NeuraNet

P. Baillehache

August 13, 2018

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.

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 "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 NN_NBPARAMBASE float all in [-1,1]
// 'x' is the input value, in [-1,1]
// NNBaseFun(param,x)=
// {tan(param[0]*NN_THETA)*(x+param[1])+param[2]}[-1,1]
// The generic base function returns a value in [-1,1]
#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 int _nbMaxHidVal;
  // Nb max of base functions
  const int _nbMaxBases;
  // Nb max of links
  const int _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)
  VecShort* _links;
  // Hidden values
  VecFloat* _hidVal;
} 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 int nbMaxHidden, const int nbMaxBases, const int 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 VecShort* const hiddenLayers);

// 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
int NNGetNbMaxHidden(const NeuraNet* const that);

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

// Get the nb max of links of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
```

```
int 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 VecShort* 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 int 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 int 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, VecShort* 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);

// ================= 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 int NNGetGAAdnFloatLength(const NeuraNet* const that)
  __attribute__((unused));
static int 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 int NNGetGAAdnIntLength(const NeuraNet* const that)
  __attribute__((unused));
static int 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 (%d==%d)",
      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 (int 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 (%d==%d)",
      GAGetLengthAdnInt(ga), NNGetGAAdnIntLength(that));
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare a vector to memorize the bounds
  VecShort2D bounds = VecShortCreateStatic2D();
  // For each gene
  for (int 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 pbmath.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 int nbMaxHidden, const int nbMaxBases, const int 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<=%d)",
      nbMaxHidden);
```

```
    PBErrCatch(NeuraNetErr);
  }
  if (nbMaxBases <= 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbMaxBases' is invalid (0<%d)",
      nbMaxBases);
    PBErrCatch(NeuraNetErr);
  }
  if (nbMaxLinks <= 0) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg, "'nbMaxLinks' is invalid (0<%d)",
      nbMaxLinks);
    PBErrCatch(NeuraNetErr);
  }
#endif
  // Declare the new NeuraNet
  NeuraNet* that = PBErrMalloc(NeuraNetErr, sizeof(NeuraNet));
  // Set properties
  *(int*)&(that->_nbInputVal) = nbInput;
  *(int*)&(that->_nbOutputVal) = nbOutput;
  *(int*)&(that->_nbMaxHidVal) = nbMaxHidden;
  *(int*)&(that->_nbMaxBases) = nbMaxBases;
  *(int*)&(that->_nbMaxLinks) = nbMaxLinks;
  that->_bases = VecFloatCreate(nbMaxBases * NN_NBPARAMBASE);
  that->_links = VecShortCreate(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 VecShort* 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
  int nbHiddenVal = 0;
  int nbBases = 0;
  int nbLinks = 0;
  int nbHiddenLayer = 0;
  // If there are hidden layers
  if (hiddenLayers != NULL) {
    // Get the number of hidden layers
    nbHiddenLayer = VecGetDim(hiddenLayers);
    // Declare two variables for computation
    int nIn = nbIn;
    int nOut = 0;
    // Calculate the nb of links and hidden values
    for (int 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
  int iLink = 0;
  // Declare variables for computation
  int shiftIn = 0;
  int shiftOut = nbIn;
  int nIn = nbIn;
  int nOut = 0;
  // Loop on hidden layers
  for (int iLayer = 0; iLayer <= nbHiddenLayer; ++iLayer) {
    // Init the links
    if (iLayer < nbHiddenLayer)
      nOut = VecGet(hiddenLayers, iLayer);
    else
      nOut = nbOut;
    for (int iIn = 0; iIn < nIn; ++iIn) {
      for (int iOut = 0; iOut < nOut; ++iOut) {
        int 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;
}

// 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 (%d!=%d)",
      VecGetDim(input), that->_nbInputVal);
    PBErrCatch(NeuraNetErr);
  }
  if (VecGetDim(output) != that->_nbOutputVal) {
    NeuraNetErr->_type = PBErrTypeInvalidArg;
    sprintf(NeuraNetErr->_msg,
      "'output' 's dimension is invalid (%d!=%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
    int startHid = NNGetNbInput(that);
    int startOut = NNGetNbMaxHidden(that) + NNGetNbInput(that);
    // Declare a variable to memorize the previous link
    int prevLink[2] = {-1, -1};
```

```
    // Declare a variable to memorize the previous output value
    float prevOut = 1.0;
    // Loop on links
    int iLink = 0;
    while (iLink < NNGetNbMaxLinks(that) &&
      VecGet(that->_links, NN_NBPARAMLINK * iLink) != -1) {
      // Declare a variable for optimization
      int 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) {
          int iVal = prevLink[1] - startHid;
          float nVal = MIN(1.0, MAX(-1.0, VecGet(that->_hidVal, iVal) + prevOut));
          VecSet(that->_hidVal, iVal, nVal);
        } else {
          int 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) {
      int iVal = prevLink[1] - startHid;
      float nVal = MIN(1.0, MAX(-1.0, VecGet(that->_hidVal, iVal) + prevOut));
      VecSet(that->_hidVal, iVal, nVal);
    } else {
      int 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, "%d", that->_nbMaxHidVal);
  JSONAddProp(json, "_nbMaxHidVal", val);
  // Encode the nbMaxBases
  sprintf(val, "%d", that->_nbMaxBases);
  JSONAddProp(json, "_nbMaxBases", val);
  // Encode the nbMaxLinks
  sprintf(val, "%d", 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;
  }
  int nbMaxHidVal = atoi(JSONLabel(JSONValue(prop, 0)));
  // Decode the nbMaxBases
  prop = JSONProperty(json, "_nbMaxBases");
  if (prop == NULL) {
    return false;
  }
  int nbMaxBases = atoi(JSONLabel(JSONValue(prop, 0)));
  // Decode the nbMaxLinks
  prop = JSONProperty(json, "_nbMaxLinks");
  if (prop == NULL) {
    return false;
  }
  int nbMaxLinks = atoi(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: %d\n", that->_nbMaxHidVal);
  fprintf(stream, "nbMaxBases: %d\n", that->_nbMaxBases);
  fprintf(stream, "nbMaxLinks: %d\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, VecShort* 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 (%d!=%d)",
      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
  int maxId = NNGetNbInput(that) + NNGetNbMaxHidden(that) +
    NNGetNbOutput(that);
  // Loop on links
  for (int iLink = 0; iLink < NNGetNbMaxLinks(that) * NN_NBPARAMLINK;
    iLink += NN_NBPARAMLINK) {
    // If this link is active
    if (VecGet(links, iLink) != -1) {
      // Declare two variable to memorize the effective input and output
      int in = VecGet(links, iLink + 1);
      int out = VecGet(links, iLink + 2);
      // If the input is greater than the output
      if (in > out) {
        // Swap the input and output
        int tmp = in;
        in = out;
        out = tmp;
      }
      // 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
  int nbLink = GSetNbElem(&set);
```

```
  // If there are active links
  if (nbLink > 0) {
    // loop on active sorted links
    GSetIterForward iter = GSetIterForwardCreateStatic(&set);
    int iLink = 0;
    do {
      short *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 (int iLink = nbLink; iLink < NNGetNbMaxLinks(that); ++iLink)
    VecSet(that->_links, iLink * NN_NBPARAMLINK, -1);
  // Correct the links definition in the GenAlg to improve
  // diversity calculation
  VecCopy(links, that->_links);
  // Free the memory
  GSetFlush(&set);
}
```

3.2 pbmath-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]}[-1,1]
// The generic base function returns a value in [-1,1]
#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 MIN(1.0, MAX(-1.0,
  //  tan(param[0] * NN_THETA) * (x + param[1]) + param[2]));
  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
int 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
int 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 max of links of the NeuraNet 'that'
#if BUILDMODE != 0
inline
#endif
int 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 VecShort* 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 int 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<=%d<%d)",
      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 (%d!=%d)",
      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 int 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<=%d<%d)",
      iBase, that->_nbMaxBases * NN_NBPARAMBASE);
    PBErrCatch(NeuraNetErr);
  }
#endif
  VecSet(that->_bases, iBase, base);
}
```

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: main

# 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)/$
```

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->_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;
  VecShort* 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 = VecShortCreate(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 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 (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);
  VecShort* links = VecShortCreate(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);
    }
  VecFree(&links);
  NeuraNetFree(&nn);
  printf("UnitTestNeuraNetGetSet OK\n");
}

void UnitTestNeuraNetSaveLoad() {
  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);
  VecShort* links = VecShortCreate(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);
  NeuraNetFree(&nn);
  printf("UnitTestNeuraNetSaveLoad OK\n");
}

void UnitTestNeuraNetEvalPrint() {
  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};
  VecShort *links = VecShortCreate(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);
        }
      }
    }
  }
  NeuraNetFree(&nn);
  printf("UnitTestNeuraNetEvalPrint 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 = 3;
  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) < 100 && 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();
  UnitTestNeuraNetGetSet();
  UnitTestNeuraNetSaveLoad();
  UnitTestNeuraNetEvalPrint();
#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
UnitTestNeuraNetGetSet OK
UnitTestNeuraNetSaveLoad 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>
UnitTestNeuraNetEvalPrint OK
1 -0.589407
2 -0.549744
4 -0.358829
5 -0.310591
11 -0.310447
13 -0.294536
17 -0.289736
18 -0.272652
19 -0.267634
21 -0.258228
25 -0.257314
26 -0.254666
30 -0.254224
32 -0.253986
36 -0.231163
37 -0.194898
41 -0.188746
44 -0.187099
82 -0.186968
84 -0.186710
90 -0.176221
93 -0.170388
94 -0.170171
95 -0.168599
97 -0.168410
99 -0.168334
100 -0.165676
108 -0.164237
112 -0.163052
138 -0.162231
140 -0.162206
256 -0.156286
427 -0.155642
429 -0.154719
445 -0.153960
462 -0.153806
488 -0.152242
499 -0.152129
502 -0.152017
666 -0.151985
1063 -0.151534
1828 -0.151175
2982 -0.150823
2995 -0.150756
3157 -0.150019
```

```
3262 -0.149638
3279 -0.149605
3286 -0.149587
4378 -0.149536
4409 -0.149486
4544 -0.149472
5115 -0.149416
5225 -0.148970
5231 -0.148939
5240 -0.148746
5326 -0.148682
5448 -0.148668
7855 -0.148618
8307 -0.148275
8748 -0.135853
8755 -0.116550
8764 -0.078999
8770 -0.071358
9107 -0.060294
9140 -0.057081
9147 -0.039839
9151 -0.037915
9153 -0.029160
9220 -0.028635
9240 -0.028070
9256 -0.019758
9723 -0.017609
9813 -0.014870
10196 -0.014759
10384 -0.014370
best after 30000 epochs: -0.014370
nbInput: 3
nbOutput: 3
nbHidden: 3
nbMaxBases: 3
nbMaxLinks: 7
bases: <-0.490,0.939,0.919,0.496,0.915,-0.867,0.303,0.226,-0.135>
links: <1,0,5,2,0,5,2,1,6,1,1,7,1,3,3,0,3,7,0,5,6>
hidden values: <0.036,0.000,0.508>
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","0.09
  },
  "_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":"3",
  "_nbMaxLinks":"7",
  "_bases":{
    "_dim":"9",
    "_val":["-0.490231","0.939080","0.918935","0.495953","0.914827","-0.867214","0.302704","0.226264","-0.134999"]
  },
  "_links":{
    "_dim":"21",
    "_val":["1","0","5","2","0","5","2","1","6","1","1","7","1","3","3","0","3","7","0","5","6"]
  }
}
```


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 20
#define NB_MAXLINK 20
#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 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 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"
  };

// 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 {
  ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
    NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
  GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  GAInit(ga);

  // Init all the links except the first one to deactivated
  GSetIterForward iter = GSetIterForwardCreateStatic(GAAdns(ga));
  do {
    GenAlgAdn* adn = GSetIterGet(&iter);
    for (int iGene = 3; iGene < VecGetDim(GAAdnAdnI(adn)); iGene += 3)
      GAAdnSetGeneI(adn, iGene, -1);
  } while (GSetIterStep(&iter));

}
// 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");
if (SAVE_GA_EVERY > 0)
  printf("Will save GenAlg every %d epochs\n", SAVE_GA_EVERY);
else
  printf("Will not save GenAlg\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = bestVal;
// Declare a variable to manage the save of GenAlg
int delaySave = 0;
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  // For each adn in the GenAlg
  for (int 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;
  // Display infos about the current epoch
  //printf("ep%lu ent%3d(age%6lu) val%.4f bestEpo%.4f  bestAll%.4f         \r",
    //GAGetCurEpoch(ga), iEnt, GAAdnGetAge(adn), value, curBest,
    //bestVal);
  //fflush(stdout);
}
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);
// If there has been improvement during this epoch
if (curBest > bestVal) {
  bestVal = curBest;
  // Display info about the improvment
  printf("Improvement at epoch %lu: %f (in %d:%d:%d:%ds)  \n",
    GAGetCurEpoch(ga), bestVal, day, hour, min, sec);
  fflush(stdout);
  // Set the links and base functions of the NeuraNet according
  // to the best adn
  if (GABestAdnF(ga) != NULL)
    NNSetBases(nn, GABestAdnF(ga));
  if (GABestAdnI(ga) != NULL)
    NNSetLinks(nn, GABestAdnI(ga));
  // 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 {
  printf("epoch %lu (in %d:%d:%d:%ds)     \r",
    GAGetCurEpoch(ga), day, hour, min, sec);
  fflush(stdout);
}
++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;
    }
  }
  // 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);
  // 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;
}
```

learning:

```
Learning...
Will stop when curEpoch >= 2000 or bestVal >= 0.999000
Will save the best NeuraNet in ./bestnn.txt at each improvement
Will not save GenAlg
Improvement at epoch 1: 0.666667 (in 0:0:0:0s)
epoch 2 (in 0:0:0:0s)
epoch 3 (in 0:0:0:0s)
epoch 4 (in 0:0:0:0s)
epoch 5 (in 0:0:0:0s)
epoch 6 (in 0:0:0:0s)
epoch 7 (in 0:0:0:0s)
epoch 8 (in 0:0:0:0s)
epoch 9 (in 0:0:0:0s)
epoch 10 (in 0:0:0:0s)
epoch 11 (in 0:0:0:0s)
epoch 12 (in 0:0:0:0s)
epoch 13 (in 0:0:0:0s)
epoch 14 (in 0:0:0:0s)
epoch 15 (in 0:0:0:0s)
epoch 16 (in 0:0:0:0s)
epoch 17 (in 0:0:0:0s)
epoch 18 (in 0:0:0:0s)
epoch 19 (in 0:0:0:0s)
epoch 20 (in 0:0:0:0s)
epoch 21 (in 0:0:0:0s)
epoch 22 (in 0:0:0:0s)
epoch 23 (in 0:0:0:0s)
epoch 24 (in 0:0:0:0s)
epoch 25 (in 0:0:0:0s)
epoch 26 (in 0:0:0:0s)
epoch 27 (in 0:0:0:0s)
epoch 28 (in 0:0:0:0s)
epoch 29 (in 0:0:0:0s)
epoch 30 (in 0:0:0:0s)
epoch 31 (in 0:0:0:0s)
epoch 32 (in 0:0:0:0s)
epoch 33 (in 0:0:0:0s)
epoch 34 (in 0:0:0:0s)
epoch 35 (in 0:0:0:0s)
epoch 36 (in 0:0:0:0s)
epoch 37 (in 0:0:0:0s)
```

```
epoch 38 (in 0:0:0:0s)
epoch 39 (in 0:0:0:0s)
epoch 40 (in 0:0:0:0s)
epoch 41 (in 0:0:0:0s)
epoch 42 (in 0:0:0:0s)
epoch 43 (in 0:0:0:0s)
epoch 44 (in 0:0:0:0s)
epoch 45 (in 0:0:0:0s)
epoch 46 (in 0:0:0:0s)
epoch 47 (in 0:0:0:0s)
epoch 48 (in 0:0:0:0s)
epoch 49 (in 0:0:0:0s)
epoch 50 (in 0:0:0:0s)
epoch 51 (in 0:0:0:0s)
epoch 52 (in 0:0:0:0s)
epoch 53 (in 0:0:0:0s)
epoch 54 (in 0:0:0:0s)
epoch 55 (in 0:0:0:0s)
epoch 56 (in 0:0:0:0s)
epoch 57 (in 0:0:0:0s)
epoch 58 (in 0:0:0:0s)
epoch 59 (in 0:0:0:0s)
epoch 60 (in 0:0:0:0s)
epoch 61 (in 0:0:0:0s)
epoch 62 (in 0:0:0:0s)
epoch 63 (in 0:0:0:0s)
epoch 64 (in 0:0:0:0s)
epoch 65 (in 0:0:0:0s)
epoch 66 (in 0:0:0:0s)
epoch 67 (in 0:0:0:0s)
epoch 68 (in 0:0:0:0s)
epoch 69 (in 0:0:0:0s)
epoch 70 (in 0:0:0:0s)
epoch 71 (in 0:0:0:0s)
epoch 72 (in 0:0:0:0s)
epoch 73 (in 0:0:0:0s)
epoch 74 (in 0:0:0:0s)
epoch 75 (in 0:0:0:0s)
epoch 76 (in 0:0:0:0s)
epoch 77 (in 0:0:0:0s)
epoch 78 (in 0:0:0:0s)
epoch 79 (in 0:0:0:0s)
epoch 80 (in 0:0:0:0s)
epoch 81 (in 0:0:0:0s)
epoch 82 (in 0:0:0:0s)
epoch 83 (in 0:0:0:0s)
epoch 84 (in 0:0:0:0s)
epoch 85 (in 0:0:0:0s)
epoch 86 (in 0:0:0:0s)
epoch 87 (in 0:0:0:0s)
epoch 88 (in 0:0:0:0s)
epoch 89 (in 0:0:0:0s)
epoch 90 (in 0:0:0:0s)
epoch 91 (in 0:0:0:0s)
epoch 92 (in 0:0:0:0s)
epoch 93 (in 0:0:0:0s)
epoch 94 (in 0:0:0:0s)
epoch 95 (in 0:0:0:0s)
epoch 96 (in 0:0:0:0s)
epoch 97 (in 0:0:0:0s)
epoch 98 (in 0:0:0:0s)
epoch 99 (in 0:0:0:0s)
```

```
epoch 100 (in 0:0:0:0s)
epoch 101 (in 0:0:0:0s)
epoch 102 (in 0:0:0:0s)
epoch 103 (in 0:0:0:0s)
epoch 104 (in 0:0:0:0s)
epoch 105 (in 0:0:0:0s)
epoch 106 (in 0:0:0:0s)
epoch 107 (in 0:0:0:0s)
epoch 108 (in 0:0:0:0s)
epoch 109 (in 0:0:0:0s)
epoch 110 (in 0:0:0:0s)
epoch 111 (in 0:0:0:0s)
epoch 112 (in 0:0:0:0s)
epoch 113 (in 0:0:0:0s)
epoch 114 (in 0:0:0:0s)
epoch 115 (in 0:0:0:0s)
epoch 116 (in 0:0:0:0s)
epoch 117 (in 0:0:0:0s)
epoch 118 (in 0:0:0:0s)
epoch 119 (in 0:0:0:0s)
epoch 120 (in 0:0:0:0s)
epoch 121 (in 0:0:0:1s)
epoch 122 (in 0:0:0:1s)
epoch 123 (in 0:0:0:1s)
epoch 124 (in 0:0:0:1s)
epoch 125 (in 0:0:0:1s)
epoch 126 (in 0:0:0:1s)
epoch 127 (in 0:0:0:1s)
epoch 128 (in 0:0:0:1s)
epoch 129 (in 0:0:0:1s)
epoch 130 (in 0:0:0:1s)
epoch 131 (in 0:0:0:1s)
epoch 132 (in 0:0:0:1s)
epoch 133 (in 0:0:0:1s)
epoch 134 (in 0:0:0:1s)
epoch 135 (in 0:0:0:1s)
epoch 136 (in 0:0:0:1s)
epoch 137 (in 0:0:0:1s)
epoch 138 (in 0:0:0:1s)
epoch 139 (in 0:0:0:1s)
epoch 140 (in 0:0:0:1s)
epoch 141 (in 0:0:0:1s)
epoch 142 (in 0:0:0:1s)
epoch 143 (in 0:0:0:1s)
epoch 144 (in 0:0:0:1s)
epoch 145 (in 0:0:0:1s)
epoch 146 (in 0:0:0:1s)
epoch 147 (in 0:0:0:1s)
epoch 148 (in 0:0:0:1s)
epoch 149 (in 0:0:0:1s)
epoch 150 (in 0:0:0:1s)
epoch 151 (in 0:0:0:1s)
epoch 152 (in 0:0:0:1s)
epoch 153 (in 0:0:0:1s)
epoch 154 (in 0:0:0:1s)
epoch 155 (in 0:0:0:1s)
epoch 156 (in 0:0:0:1s)
epoch 157 (in 0:0:0:1s)
epoch 158 (in 0:0:0:1s)
epoch 159 (in 0:0:0:1s)
epoch 160 (in 0:0:0:1s)
epoch 161 (in 0:0:0:1s)
```

```
epoch 162 (in 0:0:0:1s)
epoch 163 (in 0:0:0:1s)
epoch 164 (in 0:0:0:1s)
epoch 165 (in 0:0:0:1s)
epoch 166 (in 0:0:0:1s)
epoch 167 (in 0:0:0:1s)
epoch 168 (in 0:0:0:1s)
epoch 169 (in 0:0:0:1s)
epoch 170 (in 0:0:0:1s)
epoch 171 (in 0:0:0:1s)
epoch 172 (in 0:0:0:1s)
epoch 173 (in 0:0:0:1s)
epoch 174 (in 0:0:0:1s)
epoch 175 (in 0:0:0:1s)
epoch 176 (in 0:0:0:1s)
epoch 177 (in 0:0:0:1s)
epoch 178 (in 0:0:0:1s)
epoch 179 (in 0:0:0:1s)
epoch 180 (in 0:0:0:1s)
epoch 181 (in 0:0:0:1s)
epoch 182 (in 0:0:0:1s)
epoch 183 (in 0:0:0:1s)
epoch 184 (in 0:0:0:1s)
epoch 185 (in 0:0:0:1s)
epoch 186 (in 0:0:0:1s)
epoch 187 (in 0:0:0:1s)
epoch 188 (in 0:0:0:1s)
epoch 189 (in 0:0:0:1s)
epoch 190 (in 0:0:0:1s)
epoch 191 (in 0:0:0:1s)
epoch 192 (in 0:0:0:1s)
epoch 193 (in 0:0:0:1s)
epoch 194 (in 0:0:0:1s)
epoch 195 (in 0:0:0:1s)
epoch 196 (in 0:0:0:1s)
epoch 197 (in 0:0:0:1s)
epoch 198 (in 0:0:0:1s)
epoch 199 (in 0:0:0:1s)
epoch 200 (in 0:0:0:1s)
epoch 201 (in 0:0:0:1s)
epoch 202 (in 0:0:0:1s)
epoch 203 (in 0:0:0:1s)
epoch 204 (in 0:0:0:1s)
epoch 205 (in 0:0:0:1s)
epoch 206 (in 0:0:0:1s)
epoch 207 (in 0:0:0:1s)
epoch 208 (in 0:0:0:1s)
epoch 209 (in 0:0:0:1s)
epoch 210 (in 0:0:0:1s)
epoch 211 (in 0:0:0:1s)
epoch 212 (in 0:0:0:1s)
epoch 213 (in 0:0:0:1s)
epoch 214 (in 0:0:0:1s)
epoch 215 (in 0:0:0:1s)
epoch 216 (in 0:0:0:1s)
epoch 217 (in 0:0:0:1s)
epoch 218 (in 0:0:0:1s)
epoch 219 (in 0:0:0:1s)
epoch 220 (in 0:0:0:1s)
epoch 221 (in 0:0:0:1s)
epoch 222 (in 0:0:0:1s)
epoch 223 (in 0:0:0:1s)
```

```
epoch 224 (in 0:0:0:1s)
epoch 225 (in 0:0:0:1s)
epoch 226 (in 0:0:0:1s)
epoch 227 (in 0:0:0:1s)
epoch 228 (in 0:0:0:1s)
epoch 229 (in 0:0:0:1s)
epoch 230 (in 0:0:0:1s)
epoch 231 (in 0:0:0:1s)
epoch 232 (in 0:0:0:1s)
epoch 233 (in 0:0:0:1s)
epoch 234 (in 0:0:0:1s)
epoch 235 (in 0:0:0:1s)
epoch 236 (in 0:0:0:1s)
epoch 237 (in 0:0:0:1s)
epoch 238 (in 0:0:0:1s)
epoch 239 (in 0:0:0:1s)
epoch 240 (in 0:0:0:1s)
epoch 241 (in 0:0:0:1s)
epoch 242 (in 0:0:0:1s)
epoch 243 (in 0:0:0:1s)
epoch 244 (in 0:0:0:1s)
epoch 245 (in 0:0:0:1s)
epoch 246 (in 0:0:0:1s)
epoch 247 (in 0:0:0:1s)
epoch 248 (in 0:0:0:1s)
epoch 249 (in 0:0:0:1s)
epoch 250 (in 0:0:0:1s)
epoch 251 (in 0:0:0:1s)
epoch 252 (in 0:0:0:1s)
epoch 253 (in 0:0:0:1s)
epoch 254 (in 0:0:0:1s)
epoch 255 (in 0:0:0:1s)
epoch 256 (in 0:0:0:1s)
epoch 257 (in 0:0:0:1s)
epoch 258 (in 0:0:0:1s)
epoch 259 (in 0:0:0:1s)
epoch 260 (in 0:0:0:1s)
epoch 261 (in 0:0:0:1s)
epoch 262 (in 0:0:0:1s)
epoch 263 (in 0:0:0:1s)
epoch 264 (in 0:0:0:1s)
epoch 265 (in 0:0:0:1s)
epoch 266 (in 0:0:0:1s)
epoch 267 (in 0:0:0:1s)
epoch 268 (in 0:0:0:1s)
epoch 269 (in 0:0:0:1s)
epoch 270 (in 0:0:0:1s)
epoch 271 (in 0:0:0:1s)
epoch 272 (in 0:0:0:1s)
epoch 273 (in 0:0:0:1s)
epoch 274 (in 0:0:0:1s)
epoch 275 (in 0:0:0:1s)
epoch 276 (in 0:0:0:1s)
epoch 277 (in 0:0:0:1s)
epoch 278 (in 0:0:0:1s)
epoch 279 (in 0:0:0:1s)
epoch 280 (in 0:0:0:1s)
epoch 281 (in 0:0:0:1s)
epoch 282 (in 0:0:0:1s)
epoch 283 (in 0:0:0:1s)
epoch 284 (in 0:0:0:1s)
epoch 285 (in 0:0:0:1s)
```

```
epoch 286 (in 0:0:0:1s)
epoch 287 (in 0:0:0:1s)
epoch 288 (in 0:0:0:1s)
epoch 289 (in 0:0:0:1s)
epoch 290 (in 0:0:0:1s)
epoch 291 (in 0:0:0:1s)
epoch 292 (in 0:0:0:1s)
epoch 293 (in 0:0:0:1s)
epoch 294 (in 0:0:0:1s)
epoch 295 (in 0:0:0:1s)
epoch 296 (in 0:0:0:1s)
epoch 297 (in 0:0:0:1s)
epoch 298 (in 0:0:0:1s)
epoch 299 (in 0:0:0:1s)
epoch 300 (in 0:0:0:1s)
epoch 301 (in 0:0:0:1s)
epoch 302 (in 0:0:0:1s)
epoch 303 (in 0:0:0:1s)
epoch 304 (in 0:0:0:1s)
epoch 305 (in 0:0:0:1s)
epoch 306 (in 0:0:0:1s)
epoch 307 (in 0:0:0:1s)
epoch 308 (in 0:0:0:1s)
epoch 309 (in 0:0:0:2s)
epoch 310 (in 0:0:0:2s)
epoch 311 (in 0:0:0:2s)
epoch 312 (in 0:0:0:2s)
epoch 313 (in 0:0:0:2s)
epoch 314 (in 0:0:0:2s)
epoch 315 (in 0:0:0:2s)
epoch 316 (in 0:0:0:2s)
epoch 317 (in 0:0:0:2s)
epoch 318 (in 0:0:0:2s)
epoch 319 (in 0:0:0:2s)
epoch 320 (in 0:0:0:2s)
epoch 321 (in 0:0:0:2s)
epoch 322 (in 0:0:0:2s)
epoch 323 (in 0:0:0:2s)
epoch 324 (in 0:0:0:2s)
epoch 325 (in 0:0:0:2s)
epoch 326 (in 0:0:0:2s)
epoch 327 (in 0:0:0:2s)
epoch 328 (in 0:0:0:2s)
epoch 329 (in 0:0:0:2s)
epoch 330 (in 0:0:0:2s)
epoch 331 (in 0:0:0:2s)
epoch 332 (in 0:0:0:2s)
epoch 333 (in 0:0:0:2s)
epoch 334 (in 0:0:0:2s)
epoch 335 (in 0:0:0:2s)
epoch 336 (in 0:0:0:2s)
epoch 337 (in 0:0:0:2s)
epoch 338 (in 0:0:0:2s)
epoch 339 (in 0:0:0:2s)
epoch 340 (in 0:0:0:2s)
epoch 341 (in 0:0:0:2s)
epoch 342 (in 0:0:0:2s)
epoch 343 (in 0:0:0:2s)
epoch 344 (in 0:0:0:2s)
epoch 345 (in 0:0:0:2s)
epoch 346 (in 0:0:0:2s)
epoch 347 (in 0:0:0:2s)
```

```
epoch 348 (in 0:0:0:2s)
epoch 349 (in 0:0:0:2s)
epoch 350 (in 0:0:0:2s)
epoch 351 (in 0:0:0:2s)
epoch 352 (in 0:0:0:2s)
epoch 353 (in 0:0:0:2s)
epoch 354 (in 0:0:0:2s)
epoch 355 (in 0:0:0:2s)
epoch 356 (in 0:0:0:2s)
epoch 357 (in 0:0:0:2s)
epoch 358 (in 0:0:0:2s)
epoch 359 (in 0:0:0:2s)
epoch 360 (in 0:0:0:2s)
epoch 361 (in 0:0:0:2s)
epoch 362 (in 0:0:0:2s)
epoch 363 (in 0:0:0:2s)
epoch 364 (in 0:0:0:2s)
epoch 365 (in 0:0:0:2s)
epoch 366 (in 0:0:0:2s)
epoch 367 (in 0:0:0:2s)
epoch 368 (in 0:0:0:2s)
epoch 369 (in 0:0:0:2s)
epoch 370 (in 0:0:0:2s)
epoch 371 (in 0:0:0:2s)
epoch 372 (in 0:0:0:2s)
epoch 373 (in 0:0:0:2s)
epoch 374 (in 0:0:0:2s)
epoch 375 (in 0:0:0:2s)
epoch 376 (in 0:0:0:2s)
epoch 377 (in 0:0:0:2s)
epoch 378 (in 0:0:0:2s)
epoch 379 (in 0:0:0:2s)
epoch 380 (in 0:0:0:2s)
epoch 381 (in 0:0:0:2s)
epoch 382 (in 0:0:0:2s)
epoch 383 (in 0:0:0:2s)
epoch 384 (in 0:0:0:2s)
epoch 385 (in 0:0:0:2s)
epoch 386 (in 0:0:0:2s)
epoch 387 (in 0:0:0:2s)
epoch 388 (in 0:0:0:2s)
epoch 389 (in 0:0:0:2s)
epoch 390 (in 0:0:0:2s)
epoch 391 (in 0:0:0:2s)
epoch 392 (in 0:0:0:2s)
epoch 393 (in 0:0:0:2s)
epoch 394 (in 0:0:0:2s)
epoch 395 (in 0:0:0:2s)
epoch 396 (in 0:0:0:2s)
epoch 397 (in 0:0:0:2s)
epoch 398 (in 0:0:0:2s)
epoch 399 (in 0:0:0:2s)
epoch 400 (in 0:0:0:2s)
epoch 401 (in 0:0:0:2s)
epoch 402 (in 0:0:0:2s)
epoch 403 (in 0:0:0:2s)
epoch 404 (in 0:0:0:2s)
epoch 405 (in 0:0:0:2s)
epoch 406 (in 0:0:0:2s)
epoch 407 (in 0:0:0:2s)
epoch 408 (in 0:0:0:2s)
epoch 409 (in 0:0:0:2s)
```

```
epoch 410 (in 0:0:0:2s)
epoch 411 (in 0:0:0:2s)
epoch 412 (in 0:0:0:2s)
epoch 413 (in 0:0:0:2s)
epoch 414 (in 0:0:0:2s)
epoch 415 (in 0:0:0:2s)
epoch 416 (in 0:0:0:2s)
epoch 417 (in 0:0:0:2s)
epoch 418 (in 0:0:0:2s)
epoch 419 (in 0:0:0:2s)
epoch 420 (in 0:0:0:2s)
epoch 421 (in 0:0:0:2s)
epoch 422 (in 0:0:0:2s)
epoch 423 (in 0:0:0:2s)
epoch 424 (in 0:0:0:2s)
epoch 425 (in 0:0:0:2s)
epoch 426 (in 0:0:0:2s)
epoch 427 (in 0:0:0:2s)
epoch 428 (in 0:0:0:2s)
epoch 429 (in 0:0:0:2s)
epoch 430 (in 0:0:0:2s)
epoch 431 (in 0:0:0:2s)
epoch 432 (in 0:0:0:2s)
epoch 433 (in 0:0:0:2s)
epoch 434 (in 0:0:0:2s)
epoch 435 (in 0:0:0:2s)
epoch 436 (in 0:0:0:2s)
epoch 437 (in 0:0:0:2s)
epoch 438 (in 0:0:0:2s)
epoch 439 (in 0:0:0:2s)
epoch 440 (in 0:0:0:2s)
epoch 441 (in 0:0:0:2s)
epoch 442 (in 0:0:0:2s)
epoch 443 (in 0:0:0:2s)
epoch 444 (in 0:0:0:2s)
epoch 445 (in 0:0:0:2s)
epoch 446 (in 0:0:0:2s)
epoch 447 (in 0:0:0:2s)
epoch 448 (in 0:0:0:2s)
epoch 449 (in 0:0:0:2s)
epoch 450 (in 0:0:0:2s)
epoch 451 (in 0:0:0:2s)
epoch 452 (in 0:0:0:2s)
epoch 453 (in 0:0:0:2s)
epoch 454 (in 0:0:0:2s)
epoch 455 (in 0:0:0:2s)
epoch 456 (in 0:0:0:2s)
epoch 457 (in 0:0:0:2s)
epoch 458 (in 0:0:0:2s)
epoch 459 (in 0:0:0:2s)
epoch 460 (in 0:0:0:2s)
epoch 461 (in 0:0:0:2s)
epoch 462 (in 0:0:0:2s)
epoch 463 (in 0:0:0:2s)
epoch 464 (in 0:0:0:2s)
epoch 465 (in 0:0:0:2s)
epoch 466 (in 0:0:0:2s)
epoch 467 (in 0:0:0:2s)
epoch 468 (in 0:0:0:2s)
epoch 469 (in 0:0:0:2s)
epoch 470 (in 0:0:0:2s)
epoch 471 (in 0:0:0:2s)
```

```
epoch 472 (in 0:0:0:2s)
epoch 473 (in 0:0:0:2s)
epoch 474 (in 0:0:0:2s)
epoch 475 (in 0:0:0:2s)
epoch 476 (in 0:0:0:2s)
epoch 477 (in 0:0:0:2s)
epoch 478 (in 0:0:0:2s)
epoch 479 (in 0:0:0:2s)
epoch 480 (in 0:0:0:2s)
epoch 481 (in 0:0:0:2s)
epoch 482 (in 0:0:0:2s)
epoch 483 (in 0:0:0:2s)
epoch 484 (in 0:0:0:2s)
epoch 485 (in 0:0:0:3s)
epoch 486 (in 0:0:0:3s)
epoch 487 (in 0:0:0:3s)
epoch 488 (in 0:0:0:3s)
epoch 489 (in 0:0:0:3s)
epoch 490 (in 0:0:0:3s)
epoch 491 (in 0:0:0:3s)
epoch 492 (in 0:0:0:3s)
epoch 493 (in 0:0:0:3s)
epoch 494 (in 0:0:0:3s)
epoch 495 (in 0:0:0:3s)
epoch 496 (in 0:0:0:3s)
epoch 497 (in 0:0:0:3s)
epoch 498 (in 0:0:0:3s)
epoch 499 (in 0:0:0:3s)
epoch 500 (in 0:0:0:3s)
epoch 501 (in 0:0:0:3s)
epoch 502 (in 0:0:0:3s)
epoch 503 (in 0:0:0:3s)
epoch 504 (in 0:0:0:3s)
epoch 505 (in 0:0:0:3s)
epoch 506 (in 0:0:0:3s)
epoch 507 (in 0:0:0:3s)
epoch 508 (in 0:0:0:3s)
epoch 509 (in 0:0:0:3s)
epoch 510 (in 0:0:0:3s)
epoch 511 (in 0:0:0:3s)
epoch 512 (in 0:0:0:3s)
epoch 513 (in 0:0:0:3s)
epoch 514 (in 0:0:0:3s)
epoch 515 (in 0:0:0:3s)
epoch 516 (in 0:0:0:3s)
Improvement at epoch 517: 0.693333 (in 0:0:0:3s)
Improvement at epoch 518: 0.706667 (in 0:0:0:3s)
epoch 519 (in 0:0:0:3s)
epoch 520 (in 0:0:0:3s)
Improvement at epoch 521: 0.960000 (in 0:0:0:3s)
Improvement at epoch 522: 0.973333 (in 0:0:0:3s)
epoch 523 (in 0:0:0:3s)
epoch 524 (in 0:0:0:3s)
epoch 525 (in 0:0:0:3s)
epoch 526 (in 0:0:0:3s)
epoch 527 (in 0:0:0:3s)
epoch 528 (in 0:0:0:3s)
epoch 529 (in 0:0:0:3s)
epoch 530 (in 0:0:0:3s)
epoch 531 (in 0:0:0:3s)
epoch 532 (in 0:0:0:3s)
epoch 533 (in 0:0:0:3s)
```

```
epoch 534 (in 0:0:0:3s)
epoch 535 (in 0:0:0:3s)
epoch 536 (in 0:0:0:3s)
epoch 537 (in 0:0:0:3s)
epoch 538 (in 0:0:0:3s)
epoch 539 (in 0:0:0:3s)
epoch 540 (in 0:0:0:3s)
epoch 541 (in 0:0:0:3s)
epoch 542 (in 0:0:0:3s)
epoch 543 (in 0:0:0:3s)
epoch 544 (in 0:0:0:3s)
epoch 545 (in 0:0:0:3s)
epoch 546 (in 0:0:0:3s)
epoch 547 (in 0:0:0:3s)
epoch 548 (in 0:0:0:3s)
epoch 549 (in 0:0:0:3s)
epoch 550 (in 0:0:0:3s)
epoch 551 (in 0:0:0:3s)
epoch 552 (in 0:0:0:3s)
epoch 553 (in 0:0:0:3s)
epoch 554 (in 0:0:0:3s)
epoch 555 (in 0:0:0:3s)
epoch 556 (in 0:0:0:3s)
epoch 557 (in 0:0:0:3s)
epoch 558 (in 0:0:0:3s)
epoch 559 (in 0:0:0:3s)
epoch 560 (in 0:0:0:3s)
epoch 561 (in 0:0:0:3s)
epoch 562 (in 0:0:0:3s)
epoch 563 (in 0:0:0:3s)
epoch 564 (in 0:0:0:3s)
epoch 565 (in 0:0:0:3s)
epoch 566 (in 0:0:0:3s)
epoch 567 (in 0:0:0:3s)
epoch 568 (in 0:0:0:3s)
epoch 569 (in 0:0:0:3s)
epoch 570 (in 0:0:0:3s)
epoch 571 (in 0:0:0:3s)
epoch 572 (in 0:0:0:3s)
epoch 573 (in 0:0:0:3s)
epoch 574 (in 0:0:0:3s)
epoch 575 (in 0:0:0:3s)
epoch 576 (in 0:0:0:3s)
epoch 577 (in 0:0:0:3s)
epoch 578 (in 0:0:0:3s)
epoch 579 (in 0:0:0:3s)
epoch 580 (in 0:0:0:3s)
epoch 581 (in 0:0:0:3s)
epoch 582 (in 0:0:0:3s)
epoch 583 (in 0:0:0:3s)
epoch 584 (in 0:0:0:3s)
epoch 585 (in 0:0:0:3s)
epoch 586 (in 0:0:0:3s)
epoch 587 (in 0:0:0:3s)
epoch 588 (in 0:0:0:3s)
epoch 589 (in 0:0:0:3s)
epoch 590 (in 0:0:0:3s)
epoch 591 (in 0:0:0:3s)
epoch 592 (in 0:0:0:3s)
epoch 593 (in 0:0:0:3s)
epoch 594 (in 0:0:0:3s)
epoch 595 (in 0:0:0:3s)
```

```
epoch 596 (in 0:0:0:3s)
epoch 597 (in 0:0:0:3s)
epoch 598 (in 0:0:0:3s)
epoch 599 (in 0:0:0:3s)
epoch 600 (in 0:0:0:3s)
epoch 601 (in 0:0:0:3s)
epoch 602 (in 0:0:0:3s)
epoch 603 (in 0:0:0:3s)
epoch 604 (in 0:0:0:3s)
epoch 605 (in 0:0:0:3s)
epoch 606 (in 0:0:0:3s)
epoch 607 (in 0:0:0:3s)
epoch 608 (in 0:0:0:3s)
epoch 609 (in 0:0:0:3s)
epoch 610 (in 0:0:0:3s)
epoch 611 (in 0:0:0:3s)
epoch 612 (in 0:0:0:3s)
epoch 613 (in 0:0:0:3s)
epoch 614 (in 0:0:0:3s)
epoch 615 (in 0:0:0:3s)
epoch 616 (in 0:0:0:3s)
epoch 617 (in 0:0:0:3s)
epoch 618 (in 0:0:0:3s)
epoch 619 (in 0:0:0:3s)
epoch 620 (in 0:0:0:3s)
epoch 621 (in 0:0:0:3s)
epoch 622 (in 0:0:0:3s)
epoch 623 (in 0:0:0:3s)
epoch 624 (in 0:0:0:3s)
epoch 625 (in 0:0:0:3s)
epoch 626 (in 0:0:0:3s)
epoch 627 (in 0:0:0:3s)
epoch 628 (in 0:0:0:3s)
epoch 629 (in 0:0:0:3s)
epoch 630 (in 0:0:0:3s)
epoch 631 (in 0:0:0:3s)
epoch 632 (in 0:0:0:3s)
epoch 633 (in 0:0:0:3s)
epoch 634 (in 0:0:0:3s)
epoch 635 (in 0:0:0:3s)
epoch 636 (in 0:0:0:3s)
epoch 637 (in 0:0:0:3s)
epoch 638 (in 0:0:0:3s)
epoch 639 (in 0:0:0:3s)
epoch 640 (in 0:0:0:3s)
epoch 641 (in 0:0:0:3s)
epoch 642 (in 0:0:0:3s)
epoch 643 (in 0:0:0:3s)
epoch 644 (in 0:0:0:3s)
epoch 645 (in 0:0:0:3s)
epoch 646 (in 0:0:0:3s)
epoch 647 (in 0:0:0:3s)
epoch 648 (in 0:0:0:3s)
epoch 649 (in 0:0:0:3s)
epoch 650 (in 0:0:0:3s)
epoch 651 (in 0:0:0:3s)
epoch 652 (in 0:0:0:3s)
epoch 653 (in 0:0:0:3s)
epoch 654 (in 0:0:0:3s)
epoch 655 (in 0:0:0:3s)
epoch 656 (in 0:0:0:3s)
epoch 657 (in 0:0:0:3s)
```

```
epoch 658 (in 0:0:0:3s)
epoch 659 (in 0:0:0:3s)
epoch 660 (in 0:0:0:3s)
epoch 661 (in 0:0:0:3s)
epoch 662 (in 0:0:0:3s)
epoch 663 (in 0:0:0:3s)
epoch 664 (in 0:0:0:3s)
epoch 665 (in 0:0:0:3s)
epoch 666 (in 0:0:0:3s)
epoch 667 (in 0:0:0:3s)
epoch 668 (in 0:0:0:3s)
epoch 669 (in 0:0:0:3s)
epoch 670 (in 0:0:0:3s)
epoch 671 (in 0:0:0:3s)
epoch 672 (in 0:0:0:3s)
epoch 673 (in 0:0:0:3s)
epoch 674 (in 0:0:0:3s)
epoch 675 (in 0:0:0:3s)
epoch 676 (in 0:0:0:3s)
epoch 677 (in 0:0:0:3s)
epoch 678 (in 0:0:0:3s)
epoch 679 (in 0:0:0:3s)
epoch 680 (in 0:0:0:3s)
epoch 681 (in 0:0:0:3s)
epoch 682 (in 0:0:0:3s)
epoch 683 (in 0:0:0:3s)
epoch 684 (in 0:0:0:3s)
epoch 685 (in 0:0:0:3s)
epoch 686 (in 0:0:0:3s)
epoch 687 (in 0:0:0:3s)
epoch 688 (in 0:0:0:3s)
epoch 689 (in 0:0:0:3s)
epoch 690 (in 0:0:0:3s)
epoch 691 (in 0:0:0:3s)
epoch 692 (in 0:0:0:3s)
epoch 693 (in 0:0:0:3s)
epoch 694 (in 0:0:0:3s)
epoch 695 (in 0:0:0:3s)
epoch 696 (in 0:0:0:3s)
epoch 697 (in 0:0:0:3s)
epoch 698 (in 0:0:0:3s)
epoch 699 (in 0:0:0:3s)
epoch 700 (in 0:0:0:3s)
epoch 701 (in 0:0:0:4s)
epoch 702 (in 0:0:0:4s)
epoch 703 (in 0:0:0:4s)
epoch 704 (in 0:0:0:4s)
epoch 705 (in 0:0:0:4s)
epoch 706 (in 0:0:0:4s)
epoch 707 (in 0:0:0:4s)
epoch 708 (in 0:0:0:4s)
epoch 709 (in 0:0:0:4s)
epoch 710 (in 0:0:0:4s)
epoch 711 (in 0:0:0:4s)
epoch 712 (in 0:0:0:4s)
epoch 713 (in 0:0:0:4s)
epoch 714 (in 0:0:0:4s)
epoch 715 (in 0:0:0:4s)
epoch 716 (in 0:0:0:4s)
epoch 717 (in 0:0:0:4s)
epoch 718 (in 0:0:0:4s)
epoch 719 (in 0:0:0:4s)
```

```
epoch 720 (in 0:0:0:4s)
epoch 721 (in 0:0:0:4s)
epoch 722 (in 0:0:0:4s)
epoch 723 (in 0:0:0:4s)
epoch 724 (in 0:0:0:4s)
epoch 725 (in 0:0:0:4s)
epoch 726 (in 0:0:0:4s)
epoch 727 (in 0:0:0:4s)
epoch 728 (in 0:0:0:4s)
epoch 729 (in 0:0:0:4s)
epoch 730 (in 0:0:0:4s)
epoch 731 (in 0:0:0:4s)
epoch 732 (in 0:0:0:4s)
epoch 733 (in 0:0:0:4s)
epoch 734 (in 0:0:0:4s)
epoch 735 (in 0:0:0:4s)
epoch 736 (in 0:0:0:4s)
epoch 737 (in 0:0:0:4s)
epoch 738 (in 0:0:0:4s)
epoch 739 (in 0:0:0:4s)
epoch 740 (in 0:0:0:4s)
epoch 741 (in 0:0:0:4s)
epoch 742 (in 0:0:0:4s)
epoch 743 (in 0:0:0:4s)
epoch 744 (in 0:0:0:4s)
epoch 745 (in 0:0:0:4s)
epoch 746 (in 0:0:0:4s)
epoch 747 (in 0:0:0:4s)
epoch 748 (in 0:0:0:4s)
epoch 749 (in 0:0:0:4s)
epoch 750 (in 0:0:0:4s)
epoch 751 (in 0:0:0:4s)
epoch 752 (in 0:0:0:4s)
epoch 753 (in 0:0:0:4s)
epoch 754 (in 0:0:0:4s)
epoch 755 (in 0:0:0:4s)
epoch 756 (in 0:0:0:4s)
epoch 757 (in 0:0:0:4s)
epoch 758 (in 0:0:0:4s)
epoch 759 (in 0:0:0:4s)
epoch 760 (in 0:0:0:4s)
epoch 761 (in 0:0:0:4s)
epoch 762 (in 0:0:0:4s)
epoch 763 (in 0:0:0:4s)
epoch 764 (in 0:0:0:4s)
epoch 765 (in 0:0:0:4s)
epoch 766 (in 0:0:0:4s)
epoch 767 (in 0:0:0:4s)
epoch 768 (in 0:0:0:4s)
epoch 769 (in 0:0:0:4s)
epoch 770 (in 0:0:0:4s)
epoch 771 (in 0:0:0:4s)
epoch 772 (in 0:0:0:4s)
epoch 773 (in 0:0:0:4s)
epoch 774 (in 0:0:0:4s)
epoch 775 (in 0:0:0:4s)
epoch 776 (in 0:0:0:4s)
epoch 777 (in 0:0:0:4s)
epoch 778 (in 0:0:0:4s)
epoch 779 (in 0:0:0:4s)
epoch 780 (in 0:0:0:4s)
epoch 781 (in 0:0:0:4s)
```

```
epoch 782 (in 0:0:0:4s)
epoch 783 (in 0:0:0:4s)
epoch 784 (in 0:0:0:4s)
epoch 785 (in 0:0:0:4s)
epoch 786 (in 0:0:0:4s)
epoch 787 (in 0:0:0:4s)
epoch 788 (in 0:0:0:4s)
epoch 789 (in 0:0:0:4s)
epoch 790 (in 0:0:0:4s)
epoch 791 (in 0:0:0:4s)
epoch 792 (in 0:0:0:4s)
epoch 793 (in 0:0:0:4s)
epoch 794 (in 0:0:0:4s)
epoch 795 (in 0:0:0:4s)
epoch 796 (in 0:0:0:4s)
epoch 797 (in 0:0:0:4s)
epoch 798 (in 0:0:0:4s)
epoch 799 (in 0:0:0:4s)
epoch 800 (in 0:0:0:4s)
epoch 801 (in 0:0:0:4s)
epoch 802 (in 0:0:0:4s)
epoch 803 (in 0:0:0:4s)
epoch 804 (in 0:0:0:4s)
epoch 805 (in 0:0:0:4s)
epoch 806 (in 0:0:0:4s)
epoch 807 (in 0:0:0:4s)
epoch 808 (in 0:0:0:4s)
epoch 809 (in 0:0:0:4s)
epoch 810 (in 0:0:0:4s)
epoch 811 (in 0:0:0:4s)
epoch 812 (in 0:0:0:4s)
epoch 813 (in 0:0:0:4s)
epoch 814 (in 0:0:0:4s)
epoch 815 (in 0:0:0:4s)
epoch 816 (in 0:0:0:4s)
epoch 817 (in 0:0:0:4s)
epoch 818 (in 0:0:0:4s)
epoch 819 (in 0:0:0:4s)
epoch 820 (in 0:0:0:4s)
epoch 821 (in 0:0:0:4s)
epoch 822 (in 0:0:0:4s)
epoch 823 (in 0:0:0:4s)
epoch 824 (in 0:0:0:4s)
epoch 825 (in 0:0:0:4s)
epoch 826 (in 0:0:0:4s)
epoch 827 (in 0:0:0:4s)
epoch 828 (in 0:0:0:4s)
epoch 829 (in 0:0:0:4s)
epoch 830 (in 0:0:0:4s)
epoch 831 (in 0:0:0:4s)
epoch 832 (in 0:0:0:4s)
epoch 833 (in 0:0:0:4s)
epoch 834 (in 0:0:0:4s)
epoch 835 (in 0:0:0:4s)
epoch 836 (in 0:0:0:4s)
epoch 837 (in 0:0:0:4s)
epoch 838 (in 0:0:0:4s)
epoch 839 (in 0:0:0:4s)
epoch 840 (in 0:0:0:4s)
epoch 841 (in 0:0:0:4s)
epoch 842 (in 0:0:0:4s)
epoch 843 (in 0:0:0:4s)
```

```
epoch 844 (in 0:0:0:4s)
epoch 845 (in 0:0:0:4s)
epoch 846 (in 0:0:0:4s)
epoch 847 (in 0:0:0:4s)
epoch 848 (in 0:0:0:4s)
epoch 849 (in 0:0:0:4s)
epoch 850 (in 0:0:0:4s)
epoch 851 (in 0:0:0:4s)
epoch 852 (in 0:0:0:4s)
epoch 853 (in 0:0:0:4s)
epoch 854 (in 0:0:0:4s)
epoch 855 (in 0:0:0:4s)
epoch 856 (in 0:0:0:4s)
epoch 857 (in 0:0:0:4s)
epoch 858 (in 0:0:0:4s)
epoch 859 (in 0:0:0:4s)
epoch 860 (in 0:0:0:4s)
epoch 861 (in 0:0:0:4s)
epoch 862 (in 0:0:0:4s)
epoch 863 (in 0:0:0:4s)
epoch 864 (in 0:0:0:4s)
epoch 865 (in 0:0:0:4s)
epoch 866 (in 0:0:0:4s)
epoch 867 (in 0:0:0:4s)
epoch 868 (in 0:0:0:4s)
epoch 869 (in 0:0:0:4s)
epoch 870 (in 0:0:0:4s)
epoch 871 (in 0:0:0:4s)
epoch 872 (in 0:0:0:4s)
epoch 873 (in 0:0:0:4s)
epoch 874 (in 0:0:0:4s)
epoch 875 (in 0:0:0:4s)
epoch 876 (in 0:0:0:4s)
epoch 877 (in 0:0:0:4s)
epoch 878 (in 0:0:0:4s)
epoch 879 (in 0:0:0:4s)
epoch 880 (in 0:0:0:4s)
epoch 881 (in 0:0:0:4s)
epoch 882 (in 0:0:0:4s)
epoch 883 (in 0:0:0:4s)
epoch 884 (in 0:0:0:4s)
epoch 885 (in 0:0:0:4s)
epoch 886 (in 0:0:0:4s)
epoch 887 (in 0:0:0:4s)
epoch 888 (in 0:0:0:4s)
epoch 889 (in 0:0:0:4s)
epoch 890 (in 0:0:0:4s)
epoch 891 (in 0:0:0:4s)
epoch 892 (in 0:0:0:4s)
epoch 893 (in 0:0:0:4s)
epoch 894 (in 0:0:0:5s)
epoch 895 (in 0:0:0:5s)
epoch 896 (in 0:0:0:5s)
epoch 897 (in 0:0:0:5s)
epoch 898 (in 0:0:0:5s)
epoch 899 (in 0:0:0:5s)
epoch 900 (in 0:0:0:5s)
epoch 901 (in 0:0:0:5s)
epoch 902 (in 0:0:0:5s)
epoch 903 (in 0:0:0:5s)
epoch 904 (in 0:0:0:5s)
epoch 905 (in 0:0:0:5s)
```

```
epoch 906 (in 0:0:0:5s)
epoch 907 (in 0:0:0:5s)
epoch 908 (in 0:0:0:5s)
epoch 909 (in 0:0:0:5s)
epoch 910 (in 0:0:0:5s)
epoch 911 (in 0:0:0:5s)
epoch 912 (in 0:0:0:5s)
epoch 913 (in 0:0:0:5s)
epoch 914 (in 0:0:0:5s)
epoch 915 (in 0:0:0:5s)
epoch 916 (in 0:0:0:5s)
epoch 917 (in 0:0:0:5s)
epoch 918 (in 0:0:0:5s)
epoch 919 (in 0:0:0:5s)
epoch 920 (in 0:0:0:5s)
epoch 921 (in 0:0:0:5s)
epoch 922 (in 0:0:0:5s)
epoch 923 (in 0:0:0:5s)
epoch 924 (in 0:0:0:5s)
epoch 925 (in 0:0:0:5s)
epoch 926 (in 0:0:0:5s)
epoch 927 (in 0:0:0:5s)
epoch 928 (in 0:0:0:5s)
epoch 929 (in 0:0:0:5s)
epoch 930 (in 0:0:0:5s)
epoch 931 (in 0:0:0:5s)
epoch 932 (in 0:0:0:5s)
epoch 933 (in 0:0:0:5s)
epoch 934 (in 0:0:0:5s)
epoch 935 (in 0:0:0:5s)
epoch 936 (in 0:0:0:5s)
epoch 937 (in 0:0:0:5s)
epoch 938 (in 0:0:0:5s)
epoch 939 (in 0:0:0:5s)
epoch 940 (in 0:0:0:5s)
epoch 941 (in 0:0:0:5s)
epoch 942 (in 0:0:0:5s)
epoch 943 (in 0:0:0:5s)
epoch 944 (in 0:0:0:5s)
epoch 945 (in 0:0:0:5s)
epoch 946 (in 0:0:0:5s)
epoch 947 (in 0:0:0:5s)
epoch 948 (in 0:0:0:5s)
epoch 949 (in 0:0:0:5s)
epoch 950 (in 0:0:0:5s)
epoch 951 (in 0:0:0:5s)
epoch 952 (in 0:0:0:5s)
epoch 953 (in 0:0:0:5s)
epoch 954 (in 0:0:0:5s)
epoch 955 (in 0:0:0:5s)
epoch 956 (in 0:0:0:5s)
epoch 957 (in 0:0:0:5s)
epoch 958 (in 0:0:0:5s)
epoch 959 (in 0:0:0:5s)
epoch 960 (in 0:0:0:5s)
epoch 961 (in 0:0:0:5s)
epoch 962 (in 0:0:0:5s)
epoch 963 (in 0:0:0:5s)
epoch 964 (in 0:0:0:5s)
epoch 965 (in 0:0:0:5s)
epoch 966 (in 0:0:0:5s)
epoch 967 (in 0:0:0:5s)
```

```
epoch 968 (in 0:0:0:5s)
epoch 969 (in 0:0:0:5s)
epoch 970 (in 0:0:0:5s)
epoch 971 (in 0:0:0:5s)
epoch 972 (in 0:0:0:5s)
epoch 973 (in 0:0:0:5s)
epoch 974 (in 0:0:0:5s)
epoch 975 (in 0:0:0:5s)
epoch 976 (in 0:0:0:5s)
epoch 977 (in 0:0:0:5s)
epoch 978 (in 0:0:0:5s)
epoch 979 (in 0:0:0:5s)
epoch 980 (in 0:0:0:5s)
epoch 981 (in 0:0:0:5s)
epoch 982 (in 0:0:0:5s)
epoch 983 (in 0:0:0:5s)
epoch 984 (in 0:0:0:5s)
epoch 985 (in 0:0:0:5s)
epoch 986 (in 0:0:0:5s)
epoch 987 (in 0:0:0:5s)
epoch 988 (in 0:0:0:5s)
epoch 989 (in 0:0:0:5s)
epoch 990 (in 0:0:0:5s)
epoch 991 (in 0:0:0:5s)
epoch 992 (in 0:0:0:5s)
epoch 993 (in 0:0:0:5s)
epoch 994 (in 0:0:0:5s)
epoch 995 (in 0:0:0:5s)
epoch 996 (in 0:0:0:5s)
epoch 997 (in 0:0:0:5s)
epoch 998 (in 0:0:0:5s)
epoch 999 (in 0:0:0:5s)
epoch 1000 (in 0:0:0:5s)
epoch 1001 (in 0:0:0:5s)
epoch 1002 (in 0:0:0:5s)
epoch 1003 (in 0:0:0:5s)
epoch 1004 (in 0:0:0:5s)
epoch 1005 (in 0:0:0:5s)
epoch 1006 (in 0:0:0:5s)
epoch 1007 (in 0:0:0:5s)
epoch 1008 (in 0:0:0:5s)
epoch 1009 (in 0:0:0:5s)
epoch 1010 (in 0:0:0:5s)
epoch 1011 (in 0:0:0:5s)
epoch 1012 (in 0:0:0:5s)
epoch 1013 (in 0:0:0:5s)
epoch 1014 (in 0:0:0:5s)
epoch 1015 (in 0:0:0:5s)
epoch 1016 (in 0:0:0:5s)
epoch 1017 (in 0:0:0:5s)
epoch 1018 (in 0:0:0:5s)
epoch 1019 (in 0:0:0:5s)
epoch 1020 (in 0:0:0:5s)
epoch 1021 (in 0:0:0:5s)
epoch 1022 (in 0:0:0:5s)
epoch 1023 (in 0:0:0:5s)
epoch 1024 (in 0:0:0:5s)
epoch 1025 (in 0:0:0:5s)
epoch 1026 (in 0:0:0:5s)
epoch 1027 (in 0:0:0:5s)
epoch 1028 (in 0:0:0:5s)
epoch 1029 (in 0:0:0:5s)
```

```
epoch 1030 (in 0:0:0:5s)
epoch 1031 (in 0:0:0:5s)
epoch 1032 (in 0:0:0:5s)
epoch 1033 (in 0:0:0:5s)
epoch 1034 (in 0:0:0:5s)
epoch 1035 (in 0:0:0:5s)
epoch 1036 (in 0:0:0:5s)
epoch 1037 (in 0:0:0:5s)
epoch 1038 (in 0:0:0:5s)
epoch 1039 (in 0:0:0:5s)
epoch 1040 (in 0:0:0:5s)
epoch 1041 (in 0:0:0:5s)
epoch 1042 (in 0:0:0:5s)
epoch 1043 (in 0:0:0:5s)
epoch 1044 (in 0:0:0:5s)
epoch 1045 (in 0:0:0:5s)
epoch 1046 (in 0:0:0:5s)
epoch 1047 (in 0:0:0:5s)
epoch 1048 (in 0:0:0:5s)
epoch 1049 (in 0:0:0:5s)
epoch 1050 (in 0:0:0:5s)
epoch 1051 (in 0:0:0:5s)
epoch 1052 (in 0:0:0:5s)
epoch 1053 (in 0:0:0:5s)
epoch 1054 (in 0:0:0:5s)
epoch 1055 (in 0:0:0:5s)
epoch 1056 (in 0:0:0:5s)
epoch 1057 (in 0:0:0:5s)
epoch 1058 (in 0:0:0:5s)
epoch 1059 (in 0:0:0:5s)
epoch 1060 (in 0:0:0:5s)
epoch 1061 (in 0:0:0:5s)
epoch 1062 (in 0:0:0:5s)
epoch 1063 (in 0:0:0:5s)
epoch 1064 (in 0:0:0:5s)
epoch 1065 (in 0:0:0:5s)
epoch 1066 (in 0:0:0:5s)
epoch 1067 (in 0:0:0:5s)
epoch 1068 (in 0:0:0:5s)
epoch 1069 (in 0:0:0:5s)
epoch 1070 (in 0:0:0:5s)
epoch 1071 (in 0:0:0:5s)
epoch 1072 (in 0:0:0:5s)
epoch 1073 (in 0:0:0:5s)
epoch 1074 (in 0:0:0:5s)
epoch 1075 (in 0:0:0:5s)
epoch 1076 (in 0:0:0:5s)
epoch 1077 (in 0:0:0:5s)
epoch 1078 (in 0:0:0:5s)
epoch 1079 (in 0:0:0:5s)
epoch 1080 (in 0:0:0:5s)
epoch 1081 (in 0:0:0:5s)
epoch 1082 (in 0:0:0:5s)
epoch 1083 (in 0:0:0:5s)
epoch 1084 (in 0:0:0:5s)
epoch 1085 (in 0:0:0:5s)
epoch 1086 (in 0:0:0:5s)
epoch 1087 (in 0:0:0:5s)
epoch 1088 (in 0:0:0:5s)
epoch 1089 (in 0:0:0:5s)
epoch 1090 (in 0:0:0:5s)
epoch 1091 (in 0:0:0:5s)
```

```
epoch 1092 (in 0:0:0:5s)
epoch 1093 (in 0:0:0:5s)
epoch 1094 (in 0:0:0:5s)
epoch 1095 (in 0:0:0:5s)
epoch 1096 (in 0:0:0:5s)
epoch 1097 (in 0:0:0:5s)
epoch 1098 (in 0:0:0:5s)
epoch 1099 (in 0:0:0:6s)
epoch 1100 (in 0:0:0:6s)
epoch 1101 (in 0:0:0:6s)
epoch 1102 (in 0:0:0:6s)
epoch 1103 (in 0:0:0:6s)
epoch 1104 (in 0:0:0:6s)
epoch 1105 (in 0:0:0:6s)
epoch 1106 (in 0:0:0:6s)
epoch 1107 (in 0:0:0:6s)
epoch 1108 (in 0:0:0:6s)
epoch 1109 (in 0:0:0:6s)
epoch 1110 (in 0:0:0:6s)
epoch 1111 (in 0:0:0:6s)
epoch 1112 (in 0:0:0:6s)
epoch 1113 (in 0:0:0:6s)
epoch 1114 (in 0:0:0:6s)
epoch 1115 (in 0:0:0:6s)
epoch 1116 (in 0:0:0:6s)
epoch 1117 (in 0:0:0:6s)
epoch 1118 (in 0:0:0:6s)
epoch 1119 (in 0:0:0:6s)
epoch 1120 (in 0:0:0:6s)
epoch 1121 (in 0:0:0:6s)
epoch 1122 (in 0:0:0:6s)
epoch 1123 (in 0:0:0:6s)
epoch 1124 (in 0:0:0:6s)
epoch 1125 (in 0:0:0:6s)
epoch 1126 (in 0:0:0:6s)
epoch 1127 (in 0:0:0:6s)
epoch 1128 (in 0:0:0:6s)
epoch 1129 (in 0:0:0:6s)
epoch 1130 (in 0:0:0:6s)
epoch 1131 (in 0:0:0:6s)
epoch 1132 (in 0:0:0:6s)
epoch 1133 (in 0:0:0:6s)
epoch 1134 (in 0:0:0:6s)
epoch 1135 (in 0:0:0:6s)
epoch 1136 (in 0:0:0:6s)
epoch 1137 (in 0:0:0:6s)
epoch 1138 (in 0:0:0:6s)
epoch 1139 (in 0:0:0:6s)
epoch 1140 (in 0:0:0:6s)
epoch 1141 (in 0:0:0:6s)
epoch 1142 (in 0:0:0:6s)
epoch 1143 (in 0:0:0:6s)
epoch 1144 (in 0:0:0:6s)
epoch 1145 (in 0:0:0:6s)
epoch 1146 (in 0:0:0:6s)
epoch 1147 (in 0:0:0:6s)
epoch 1148 (in 0:0:0:6s)
epoch 1149 (in 0:0:0:6s)
epoch 1150 (in 0:0:0:6s)
epoch 1151 (in 0:0:0:6s)
epoch 1152 (in 0:0:0:6s)
epoch 1153 (in 0:0:0:6s)
```

```
epoch 1154 (in 0:0:0:6s)
epoch 1155 (in 0:0:0:6s)
epoch 1156 (in 0:0:0:6s)
epoch 1157 (in 0:0:0:6s)
epoch 1158 (in 0:0:0:6s)
epoch 1159 (in 0:0:0:6s)
epoch 1160 (in 0:0:0:6s)
epoch 1161 (in 0:0:0:6s)
epoch 1162 (in 0:0:0:6s)
epoch 1163 (in 0:0:0:6s)
epoch 1164 (in 0:0:0:6s)
epoch 1165 (in 0:0:0:6s)
epoch 1166 (in 0:0:0:6s)
epoch 1167 (in 0:0:0:6s)
epoch 1168 (in 0:0:0:6s)
epoch 1169 (in 0:0:0:6s)
epoch 1170 (in 0:0:0:6s)
epoch 1171 (in 0:0:0:6s)
epoch 1172 (in 0:0:0:6s)
epoch 1173 (in 0:0:0:6s)
epoch 1174 (in 0:0:0:6s)
epoch 1175 (in 0:0:0:6s)
epoch 1176 (in 0:0:0:6s)
epoch 1177 (in 0:0:0:6s)
epoch 1178 (in 0:0:0:6s)
epoch 1179 (in 0:0:0:6s)
epoch 1180 (in 0:0:0:6s)
epoch 1181 (in 0:0:0:6s)
epoch 1182 (in 0:0:0:6s)
epoch 1183 (in 0:0:0:6s)
epoch 1184 (in 0:0:0:6s)
epoch 1185 (in 0:0:0:6s)
epoch 1186 (in 0:0:0:6s)
epoch 1187 (in 0:0:0:6s)
epoch 1188 (in 0:0:0:6s)
epoch 1189 (in 0:0:0:6s)
epoch 1190 (in 0:0:0:6s)
epoch 1191 (in 0:0:0:6s)
epoch 1192 (in 0:0:0:6s)
epoch 1193 (in 0:0:0:6s)
epoch 1194 (in 0:0:0:6s)
epoch 1195 (in 0:0:0:6s)
epoch 1196 (in 0:0:0:6s)
epoch 1197 (in 0:0:0:6s)
epoch 1198 (in 0:0:0:6s)
epoch 1199 (in 0:0:0:6s)
epoch 1200 (in 0:0:0:6s)
epoch 1201 (in 0:0:0:6s)
epoch 1202 (in 0:0:0:6s)
epoch 1203 (in 0:0:0:6s)
epoch 1204 (in 0:0:0:6s)
epoch 1205 (in 0:0:0:6s)
epoch 1206 (in 0:0:0:6s)
epoch 1207 (in 0:0:0:6s)
epoch 1208 (in 0:0:0:6s)
epoch 1209 (in 0:0:0:6s)
epoch 1210 (in 0:0:0:6s)
epoch 1211 (in 0:0:0:6s)
epoch 1212 (in 0:0:0:6s)
epoch 1213 (in 0:0:0:6s)
epoch 1214 (in 0:0:0:6s)
epoch 1215 (in 0:0:0:6s)
```

```
epoch 1216 (in 0:0:0:6s)
epoch 1217 (in 0:0:0:6s)
epoch 1218 (in 0:0:0:6s)
epoch 1219 (in 0:0:0:6s)
epoch 1220 (in 0:0:0:6s)
epoch 1221 (in 0:0:0:6s)
epoch 1222 (in 0:0:0:6s)
epoch 1223 (in 0:0:0:6s)
epoch 1224 (in 0:0:0:6s)
epoch 1225 (in 0:0:0:6s)
epoch 1226 (in 0:0:0:6s)
epoch 1227 (in 0:0:0:6s)
epoch 1228 (in 0:0:0:6s)
epoch 1229 (in 0:0:0:6s)
epoch 1230 (in 0:0:0:6s)
epoch 1231 (in 0:0:0:6s)
epoch 1232 (in 0:0:0:6s)
epoch 1233 (in 0:0:0:6s)
epoch 1234 (in 0:0:0:6s)
epoch 1235 (in 0:0:0:6s)
epoch 1236 (in 0:0:0:6s)
epoch 1237 (in 0:0:0:6s)
epoch 1238 (in 0:0:0:6s)
epoch 1239 (in 0:0:0:6s)
epoch 1240 (in 0:0:0:6s)
epoch 1241 (in 0:0:0:6s)
epoch 1242 (in 0:0:0:6s)
epoch 1243 (in 0:0:0:6s)
epoch 1244 (in 0:0:0:6s)
epoch 1245 (in 0:0:0:6s)
epoch 1246 (in 0:0:0:6s)
epoch 1247 (in 0:0:0:6s)
epoch 1248 (in 0:0:0:6s)
epoch 1249 (in 0:0:0:6s)
epoch 1250 (in 0:0:0:6s)
epoch 1251 (in 0:0:0:6s)
epoch 1252 (in 0:0:0:6s)
epoch 1253 (in 0:0:0:6s)
epoch 1254 (in 0:0:0:6s)
epoch 1255 (in 0:0:0:6s)
epoch 1256 (in 0:0:0:6s)
epoch 1257 (in 0:0:0:6s)
epoch 1258 (in 0:0:0:6s)
epoch 1259 (in 0:0:0:6s)
epoch 1260 (in 0:0:0:6s)
epoch 1261 (in 0:0:0:6s)
epoch 1262 (in 0:0:0:6s)
epoch 1263 (in 0:0:0:6s)
epoch 1264 (in 0:0:0:6s)
epoch 1265 (in 0:0:0:6s)
epoch 1266 (in 0:0:0:6s)
epoch 1267 (in 0:0:0:6s)
epoch 1268 (in 0:0:0:6s)
epoch 1269 (in 0:0:0:6s)
epoch 1270 (in 0:0:0:6s)
epoch 1271 (in 0:0:0:6s)
epoch 1272 (in 0:0:0:6s)
epoch 1273 (in 0:0:0:6s)
epoch 1274 (in 0:0:0:6s)
epoch 1275 (in 0:0:0:6s)
epoch 1276 (in 0:0:0:6s)
epoch 1277 (in 0:0:0:6s)
```

```
epoch 1278 (in 0:0:0:6s)
epoch 1279 (in 0:0:0:6s)
epoch 1280 (in 0:0:0:6s)
epoch 1281 (in 0:0:0:6s)
epoch 1282 (in 0:0:0:6s)
epoch 1283 (in 0:0:0:6s)
epoch 1284 (in 0:0:0:6s)
epoch 1285 (in 0:0:0:6s)
epoch 1286 (in 0:0:0:6s)
epoch 1287 (in 0:0:0:6s)
epoch 1288 (in 0:0:0:6s)
epoch 1289 (in 0:0:0:6s)
epoch 1290 (in 0:0:0:6s)
epoch 1291 (in 0:0:0:6s)
epoch 1292 (in 0:0:0:6s)
epoch 1293 (in 0:0:0:6s)
epoch 1294 (in 0:0:0:6s)
epoch 1295 (in 0:0:0:6s)
epoch 1296 (in 0:0:0:7s)
epoch 1297 (in 0:0:0:7s)
epoch 1298 (in 0:0:0:7s)
epoch 1299 (in 0:0:0:7s)
epoch 1300 (in 0:0:0:7s)
epoch 1301 (in 0:0:0:7s)
epoch 1302 (in 0:0:0:7s)
epoch 1303 (in 0:0:0:7s)
epoch 1304 (in 0:0:0:7s)
epoch 1305 (in 0:0:0:7s)
epoch 1306 (in 0:0:0:7s)
epoch 1307 (in 0:0:0:7s)
epoch 1308 (in 0:0:0:7s)
epoch 1309 (in 0:0:0:7s)
epoch 1310 (in 0:0:0:7s)
epoch 1311 (in 0:0:0:7s)
epoch 1312 (in 0:0:0:7s)
epoch 1313 (in 0:0:0:7s)
epoch 1314 (in 0:0:0:7s)
epoch 1315 (in 0:0:0:7s)
epoch 1316 (in 0:0:0:7s)
epoch 1317 (in 0:0:0:7s)
epoch 1318 (in 0:0:0:7s)
epoch 1319 (in 0:0:0:7s)
epoch 1320 (in 0:0:0:7s)
epoch 1321 (in 0:0:0:7s)
epoch 1322 (in 0:0:0:7s)
epoch 1323 (in 0:0:0:7s)
epoch 1324 (in 0:0:0:7s)
epoch 1325 (in 0:0:0:7s)
epoch 1326 (in 0:0:0:7s)
epoch 1327 (in 0:0:0:7s)
epoch 1328 (in 0:0:0:7s)
epoch 1329 (in 0:0:0:7s)
epoch 1330 (in 0:0:0:7s)
epoch 1331 (in 0:0:0:7s)
epoch 1332 (in 0:0:0:7s)
epoch 1333 (in 0:0:0:7s)
epoch 1334 (in 0:0:0:7s)
epoch 1335 (in 0:0:0:7s)
epoch 1336 (in 0:0:0:7s)
epoch 1337 (in 0:0:0:7s)
epoch 1338 (in 0:0:0:7s)
epoch 1339 (in 0:0:0:7s)
```

```
epoch 1340 (in 0:0:0:7s)
epoch 1341 (in 0:0:0:7s)
epoch 1342 (in 0:0:0:7s)
epoch 1343 (in 0:0:0:7s)
epoch 1344 (in 0:0:0:7s)
epoch 1345 (in 0:0:0:7s)
epoch 1346 (in 0:0:0:7s)
epoch 1347 (in 0:0:0:7s)
epoch 1348 (in 0:0:0:7s)
epoch 1349 (in 0:0:0:7s)
epoch 1350 (in 0:0:0:7s)
epoch 1351 (in 0:0:0:7s)
epoch 1352 (in 0:0:0:7s)
epoch 1353 (in 0:0:0:7s)
epoch 1354 (in 0:0:0:7s)
epoch 1355 (in 0:0:0:7s)
epoch 1356 (in 0:0:0:7s)
epoch 1357 (in 0:0:0:7s)
epoch 1358 (in 0:0:0:7s)
epoch 1359 (in 0:0:0:7s)
epoch 1360 (in 0:0:0:7s)
epoch 1361 (in 0:0:0:7s)
epoch 1362 (in 0:0:0:7s)
epoch 1363 (in 0:0:0:7s)
epoch 1364 (in 0:0:0:7s)
epoch 1365 (in 0:0:0:7s)
epoch 1366 (in 0:0:0:7s)
epoch 1367 (in 0:0:0:7s)
epoch 1368 (in 0:0:0:7s)
epoch 1369 (in 0:0:0:7s)
epoch 1370 (in 0:0:0:7s)
epoch 1371 (in 0:0:0:7s)
epoch 1372 (in 0:0:0:7s)
epoch 1373 (in 0:0:0:7s)
epoch 1374 (in 0:0:0:7s)
epoch 1375 (in 0:0:0:7s)
epoch 1376 (in 0:0:0:7s)
epoch 1377 (in 0:0:0:7s)
epoch 1378 (in 0:0:0:7s)
epoch 1379 (in 0:0:0:7s)
epoch 1380 (in 0:0:0:7s)
epoch 1381 (in 0:0:0:7s)
epoch 1382 (in 0:0:0:7s)
epoch 1383 (in 0:0:0:7s)
epoch 1384 (in 0:0:0:7s)
epoch 1385 (in 0:0:0:7s)
epoch 1386 (in 0:0:0:7s)
epoch 1387 (in 0:0:0:7s)
epoch 1388 (in 0:0:0:7s)
epoch 1389 (in 0:0:0:7s)
epoch 1390 (in 0:0:0:7s)
epoch 1391 (in 0:0:0:7s)
epoch 1392 (in 0:0:0:7s)
epoch 1393 (in 0:0:0:7s)
epoch 1394 (in 0:0:0:7s)
epoch 1395 (in 0:0:0:7s)
epoch 1396 (in 0:0:0:7s)
epoch 1397 (in 0:0:0:7s)
epoch 1398 (in 0:0:0:7s)
epoch 1399 (in 0:0:0:7s)
epoch 1400 (in 0:0:0:7s)
epoch 1401 (in 0:0:0:7s)
```

```
epoch 1402 (in 0:0:0:7s)
epoch 1403 (in 0:0:0:7s)
epoch 1404 (in 0:0:0:7s)
epoch 1405 (in 0:0:0:7s)
epoch 1406 (in 0:0:0:7s)
epoch 1407 (in 0:0:0:7s)
epoch 1408 (in 0:0:0:7s)
epoch 1409 (in 0:0:0:7s)
epoch 1410 (in 0:0:0:7s)
epoch 1411 (in 0:0:0:7s)
epoch 1412 (in 0:0:0:7s)
epoch 1413 (in 0:0:0:7s)
epoch 1414 (in 0:0:0:7s)
epoch 1415 (in 0:0:0:7s)
epoch 1416 (in 0:0:0:7s)
epoch 1417 (in 0:0:0:7s)
epoch 1418 (in 0:0:0:7s)
epoch 1419 (in 0:0:0:7s)
epoch 1420 (in 0:0:0:7s)
epoch 1421 (in 0:0:0:7s)
epoch 1422 (in 0:0:0:7s)
epoch 1423 (in 0:0:0:7s)
epoch 1424 (in 0:0:0:7s)
epoch 1425 (in 0:0:0:7s)
epoch 1426 (in 0:0:0:7s)
epoch 1427 (in 0:0:0:7s)
epoch 1428 (in 0:0:0:7s)
epoch 1429 (in 0:0:0:7s)
epoch 1430 (in 0:0:0:7s)
epoch 1431 (in 0:0:0:7s)
epoch 1432 (in 0:0:0:7s)
epoch 1433 (in 0:0:0:7s)
epoch 1434 (in 0:0:0:7s)
epoch 1435 (in 0:0:0:7s)
epoch 1436 (in 0:0:0:7s)
epoch 1437 (in 0:0:0:7s)
epoch 1438 (in 0:0:0:7s)
epoch 1439 (in 0:0:0:7s)
epoch 1440 (in 0:0:0:7s)
epoch 1441 (in 0:0:0:7s)
epoch 1442 (in 0:0:0:7s)
epoch 1443 (in 0:0:0:7s)
epoch 1444 (in 0:0:0:7s)
epoch 1445 (in 0:0:0:7s)
epoch 1446 (in 0:0:0:7s)
epoch 1447 (in 0:0:0:7s)
epoch 1448 (in 0:0:0:7s)
epoch 1449 (in 0:0:0:7s)
epoch 1450 (in 0:0:0:7s)
epoch 1451 (in 0:0:0:7s)
epoch 1452 (in 0:0:0:7s)
epoch 1453 (in 0:0:0:7s)
epoch 1454 (in 0:0:0:7s)
epoch 1455 (in 0:0:0:7s)
epoch 1456 (in 0:0:0:7s)
epoch 1457 (in 0:0:0:7s)
epoch 1458 (in 0:0:0:7s)
epoch 1459 (in 0:0:0:7s)
epoch 1460 (in 0:0:0:7s)
epoch 1461 (in 0:0:0:7s)
epoch 1462 (in 0:0:0:7s)
epoch 1463 (in 0:0:0:7s)
```

```
epoch 1464 (in 0:0:0:7s)
epoch 1465 (in 0:0:0:7s)
epoch 1466 (in 0:0:0:7s)
epoch 1467 (in 0:0:0:7s)
epoch 1468 (in 0:0:0:7s)
epoch 1469 (in 0:0:0:7s)
epoch 1470 (in 0:0:0:7s)
epoch 1471 (in 0:0:0:7s)
epoch 1472 (in 0:0:0:7s)
epoch 1473 (in 0:0:0:7s)
epoch 1474 (in 0:0:0:7s)
epoch 1475 (in 0:0:0:7s)
epoch 1476 (in 0:0:0:7s)
epoch 1477 (in 0:0:0:7s)
epoch 1478 (in 0:0:0:7s)
epoch 1479 (in 0:0:0:7s)
epoch 1480 (in 0:0:0:7s)
epoch 1481 (in 0:0:0:7s)
epoch 1482 (in 0:0:0:7s)
epoch 1483 (in 0:0:0:7s)
epoch 1484 (in 0:0:0:7s)
epoch 1485 (in 0:0:0:7s)
epoch 1486 (in 0:0:0:7s)
epoch 1487 (in 0:0:0:7s)
epoch 1488 (in 0:0:0:7s)
epoch 1489 (in 0:0:0:7s)
epoch 1490 (in 0:0:0:7s)
epoch 1491 (in 0:0:0:7s)
epoch 1492 (in 0:0:0:8s)
epoch 1493 (in 0:0:0:8s)
epoch 1494 (in 0:0:0:8s)
epoch 1495 (in 0:0:0:8s)
epoch 1496 (in 0:0:0:8s)
epoch 1497 (in 0:0:0:8s)
epoch 1498 (in 0:0:0:8s)
epoch 1499 (in 0:0:0:8s)
epoch 1500 (in 0:0:0:8s)
epoch 1501 (in 0:0:0:8s)
epoch 1502 (in 0:0:0:8s)
epoch 1503 (in 0:0:0:8s)
epoch 1504 (in 0:0:0:8s)
epoch 1505 (in 0:0:0:8s)
epoch 1506 (in 0:0:0:8s)
epoch 1507 (in 0:0:0:8s)
epoch 1508 (in 0:0:0:8s)
epoch 1509 (in 0:0:0:8s)
epoch 1510 (in 0:0:0:8s)
epoch 1511 (in 0:0:0:8s)
epoch 1512 (in 0:0:0:8s)
epoch 1513 (in 0:0:0:8s)
epoch 1514 (in 0:0:0:8s)
epoch 1515 (in 0:0:0:8s)
epoch 1516 (in 0:0:0:8s)
epoch 1517 (in 0:0:0:8s)
epoch 1518 (in 0:0:0:8s)
epoch 1519 (in 0:0:0:8s)
epoch 1520 (in 0:0:0:8s)
epoch 1521 (in 0:0:0:8s)
epoch 1522 (in 0:0:0:8s)
epoch 1523 (in 0:0:0:8s)
epoch 1524 (in 0:0:0:8s)
epoch 1525 (in 0:0:0:8s)
```

```
epoch 1526 (in 0:0:0:8s)
epoch 1527 (in 0:0:0:8s)
epoch 1528 (in 0:0:0:8s)
epoch 1529 (in 0:0:0:8s)
epoch 1530 (in 0:0:0:8s)
epoch 1531 (in 0:0:0:8s)
epoch 1532 (in 0:0:0:8s)
epoch 1533 (in 0:0:0:8s)
epoch 1534 (in 0:0:0:8s)
epoch 1535 (in 0:0:0:8s)
epoch 1536 (in 0:0:0:8s)
epoch 1537 (in 0:0:0:8s)
epoch 1538 (in 0:0:0:8s)
epoch 1539 (in 0:0:0:8s)
epoch 1540 (in 0:0:0:8s)
epoch 1541 (in 0:0:0:8s)
epoch 1542 (in 0:0:0:8s)
epoch 1543 (in 0:0:0:8s)
epoch 1544 (in 0:0:0:8s)
epoch 1545 (in 0:0:0:8s)
epoch 1546 (in 0:0:0:8s)
epoch 1547 (in 0:0:0:8s)
epoch 1548 (in 0:0:0:8s)
epoch 1549 (in 0:0:0:8s)
epoch 1550 (in 0:0:0:8s)
epoch 1551 (in 0:0:0:8s)
epoch 1552 (in 0:0:0:8s)
epoch 1553 (in 0:0:0:8s)
epoch 1554 (in 0:0:0:8s)
epoch 1555 (in 0:0:0:8s)
epoch 1556 (in 0:0:0:8s)
epoch 1557 (in 0:0:0:8s)
epoch 1558 (in 0:0:0:8s)
epoch 1559 (in 0:0:0:8s)
epoch 1560 (in 0:0:0:8s)
epoch 1561 (in 0:0:0:8s)
epoch 1562 (in 0:0:0:8s)
epoch 1563 (in 0:0:0:8s)
epoch 1564 (in 0:0:0:8s)
epoch 1565 (in 0:0:0:8s)
epoch 1566 (in 0:0:0:8s)
epoch 1567 (in 0:0:0:8s)
epoch 1568 (in 0:0:0:8s)
epoch 1569 (in 0:0:0:8s)
epoch 1570 (in 0:0:0:8s)
epoch 1571 (in 0:0:0:8s)
epoch 1572 (in 0:0:0:8s)
epoch 1573 (in 0:0:0:8s)
epoch 1574 (in 0:0:0:8s)
epoch 1575 (in 0:0:0:8s)
epoch 1576 (in 0:0:0:8s)
epoch 1577 (in 0:0:0:8s)
epoch 1578 (in 0:0:0:8s)
epoch 1579 (in 0:0:0:8s)
epoch 1580 (in 0:0:0:8s)
epoch 1581 (in 0:0:0:8s)
epoch 1582 (in 0:0:0:8s)
epoch 1583 (in 0:0:0:8s)
epoch 1584 (in 0:0:0:8s)
epoch 1585 (in 0:0:0:8s)
epoch 1586 (in 0:0:0:8s)
epoch 1587 (in 0:0:0:8s)
```

```
epoch 1588 (in 0:0:0:8s)
epoch 1589 (in 0:0:0:8s)
epoch 1590 (in 0:0:0:8s)
epoch 1591 (in 0:0:0:8s)
epoch 1592 (in 0:0:0:8s)
epoch 1593 (in 0:0:0:8s)
epoch 1594 (in 0:0:0:8s)
epoch 1595 (in 0:0:0:8s)
epoch 1596 (in 0:0:0:8s)
epoch 1597 (in 0:0:0:8s)
epoch 1598 (in 0:0:0:8s)
epoch 1599 (in 0:0:0:8s)
epoch 1600 (in 0:0:0:8s)
epoch 1601 (in 0:0:0:8s)
epoch 1602 (in 0:0:0:8s)
epoch 1603 (in 0:0:0:8s)
epoch 1604 (in 0:0:0:8s)
epoch 1605 (in 0:0:0:8s)
epoch 1606 (in 0:0:0:8s)
epoch 1607 (in 0:0:0:8s)
epoch 1608 (in 0:0:0:8s)
epoch 1609 (in 0:0:0:8s)
epoch 1610 (in 0:0:0:8s)
epoch 1611 (in 0:0:0:8s)
epoch 1612 (in 0:0:0:8s)
epoch 1613 (in 0:0:0:8s)
epoch 1614 (in 0:0:0:8s)
epoch 1615 (in 0:0:0:8s)
epoch 1616 (in 0:0:0:8s)
epoch 1617 (in 0:0:0:8s)
epoch 1618 (in 0:0:0:8s)
epoch 1619 (in 0:0:0:8s)
epoch 1620 (in 0:0:0:8s)
epoch 1621 (in 0:0:0:8s)
epoch 1622 (in 0:0:0:8s)
epoch 1623 (in 0:0:0:8s)
epoch 1624 (in 0:0:0:8s)
epoch 1625 (in 0:0:0:8s)
epoch 1626 (in 0:0:0:8s)
epoch 1627 (in 0:0:0:8s)
epoch 1628 (in 0:0:0:8s)
epoch 1629 (in 0:0:0:8s)
epoch 1630 (in 0:0:0:8s)
epoch 1631 (in 0:0:0:8s)
epoch 1632 (in 0:0:0:8s)
epoch 1633 (in 0:0:0:8s)
epoch 1634 (in 0:0:0:8s)
epoch 1635 (in 0:0:0:8s)
epoch 1636 (in 0:0:0:8s)
epoch 1637 (in 0:0:0:8s)
epoch 1638 (in 0:0:0:8s)
epoch 1639 (in 0:0:0:8s)
epoch 1640 (in 0:0:0:8s)
epoch 1641 (in 0:0:0:8s)
epoch 1642 (in 0:0:0:8s)
epoch 1643 (in 0:0:0:8s)
epoch 1644 (in 0:0:0:8s)
epoch 1645 (in 0:0:0:8s)
epoch 1646 (in 0:0:0:8s)
epoch 1647 (in 0:0:0:8s)
epoch 1648 (in 0:0:0:8s)
epoch 1649 (in 0:0:0:8s)
```

```
epoch 1650 (in 0:0:0:8s)
epoch 1651 (in 0:0:0:8s)
epoch 1652 (in 0:0:0:8s)
epoch 1653 (in 0:0:0:8s)
epoch 1654 (in 0:0:0:8s)
epoch 1655 (in 0:0:0:8s)
epoch 1656 (in 0:0:0:8s)
epoch 1657 (in 0:0:0:8s)
epoch 1658 (in 0:0:0:8s)
epoch 1659 (in 0:0:0:8s)
epoch 1660 (in 0:0:0:8s)
epoch 1661 (in 0:0:0:8s)
epoch 1662 (in 0:0:0:8s)
epoch 1663 (in 0:0:0:8s)
epoch 1664 (in 0:0:0:8s)
epoch 1665 (in 0:0:0:8s)
epoch 1666 (in 0:0:0:8s)
epoch 1667 (in 0:0:0:8s)
epoch 1668 (in 0:0:0:8s)
epoch 1669 (in 0:0:0:8s)
epoch 1670 (in 0:0:0:8s)
epoch 1671 (in 0:0:0:8s)
epoch 1672 (in 0:0:0:8s)
epoch 1673 (in 0:0:0:8s)
epoch 1674 (in 0:0:0:8s)
epoch 1675 (in 0:0:0:8s)
epoch 1676 (in 0:0:0:8s)
epoch 1677 (in 0:0:0:9s)
epoch 1678 (in 0:0:0:9s)
epoch 1679 (in 0:0:0:9s)
epoch 1680 (in 0:0:0:9s)
epoch 1681 (in 0:0:0:9s)
epoch 1682 (in 0:0:0:9s)
epoch 1683 (in 0:0:0:9s)
epoch 1684 (in 0:0:0:9s)
epoch 1685 (in 0:0:0:9s)
epoch 1686 (in 0:0:0:9s)
epoch 1687 (in 0:0:0:9s)
epoch 1688 (in 0:0:0:9s)
epoch 1689 (in 0:0:0:9s)
epoch 1690 (in 0:0:0:9s)
epoch 1691 (in 0:0:0:9s)
epoch 1692 (in 0:0:0:9s)
epoch 1693 (in 0:0:0:9s)
epoch 1694 (in 0:0:0:9s)
epoch 1695 (in 0:0:0:9s)
epoch 1696 (in 0:0:0:9s)
epoch 1697 (in 0:0:0:9s)
epoch 1698 (in 0:0:0:9s)
epoch 1699 (in 0:0:0:9s)
epoch 1700 (in 0:0:0:9s)
epoch 1701 (in 0:0:0:9s)
epoch 1702 (in 0:0:0:9s)
epoch 1703 (in 0:0:0:9s)
epoch 1704 (in 0:0:0:9s)
epoch 1705 (in 0:0:0:9s)
epoch 1706 (in 0:0:0:9s)
epoch 1707 (in 0:0:0:9s)
epoch 1708 (in 0:0:0:9s)
epoch 1709 (in 0:0:0:9s)
epoch 1710 (in 0:0:0:9s)
epoch 1711 (in 0:0:0:9s)
```

```
epoch 1712 (in 0:0:0:9s)
epoch 1713 (in 0:0:0:9s)
epoch 1714 (in 0:0:0:9s)
epoch 1715 (in 0:0:0:9s)
epoch 1716 (in 0:0:0:9s)
epoch 1717 (in 0:0:0:9s)
epoch 1718 (in 0:0:0:9s)
epoch 1719 (in 0:0:0:9s)
epoch 1720 (in 0:0:0:9s)
epoch 1721 (in 0:0:0:9s)
epoch 1722 (in 0:0:0:9s)
epoch 1723 (in 0:0:0:9s)
epoch 1724 (in 0:0:0:9s)
epoch 1725 (in 0:0:0:9s)
epoch 1726 (in 0:0:0:9s)
epoch 1727 (in 0:0:0:9s)
epoch 1728 (in 0:0:0:9s)
epoch 1729 (in 0:0:0:9s)
epoch 1730 (in 0:0:0:9s)
epoch 1731 (in 0:0:0:9s)
epoch 1732 (in 0:0:0:9s)
epoch 1733 (in 0:0:0:9s)
epoch 1734 (in 0:0:0:9s)
epoch 1735 (in 0:0:0:9s)
epoch 1736 (in 0:0:0:9s)
epoch 1737 (in 0:0:0:9s)
epoch 1738 (in 0:0:0:9s)
epoch 1739 (in 0:0:0:9s)
epoch 1740 (in 0:0:0:9s)
epoch 1741 (in 0:0:0:9s)
epoch 1742 (in 0:0:0:9s)
epoch 1743 (in 0:0:0:9s)
epoch 1744 (in 0:0:0:9s)
epoch 1745 (in 0:0:0:9s)
epoch 1746 (in 0:0:0:9s)
epoch 1747 (in 0:0:0:9s)
epoch 1748 (in 0:0:0:9s)
epoch 1749 (in 0:0:0:9s)
epoch 1750 (in 0:0:0:9s)
epoch 1751 (in 0:0:0:9s)
epoch 1752 (in 0:0:0:9s)
epoch 1753 (in 0:0:0:9s)
epoch 1754 (in 0:0:0:9s)
epoch 1755 (in 0:0:0:9s)
epoch 1756 (in 0:0:0:9s)
epoch 1757 (in 0:0:0:9s)
epoch 1758 (in 0:0:0:9s)
epoch 1759 (in 0:0:0:9s)
epoch 1760 (in 0:0:0:9s)
epoch 1761 (in 0:0:0:9s)
epoch 1762 (in 0:0:0:9s)
epoch 1763 (in 0:0:0:9s)
epoch 1764 (in 0:0:0:9s)
epoch 1765 (in 0:0:0:9s)
epoch 1766 (in 0:0:0:9s)
epoch 1767 (in 0:0:0:9s)
epoch 1768 (in 0:0:0:9s)
epoch 1769 (in 0:0:0:9s)
epoch 1770 (in 0:0:0:9s)
epoch 1771 (in 0:0:0:9s)
epoch 1772 (in 0:0:0:9s)
epoch 1773 (in 0:0:0:9s)
```

```
epoch 1774 (in 0:0:0:9s)
epoch 1775 (in 0:0:0:9s)
epoch 1776 (in 0:0:0:9s)
epoch 1777 (in 0:0:0:9s)
epoch 1778 (in 0:0:0:9s)
epoch 1779 (in 0:0:0:9s)
epoch 1780 (in 0:0:0:9s)
epoch 1781 (in 0:0:0:9s)
epoch 1782 (in 0:0:0:9s)
epoch 1783 (in 0:0:0:9s)
epoch 1784 (in 0:0:0:9s)
epoch 1785 (in 0:0:0:9s)
epoch 1786 (in 0:0:0:9s)
epoch 1787 (in 0:0:0:9s)
epoch 1788 (in 0:0:0:9s)
epoch 1789 (in 0:0:0:9s)
epoch 1790 (in 0:0:0:9s)
epoch 1791 (in 0:0:0:9s)
epoch 1792 (in 0:0:0:9s)
epoch 1793 (in 0:0:0:9s)
epoch 1794 (in 0:0:0:9s)
epoch 1795 (in 0:0:0:9s)
epoch 1796 (in 0:0:0:9s)
epoch 1797 (in 0:0:0:9s)
epoch 1798 (in 0:0:0:9s)
epoch 1799 (in 0:0:0:9s)
epoch 1800 (in 0:0:0:9s)
epoch 1801 (in 0:0:0:9s)
epoch 1802 (in 0:0:0:9s)
epoch 1803 (in 0:0:0:9s)
epoch 1804 (in 0:0:0:9s)
epoch 1805 (in 0:0:0:9s)
epoch 1806 (in 0:0:0:9s)
epoch 1807 (in 0:0:0:9s)
epoch 1808 (in 0:0:0:9s)
epoch 1809 (in 0:0:0:9s)
epoch 1810 (in 0:0:0:9s)
epoch 1811 (in 0:0:0:9s)
epoch 1812 (in 0:0:0:9s)
epoch 1813 (in 0:0:0:9s)
epoch 1814 (in 0:0:0:9s)
epoch 1815 (in 0:0:0:9s)
epoch 1816 (in 0:0:0:9s)
epoch 1817 (in 0:0:0:9s)
epoch 1818 (in 0:0:0:9s)
epoch 1819 (in 0:0:0:9s)
epoch 1820 (in 0:0:0:9s)
epoch 1821 (in 0:0:0:9s)
epoch 1822 (in 0:0:0:9s)
epoch 1823 (in 0:0:0:9s)
epoch 1824 (in 0:0:0:9s)
epoch 1825 (in 0:0:0:9s)
epoch 1826 (in 0:0:0:9s)
epoch 1827 (in 0:0:0:9s)
epoch 1828 (in 0:0:0:9s)
epoch 1829 (in 0:0:0:9s)
epoch 1830 (in 0:0:0:9s)
epoch 1831 (in 0:0:0:9s)
epoch 1832 (in 0:0:0:9s)
epoch 1833 (in 0:0:0:9s)
epoch 1834 (in 0:0:0:9s)
epoch 1835 (in 0:0:0:9s)
```

```
epoch 1836 (in 0:0:0:9s)
epoch 1837 (in 0:0:0:9s)
epoch 1838 (in 0:0:0:9s)
epoch 1839 (in 0:0:0:9s)
epoch 1840 (in 0:0:0:9s)
epoch 1841 (in 0:0:0:9s)
epoch 1842 (in 0:0:0:9s)
epoch 1843 (in 0:0:0:9s)
epoch 1844 (in 0:0:0:9s)
epoch 1845 (in 0:0:0:9s)
epoch 1846 (in 0:0:0:9s)
epoch 1847 (in 0:0:0:9s)
epoch 1848 (in 0:0:0:9s)
epoch 1849 (in 0:0:0:9s)
epoch 1850 (in 0:0:0:9s)
epoch 1851 (in 0:0:0:9s)
epoch 1852 (in 0:0:0:9s)
epoch 1853 (in 0:0:0:9s)
epoch 1854 (in 0:0:0:10s)
epoch 1855 (in 0:0:0:10s)
epoch 1856 (in 0:0:0:10s)
epoch 1857 (in 0:0:0:10s)
epoch 1858 (in 0:0:0:10s)
epoch 1859 (in 0:0:0:10s)
epoch 1860 (in 0:0:0:10s)
epoch 1861 (in 0:0:0:10s)
epoch 1862 (in 0:0:0:10s)
epoch 1863 (in 0:0:0:10s)
epoch 1864 (in 0:0:0:10s)
epoch 1865 (in 0:0:0:10s)
epoch 1866 (in 0:0:0:10s)
epoch 1867 (in 0:0:0:10s)
epoch 1868 (in 0:0:0:10s)
epoch 1869 (in 0:0:0:10s)
epoch 1870 (in 0:0:0:10s)
epoch 1871 (in 0:0:0:10s)
epoch 1872 (in 0:0:0:10s)
epoch 1873 (in 0:0:0:10s)
epoch 1874 (in 0:0:0:10s)
epoch 1875 (in 0:0:0:10s)
epoch 1876 (in 0:0:0:10s)
epoch 1877 (in 0:0:0:10s)
epoch 1878 (in 0:0:0:10s)
epoch 1879 (in 0:0:0:10s)
epoch 1880 (in 0:0:0:10s)
epoch 1881 (in 0:0:0:10s)
epoch 1882 (in 0:0:0:10s)
epoch 1883 (in 0:0:0:10s)
epoch 1884 (in 0:0:0:10s)
epoch 1885 (in 0:0:0:10s)
epoch 1886 (in 0:0:0:10s)
epoch 1887 (in 0:0:0:10s)
epoch 1888 (in 0:0:0:10s)
epoch 1889 (in 0:0:0:10s)
epoch 1890 (in 0:0:0:10s)
epoch 1891 (in 0:0:0:10s)
epoch 1892 (in 0:0:0:10s)
epoch 1893 (in 0:0:0:10s)
epoch 1894 (in 0:0:0:10s)
epoch 1895 (in 0:0:0:10s)
epoch 1896 (in 0:0:0:10s)
epoch 1897 (in 0:0:0:10s)
```

```
epoch 1898 (in 0:0:0:10s)
epoch 1899 (in 0:0:0:10s)
epoch 1900 (in 0:0:0:10s)
epoch 1901 (in 0:0:0:10s)
epoch 1902 (in 0:0:0:10s)
epoch 1903 (in 0:0:0:10s)
epoch 1904 (in 0:0:0:10s)
epoch 1905 (in 0:0:0:10s)
epoch 1906 (in 0:0:0:10s)
epoch 1907 (in 0:0:0:10s)
epoch 1908 (in 0:0:0:10s)
epoch 1909 (in 0:0:0:10s)
epoch 1910 (in 0:0:0:10s)
epoch 1911 (in 0:0:0:10s)
epoch 1912 (in 0:0:0:10s)
epoch 1913 (in 0:0:0:10s)
epoch 1914 (in 0:0:0:10s)
epoch 1915 (in 0:0:0:10s)
epoch 1916 (in 0:0:0:10s)
epoch 1917 (in 0:0:0:10s)
epoch 1918 (in 0:0:0:10s)
epoch 1919 (in 0:0:0:10s)
epoch 1920 (in 0:0:0:10s)
epoch 1921 (in 0:0:0:10s)
epoch 1922 (in 0:0:0:10s)
epoch 1923 (in 0:0:0:10s)
epoch 1924 (in 0:0:0:10s)
epoch 1925 (in 0:0:0:10s)
epoch 1926 (in 0:0:0:10s)
epoch 1927 (in 0:0:0:10s)
epoch 1928 (in 0:0:0:10s)
epoch 1929 (in 0:0:0:10s)
epoch 1930 (in 0:0:0:10s)
epoch 1931 (in 0:0:0:10s)
epoch 1932 (in 0:0:0:10s)
epoch 1933 (in 0:0:0:10s)
epoch 1934 (in 0:0:0:10s)
epoch 1935 (in 0:0:0:10s)
epoch 1936 (in 0:0:0:10s)
epoch 1937 (in 0:0:0:10s)
epoch 1938 (in 0:0:0:10s)
epoch 1939 (in 0:0:0:10s)
epoch 1940 (in 0:0:0:10s)
epoch 1941 (in 0:0:0:10s)
epoch 1942 (in 0:0:0:10s)
epoch 1943 (in 0:0:0:10s)
epoch 1944 (in 0:0:0:10s)
epoch 1945 (in 0:0:0:10s)
epoch 1946 (in 0:0:0:10s)
epoch 1947 (in 0:0:0:10s)
epoch 1948 (in 0:0:0:10s)
epoch 1949 (in 0:0:0:10s)
epoch 1950 (in 0:0:0:10s)
epoch 1951 (in 0:0:0:10s)
epoch 1952 (in 0:0:0:10s)
epoch 1953 (in 0:0:0:10s)
epoch 1954 (in 0:0:0:10s)
epoch 1955 (in 0:0:0:10s)
epoch 1956 (in 0:0:0:10s)
epoch 1957 (in 0:0:0:10s)
epoch 1958 (in 0:0:0:10s)
epoch 1959 (in 0:0:0:10s)
```

```
epoch 1960 (in 0:0:0:10s)
epoch 1961 (in 0:0:0:10s)
epoch 1962 (in 0:0:0:10s)
epoch 1963 (in 0:0:0:10s)
epoch 1964 (in 0:0:0:10s)
epoch 1965 (in 0:0:0:10s)
epoch 1966 (in 0:0:0:10s)
epoch 1967 (in 0:0:0:10s)
epoch 1968 (in 0:0:0:10s)
epoch 1969 (in 0:0:0:10s)
epoch 1970 (in 0:0:0:10s)
epoch 1971 (in 0:0:0:10s)
epoch 1972 (in 0:0:0:10s)
epoch 1973 (in 0:0:0:10s)
epoch 1974 (in 0:0:0:10s)
epoch 1975 (in 0:0:0:10s)
epoch 1976 (in 0:0:0:10s)
epoch 1977 (in 0:0:0:10s)
epoch 1978 (in 0:0:0:10s)
epoch 1979 (in 0:0:0:10s)
epoch 1980 (in 0:0:0:10s)
epoch 1981 (in 0:0:0:10s)
epoch 1982 (in 0:0:0:10s)
epoch 1983 (in 0:0:0:10s)
epoch 1984 (in 0:0:0:10s)
epoch 1985 (in 0:0:0:10s)
epoch 1986 (in 0:0:0:10s)
epoch 1987 (in 0:0:0:10s)
epoch 1988 (in 0:0:0:10s)
epoch 1989 (in 0:0:0:10s)
epoch 1990 (in 0:0:0:10s)
epoch 1991 (in 0:0:0:10s)
epoch 1992 (in 0:0:0:10s)
epoch 1993 (in 0:0:0:10s)
epoch 1994 (in 0:0:0:10s)
epoch 1995 (in 0:0:0:10s)
epoch 1996 (in 0:0:0:10s)
epoch 1997 (in 0:0:0:10s)
epoch 1998 (in 0:0:0:10s)
epoch 1999 (in 0:0:0:10s)
epoch 2000 (in 0:0:0:10s)
Learning complete (in 0:0:0:10s)
```

validation (with confidence vector):

```
#74 pred2 real2 <-6.035,-0.241,-0.212> OK
#73 pred2 real2 <-8.584,-0.368,-0.212> OK
#72 pred2 real2 <-7.054,-0.292,-0.212> OK
#71 pred2 real2 <-6.545,-0.266,-0.212> OK
#70 pred2 real2 <-8.584,-0.368,-0.212> OK
#69 pred2 real2 <-9.604,-0.419,-0.212> OK
#68 pred2 real2 <-8.584,-0.368,-0.212> OK
#67 pred2 real2 <-6.545,-0.266,-0.212> OK
#66 pred2 real2 <-8.584,-0.368,-0.212> OK
#65 pred2 real2 <-9.094,-0.393,-0.212> OK
#64 pred2 real2 <-7.564,-0.317,-0.212> OK
#63 pred2 real2 <-6.035,-0.241,-0.212> OK
#62 pred2 real2 <-6.035,-0.241,-0.212> OK
#61 pred2 real2 <-9.094,-0.393,-0.212> OK
#60 pred2 real2 <-8.584,-0.368,-0.212> OK
#59 pred1 real2 <-3.995,-0.139,-0.212> NG
```

```
#58 pred1 real2 <-4.505,-0.164,-0.212> NG
#57 pred2 real2 <-8.074,-0.342,-0.212> OK
#56 pred2 real2 <-7.054,-0.292,-0.212> OK
#55 pred2 real2 <-6.545,-0.266,-0.212> OK
#54 pred1 real2 <-5.015,-0.190,-0.212> NG
#53 pred2 real2 <-7.564,-0.317,-0.212> OK
#52 pred2 real2 <-6.035,-0.241,-0.212> OK
#51 pred2 real2 <-6.035,-0.241,-0.212> OK
#50 pred2 real2 <-6.035,-0.241,-0.212> OK
#49 pred1 real1 <-3.485,-0.114,-0.212> OK
#48 pred1 real1 <-2.466,-0.063,-0.212> OK
#47 pred1 real1 <-3.485,-0.114,-0.212> OK
#46 pred1 real1 <-3.485,-0.114,-0.212> OK
#45 pred1 real1 <-2.976,-0.088,-0.212> OK
#44 pred1 real1 <-3.485,-0.114,-0.212> OK
#43 pred1 real1 <-1.956,-0.037,-0.212> OK
#42 pred1 real1 <-2.976,-0.088,-0.212> OK
#41 pred1 real1 <-3.995,-0.139,-0.212> OK
#40 pred1 real1 <-2.976,-0.088,-0.212> OK
#39 pred1 real1 <-3.485,-0.114,-0.212> OK
#38 pred1 real1 <-3.485,-0.114,-0.212> OK
#37 pred1 real1 <-3.485,-0.114,-0.212> OK
#36 pred1 real1 <-4.505,-0.164,-0.212> OK
#35 pred1 real1 <-5.015,-0.190,-0.212> OK
#34 pred1 real1 <-4.505,-0.164,-0.212> OK
#33 pred1 real1 <-5.015,-0.190,-0.212> OK
#32 pred1 real1 <-2.976,-0.088,-0.212> OK
#31 pred1 real1 <-1.956,-0.037,-0.212> OK
#30 pred1 real1 <-2.466,-0.063,-0.212> OK
#29 pred1 real1 <-1.956,-0.037,-0.212> OK
#28 pred1 real1 <-4.505,-0.164,-0.212> OK
#27 pred2 real1 <-5.525,-0.215,-0.212> NG
#26 pred1 real1 <-3.995,-0.139,-0.212> OK
#25 pred1 real1 <-3.995,-0.139,-0.212> OK
#24 pred0 real0 <2.123,0.166,-0.212> OK
#23 pred0 real0 <2.123,0.166,-0.212> OK
#22 pred0 real0 <2.123,0.166,-0.212> OK
#21 pred0 real0 <2.123,0.166,-0.212> OK
#20 pred0 real0 <1.613,0.141,-0.212> OK
#19 pred0 real0 <1.103,0.115,-0.212> OK
#18 pred0 real0 <0.084,0.065,-0.212> OK
#17 pred0 real0 <2.123,0.166,-0.212> OK
#16 pred0 real0 <1.613,0.141,-0.212> OK
#15 pred0 real0 <1.613,0.141,-0.212> OK
#14 pred0 real0 <2.123,0.166,-0.212> OK
#13 pred0 real0 <2.123,0.166,-0.212> OK
#12 pred0 real0 <2.633,0.192,-0.212> OK
#11 pred0 real0 <2.123,0.166,-0.212> OK
#10 pred0 real0 <2.123,0.166,-0.212> OK
#9 pred0 real0 <2.123,0.166,-0.212> OK
#8 pred0 real0 <2.123,0.166,-0.212> OK
#7 pred0 real0 <2.633,0.192,-0.212> OK
#6 pred0 real0 <1.103,0.115,-0.212> OK
#5 pred0 real0 <2.123,0.166,-0.212> OK
#4 pred0 real0 <2.123,0.166,-0.212> OK
#3 pred0 real0 <2.123,0.166,-0.212> OK
#2 pred0 real0 <2.123,0.166,-0.212> OK
#1 pred0 real0 <1.103,0.115,-0.212> OK
#0 pred0 real0 <2.123,0.166,-0.212> OK
Value: 0.946667
```

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 50
#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 500
// 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 {
    ga = GenAlgCreate(ADN_SIZE_POOL, ADN_SIZE_ELITE,
      NNGetGAAdnFloatLength(nn), NNGetGAAdnIntLength(nn));
    GASetTypeNeuraNet(ga, NB_INPUT, NB_MAXHIDDEN, NB_OUTPUT);
```

```
  NNSetGABoundsBases(nn, ga);
  NNSetGABoundsLinks(nn, ga);
  GAInit(ga);
  // Init all the links except the first one to deactivated
  GSetIterForward iter = GSetIterForwardCreateStatic(GAAdns(ga));
  do {
    GenAlgAdn* adn = GSetIterGet(&iter);
    for (int iGene = 3; iGene < VecGetDim(GAAdnAdnI(adn)); iGene += 3)
      GAAdnSetGeneI(adn, iGene, -1);
  } while (GSetIterStep(&iter));
}
// Start learning process
printf("Learning...\n");
printf("Will stop when curEpoch >= %lu or bestVal >= %f\n",
  limitEpoch, STOP_LEARNING_AT_VAL);
printf("Best NeuraNet saved in ./bestnn.txt at each improvement\n");
if (SAVE_GA_EVERY > 0)
  printf("Will save GenAlg every %d epochs\n", SAVE_GA_EVERY);
else
  printf("Will not save GenAlg\n");
fflush(stdout);
// Declare a variable to memorize the best value in the current epoch
float curBest = bestVal;
// Declare a variable to manage the save of GenAlg
int delaySave = 0;
while (bestVal < STOP_LEARNING_AT_VAL &&
  GAGetCurEpoch(ga) < limitEpoch) {
  // For each adn in the GenAlg
  for (int 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;
    // Display infos about the current epoch
    //printf("ep%lu ent%3d(age%6lu) val%.4f bestEpo%.4f bestAll%.4f         \r",
    //  GAGetCurEpoch(ga), iEnt, GAAdnGetAge(adn), value, curBest,
    //  bestVal);
    //fflush(stdout);
  }
  // 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);
```

```
  // If there has been improvement during this epoch
  if (curBest > bestVal) {
    bestVal = curBest;
    // Display info about the improvment
    printf("Improvement at epoch %lu: %f (in %d:%d:%d:%ds)  \n",
      GAGetCurEpoch(ga), bestVal, day, hour, min, sec);
    fflush(stdout);
    // Set the links and base functions of the NeuraNet according
    // to the best adn
    if (GABestAdnF(ga) != NULL)
      NNSetBases(nn, GABestAdnF(ga));
    if (GABestAdnI(ga) != NULL)
      NNSetLinks(nn, GABestAdnI(ga));
    // 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 {
    printf("epoch %lu (in %d:%d:%d:%ds)     \r",
      GAGetCurEpoch(ga), day, hour, min, sec);
    fflush(stdout);
  }
  ++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;
  }
}
// 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);
// 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;
}
```

learning:

```
Learning...
Will stop when curEpoch >= 500 or bestVal >= -0.010000
Best NeuraNet saved in ./bestnn.txt at each improvement
Will not save GenAlg
Improvement at epoch 1: -2.274210 (in 0:0:0:0s)
epoch 2 (in 0:0:0:0s)
epoch 3 (in 0:0:0:0s)
epoch 4 (in 0:0:0:1s)
epoch 5 (in 0:0:0:1s)
epoch 6 (in 0:0:0:1s)
epoch 7 (in 0:0:0:2s)
epoch 8 (in 0:0:0:2s)
epoch 9 (in 0:0:0:2s)
epoch 10 (in 0:0:0:3s)
epoch 11 (in 0:0:0:3s)
epoch 12 (in 0:0:0:4s)
epoch 13 (in 0:0:0:4s)
epoch 14 (in 0:0:0:4s)
epoch 15 (in 0:0:0:5s)
epoch 16 (in 0:0:0:5s)
epoch 17 (in 0:0:0:6s)
```

```
epoch 18 (in 0:0:0:6s)
Improvement at epoch 19: -2.270460 (in 0:0:0:7s)
epoch 20 (in 0:0:0:7s)
Improvement at epoch 21: -2.269608 (in 0:0:0:8s)
epoch 22 (in 0:0:0:8s)
epoch 23 (in 0:0:0:9s)
epoch 24 (in 0:0:0:9s)
Improvement at epoch 25: -2.268400 (in 0:0:0:10s)
epoch 26 (in 0:0:0:10s)
epoch 27 (in 0:0:0:11s)
epoch 28 (in 0:0:0:12s)
epoch 29 (in 0:0:0:12s)
Improvement at epoch 30: -2.265970 (in 0:0:0:13s)
Improvement at epoch 31: -2.252735 (in 0:0:0:14s)
Improvement at epoch 32: -2.231195 (in 0:0:0:14s)
Improvement at epoch 33: -2.208897 (in 0:0:0:15s)
epoch 34 (in 0:0:0:16s)
Improvement at epoch 35: -2.191270 (in 0:0:0:17s)
Improvement at epoch 36: -2.152903 (in 0:0:0:18s)
Improvement at epoch 37: -2.001125 (in 0:0:0:19s)
Improvement at epoch 38: -1.954206 (in 0:0:0:19s)
epoch 39 (in 0:0:0:20s)
epoch 40 (in 0:0:0:21s)
epoch 41 (in 0:0:0:22s)
epoch 42 (in 0:0:0:22s)
Improvement at epoch 43: -1.946620 (in 0:0:0:23s)
epoch 44 (in 0:0:0:24s)
epoch 45 (in 0:0:0:25s)
Improvement at epoch 46: -1.935281 (in 0:0:0:25s)
epoch 47 (in 0:0:0:26s)
epoch 48 (in 0:0:0:27s)
epoch 49 (in 0:0:0:28s)
epoch 50 (in 0:0:0:28s)
epoch 51 (in 0:0:0:29s)
epoch 52 (in 0:0:0:30s)
epoch 53 (in 0:0:0:31s)
epoch 54 (in 0:0:0:31s)
epoch 55 (in 0:0:0:32s)
epoch 56 (in 0:0:0:33s)
Improvement at epoch 57: -1.930344 (in 0:0:0:34s)
epoch 58 (in 0:0:0:35s)
epoch 59 (in 0:0:0:35s)
epoch 60 (in 0:0:0:36s)
Improvement at epoch 61: -1.926259 (in 0:0:0:37s)
Improvement at epoch 62: -1.921069 (in 0:0:0:38s)
epoch 63 (in 0:0:0:38s)
epoch 64 (in 0:0:0:39s)
epoch 65 (in 0:0:0:40s)
epoch 66 (in 0:0:0:41s)
Improvement at epoch 67: -1.920776 (in 0:0:0:42s)
epoch 68 (in 0:0:0:42s)
epoch 69 (in 0:0:0:43s)
epoch 70 (in 0:0:0:44s)
epoch 71 (in 0:0:0:45s)
epoch 72 (in 0:0:0:46s)
epoch 73 (in 0:0:0:46s)
Improvement at epoch 74: -1.920237 (in 0:0:0:47s)
epoch 75 (in 0:0:0:48s)
epoch 76 (in 0:0:0:49s)
epoch 77 (in 0:0:0:50s)
epoch 78 (in 0:0:0:51s)
epoch 79 (in 0:0:0:52s)
```

```
epoch 80 (in 0:0:0:52s)
epoch 81 (in 0:0:0:53s)
epoch 82 (in 0:0:0:54s)
epoch 83 (in 0:0:0:55s)
epoch 84 (in 0:0:0:56s)
epoch 85 (in 0:0:0:57s)
epoch 86 (in 0:0:0:58s)
epoch 87 (in 0:0:0:59s)
epoch 88 (in 0:0:1:0s)
epoch 89 (in 0:0:1:0s)
Improvement at epoch 90: -1.920055 (in 0:0:1:1s)
epoch 91 (in 0:0:1:2s)
epoch 92 (in 0:0:1:3s)
epoch 93 (in 0:0:1:4s)
epoch 94 (in 0:0:1:5s)
epoch 95 (in 0:0:1:6s)
epoch 96 (in 0:0:1:7s)
epoch 97 (in 0:0:1:8s)
Improvement at epoch 98: -1.919110 (in 0:0:1:9s)
epoch 99 (in 0:0:1:10s)
epoch 100 (in 0:0:1:11s)
epoch 101 (in 0:0:1:12s)
epoch 102 (in 0:0:1:13s)
Improvement at epoch 103: -1.918963 (in 0:0:1:13s)
epoch 104 (in 0:0:1:14s)
epoch 105 (in 0:0:1:15s)
epoch 106 (in 0:0:1:16s)
epoch 107 (in 0:0:1:17s)
epoch 108 (in 0:0:1:18s)
epoch 109 (in 0:0:1:19s)
epoch 110 (in 0:0:1:20s)
epoch 111 (in 0:0:1:21s)
epoch 112 (in 0:0:1:22s)
epoch 113 (in 0:0:1:23s)
epoch 114 (in 0:0:1:24s)
epoch 115 (in 0:0:1:25s)
epoch 116 (in 0:0:1:26s)
epoch 117 (in 0:0:1:27s)
epoch 118 (in 0:0:1:28s)
epoch 119 (in 0:0:1:29s)
epoch 120 (in 0:0:1:30s)
epoch 121 (in 0:0:1:31s)
Improvement at epoch 122: -1.918518 (in 0:0:1:32s)
epoch 123 (in 0:0:1:33s)
epoch 124 (in 0:0:1:34s)
epoch 125 (in 0:0:1:35s)
epoch 126 (in 0:0:1:36s)
epoch 127 (in 0:0:1:38s)
epoch 128 (in 0:0:1:39s)
epoch 129 (in 0:0:1:40s)
epoch 130 (in 0:0:1:41s)
epoch 131 (in 0:0:1:43s)
epoch 132 (in 0:0:1:44s)
Improvement at epoch 133: -1.918455 (in 0:0:1:45s)
Improvement at epoch 134: -1.918369 (in 0:0:1:47s)
epoch 135 (in 0:0:1:48s)
epoch 136 (in 0:0:1:50s)
epoch 137 (in 0:0:1:51s)
epoch 138 (in 0:0:1:53s)
epoch 139 (in 0:0:1:54s)
Improvement at epoch 140: -1.918289 (in 0:0:1:56s)
epoch 141 (in 0:0:1:57s)
```

```
epoch 142 (in 0:0:1:59s)
epoch 143 (in 0:0:2:0s)
epoch 144 (in 0:0:2:2s)
epoch 145 (in 0:0:2:4s)
epoch 146 (in 0:0:2:5s)
epoch 147 (in 0:0:2:7s)
epoch 148 (in 0:0:2:9s)
epoch 149 (in 0:0:2:10s)
epoch 150 (in 0:0:2:12s)
epoch 151 (in 0:0:2:14s)
epoch 152 (in 0:0:2:15s)
epoch 153 (in 0:0:2:17s)
epoch 154 (in 0:0:2:19s)
epoch 155 (in 0:0:2:20s)
epoch 156 (in 0:0:2:22s)
epoch 157 (in 0:0:2:24s)
epoch 158 (in 0:0:2:26s)
epoch 159 (in 0:0:2:27s)
epoch 160 (in 0:0:2:29s)
epoch 161 (in 0:0:2:31s)
epoch 162 (in 0:0:2:33s)
epoch 163 (in 0:0:2:34s)
epoch 164 (in 0:0:2:36s)
epoch 165 (in 0:0:2:38s)
epoch 166 (in 0:0:2:40s)
epoch 167 (in 0:0:2:42s)
epoch 168 (in 0:0:2:44s)
epoch 169 (in 0:0:2:46s)
epoch 170 (in 0:0:2:48s)
epoch 171 (in 0:0:2:50s)
epoch 172 (in 0:0:2:52s)
epoch 173 (in 0:0:2:54s)
Improvement at epoch 174: -1.917049 (in 0:0:2:56s)
Improvement at epoch 175: -1.916214 (in 0:0:2:58s)
Improvement at epoch 176: -1.916184 (in 0:0:3:0s)
Improvement at epoch 177: -1.914361 (in 0:0:3:3s)
Improvement at epoch 178: -1.909290 (in 0:0:3:5s)
Improvement at epoch 179: -1.907294 (in 0:0:3:7s)
epoch 180 (in 0:0:3:9s)
epoch 181 (in 0:0:3:12s)
epoch 182 (in 0:0:3:14s)
Improvement at epoch 183: -1.907138 (in 0:0:3:16s)
epoch 184 (in 0:0:3:19s)
epoch 185 (in 0:0:3:21s)
epoch 186 (in 0:0:3:24s)
epoch 187 (in 0:0:3:27s)
epoch 188 (in 0:0:3:29s)
Improvement at epoch 189: -1.897884 (in 0:0:3:32s)
Improvement at epoch 190: -1.897802 (in 0:0:3:34s)
epoch 191 (in 0:0:3:37s)
Improvement at epoch 192: -1.897552 (in 0:0:3:40s)
Improvement at epoch 193: -1.896717 (in 0:0:3:42s)
Improvement at epoch 194: -1.890286 (in 0:0:3:45s)
Improvement at epoch 195: -1.890038 (in 0:0:3:47s)
epoch 196 (in 0:0:3:50s)
epoch 197 (in 0:0:3:53s)
Improvement at epoch 198: -1.888861 (in 0:0:3:56s)
Improvement at epoch 199: -1.886302 (in 0:0:3:59s)
epoch 200 (in 0:0:4:1s)
Improvement at epoch 201: -1.885995 (in 0:0:4:3s)
Improvement at epoch 202: -1.885358 (in 0:0:4:5s)
epoch 203 (in 0:0:4:8s)
```

```
epoch 204 (in 0:0:4:10s)
epoch 205 (in 0:0:4:12s)
epoch 206 (in 0:0:4:14s)
epoch 207 (in 0:0:4:16s)
Improvement at epoch 208: -1.885347 (in 0:0:4:18s)
epoch 209 (in 0:0:4:20s)
epoch 210 (in 0:0:4:22s)
epoch 211 (in 0:0:4:25s)
epoch 212 (in 0:0:4:27s)
Improvement at epoch 213: -1.885246 (in 0:0:4:29s)
epoch 214 (in 0:0:4:31s)
epoch 215 (in 0:0:4:33s)
epoch 216 (in 0:0:4:35s)
Improvement at epoch 217: -1.885143 (in 0:0:4:37s)
Improvement at epoch 218: -1.885031 (in 0:0:4:40s)
Improvement at epoch 219: -1.885029 (in 0:0:4:42s)
epoch 220 (in 0:0:4:44s)
epoch 221 (in 0:0:4:47s)
epoch 222 (in 0:0:4:49s)
Improvement at epoch 223: -1.884518 (in 0:0:4:52s)
Improvement at epoch 224: -1.883870 (in 0:0:4:54s)
epoch 225 (in 0:0:4:57s)
Improvement at epoch 226: -1.883870 (in 0:0:5:0s)
epoch 227 (in 0:0:5:3s)
epoch 228 (in 0:0:5:5s)
epoch 229 (in 0:0:5:8s)
Improvement at epoch 230: -1.881945 (in 0:0:5:11s)
Improvement at epoch 231: -1.880978 (in 0:0:5:14s)
epoch 232 (in 0:0:5:17s)
Improvement at epoch 233: -1.880970 (in 0:0:5:19s)
Improvement at epoch 234: -1.879663 (in 0:0:5:22s)
Improvement at epoch 235: -1.879652 (in 0:0:5:25s)
Improvement at epoch 236: -1.879617 (in 0:0:5:28s)
Improvement at epoch 237: -1.879617 (in 0:0:5:31s)
epoch 238 (in 0:0:5:34s)
epoch 239 (in 0:0:5:36s)
epoch 240 (in 0:0:5:39s)
epoch 241 (in 0:0:5:42s)
epoch 242 (in 0:0:5:45s)
epoch 243 (in 0:0:5:48s)
epoch 244 (in 0:0:5:51s)
epoch 245 (in 0:0:5:53s)
Improvement at epoch 246: -1.878580 (in 0:0:5:56s)
epoch 247 (in 0:0:5:59s)
epoch 248 (in 0:0:6:2s)
epoch 249 (in 0:0:6:5s)
epoch 250 (in 0:0:6:8s)
epoch 251 (in 0:0:6:11s)
epoch 252 (in 0:0:6:14s)
Improvement at epoch 253: -1.875958 (in 0:0:6:17s)
epoch 254 (in 0:0:6:20s)
Improvement at epoch 255: -1.873699 (in 0:0:6:23s)
epoch 256 (in 0:0:6:25s)
Improvement at epoch 257: -1.867663 (in 0:0:6:28s)
Improvement at epoch 258: -1.867663 (in 0:0:6:31s)
Improvement at epoch 259: -1.858982 (in 0:0:6:33s)
Improvement at epoch 260: -1.858982 (in 0:0:6:36s)
epoch 261 (in 0:0:6:38s)
Improvement at epoch 262: -1.858916 (in 0:0:6:41s)
Improvement at epoch 263: -1.857564 (in 0:0:6:43s)
epoch 264 (in 0:0:6:45s)
Improvement at epoch 265: -1.857564 (in 0:0:6:48s)
```

```
Improvement at epoch 266: -1.856858 (in 0:0:6:51s)
epoch 267 (in 0:0:6:53s)
epoch 268 (in 0:0:6:56s)
epoch 269 (in 0:0:6:58s)
epoch 270 (in 0:0:7:0s)
epoch 271 (in 0:0:7:3s)
epoch 272 (in 0:0:7:5s)
epoch 273 (in 0:0:7:8s)
epoch 274 (in 0:0:7:10s)
epoch 275 (in 0:0:7:13s)
epoch 276 (in 0:0:7:15s)
epoch 277 (in 0:0:7:17s)
epoch 278 (in 0:0:7:20s)
Improvement at epoch 279: -1.856853 (in 0:0:7:22s)
epoch 280 (in 0:0:7:25s)
epoch 281 (in 0:0:7:27s)
Improvement at epoch 282: -1.856812 (in 0:0:7:30s)
epoch 283 (in 0:0:7:32s)
epoch 284 (in 0:0:7:35s)
epoch 285 (in 0:0:7:37s)
epoch 286 (in 0:0:7:40s)
epoch 287 (in 0:0:7:43s)
Improvement at epoch 288: -1.856737 (in 0:0:7:45s)
epoch 289 (in 0:0:7:48s)
Improvement at epoch 290: -1.856680 (in 0:0:7:51s)
epoch 291 (in 0:0:7:53s)
Improvement at epoch 292: -1.856669 (in 0:0:7:56s)
epoch 293 (in 0:0:7:59s)
epoch 294 (in 0:0:8:1s)
Improvement at epoch 295: -1.856112 (in 0:0:8:4s)
epoch 296 (in 0:0:8:7s)
Improvement at epoch 297: -1.855945 (in 0:0:8:9s)
Improvement at epoch 298: -1.855619 (in 0:0:8:11s)
Improvement at epoch 299: -1.855411 (in 0:0:8:14s)
epoch 300 (in 0:0:8:16s)
epoch 301 (in 0:0:8:19s)
epoch 302 (in 0:0:8:21s)
epoch 303 (in 0:0:8:24s)
Improvement at epoch 304: -1.855353 (in 0:0:8:26s)
epoch 305 (in 0:0:8:29s)
epoch 306 (in 0:0:8:31s)
epoch 307 (in 0:0:8:33s)
Improvement at epoch 308: -1.855281 (in 0:0:8:35s)
epoch 309 (in 0:0:8:38s)
epoch 310 (in 0:0:8:40s)
epoch 311 (in 0:0:8:42s)
epoch 312 (in 0:0:8:45s)
Improvement at epoch 313: -1.854960 (in 0:0:8:47s)
Improvement at epoch 314: -1.853391 (in 0:0:8:50s)
epoch 315 (in 0:0:8:52s)
epoch 316 (in 0:0:8:55s)
epoch 317 (in 0:0:8:57s)
Improvement at epoch 318: -1.851903 (in 0:0:9:0s)
epoch 319 (in 0:0:9:2s)
epoch 320 (in 0:0:9:4s)
Improvement at epoch 321: -1.851760 (in 0:0:9:6s)
Improvement at epoch 322: -1.851756 (in 0:0:9:9s)
epoch 323 (in 0:0:9:11s)
epoch 324 (in 0:0:9:13s)
Improvement at epoch 325: -1.851628 (in 0:0:9:16s)
Improvement at epoch 326: -1.850918 (in 0:0:9:18s)
epoch 327 (in 0:0:9:20s)
```

```
Improvement at epoch 328: -1.850391 (in 0:0:9:23s)
Improvement at epoch 329: -1.850352 (in 0:0:9:25s)
Improvement at epoch 330: -1.850336 (in 0:0:9:28s)
Improvement at epoch 331: -1.848890 (in 0:0:9:31s)
epoch 332 (in 0:0:9:33s)
epoch 333 (in 0:0:9:36s)
epoch 334 (in 0:0:9:38s)
epoch 335 (in 0:0:9:41s)
epoch 336 (in 0:0:9:44s)
epoch 337 (in 0:0:9:46s)
epoch 338 (in 0:0:9:49s)
epoch 339 (in 0:0:9:51s)
epoch 340 (in 0:0:9:54s)
epoch 341 (in 0:0:9:57s)
epoch 342 (in 0:0:9:59s)
epoch 343 (in 0:0:10:2s)
epoch 344 (in 0:0:10:5s)
epoch 345 (in 0:0:10:7s)
epoch 346 (in 0:0:10:10s)
epoch 347 (in 0:0:10:13s)
epoch 348 (in 0:0:10:15s)
epoch 349 (in 0:0:10:18s)
epoch 350 (in 0:0:10:21s)
epoch 351 (in 0:0:10:23s)
Improvement at epoch 352: -1.848323 (in 0:0:10:26s)
epoch 353 (in 0:0:10:29s)
epoch 354 (in 0:0:10:31s)
epoch 355 (in 0:0:10:34s)
epoch 356 (in 0:0:10:37s)
epoch 357 (in 0:0:10:40s)
epoch 358 (in 0:0:10:42s)
epoch 359 (in 0:0:10:45s)
epoch 360 (in 0:0:10:48s)
epoch 361 (in 0:0:10:50s)
epoch 362 (in 0:0:10:53s)
epoch 363 (in 0:0:10:56s)
epoch 364 (in 0:0:10:58s)
Improvement at epoch 365: -1.847722 (in 0:0:11:1s)
epoch 366 (in 0:0:11:4s)
Improvement at epoch 367: -1.847674 (in 0:0:11:7s)
epoch 368 (in 0:0:11:9s)
Improvement at epoch 369: -1.847639 (in 0:0:11:12s)
Improvement at epoch 370: -1.847514 (in 0:0:11:15s)
epoch 371 (in 0:0:11:18s)
Improvement at epoch 372: -1.847437 (in 0:0:11:21s)
Improvement at epoch 373: -1.847405 (in 0:0:11:23s)
Improvement at epoch 374: -1.847137 (in 0:0:11:26s)
Improvement at epoch 375: -1.845598 (in 0:0:11:29s)
epoch 376 (in 0:0:11:32s)
Improvement at epoch 377: -1.845489 (in 0:0:11:35s)
epoch 378 (in 0:0:11:38s)
Improvement at epoch 379: -1.839827 (in 0:0:11:41s)
Improvement at epoch 380: -1.838284 (in 0:0:11:44s)
Improvement at epoch 381: -1.831431 (in 0:0:11:46s)
Improvement at epoch 382: -1.829923 (in 0:0:11:49s)
Improvement at epoch 383: -1.823335 (in 0:0:11:52s)
epoch 384 (in 0:0:11:54s)
Improvement at epoch 385: -1.822885 (in 0:0:11:57s)
epoch 386 (in 0:0:11:59s)
Improvement at epoch 387: -1.822437 (in 0:0:12:2s)
Improvement at epoch 388: -1.819679 (in 0:0:12:4s)
Improvement at epoch 389: -1.818805 (in 0:0:12:7s)
```

```
Improvement at epoch 390: -1.813942 (in 0:0:12:9s)
epoch 391 (in 0:0:12:11s)
epoch 392 (in 0:0:12:14s)
epoch 393 (in 0:0:12:16s)
epoch 394 (in 0:0:12:19s)
Improvement at epoch 395: -1.812989 (in 0:0:12:21s)
epoch 396 (in 0:0:12:23s)
Improvement at epoch 397: -1.812355 (in 0:0:12:26s)
epoch 398 (in 0:0:12:28s)
Improvement at epoch 399: -1.799787 (in 0:0:12:31s)
epoch 400 (in 0:0:12:33s)
Improvement at epoch 401: -1.798370 (in 0:0:12:35s)
Improvement at epoch 402: -1.786153 (in 0:0:12:38s)
Improvement at epoch 403: -1.778480 (in 0:0:12:40s)
Improvement at epoch 404: -1.770116 (in 0:0:12:43s)
Improvement at epoch 405: -1.768147 (in 0:0:12:46s)
epoch 406 (in 0:0:12:48s)
Improvement at epoch 407: -1.761949 (in 0:0:12:51s)
epoch 408 (in 0:0:12:53s)
Improvement at epoch 409: -1.759806 (in 0:0:12:56s)
Improvement at epoch 410: -1.754465 (in 0:0:12:58s)
epoch 411 (in 0:0:13:1s)
epoch 412 (in 0:0:13:4s)
epoch 413 (in 0:0:13:7s)
Improvement at epoch 414: -1.754370 (in 0:0:13:10s)
Improvement at epoch 415: -1.753957 (in 0:0:13:13s)
Improvement at epoch 416: -1.752482 (in 0:0:13:16s)
epoch 417 (in 0:0:13:19s)
Improvement at epoch 418: -1.750418 (in 0:0:13:22s)
Improvement at epoch 419: -1.749863 (in 0:0:13:25s)
epoch 420 (in 0:0:13:28s)
epoch 421 (in 0:0:13:31s)
Improvement at epoch 422: -1.749056 (in 0:0:13:33s)
Improvement at epoch 423: -1.745777 (in 0:0:13:36s)
Improvement at epoch 424: -1.743961 (in 0:0:13:39s)
epoch 425 (in 0:0:13:42s)
epoch 426 (in 0:0:13:45s)
epoch 427 (in 0:0:13:48s)
Improvement at epoch 428: -1.743861 (in 0:0:13:50s)
Improvement at epoch 429: -1.741271 (in 0:0:13:53s)
Improvement at epoch 430: -1.726061 (in 0:0:13:56s)
Improvement at epoch 431: -1.714161 (in 0:0:13:59s)
Improvement at epoch 432: -1.709199 (in 0:0:14:2s)
Improvement at epoch 433: -1.702927 (in 0:0:14:5s)
epoch 434 (in 0:0:14:8s)
Improvement at epoch 435: -1.700296 (in 0:0:14:11s)
Improvement at epoch 436: -1.700281 (in 0:0:14:14s)
epoch 437 (in 0:0:14:17s)
epoch 438 (in 0:0:14:19s)
Improvement at epoch 439: -1.698429 (in 0:0:14:22s)
Improvement at epoch 440: -1.690329 (in 0:0:14:25s)
epoch 441 (in 0:0:14:28s)
epoch 442 (in 0:0:14:31s)
Improvement at epoch 443: -1.688173 (in 0:0:14:34s)
epoch 444 (in 0:0:14:37s)
epoch 445 (in 0:0:14:39s)
Improvement at epoch 446: -1.684357 (in 0:0:14:42s)
Improvement at epoch 447: -1.664089 (in 0:0:14:45s)
epoch 448 (in 0:0:14:48s)
epoch 449 (in 0:0:14:51s)
epoch 450 (in 0:0:14:54s)
Improvement at epoch 451: -1.661744 (in 0:0:14:56s)
```

```
Improvement at epoch 452: -1.659153 (in 0:0:14:59s)
epoch 453 (in 0:0:15:2s)
epoch 454 (in 0:0:15:4s)
epoch 455 (in 0:0:15:7s)
epoch 456 (in 0:0:15:10s)
Improvement at epoch 457: -1.656781 (in 0:0:15:12s)
epoch 458 (in 0:0:15:15s)
epoch 459 (in 0:0:15:18s)
epoch 460 (in 0:0:15:20s)
epoch 461 (in 0:0:15:23s)
epoch 462 (in 0:0:15:26s)
epoch 463 (in 0:0:15:29s)
epoch 464 (in 0:0:15:31s)
epoch 465 (in 0:0:15:34s)
epoch 466 (in 0:0:15:37s)
epoch 467 (in 0:0:15:39s)
epoch 468 (in 0:0:15:42s)
Improvement at epoch 469: -1.651396 (in 0:0:15:45s)
epoch 470 (in 0:0:15:47s)
epoch 471 (in 0:0:15:50s)
epoch 472 (in 0:0:15:53s)
Improvement at epoch 473: -1.645951 (in 0:0:15:56s)
epoch 474 (in 0:0:15:58s)
epoch 475 (in 0:0:16:1s)
epoch 476 (in 0:0:16:4s)
Improvement at epoch 477: -1.645103 (in 0:0:16:7s)
epoch 478 (in 0:0:16:9s)
epoch 479 (in 0:0:16:12s)
epoch 480 (in 0:0:16:15s)
Improvement at epoch 481: -1.644703 (in 0:0:16:18s)
Improvement at epoch 482: -1.643883 (in 0:0:16:21s)
Improvement at epoch 483: -1.643422 (in 0:0:16:23s)
epoch 484 (in 0:0:16:26s)
epoch 485 (in 0:0:16:29s)
epoch 486 (in 0:0:16:32s)
epoch 487 (in 0:0:16:35s)
Improvement at epoch 488: -1.643393 (in 0:0:16:37s)
Improvement at epoch 489: -1.640155 (in 0:0:16:40s)
epoch 490 (in 0:0:16:43s)
epoch 491 (in 0:0:16:46s)
epoch 492 (in 0:0:16:48s)
epoch 493 (in 0:0:16:51s)
epoch 494 (in 0:0:16:54s)
epoch 495 (in 0:0:16:57s)
epoch 496 (in 0:0:17:0s)
epoch 497 (in 0:0:17:2s)
Improvement at epoch 498: -1.638433 (in 0:0:17:5s)
epoch 499 (in 0:0:17:8s)
epoch 500 (in 0:0:17:11s)
Learning complete (in 0:0:17:11s)
```

validation:

```
age_err count cumul_perc
 0   274 0.232795
 1   458 0.621920
 2   249 0.833475
 3   100 0.918437
 4    29 0.943076
 5    27 0.966015
 6    15 0.978760
```

```
 7   13 0.989805
 8    7 0.995752
 9    2 0.997451
10    2 0.999150
11    1 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: -1.542048
```