) (in 00:00:17:06s)
Improvement at epoch 00737: 0.790714(073) (in 00:00:17:10s)
Improvement at epoch 00751: 0.792799(020) (in 00:00:17:31s)
Improvement at epoch 00790: 0.793874(073) (in 00:00:18:31s)
Improvement at epoch 00791: 0.796630(031) (in 00:00:18:32s)
Improvement at epoch 00802: 0.797198(040) (in 00:00:18:49s)
Improvement at epoch 00806: 0.799274(078) (in 00:00:18:56s)
Improvement at epoch 00807: 0.800357(087) (in 00:00:18:57s)
Improvement at epoch 00829: 0.800360(090) (in 00:00:19:31s)
Improvement at epoch 00845: 0.800892(077) (in 00:00:19:56s)
Improvement at epoch 00860: 0.802398(072) (in 00:00:20:19s)
Improvement at epoch 00877: 0.804012(062) (in 00:00:20:45s)
Improvement at epoch 00920: 0.804024(041) (in 00:00:21:53s)
Improvement at epoch 00921: 0.804528(094) (in 00:00:21:54s)
Improvement at epoch 00937: 0.805600(057) (in 00:00:22:19s)
Improvement at epoch 00972: 0.806652(096) (in 00:00:23:15s)
Improvement at epoch 01035: 0.807146(075) (in 00:00:25:09s)
Improvement at epoch 01046: 0.807678(082) (in 00:00:25:31s)
Improvement at epoch 01077: 0.807684(035) (in 00:00:26:32s)
Improvement at epoch 01082: 0.807732(045) (in 00:00:26:43s)
Improvement at epoch 01084: 0.808134(077) (in 00:00:26:47s)
Improvement at epoch 01090: 0.808232(037) (in 00:00:26:59s)
Improvement at epoch 01107: 0.808670(078) (in 00:00:27:33s)
Improvement at epoch 01109: 0.808697(022) (in 00:00:27:36s)
Improvement at epoch 01114: 0.809203(035) (in 00:00:27:47s)
Improvement at epoch 01116: 0.810275(030) (in 00:00:27:51s)
Improvement at epoch 01147: 0.811310(075) (in 00:00:28:54s)
Improvement at epoch 01164: 0.811827(049) (in 00:00:29:29s)
Improvement at epoch 01172: 0.812860(029) (in 00:00:29:44s)
Improvement at epoch 02869: 0.812917(020) (in 00:01:24:12s)
Improvement at epoch 02876: 0.813435(069) (in 00:01:24:24s)
Improvement at epoch 02881: 0.813450(020) (in 00:01:24:33s)
Improvement at epoch 02905: 0.813465(024) (in 00:01:25:18s)
Improvement at epoch 02907: 0.814002(047) (in 00:01:25:22s)
Improvement at epoch 02918: 0.814529(079) (in 00:01:25:43s)
Improvement at epoch 02983: 0.815055(068) (in 00:01:28:01s)
Improvement at epoch 03004: 0.815576(085) (in 00:01:28:46s)
Improvement at epoch 03134: 0.815579(085) (in 00:01:33:17s)
Improvement at epoch 03762: 0.816067(084) (in 00:01:53:41s)
Improvement at epoch 03785: 0.816597(039) (in 00:01:54:18s)
Improvement at epoch 03800: 0.816619(059) (in 00:01:54:43s)
Improvement at epoch 03804: 0.817128(026) (in 00:01:54:50s)
Improvement at epoch 03843: 0.817644(023) (in 00:01:55:53s)
Improvement at epoch 03877: 0.818168(068) (in 00:01:56:48s)
Improvement at epoch 03889: 0.818700(081) (in 00:01:57:08s)
Improvement at epoch 03910: 0.818886(079) (in 00:01:57:42s)
Improvement at epoch 03912: 0.819215(079) (in 00:01:57:45s)
```

```
Improvement at epoch 03918: 0.819230(087) (in 00:01:57:55s)
Improvement at epoch 03924: 0.819421(054) (in 00:01:58:05s)
Improvement at epoch 03927: 0.819747(094) (in 00:01:58:10s)
Improvement at epoch 03928: 0.819947(097) (in 00:01:58:11s)
Improvement at epoch 03940: 0.819950(063) (in 00:01:58:31s)
Improvement at epoch 03941: 0.819961(081) (in 00:01:58:33s)
Improvement at epoch 03953: 0.820461(075) (in 00:01:58:52s)
Improvement at epoch 03973: 0.820464(020) (in 00:01:59:25s)
Improvement at epoch 04003: 0.820469(072) (in 00:02:00:15s)
Improvement at epoch 04019: 0.820978(052) (in 00:02:00:41s)
Improvement at epoch 04047: 0.821495(093) (in 00:02:01:27s)
Improvement at epoch 04075: 0.821514(040) (in 00:02:02:23s)
Improvement at epoch 04086: 0.821990(075) (in 00:02:02:46s)
Improvement at epoch 04092: 0.822028(065) (in 00:02:02:57s)
Improvement at epoch 04141: 0.822506(078) (in 00:02:04:33s)
Improvement at epoch 04227: 0.822971(037) (in 00:02:06:58s)
Improvement at epoch 04230: 0.823523(065) (in 00:02:07:03s)
Improvement at epoch 04248: 0.824054(035) (in 00:02:07:34s)
Improvement at epoch 04259: 0.824055(044) (in 00:02:07:52s)
Improvement at epoch 04265: 0.824076(099) (in 00:02:08:02s)
Improvement at epoch 04321: 0.825703(025) (in 00:02:09:36s)
Improvement at epoch 04322: 0.826703(073) (in 00:02:09:38s)
Improvement at epoch 04338: 0.827200(098) (in 00:02:10:05s)
Improvement at epoch 04339: 0.828269(092) (in 00:02:10:07s)
Improvement at epoch 04344: 0.828783(023) (in 00:02:10:15s)
Improvement at epoch 04351: 0.828803(070) (in 00:02:10:26s)
Improvement at epoch 04352: 0.829819(062) (in 00:02:10:28s)
Improvement at epoch 04374: 0.830348(087) (in 00:02:11:05s)
Improvement at epoch 04434: 0.830869(090) (in 00:02:12:45s)
Improvement at epoch 04435: 0.831386(053) (in 00:02:12:46s)

Learning complete (in 0:2:27:58s)
```

checktest.txt:

```
        0 ( 178): 0.977528
        1 ( 182): 0.670330
        2 ( 177): 0.926554
        3 ( 183): 0.852459
        4 ( 181): 0.966851
        5 ( 182): 0.923077
        6 ( 181): 0.944751
        7 ( 179): 0.854749
        8 ( 174): 0.821839
        9 ( 180): 0.761111
Value: 0.869783
```

checkall.txt:

```
        0 ( 555): 0.983784
        1 ( 571): 0.716287
        2 ( 557): 0.940754
        3 ( 572): 0.921329
        4 ( 568): 0.929577
        5 ( 558): 0.930108
        6 ( 558): 0.969534
        7 ( 566): 0.927562
        8 ( 553): 0.880651
        9 ( 562): 0.809609
Value: 0.900534
```

8 nn2cloud

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "neuranet.h"

int main(int argc, char **argv) {
  char* NNUrl = NULL;
  // Decode arguments
  bool flagPrune = false;
  for (int iArg = 0; iArg < argc; ++iArg) {
    if (strcmp(argv[iArg] , "-nn") == 0 && iArg + 1 < argc) {
      NNUrl = argv[iArg + 1];
      ++iArg;
    } else if (strcmp(argv[iArg] , "-prune") == 0) {
      flagPrune = true;
    } else if (strcmp(argv[iArg] , "-help") == 0) {
      printf("arguments : -nn <NeuraNet file url>\n");
      // Stop here
      return 0;
    }
  }
  if (NNUrl == NULL)
    return 1;
  FILE* fd = fopen(NNUrl, "r");
  NeuraNet* nn = NULL;
  if (NNLoad(&nn, fd) == false) {
    fprintf(stderr, "Failed to open the NeuraNet %s\n", NNUrl);
  }
  fclose(fd);
  if (flagPrune)
    NNPrune(nn);
  char* cloudUrl = "./cloud.txt";
  if (NNSaveLinkAsCloudGraph(nn, cloudUrl) == false) {
    fprintf(stderr, "Failed to save the CloudGraph %s\n", cloudUrl);
  }
  NeuraNetFree(&nn);
  // Return success code
  return 0;
}
```