NeuraMorph

P. Baillehache

September 13, 2020

Contents

Introduction

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

It uses the `PBErr`, `PBMath`, `GSet` library.

1 Definitions

2 Interface

```
// ============ NEURAMORPH.H ================
```

```
#ifndef NEURAMORPH_H
#define NEURAMORPH_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"
#include "gdataset.h"
#include "bcurve.h"

// ----- NeuraMorphUnit

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

typedef struct NeuraMorphUnit {

  // Input indices in parent NeuraMorph
  VecLong* iInputs;

  // Output indices in parent NeuraMorph
  VecLong* iOutputs;

  // Lowest and highest values for filtering inputs
  VecFloat* lowFilters;
  VecFloat* highFilters;

  // Lowest and highest values of outputs
  VecFloat* lowOutputs;
  VecFloat* highOutputs;

  // Vector to memorize the output values
  VecFloat* outputs;

  // Transfer function
  BBody* transfer;

  // Working variable to avoid reallocation of memory at each Evaluate()
  VecFloat* unitInputs;

  // Variable to memorize the value of the unit during training
  float value;

} NeuraMorphUnit;

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

// Create a new NeuraMorphUnit between the input 'iInputs' and the
// outputs 'iOutputs'
NeuraMorphUnit* NeuraMorphUnitCreate(
  const VecLong* iInputs,
  const VecLong* iOutputs);

// Free the memory used by the NeuraMorphUnit 'that'
void NeuraMorphUnitFree(NeuraMorphUnit** that);

// Get the input indices of the NeuraMorphUnit 'that'
```

```
#if BUILDMODE != 0
static inline
#endif
const VecLong* NMUnitIInputs(const NeuraMorphUnit* that);

// Get the output indices of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
const VecLong* NMUnitIOutputs(const NeuraMorphUnit* that);

// Get the output values of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMUnitOutputs(const NeuraMorphUnit* that);

// Calculate the outputs for the 'inputs' with the NeuraMorphUnit 'that'
// Update 'that->outputs'
void NMUnitEvaluate(
  NeuraMorphUnit* that,
  const VecFloat* inputs);

// Get the number of input values of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
long NMUnitGetNbInputs(const NeuraMorphUnit* that);

// Get the number of output values of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
long NMUnitGetNbOutputs(const NeuraMorphUnit* that);

// Get the number of coefficients in the transfer function of
// the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
long NMUnitGetNbCoeffs(const NeuraMorphUnit* that);

// Get the value of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
float NMUnitGetValue(const NeuraMorphUnit* that);

// Set the value of the NeuraMorphUnit 'that' to 'val'
#if BUILDMODE != 0
static inline
#endif
void NMUnitSetValue(
  NeuraMorphUnit* that,
            float val);

// Print the NeuraMorphUnit 'that' on the 'stream'
void NMUnitPrint(
  const NeuraMorphUnit* that,
                 FILE* stream);
#define NMUnitPrintln(T, S) \
  do {NMUnitPrint(T, S);fprintf(S, "\n");} while (false)
```

```
// ----- NeuraMorph

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

typedef struct NeuraMorph {

  // Number of inputs and outputs
  long nbInput;
  long nbOutput;

  // Inputs and outputs values
  VecFloat* inputs;
  VecFloat* outputs;

  // Internal values
  VecFloat* hiddens;

  // Lowest and highest values for internal values
  VecFloat* lowHiddens;
  VecFloat* highHiddens;

  // Flag to memorize if the outputs are to be seen as one hot encoding
  bool flagOneHot;

  // GSet of NeuraMorphUnit
  GSet units;

} NeuraMorph;

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

// Create a new NeuraMorph with 'nbInput' inputs and 'nbOutput' outputs
NeuraMorph* NeuraMorphCreate(
  long nbInput,
  long nbOutput);

// Free the memory used by the NeuraMorph 'that'
void NeuraMorphFree(NeuraMorph** that);

// Get the number of input values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
long NMGetNbInput(const NeuraMorph* that);

// Get the number of output values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
long NMGetNbOutput(const NeuraMorph* that);

// Get the input values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
VecFloat* NMInputs(NeuraMorph* that);

// Get the output values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
```

```
const VecFloat* NMOutputs(const NeuraMorph* that);

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

// Get the lowest bound of hidden values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMLowHiddens(const NeuraMorph* that);

// Get the highest bound of hidden values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMHighHiddens(const NeuraMorph* that);

// Get the number of hidden values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
long NMGetNbHidden(const NeuraMorph* that);

// Set the number of hidden values of the NeuraMorph 'that' to 'nb'
#if BUILDMODE != 0
static inline
#endif
void NMSetNbHidden(
  NeuraMorph* that,
         long nb);

// Get the flag for one hot encoding of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
bool NMGetFlagOneHot(const NeuraMorph* that);

// Set the flag for one hot encoding of the NeuraMorph 'that' to 'flag'
#if BUILDMODE != 0
static inline
#endif
void NMSetFlagOneHot(
  NeuraMorph* that,
         bool flag);

// Add one NeuraMorphUnit with input and output indices 'iInputs'
// and 'iOutputs' to the NeuraMorph 'that'
// Return the created NeuraMorphUnit
NeuraMorphUnit* NMAddUnit(
     NeuraMorph* that,
  const VecLong* iInputs,
  const VecLong* iOutputs);

// Remove the NeuraMorphUnit 'unit' from the NeuraMorph 'that'
// The NeuraMorphUnit is not freed
void NMRemoveUnit(
      NeuraMorph* that,
  NeuraMorphUnit* unit);
```

```
// Burry the NeuraMorphUnits in the 'units' set into the
// NeuraMorph 'that'
// 'units' is empty after calling this function
// The NeuraMorphUnits iOutputs must point toward the NeuraMorph
// outputs
// NeuraMorphUnits' iOutputs are redirected toward new hidden values
// 'that->hiddens' is resized as necessary
void NMBurryUnits(
  NeuraMorph* that,
        GSet* units);

// Get a new vector with indices of the outputs in the NeuraMorph 'that'
VecLong* NMGetVecIOutputs(const NeuraMorph* that);

// Evaluate the NeuraMorph 'that' on the 'inputs' values
void NMEvaluate(
  NeuraMorph* that,
    VecFloat* inputs);

// ----- NeuraMorphTrainer

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

typedef struct NeuraMorphTrainer {

  // Trained NeuraMorph
  NeuraMorph* neuraMorph;

  // Training dataset
  GDataSetVecFloat* dataset;

  // Index of the dataset's category used for training and evaluation
  unsigned int iCatTraining;
  unsigned int iCatEval;

  // Depth of the training
  short depth;

  // Order of the transfer function of NeuraMorphUnit
  int order;

  // Maximum number of inputs per NeuraMorphUnit
  int nbMaxInputsUnit;

  // Threshold used to discard weakest units during training
  // in [0.0,1.0]
  float weakUnitThreshold;

  // Maximum number of unit kept at each depth
  int nbMaxUnitDepth;

  // Max level of division of values' range
  short maxLvlDiv;

  // Precomputed values to train the NeuraMorphUnit
  VecFloat** preCompInp;
  VecFloat** preCompOut;

  // Lowest and highest values for input values in the training
  // dataset
  VecFloat* lowInputs;
  VecFloat* highInputs;
```

```
  // Variable to store the result of the last evaluation
  VecFloat3D resEval;

} NeuraMorphTrainer;

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

// Create a static NeuraMorphTrainer for the NeuraMorph 'neuraMorph' and the
// GDataSet 'dataset'
// Default depth: 2
// Default iCatTraining: 0
// Default weakUnitThreshold: 0.9
NeuraMorphTrainer NeuraMorphTrainerCreateStatic(
        NeuraMorph* neuraMorph,
  GDataSetVecFloat* dataset);

// Free the memory used by the static NeuraMorphTrainer 'that'
void NeuraMorphTrainerFreeStatic(NeuraMorphTrainer* that);

// Run the training process for the NeuraMorphTrainer 'that'
void NMTrainerRun(NeuraMorphTrainer* that);

// Run the evaluation process for the NeuraMorphTrainer 'that'
void NMTrainerEval(NeuraMorphTrainer* that);

// Get the depth of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
short NMTrainerGetDepth(const NeuraMorphTrainer* that);

// Set the depth of the NeuraMorphTrainer 'that' to 'depth'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetDepth(
  NeuraMorphTrainer* that,
               short depth);

// Get the maxLvlDiv of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
short NMTrainerGetMaxLvlDiv(const NeuraMorphTrainer* that);

// Set the maxLvlDiv of the NeuraMorphTrainer 'that' to 'lvl'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetMaxLvlDiv(
  NeuraMorphTrainer* that,
               short lvl);

// Get the order of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
int NMTrainerGetOrder(const NeuraMorphTrainer* that);

// Set the order of the NeuraMorphTrainer 'that' to 'order'
#if BUILDMODE != 0
```

```
static inline
#endif
void NMTrainerSetOrder(
  NeuraMorphTrainer* that,
                int order);

// Get the nbMaxUnitDepth of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
int NMTrainerGetNbMaxUnitDepth(const NeuraMorphTrainer* that);

// Set the nbMaxUnitDepth of the NeuraMorphTrainer 'that' to 'nbMaxUnitDepth'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetNbMaxUnitDepth(
  NeuraMorphTrainer* that,
                int nbMaxUnitDepth);

// Get the nbMaxInputsUnit of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
int NMTrainerGetNbMaxInputsUnit(const NeuraMorphTrainer* that);

// Set the nbMaxInputsUnit of the NeuraMorphTrainer 'that' to 'order'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetNbMaxInputsUnit(
  NeuraMorphTrainer* that,
                int nbMaxInputsUnit);

// Get the weakness threshold of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
float NMTrainerGetWeakThreshold(const NeuraMorphTrainer* that);

// Set the weakness threshold of the NeuraMorphTrainer 'that'
// to 'iCat'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetWeakThreshold(
  NeuraMorphTrainer* that,
              float weakUnitThreshold);

// Get the index of the training category of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int NMTrainerGetICatTraining(const NeuraMorphTrainer* that);

// Set the index of the training category of the NeuraMorphTrainer 'that'
// to 'iCat'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetICatTraining(
  NeuraMorphTrainer* that,
```

```
        unsigned int iCatTraining);

// Get the index of the evaluation category of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int NMTrainerGetICatEval(const NeuraMorphTrainer* that);

// Set the index of the evaluation category of the NeuraMorphTrainer 'that'
// to 'iCat'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetICatEval(
  NeuraMorphTrainer* that,
        unsigned int iCatEval);

// Get the NeuraMorph of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
NeuraMorph* NMTrainerNeuraMorph(const NeuraMorphTrainer* that);

// Get the GDataSet of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
GDataSetVecFloat* NMTrainerDataset(const NeuraMorphTrainer* that);

// Get the result of the last evaluation of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat3D* NMTrainerResEval(const NeuraMorphTrainer* that);

// ================ static inliner ====================

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

#endif
```

3 Code

3.1 neuramorph.c

```
// ============ NEURAMORPH.C ================

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

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

// ----- NeuraMorphUnit
```

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

// Update the low and high of the hiddens of the NeuraMorph 'that' with
// the low and high of its units
void NMUpdateLowHighHiddens(NeuraMorph* that);

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

// Create a new NeuraMorphUnit between the input 'iInputs' and the
// outputs 'iOutputs'
NeuraMorphUnit* NeuraMorphUnitCreate(
  const VecLong* iInputs,
  const VecLong* iOutputs) {

#if BUILDMODE == 0

  if (iInputs == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'iInputs' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (iOutputs == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'iOutputs' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Allocate memory for the NeuraMorphUnit
  NeuraMorphUnit* that =
    PBErrMalloc(
      NeuraMorphErr,
      sizeof(NeuraMorphUnit));

  // Get the number of inputs (including the constant) and outputs
  long nbIn = VecGetDim(iInputs);
  long nbOut = VecGetDim(iOutputs);

  // Init properties
  that->iInputs = VecClone(iInputs);
  that->iOutputs = VecClone(iOutputs);
  that->lowFilters = VecFloatCreate(nbIn);
  that->highFilters = VecFloatCreate(nbIn);
  that->lowOutputs = NULL;
  that->highOutputs = NULL;
  that->outputs = VecFloatCreate(nbOut);
  VecShort2D dim = VecShortCreateStatic2D();
  VecSet(
    &dim,
    0,
    nbIn);
  VecSet(
```

```
    &dim,
    1,
    nbOut);
  that->transfer = NULL;
  that->unitInputs = VecFloatCreate(nbIn);
  that->value = 0.0;

  // Return the new NeuraMorphUnit
  return that;

}

// Free the memory used by the NeuraMorphUnit 'that'
void NeuraMorphUnitFree(NeuraMorphUnit** that) {

  // Check the input
  if (that == NULL || *that == NULL) {

    return;

  }

  // Free memory
  VecFree(&((*that)->iInputs));
  VecFree(&((*that)->iOutputs));
  VecFree(&((*that)->lowFilters));
  VecFree(&((*that)->highFilters));
  if ((*that)->lowOutputs != NULL) {

    VecFree(&((*that)->lowOutputs));

  }

  if ((*that)->highOutputs != NULL) {

    VecFree(&((*that)->highOutputs));

  }

  VecFree(&((*that)->outputs));
  BBodyFree(&((*that)->transfer));
  VecFree(&((*that)->unitInputs));
  free(*that);
  *that = NULL;

}

// Calculate the outputs for the 'inputs' with the NeuraMorphUnit 'that'
// Update 'that->outputs'
void NMUnitEvaluate(
  NeuraMorphUnit* that,
  const VecFloat* inputs) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);
```

```
  }

  if (VecGetDim(inputs) != VecGetDim(that->iInputs)) {

    NeuraMorphErr->_type = PBErrTypeInvalidArg;
    sprintf(
      NeuraMorphErr->_msg,
      "'inputs' has invalid dimension (%ld!=%ld)",
      VecGetDim(inputs),
      VecGetDim(that->iInputs));
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Reset the outputs
  VecFree(&(that->outputs));

  // Update the scaled inputs
  for (
    long iInput = 0;
    iInput < VecGetDim(that->unitInputs);
    ++iInput) {

    // Get the input value and its low/high filters
    float val =
      VecGet(
        inputs,
        iInput);
    float low =
      VecGet(
        that->lowFilters,
        iInput);
    float high =
      VecGet(
        that->highFilters,
        iInput);

    // Set the value in the unit inputs
    VecSet(
      that->unitInputs,
      iInput,
      (val - low) / (high - low));

  }

  // Apply the transfer function
  that->outputs =
    BBodyGet(
      that->transfer,
      that->unitInputs);

  // If the low and high values for outputs don't exist yet
  if (that->lowOutputs == NULL) {

    // Create the low and high values by cloning the current output
    that->lowOutputs = VecClone(that->outputs);
    that->highOutputs = VecClone(that->outputs);

  // Else, the low and high values for outputs exist
```

```
  } else {

    // Loop on the outputs
    for (
      long iOutput = 0;
      iOutput < VecGetDim(that->outputs);
      ++iOutput) {

      // Update the low and high values for this output
      float val =
        VecGet(
          that->outputs,
          iOutput);

      float curLow =
        VecGet(
          that->lowOutputs,
          iOutput);
      if (curLow > val) {

        VecSet(
          that->lowOutputs,
          iOutput,
          val);

      }

      float curHigh =
        VecGet(
          that->highOutputs,
          iOutput);
      if (curHigh < val) {

        VecSet(
          that->highOutputs,
          iOutput,
          val);

      }

    }

  }

}

// Print the NeuraMorphUnit 'that' on the 'stream'
void NMUnitPrint(
  const NeuraMorphUnit* that,
                 FILE* stream) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }
```

```
  if (stream == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'stream' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  VecPrint(
    NMUnitIInputs(that),
    stream);
  fprintf(
    stream,
    " -> ");
  VecPrint(
    NMUnitIOutputs(that),
    stream);
  fprintf(
    stream,
    " (%04.6f)",
    NMUnitGetValue(that));

}

// ----- NeuraMorph

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

// Create a new NeuraMorph with 'nbInput' inputs and 'nbOutput' outputs
NeuraMorph* NeuraMorphCreate(
  long nbInput,
  long nbOutput) {

  // Allocate memory for the NeuraMorph
  NeuraMorph* that =
    PBErrMalloc(
      NeuraMorphErr,
      sizeof(NeuraMorph));

  // Init properties
  that->nbInput = nbInput;
  that->nbOutput = nbOutput;
  that->inputs = VecFloatCreate(nbInput);
  that->outputs = VecFloatCreate(nbOutput);
  that->hiddens = NULL;
  that->lowHiddens = NULL;
  that->highHiddens = NULL;
  that->units = GSetCreateStatic();
  that->flagOneHot = false;

  // Return the NeuraMorph
  return that;

}

// Free the memory used by the NeuraMorph 'that'
void NeuraMorphFree(NeuraMorph** that) {
```

```
  // Check the input
  if (that == NULL || *that == NULL) {

    return;

  }

  // Free memory
  VecFree(&((*that)->inputs));
  VecFree(&((*that)->outputs));
  if ((*that)->hiddens != NULL) {

    VecFree(&((*that)->hiddens));
    VecFree(&((*that)->lowHiddens));
    VecFree(&((*that)->highHiddens));

  }

  while (GSetNbElem(&((*that)->units)) > 0) {

    NeuraMorphUnit* unit = GSetPop(&((*that)->units));
    NeuraMorphUnitFree(&unit);

  }

  free(*that);
  *that = NULL;

}

// Add one NeuraMorphUnit with input and output indices 'iInputs'
// and 'iOutputs' to the NeuraMorph 'that'
// Return the created NeuraMorphUnit
NeuraMorphUnit* NMAddUnit(
     NeuraMorph* that,
  const VecLong* iInputs,
  const VecLong* iOutputs) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (iInputs == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'iInputs' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (iOutputs == NULL) {
```

```
    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'iOutputs' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Create the NeuraMorphUnit
  NeuraMorphUnit* unit =
    NeuraMorphUnitCreate(
      iInputs,
      iOutputs);

  // Append the new NeuraorphUnit to the set of NeuraMorphUnit
  GSetAppend(
    &(that->units),
    unit);

  // Return the new unit
  return unit;

}

// Remove the NeuraMorphUnit 'unit' from the NeuraMorph 'that'
// The NeuraMorphUnit is not freed
void NMRemoveUnit(
      NeuraMorph* that,
  NeuraMorphUnit* unit) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Remove the NeuraorphUnit from the set of NeuraMorphUnit
  GSetRemoveAll(
    &(that->units),
    unit);

}

// Burry the NeuraMorphUnits in the 'units' set into the
// NeuraMorph 'that'
// 'units' is empty after calling this function
// The NeuraMorphUnits iOutputs must point toward the NeuraMorph
// outputs
// NeuraMorphUnits' iOutputs are redirected toward new hidden values
// 'that->hiddens' is resized as necessary
void NMBurryUnits(
```

```
  NeuraMorph* that,
        GSet* units) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Declare a variable to memorize the number of hidden values
  // to add
  long nbHiddenValues = 0;

  // While there are units to burry
  while (GSetNbElem(units) > 0) {

    // Get the unit
    NeuraMorphUnit* unit = GSetPop(units);

    // Loop on the iOutputs of the unit
    for (
      long iOutput = 0;
      iOutput < VecGetDim(NMUnitIOutputs(unit));
      ++iOutput) {

      long indice =
        VecGet(
          NMUnitIOutputs(unit),
          iOutput);
      VecSet(
        unit->iOutputs,
        iOutput,
        indice + nbHiddenValues);

    }

    // Append the unit to the set of NeuraMorphUnit
    GSetAppend(
      &(that->units),
      unit);

    // Update the number of new hidden values
    nbHiddenValues += VecGetDim(NMUnitIOutputs(unit));

  }

  // If there is already hidden values
  if (that->hiddens != NULL) {

    // Add the previous number of hidden values
    nbHiddenValues += VecGetDim(that->hiddens);

    // Free memory
    VecFree(&(that->hiddens));
```

```
    VecFree(&(that->lowHiddens));
    VecFree(&(that->highHiddens));

  }

  // If there are hidden values after burrying
  if (nbHiddenValues > 0) {

    // Resize the hiddens value vector
    that->hiddens = VecFloatCreate(nbHiddenValues);
    that->lowHiddens = VecFloatCreate(nbHiddenValues);
    that->highHiddens = VecFloatCreate(nbHiddenValues);

    // Update the low and high of the hiddens with the low and high
    // of the units
    NMUpdateLowHighHiddens(that);

  }

}

// Get a new vector with indices of the outputs in the NeuraMorph 'that'
VecLong* NMGetVecIOutputs(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Allocate memory for the result
  VecLong* iOutputs = VecLongCreate(NMGetNbOutput(that));

  // Loop on indices
  for (
    long iOutput = 0;
    iOutput < NMGetNbOutput(that);
    ++iOutput) {

    // Set the indice of this output
    VecSet(
      iOutputs,
      iOutput,
      iOutput + NMGetNbHidden(that));

  }

  // Return the result
  return iOutputs;

}

// Update the low and high of the hiddens of the NeuraMorph 'that' with
// the low and high of its units
```

```
void NMUpdateLowHighHiddens(NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Loop on the units
  GSetIterForward iter =
    GSetIterForwardCreateStatic(&(that->units));
  do {

    // Get the unit
    NeuraMorphUnit* unit = GSetIterGet(&iter);

    // Loop on the iOutputs of the unit
    for (
      long iOutput = 0;
      iOutput < VecGetDim(NMUnitIOutputs(unit));
      ++iOutput) {

      // Get the indice
      long indice =
        VecGet(
          NMUnitIOutputs(unit),
          iOutput);

      // If the indice points to a hidden value
      if (indice < NMGetNbHidden(that)) {

        // If the low and high exist
        if (
          unit->lowOutputs != NULL &&
          unit->highOutputs != NULL) {

          // Update the low and high
          float low =
            VecGet(
              unit->lowOutputs,
              iOutput);
          float high =
            VecGet(
              unit->highOutputs,
              iOutput);
          VecSet(
            that->lowHiddens,
            indice,
            low);
          VecSet(
            that->highHiddens,
            indice,
            high);
```

```
        }

      }

    }

  } while (GSetIterStep(&iter));

}

// Evaluate the NeuraMorph 'that' on the 'inputs' values
void NMEvaluate(
  NeuraMorph* that,
    VecFloat* inputs) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (inputs == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'inputs' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (VecGetDim(inputs) != VecGetDim(that->inputs)) {

    NeuraMorphErr->_type = PBErrTypeInvalidArg;
    sprintf(
      NeuraMorphErr->_msg,
      "'inputs' has invalid size (%ld==%ld)",
      VecGetDim(inputs),
      VecGetDim(that->inputs));
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Copy the inputs into the internal inputs
  VecCopy(
    that->inputs,
    inputs);

  // Reset the internal outputs
  VecSetNull(that->outputs);

  // If there are no units
  if (GSetNbElem(&(that->units)) == 0) {
```

```
  // Nothing else to do
  return;

}

// Loop on the units
GSetIterForward iter = GSetIterForwardCreateStatic(&(that->units));
do {

  // Get the unit
  NeuraMorphUnit* unit = GSetIterGet(&iter);

  // Allocate memory for inputs sent to the unit
  VecFloat* unitInputs = VecFloatCreate(NMUnitGetNbInputs(unit));

  // Loop on the input indices of the unit
  for (
    long iInput = 0;
    iInput < NMUnitGetNbInputs(unit);
    ++iInput) {

    // Get the input indice
    long indiceInput =
      VecGet(
        NMUnitIInputs(unit),
        iInput);

    // Declare a variable to memorize the input value
    float val = 0.0;

    // If this indice points toward an input
    if (indiceInput < NMGetNbInput(that)) {

      // Get the input value of the NeuraMorph for this indice
      val =
        VecGet(
          NMInputs(that),
          indiceInput);

    // Else, the indice points toward a hidden value
    } else {

      // Get the hidden value of the NeuraMorph for this indice
      val =
        VecGet(
          that->hiddens,
          indiceInput - NMGetNbInput(that));

    }

    // Set the input value for the unit for this indice
    VecSet(
      unitInputs,
      iInput,
      val);

  }

  // Evaluate the unit
  NMUnitEvaluate(
    unit,
    unitInputs);
```

```
  // Free the memory used by the unit input
  VecFree(&unitInputs);

  // Loop on the output indices of the unit
  for (
    long iOutput = 0;
    iOutput < NMUnitGetNbOutputs(unit);
    ++iOutput) {

    // Get the output value of the unit for this indice
    float val =
      VecGet(
        NMUnitOutputs(unit),
        iOutput);

    // Get the output indice
    long indiceOutput =
      VecGet(
        NMUnitIOutputs(unit),
        iOutput);

    // If the indice points toward a hidden
    if (indiceOutput < NMGetNbHidden(that)) {

      // Set the hidden value of the NeuraMorph for this indice
      VecSet(
        that->hiddens,
        indiceOutput,
        val);

    // Else, the indice points toward an output
    } else {

      // Set the output value of the NeuraMorph for this indice
      VecSet(
        that->outputs,
        indiceOutput - NMGetNbHidden(that),
        val);

    }

  }

} while (GSetIterStep(&iter));

// If the NeuraMorph is a one hot encoder
if (NMGetFlagOneHot(that) == true) {

  // Get the one hot
  long oneHot = VecGetIMaxVal(that->outputs);

  // Convert the output values
  VecSetAll(
    that->outputs,
    -1.0);
  VecSet(
    that->outputs,
    oneHot,
    1.0);

}
```

```
}

// ----- NeuraMorphTrainer

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

// Return true if the vector 'v' is a valid indices configuration
// i.e. v[i]<v[j] for all i<j
bool NMTrainerIsValidInputConfig(
  const VecLong* v,
           long iMinInput);

// Train a new NeuraMorphUnit with the interface defined by 'iInputs'
// and 'iOutputs', and add it to the set, sorted on its value
// If 'lastUnit' is true, the NeuraMorphUnit will be the last one in
// its NeuraMorph
void NMTrainerTrainUnit(
  NeuraMorphTrainer* that,
              GSet* trainedUnits,
      const VecLong* iInputs,
      const VecLong* iOutputs,
               bool lastUnit);

// Precompute the values of the NeuraMorph for each sample of the
// GDataset for the NeuraMorphTrainer 'that'
void NMTrainerPrecomputeValues(NeuraMorphTrainer* that);

// Free the precomputed values of the NeuraMorphTrainer 'that'
void NMTrainerFreePrecomputed(NeuraMorphTrainer* that);

// ================ Functions implemetation ====================

// Create a static NeuraMorphTrainer for the NeuraMorph 'neuraMorph' and the
// GDataSet 'dataset'
// Default depth: 2
// Default iCatTraining: 0
// Default weakUnitThreshold: 0.9
NeuraMorphTrainer NeuraMorphTrainerCreateStatic(
        NeuraMorph* neuraMorph,
  GDataSetVecFloat* dataset) {

#if BUILDMODE == 0

  if (neuraMorph == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'neuraMorph' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (dataset == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'dataset' is null");
    PBErrCatch(NeuraMorphErr);
```

```
  }

#endif

  // Declare the new NeuraMorphTrainer
  NeuraMorphTrainer that;

  // Init properties
  that.neuraMorph = neuraMorph;
  that.dataset = dataset;
  that.depth = 2;
  that.order = 1;
  that.nbMaxUnitDepth = 2;
  that.maxLvlDiv = 2;
  that.nbMaxInputsUnit =
    MAX(
      GDSGetNbOutputs(dataset),
      2);
  that.iCatTraining = 0;
  that.iCatEval = 1;
  that.weakUnitThreshold = 0.9;
  that.preCompInp = NULL;
  that.lowInputs = NULL;
  that.highInputs = NULL;
  that.resEval = VecFloatCreateStatic3D();

  // Return the NeuraMorphTrainer
  return that;

}

// Free the memory used by the static NeuraMorphTrainer 'that'
void NeuraMorphTrainerFreeStatic(NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  VecFree(&(that->lowInputs));
  VecFree(&(that->highInputs));

}

// Run the training process for the NeuraMorphTrainer 'that'
void NMTrainerRun(NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
```

```
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Declare a variable to memorize the minimum index needed in the
  // inputs of the new unit to ensure we do not train twice the same
  // unit
  long iMinInput = 0;

  // Loop on training depth
  for (
    short iDepth = 1;
    iDepth <= NMTrainerGetDepth(that);
    ++iDepth) {

    printf(
      "Depth %d/%d...\n",
      iDepth,
      NMTrainerGetDepth(that));

    // Get the number of available inputs for the new unit
    long nbAvailInputs =
      NMGetNbInput(NMTrainerNeuraMorph(that)) +
      NMGetNbHidden(NMTrainerNeuraMorph(that));

    printf(
      "Nb available inputs: %ld\n",
      nbAvailInputs);

    // Precompute the values to speed up the training
    NMTrainerPrecomputeValues(that);

    // Get the output indices
    VecLong* iOutputs = NMGetVecIOutputs(NMTrainerNeuraMorph(that));

    // Declare a set to memorize the trained units
    GSet trainedUnits = GSetCreateStatic();

    // Set a flag to memorize if we are at the last depth
    bool isLastDepth = (iDepth == NMTrainerGetDepth(that));

    // Get the number of inputs per unit
    long nbMaxInputsUnit =
      MIN(
        nbAvailInputs,
        NMTrainerGetNbMaxInputsUnit(that));

    // Loop on the number of inputs for the new unit
    for (
      long nbUnitInputs = 1;
      nbUnitInputs <= nbMaxInputsUnit;
      ++nbUnitInputs) {

      printf(
        "Train units with %04ld input(s)\n",
        nbUnitInputs);

      // Loop on the possible input configurations for the new units
```

```
  VecLong* iInputs = VecLongCreate(nbUnitInputs);
  VecLong* iInputsBound = VecLongCreate(nbUnitInputs);
  VecSetAll(
    iInputsBound,
    nbAvailInputs);
  bool hasStepped = true;
  do {

    bool isValidInputConfig =
      NMTrainerIsValidInputConfig(
        iInputs,
        iMinInput);
    if (isValidInputConfig == true) {

      // Train the unit
      NMTrainerTrainUnit(
        that,
        &trainedUnits,
        iInputs,
        iOutputs,
        isLastDepth);

      }

    // Step to the next input configuration
    hasStepped =
      VecStep(
        iInputs,
        iInputsBound);

  } while (hasStepped);

  // Free memory
  VecFree(&iInputs);
  VecFree(&iInputsBound);

}

// If this is the last depth
if (isLastDepth == true) {

  // Add the best of all units to the NeuraMorph
  NeuraMorphUnit* bestUnit = GSetDrop(&trainedUnits);
  GSetAppend(
    &(NMTrainerNeuraMorph(that)->units),
    bestUnit);

  printf("Add the last unit\n");
  NMUnitPrintln(
    bestUnit,
    stdout);

  // Discard all other units
  while (GSetNbElem(&trainedUnits) > 0) {

    NeuraMorphUnit* unit = GSetPop(&trainedUnits);
    NeuraMorphUnitFree(&unit);

  }

// Else, this is not the last depth
} else {
```

```
      // Get the value of the weakest and strongest units
      float weakVal = GSetElemGetSortVal(GSetHeadElem(&trainedUnits));
      float strongVal = GSetElemGetSortVal(GSetTailElem(&trainedUnits));

      // Get the threshold to discard the weakest units
      float threshold =
        weakVal + (strongVal - weakVal) *
        NMTrainerGetWeakThreshold(that);

      // Discard the weakest units
      long nbTrainedUnits = GSetNbElem(&trainedUnits);
      while (
        GSetElemGetSortVal(GSetHeadElem(&trainedUnits)) < threshold
        || GSetNbElem(&trainedUnits) > NMTrainerGetNbMaxUnitDepth(that)) {

        NeuraMorphUnit* unit = GSetPop(&trainedUnits);
        NeuraMorphUnitFree(&unit);

      }

      // Displayed the burried units
      printf(
        "Burry %ld out of %ld unit(s)\n",
        GSetNbElem(&trainedUnits),
        nbTrainedUnits);
      GSetIterForward iter = GSetIterForwardCreateStatic(&trainedUnits);
      do {

        NeuraMorphUnit* unit = GSetIterGet(&iter);
        NMUnitPrintln(
          unit,
          stdout);

      } while (GSetIterStep(&iter));

      // Burry the remaining units
      NMBurryUnits(
        NMTrainerNeuraMorph(that),
        &trainedUnits);

    }

    // Update the minimum index of a valid configuration
    iMinInput = nbAvailInputs;

    // Free memory
    VecFree(&iOutputs);
    NMTrainerFreePrecomputed(that);

  }

}

// Return true if the vector 'v' is a valid indices configuration
// i.e. v[i]<v[j] for all i<j and there exists i such as
// v[i]>=iMinInput
bool NMTrainerIsValidInputConfig(
  const VecLong* v,
            long iMinInput) {

#if BUILDMODE == 0
```

```
  if (v == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'v' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  bool noveltyCond = false;
  long a =
    VecGet(
      v,
      0);
  if (a >= iMinInput) {

    noveltyCond = true;

  }

  for (
    long i = 1;
    i < VecGetDim(v);
    ++i) {

    long b =
      VecGet(
        v,
        i);
    if (a >= b) {

      return false;

    }

    a = b;

    if (a >= iMinInput) {

      noveltyCond = true;

    }

  }

  return noveltyCond;

}

// Train a new NeuraMorphUnit with the interface defined by 'iInputs'
// and 'iOutputs', and add it to the set, sorted on its value
// If 'lastUnit' is true, the NeuraMorphUnit will be the last one in
// its NeuraMorph
void NMTrainerTrainUnit(
  NeuraMorphTrainer* that,
              GSet* trainedUnits,
      const VecLong* iInputs,
      const VecLong* iOutputs,
```

```
                bool lastUnit) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (trainedUnits == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'trainedUnits' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (iInputs == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'iInputs' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (iOutputs == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'iOutputs' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Get the number of inputs
  long nbInputs = VecGetDim(iInputs);

  // Loop on the division levels
  // (None for the last unit)
  VecShort* curDivLvl = VecShortCreate(nbInputs);
  VecShort* divLvlBound = VecShortCreate(nbInputs);
  if (lastUnit == true) {

    VecSetAll(
      divLvlBound,
      0);

  } else {

    VecSetAll(
```

```
    divLvlBound,
    NMTrainerGetMaxLvlDiv(that));

}

bool flagStepDivLvl = true;
do {

  // Get the bounds for the number of division for each input
  // at current levels
  VecShort* divBound = VecShortCreate(nbInputs);
  for (
    long iInput = nbInputs;
    iInput--;) {

    short lvl =
      VecGet(
        curDivLvl,
        iInput);
    short bound =
      powi(
        2,
        lvl);
    VecSet(
      divBound,
      iInput,
      bound);

  }

  // Loop on the combination of divisions
  VecShort* curDiv = VecShortCreate(nbInputs);
  bool flagStepDiv = true;
  do {

    // Create the unit
    NeuraMorphUnit* unit =
      NeuraMorphUnitCreate(
        iInputs,
        iOutputs);

    // Loop on the inputs of the unit
    for (
      long iInput = nbInputs;
      iInput--;) {

      // Get the indice of this input in the NeuraMorph
      short jInput =
        VecGet(
          NMUnitIInputs(unit),
          iInput);

      // Declare variables to memorize the lowest and highest
      // values for this input
      float low = 0.0;
      float high = 0.0;

      // If this input is an input in the NeuraMorph
      if (jInput < NMGetNbInput(NMTrainerNeuraMorph(that))) {

        low =
          VecGet(
```

```
        that->lowInputs,
        jInput);
    high =
      VecGet(
        that->highInputs,
        jInput);

  // Else, this input is an hidden value in the NeuraMorph
  } else {

    low =
      VecGet(
        NMLowHiddens(NMTrainerNeuraMorph(that)),
        jInput - NMGetNbInput(NMTrainerNeuraMorph(that)));
    high =
      VecGet(
        NMHighHiddens(NMTrainerNeuraMorph(that)),
        jInput - NMGetNbInput(NMTrainerNeuraMorph(that)));

  }

  // Get the filter values for the current division
  float lowFilter =
    low + (high - low) *
    (float)VecGet(
      curDiv,
      iInput) /
    (float)VecGet(
      divBound,
      iInput);
  float highFilter =
    low + (high - low) *
    (float)(VecGet(
      curDiv,
      iInput) + 1) /
    (float)VecGet(
      divBound,
      iInput);

  // Set the filter values in the unit
  VecSet(
    unit->lowFilters,
    iInput,
    lowFilter);
  VecSet(
    unit->highFilters,
    iInput,
    highFilter);

}

// Declare two GSets to extract the filtered samples
GSetVecFloat trainingInputs = GSetVecFloatCreateStatic();
GSetVecFloat trainingOutputs = GSetVecFloatCreateStatic();

// Loop on the samples of the dataset
long nbSample =
  GDSGetSizeCat(
    NMTrainerDataset(that),
    NMTrainerGetICatTraining(that));
for (
  long iSample = 0;
```

```
iSample < nbSample;
++iSample) {

// Create the sample's inputs for this unit
VecFloat* sampleInputs = VecFloatCreate(nbInputs);

// If all the input values are within the bound of the unit
bool flag = true;
for (
  long iInput = nbInputs;
  flag && iInput--;) {

  float low =
    VecGet(
      unit->lowFilters,
      iInput);
  float high =
    VecGet(
      unit->highFilters,
      iInput);
  short jInput =
    VecGet(
      iInputs,
      iInput);
  float val =
    VecGet(
      that->preCompInp[iSample],
      jInput);
  if (
    val < low ||
    val > high) {

    flag = false;

  }

  // Simultaneously, scale the inputs values toward the unit
  // input space
  val = (val - low) / (high - low);
  VecSet(
    sampleInputs,
    iInput,
    val);

}

if (flag) {

  // Add this sample to the training set for the current unit
  GSetAppend(
    &trainingInputs,
    sampleInputs);
  GSetAppend(
    &trainingOutputs,
    that->preCompOut[iSample]);

} else {

  // Free memory
  VecFree(&sampleInputs);

}
```

```
  }

  // If we have enough samples to train the unit on the current
  // combination of divisions
  if (GSetNbElem(&trainingInputs) >= NMUnitGetNbInputs(unit)) {

    // Calculate the transfer function
    float bias = 0.0;
    unit->transfer =
      BBodyFromPointCloud(
        NMTrainerGetOrder(that),
        &trainingInputs,
        &trainingOutputs,
        &bias);

    // If we could calculate the transfer function
    if (unit->transfer != NULL) {

      // Set the value of the unit
      float corrRange =
        (float)GSetNbElem(&trainingInputs) /
        (float)GDSGetSizeCat(
          NMTrainerDataset(that),
          NMTrainerGetICatTraining(that));
      NMUnitSetValue(
        unit,
        -1.0 * bias / corrRange);

      // Add the unit to the set of trained units
      GSetAddSort(
        trainedUnits,
        unit,
        NMUnitGetValue(unit));

    // Else, we couldn't calculate the transfer function
    } else {

      // Free memory
      NeuraMorphUnitFree(&unit);

    }

  }

  // Free memory
  while (GSetNbElem(&trainingInputs) > 0) {

    VecFloat* v = GSetPop(&trainingInputs);
    VecFree(&v);

  }

  GSetFlush(&trainingOutputs);

  // Move to the next combination of divisions
  flagStepDiv =
    VecStep(
      curDiv,
      divBound);

} while (flagStepDiv);
```

```
    // Free memory
    VecFree(&curDiv);
    VecFree(&divBound);

    // Move to the next division level
    flagStepDivLvl =
      VecStep(
        curDivLvl,
        divLvlBound);

  } while (flagStepDivLvl);

  // Free memory
  VecFree(&curDivLvl);
  VecFree(&divLvlBound);

}

// Precompute the values of the NeuraMorph for each sample of the
// GDataset for the NeuraMorphTrainer 'that'
void NMTrainerPrecomputeValues(NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Get the number of samples
  long nbSample =
    GDSGetSizeCat(
      NMTrainerDataset(that),
      NMTrainerGetICatTraining(that));

  // Allocate memory
  that->preCompInp =
    PBErrMalloc(
      NeuraMorphErr,
      nbSample * sizeof(VecFloat*));
  that->preCompOut =
    PBErrMalloc(
      NeuraMorphErr,
      nbSample * sizeof(VecFloat*));

  // Reset the low and high values for input
  VecFree(&(that->lowInputs));
  VecFree(&(that->highInputs));

  // Get the size of the precomputed vector
  long sizeInp =
    NMGetNbInput(NMTrainerNeuraMorph(that)) +
    NMGetNbHidden(NMTrainerNeuraMorph(that));
```

```
// Loop on the samples
long iSample = 0;
bool flagStep = true;
GDSReset(
  NMTrainerDataset(that),
  NMTrainerGetICatTraining(that));
do {

  // Get a clone of the sample's inputs
  VecFloat* inputs =
    GDSGetSampleInputs(
      NMTrainerDataset(that),
      NMTrainerGetICatTraining(that));

  // Update the low and high input values
  if (that->lowInputs == NULL) {

    that->lowInputs = VecClone(inputs);
    that->highInputs = VecClone(inputs);

  } else {

    for (
      long iInput = 0;
      iInput < VecGetDim(inputs);
      ++iInput) {

      float val =
        VecGet(
          inputs,
          iInput);

      float curLow =
        VecGet(
          that->lowInputs,
          iInput);
      if (curLow > val) {

        VecSet(
          that->lowInputs,
          iInput,
          val);

      }

      float curHigh =
        VecGet(
          that->highInputs,
          iInput);
      if (curHigh < val) {

        VecSet(
          that->highInputs,
          iInput,
          val);

      }

    }

  }
```

```
  // Get a clone of the sample's outputs
  that->preCompOut[iSample] =
    GDSGetSampleOutputs(
      NMTrainerDataset(that),
      NMTrainerGetICatTraining(that));

  // Run the NeuraMorph on the sample
  NMEvaluate(
    NMTrainerNeuraMorph(that),
    inputs);

  // Allocate memory for the precomputed vector
  that->preCompInp[iSample] = VecFloatCreate(sizeInp);

  // Copy the inputs and hidden values into the precomputed vector
  for (
    long i = NMGetNbInput(NMTrainerNeuraMorph(that));
    i--;) {

    float val =
      VecGet(
        NMInputs(NMTrainerNeuraMorph(that)),
        i);
    VecSet(
      that->preCompInp[iSample],
      i,
      val);

  }

  for (
    long i = NMGetNbHidden(NMTrainerNeuraMorph(that));
    i--;) {

    float val =
      VecGet(
        NMHiddens(NMTrainerNeuraMorph(that)),
        i);
    VecSet(
      that->preCompInp[iSample],
      i + NMGetNbInput(NMTrainerNeuraMorph(that)),
      val);

  }

  // Free memory
  VecFree(&inputs);

  // Move to the next sample
  ++iSample;
  flagStep =
    GDSStepSample(
      NMTrainerDataset(that),
      NMTrainerGetICatTraining(that));

} while (flagStep);

// Finally, update the hiddens bound if any
if (NMGetNbHidden(NMTrainerNeuraMorph(that)) > 0) {

  NMUpdateLowHighHiddens(NMTrainerNeuraMorph(that));
```

```
  }

}

// Free the precomputed hidden values of the NeuraMorphTrainer 'that'
void NMTrainerFreePrecomputed(NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // If the hidden values are not precomputed
  if (that->preCompInp == NULL) {

    // Stop here
    return;

  }

  // Get the number of samples
  long nbSample =
    GDSGetSizeCat(
      NMTrainerDataset(that),
      NMTrainerGetICatTraining(that));

  // Free memory
  for (
    long iSample = nbSample;
    iSample--;) {

    VecFree(that->preCompInp + iSample);
    VecFree(that->preCompOut + iSample);

  }

  free(that->preCompInp);
  that->preCompInp = NULL;
  free(that->preCompOut);
  that->preCompOut = NULL;

}

// Run the evaluation process for the NeuraMorphTrainer 'that'
void NMTrainerEval(NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
```

```
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  // Declare a variable to calculate the result of evaluation
  float minBias = 0.0;
  float avgBias = 0.0;
  float maxBias = 0.0;

  // Loop on the evaluation samples
  long iSample = 0;
  bool flagStep = true;
  GDSReset(
    NMTrainerDataset(that),
    NMTrainerGetICatEval(that));
  do {

    // Get a clone of the sample's inputs and outputs
    VecFloat* inputs =
      GDSGetSampleInputs(
        NMTrainerDataset(that),
        NMTrainerGetICatEval(that));
    VecFloat* outputs =
      GDSGetSampleOutputs(
        NMTrainerDataset(that),
        NMTrainerGetICatEval(that));

    // Run the NeuraMorph on the sample
    NMEvaluate(
      NMTrainerNeuraMorph(that),
      inputs);

    // Display the result
    printf(
      "%02ld ",
      iSample);
    VecPrint(
      inputs,
      stdout);
    printf(" -> ");
    VecPrint(
      outputs,
      stdout);
    printf(" : ");
    VecPrint(
      NMOutputs(NMTrainerNeuraMorph(that)),
      stdout);
    printf(" ");
    float bias =
      VecDist(
        outputs,
        NMOutputs(NMTrainerNeuraMorph(that)));
    printf(
      "%f\n",
      bias);

    // Update the result of evaluation
    avgBias += bias;
    if (iSample == 0) {
```

```
      minBias = bias;
      maxBias = bias;

    } else {

      minBias =
        MIN(
          bias,
          minBias);
      maxBias =
        MAX(
          bias,
          maxBias);

    }

    // Free memory
    VecFree(&inputs);
    VecFree(&outputs);

    // Move to the next sample
    ++iSample;
    flagStep =
      GDSStepSample(
        NMTrainerDataset(that),
        NMTrainerGetICatEval(that));

  } while (flagStep);

  // Memorize the result of evaluation
  avgBias /=
    (float)GDSGetSizeCat(
      NMTrainerDataset(that),
      NMTrainerGetICatEval(that));
  VecSet(
    &(that->resEval),
    0,
    minBias);
  VecSet(
    &(that->resEval),
    1,
    avgBias);
  VecSet(
    &(that->resEval),
    2,
    maxBias);

}
```

3.2 neuramorph-inline.c

```
// ============ NEURAMORPH-INLINE.C ================

// ----- NeuraMorphUnit

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

// Get the input indices of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
```

```
static inline
#endif
const VecLong* NMUnitIInputs(const NeuraMorphUnit* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->iInputs;

}

// Get the output indices of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
const VecLong* NMUnitIOutputs(const NeuraMorphUnit* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->iOutputs;

}

// Get the output values of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMUnitOutputs(const NeuraMorphUnit* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);
```

```
  }

#endif

  return that->outputs;

}

// Get the number of input values of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
long NMUnitGetNbInputs(const NeuraMorphUnit* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return VecGetDim(that->iInputs);

}

// Get the number of output values of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
long NMUnitGetNbOutputs(const NeuraMorphUnit* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return VecGetDim(that->iOutputs);

}

// Get the number of coefficients in the transfer function of
// the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
long NMUnitGetNbCoeffs(const NeuraMorphUnit* that) {
```

```
#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return BBodyGetNbCtrl(that->transfer);

}

// Get the value of the NeuraMorphUnit 'that'
#if BUILDMODE != 0
static inline
#endif
float NMUnitGetValue(const NeuraMorphUnit* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->value;

}

// Set the value of the NeuraMorphUnit 'that' to 'val'
#if BUILDMODE != 0
static inline
#endif
void NMUnitSetValue(
  NeuraMorphUnit* that,
            float val) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }
```

```
#endif

  that->value = val;

}

// ----- NeuraMorph

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

// Get the number of input values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
long NMGetNbInput(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->nbInput;

}

// Get the number of output values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
long NMGetNbOutput(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->nbOutput;

}

// Get the input values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
```

```
#endif
VecFloat* NMInputs(NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->inputs;

}

// Get the output values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMOutputs(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->outputs;

}

// Get the hidden values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMHiddens(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }
```

```
#endif

  return that->hiddens;

}

// Get the number of hidden values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
long NMGetNbHidden(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  long nb = 0;
  if (that->hiddens != NULL) {

    nb = VecGetDim(that->hiddens);

  }

  return nb;

}

// Set the number of hidden values of the NeuraMorph 'that' to 'nb'
#if BUILDMODE != 0
static inline
#endif
void NMSetNbHidden(
  NeuraMorph* that,
         long nb) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (nb <= 0) {

    NeuraMorphErr->_type = PBErrTypeInvalidArg;
    sprintf(
```

```
      NeuraMorphErr->_msg,
      "'nb' is invalid (%ld>0)",
      nb);
    PBErrCatch(NeuraMorphErr);

  }

#endif

  if (that->hiddens != NULL) {

    VecFree(&(that->hiddens));

  }

  that->hiddens = VecFloatCreate(nb);

}

// Get the lowest bound of hidden values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMLowHiddens(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->lowHiddens;

}

// Get the highest bound of hidden values of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* NMHighHiddens(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif
```

```
  return that->highHiddens;

}

// Get the flag for one hot encoding of the NeuraMorph 'that'
#if BUILDMODE != 0
static inline
#endif
bool NMGetFlagOneHot(const NeuraMorph* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->flagOneHot;

}

// Set the flag for one hot encoding of the NeuraMorph 'that' to 'flag'
#if BUILDMODE != 0
static inline
#endif
void NMSetFlagOneHot(
  NeuraMorph* that,
         bool flag) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->flagOneHot = flag;

}

// ----- NeuraMorphTrainer

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

// Get the depth of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
```

```
#endif
short NMTrainerGetDepth(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->depth;

}

// Set the depth of the NeuraMorphTrainer 'that' to 'depth'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetDepth(
  NeuraMorphTrainer* that,
               short depth) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (depth < 1) {

    NeuraMorphErr->_type = PBErrTypeInvalidArg;
    sprintf(
      NeuraMorphErr->_msg,
      "'depth' is invalid (%d>=1)",
      depth);
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->depth = depth;

}

// Get the maxLvlDiv of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
```

```
short NMTrainerGetMaxLvlDiv(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->maxLvlDiv;

}

// Set the maxLvlDiv of the NeuraMorphTrainer 'that' to 'lvl'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetMaxLvlDiv(
  NeuraMorphTrainer* that,
               short lvl) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (lvl < 0) {

    NeuraMorphErr->_type = PBErrTypeInvalidArg;
    sprintf(
      NeuraMorphErr->_msg,
      "'lvl' is invalid (%d>=0)",
      lvl);
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->maxLvlDiv = lvl;

}

// Get the order of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
int NMTrainerGetOrder(const NeuraMorphTrainer* that) {
```

```
#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->order;

}

// Set the order of the NeuraMorphTrainer 'that' to 'order'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetOrder(
  NeuraMorphTrainer* that,
                int order) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (order < 1) {

    NeuraMorphErr->_type = PBErrTypeInvalidArg;
    sprintf(
      NeuraMorphErr->_msg,
      "'order' is invalid (%d>=1)",
      order);
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->order = order;

}

// Get the nbMaxInputsUnit of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
int NMTrainerGetNbMaxInputsUnit(const NeuraMorphTrainer* that) {
```

```
#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->nbMaxInputsUnit;

}

// Set the nbMaxInputsUnit of the NeuraMorphTrainer 'that' to 'order'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetNbMaxInputsUnit(
  NeuraMorphTrainer* that,
                int nbMaxInputsUnit) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

  if (nbMaxInputsUnit < 2) {

    NeuraMorphErr->_type = PBErrTypeInvalidArg;
    sprintf(
      NeuraMorphErr->_msg,
      "'nbMaxInputsUnit' is invalid (%d>=2)",
      nbMaxInputsUnit);
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->nbMaxInputsUnit = nbMaxInputsUnit;

}

// Get the NeuraMorph of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
NeuraMorph* NMTrainerNeuraMorph(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0
```

```
  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->neuraMorph;

}

// Get the GDataSet of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
GDataSetVecFloat* NMTrainerDataset(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->dataset;

}

// Get the index of the training category of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int NMTrainerGetICatTraining(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->iCatTraining;
```

```
}

// Set the index of the training category of the NeuraMorphTrainer 'that'
// to 'iCat'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetICatTraining(
  NeuraMorphTrainer* that,
        unsigned int iCatTraining) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->iCatTraining = iCatTraining;

}

// Get the index of the evaluation category of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned int NMTrainerGetICatEval(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->iCatEval;

}

// Set the index of the evaluation category of the NeuraMorphTrainer 'that'
// to 'iCat'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetICatEval(
  NeuraMorphTrainer* that,
        unsigned int iCatEval) {
```

```
#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->iCatEval = iCatEval;

}

// Get the weakness threshold of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
float NMTrainerGetWeakThreshold(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->weakUnitThreshold;

}

// Set the weakness threshold of the NeuraMorphTrainer 'that'
// to 'iCat'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetWeakThreshold(
  NeuraMorphTrainer* that,
               float weakUnitThreshold) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);
```

```
  }

#endif

  that->weakUnitThreshold = weakUnitThreshold;

}

// Get the nbMaxUnitDepth of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
int NMTrainerGetNbMaxUnitDepth(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return that->nbMaxUnitDepth;

}

// Set the nbMaxUnitDepth of the NeuraMorphTrainer 'that' to 'nbMaxUnitDepth'
#if BUILDMODE != 0
static inline
#endif
void NMTrainerSetNbMaxUnitDepth(
  NeuraMorphTrainer* that,
                 int nbMaxUnitDepth) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  that->nbMaxUnitDepth = nbMaxUnitDepth;

}

// Get the result of the last evaluation of the NeuraMorphTrainer 'that'
#if BUILDMODE != 0
static inline
#endif
```

```
const VecFloat3D* NMTrainerResEval(const NeuraMorphTrainer* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    NeuraMorphErr->_type = PBErrTypeNullPointer;
    sprintf(
      NeuraMorphErr->_msg,
      "'that' is null");
    PBErrCatch(NeuraMorphErr);

  }

#endif

  return &(that->resEval);

}
```

4 Makefile

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

all: pbmake_wget main

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

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

# Rules to make the executable
repo=neuramorph
$($(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 "neuramorph.h"

void UnitTestNeuraMorphUnitCreateFree() {

  VecLong* iIn = VecLongCreate(3);
  VecSet(
    iIn,
    0,
    0);
  VecSet(
    iIn,
    1,
    1);
  VecSet(
    iIn,
    2,
    2);
  VecLong* iOut = VecLongCreate(2);
  VecSet(
    iOut,
    0,
    0);
  VecSet(
    iOut,
    1,
    1);
  NeuraMorphUnit* unit =
    NeuraMorphUnitCreate(
      iIn,
      iOut);
  bool isSame =
    ISEQUALF(
      unit->value,
      0.0);
  if (
    VecGetDim(unit->outputs) != 2 ||
    VecGetDim(unit->lowFilters) != 3 ||
    VecGetDim(unit->highFilters) != 3 ||
    VecGetDim(unit->unitInputs) != 3 ||
    isSame != true ||
    unit->lowOutputs != NULL ||
    unit->highOutputs != NULL) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphUnitCreate failed (1)");
    PBErrCatch(NeuraMorphErr);

  }

  isSame =
    VecIsEqual(
      unit->iInputs,
      iIn);
  if (isSame == false) {
```

```
    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphUnitCreate failed (2)");
    PBErrCatch(NeuraMorphErr);

  }

  isSame =
    VecIsEqual(
      unit->iOutputs,
      iOut);
  if (isSame == false) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphUnitCreate failed (3)");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphUnitFree(&unit);
  if (unit != NULL) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphUnitFree failed");
    PBErrCatch(NeuraMorphErr);

  }

  VecFree(&iIn);
  VecFree(&iOut);
  printf("UnitTestNeuraMorphUnitCreateFree OK\n");

}

void UnitTestNeuraMorphUnitGetSetPrint() {

  VecLong* iIn = VecLongCreate(3);
  VecLong* iOut = VecLongCreate(2);
  NeuraMorphUnit* unit =
    NeuraMorphUnitCreate(
      iIn,
      iOut);

  if (NMUnitIInputs(unit) != unit->iInputs) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMUnitIInputs failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMUnitIOutputs(unit) != unit->iOutputs) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
```

```
    NeuraMorphErr->_msg,
    "NMUnitIOutputs failed");
  PBErrCatch(NeuraMorphErr);

}

if (NMUnitOutputs(unit) != unit->outputs) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMUnitOutputs failed");
  PBErrCatch(NeuraMorphErr);

}

if (NMUnitGetNbInputs(unit) != 3) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMUnitGetNbInputs failed");
  PBErrCatch(NeuraMorphErr);

}

if (NMUnitGetNbOutputs(unit) != 2) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMUnitGetNbOutputs failed");
  PBErrCatch(NeuraMorphErr);

}

bool isSame =
  ISEQUALF(
    NMUnitGetValue(unit),
    0.0);
if (isSame != true) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMUnitGetValue failed");
  PBErrCatch(NeuraMorphErr);

}

NMUnitSetValue(
  unit,
  0.5);
isSame =
  ISEQUALF(
    NMUnitGetValue(unit),
    0.5);
if (isSame != true) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
```

```
      "NMUnitSetValue failed");
    PBErrCatch(NeuraMorphErr);

  }

  NMUnitPrintln(
    unit,
    stdout);

  NeuraMorphUnitFree(&unit);
  VecFree(&iIn);
  VecFree(&iOut);
  printf("UnitTestNeuraMorphUnitGetSetPrint OK\n");

}

void UnitTestNeuraMorphUnitEvaluate() {

  VecLong* iIn = VecLongCreate(3);
  VecLong* iOut = VecLongCreate(2);
  NeuraMorphUnit* unit =
    NeuraMorphUnitCreate(
      iIn,
      iOut);

  for (
    long iInput = 3;
    iInput--;) {

    VecSet(
      unit->lowFilters,
      iInput,
      0.0);
    VecSet(
      unit->highFilters,
      iInput,
      2.0);

  }

  VecShort2D dim = VecShortCreateStatic2D();
  VecSet(
    &dim,
    0,
    3);
  VecSet(
    &dim,
    1,
    2);
  unit->transfer =
    BBodyCreate(
      1,
      &dim);
  unit->transfer->_ctrl[0]->_val[0] = 1.0;
  unit->transfer->_ctrl[0]->_val[1] = 2.0;

  VecFloat* inputs = VecFloatCreate(3);
  VecSet(
    inputs,
    0,
    1.0);
  VecSet(
```

```
  inputs,
  1,
  3.0);
VecSet(
  inputs,
  2,
  1.5);

NMUnitEvaluate(
  unit,
  inputs);

float check[2];
check[0] = -0.0625;
check[1] = -0.125;
VecFloat2D checkHigh = VecFloatCreateStatic2D();
VecSet(
  &checkHigh,
  0,
  check[0]);
VecSet(
  &checkHigh,
  1,
  check[1]);
VecFloat2D checkLow = checkHigh;
for (
  long iOutput = 2;
  iOutput--;) {

  float v =
    VecGet(
      unit->outputs,
      iOutput);
  bool same =
    ISEQUALF(
      v,
      check[iOutput]);
  if (same == false) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMUnitEvaluate failed (1)");
    PBErrCatch(NeuraMorphErr);

  }

}

bool sameLow =
  VecIsEqual(
    &checkLow,
    unit->lowOutputs);
bool sameHigh =
  VecIsEqual(
    &checkHigh,
    unit->highOutputs);
if (
  sameLow == false ||
  sameHigh == false) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
```

```
    sprintf(
      NeuraMorphErr->_msg,
      "NMUnitEvaluate failed (2)");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphUnitFree(&unit);
  VecFree(&iIn);
  VecFree(&iOut);
  VecFree(&inputs);
  printf("UnitTestNeuraMorphUnitEvaluate OK\n");

}

void UnitTestNeuraMorphUnit() {

  UnitTestNeuraMorphUnitCreateFree();
  UnitTestNeuraMorphUnitGetSetPrint();
  UnitTestNeuraMorphUnitEvaluate();
  printf("UnitTestNeuraMorphUnit OK\n");

}

void UnitTestNeuraMorphCreateFree() {

  NeuraMorph* nm =
    NeuraMorphCreate(
      3,
      2);
  if (
    nm->nbInput != 3 ||
    nm->nbOutput != 2 ||
    nm->flagOneHot != false ||
    VecGetDim(nm->inputs) != 3 ||
    VecGetDim(nm->outputs) != 2 ||
    nm->hiddens != NULL ||
    GSetNbElem(&(nm->units)) != 0) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphCreate failed");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphFree(&nm);
  if (nm != NULL) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphFree failed");
    PBErrCatch(NeuraMorphErr);

  }

  printf("UnitTestNeuraMorphCreateFree OK\n");

}
```

```
void UnitTestNeuraMorphGetSet() {

  NeuraMorph* nm =
    NeuraMorphCreate(
      3,
      2);
  if (NMGetNbInput(nm) != 3) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMGetNbInput failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMGetNbOutput(nm) != 2) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMGetNbOutput failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMGetNbHidden(nm) != 0) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMGetNbHidden failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMGetFlagOneHot(nm) != false) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMGetFlagOneHot failed");
    PBErrCatch(NeuraMorphErr);

  }

  NMSetNbHidden(
    nm,
    5);
  if (NMGetNbHidden(nm) != 5) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMSetNbHidden failed");
    PBErrCatch(NeuraMorphErr);

  }

  NMSetFlagOneHot(
    nm,
    true);
```

```
if (NMGetFlagOneHot(nm) != true) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMSetFlagOneHot failed");
  PBErrCatch(NeuraMorphErr);

}

VecLong* iOuts = NMGetVecIOutputs(nm);
VecLong2D checkOuts =
  VecLongCreateStatic2D();
VecSet(
  &checkOuts,
  0,
  5);
VecSet(
  &checkOuts,
  1,
  6);
bool isSame =
  VecIsEqual(
    &checkOuts,
    iOuts);
if (isSame == false) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMGetVecIOutputs failed");
  PBErrCatch(NeuraMorphErr);

}

VecFree(&iOuts);

if (NMInputs(nm) != nm->inputs) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMInputs failed");
  PBErrCatch(NeuraMorphErr);

}

if (NMOutputs(nm) != nm->outputs) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMOutputs failed");
  PBErrCatch(NeuraMorphErr);

}

if (NMHiddens(nm) != nm->hiddens) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
```

```
      "NMHiddens failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMLowHiddens(nm) != nm->lowHiddens) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMLowHiddens failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMHighHiddens(nm) != nm->highHiddens) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMHighHiddens failed");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphFree(&nm);

  printf("UnitTestNeuraMorphGetSet OK\n");

}

void UnitTestNeuraMorphAddRemoveUnit() {

  VecLong3D iInputs = VecLongCreateStatic3D();
  VecSet(
    &iInputs,
    0,
    0);
  VecSet(
    &iInputs,
    1,
    1);
  VecSet(
    &iInputs,
    2,
    2);
  VecLong2D iOutputs = VecLongCreateStatic2D();
  VecSet(
    &iOutputs,
    0,
    0);
  VecSet(
    &iOutputs,
    1,
    1);

  NeuraMorph* nm =
    NeuraMorphCreate(
      3,
      2);

  NeuraMorphUnit* unit =
```

```
    NMAddUnit(
      nm,
      (VecLong*)&iInputs,
      (VecLong*)&iOutputs);

  bool isSameA =
    VecIsEqual(
      &iInputs,
      unit->iInputs);
  bool isSameB =
    VecIsEqual(
      &iOutputs,
      unit->iOutputs);
  if (
    GSetNbElem(&(nm->units)) != 1 ||
    GSetHead(&(nm->units)) != unit ||
    isSameA == false ||
    isSameB == false) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMAddUnit failed");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphFree(&nm);

  nm =
    NeuraMorphCreate(
      3,
      2);

  unit =
    NMAddUnit(
      nm,
      (VecLong*)&iInputs,
      (VecLong*)&iOutputs);

  NMRemoveUnit(
    nm,
    unit);

  if (GSetNbElem(&(nm->units)) != 0) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NMRemoveUnit failed");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphUnitFree(&unit);
  NeuraMorphFree(&nm);

  printf("UnitTestNeuraMorphAddRemoveUnit OK\n");

}

void UnitTestNeuraMorphBurryUnitsEvaluate() {
```

```
VecLong3D iInputs = VecLongCreateStatic3D();
VecSet(
  &iInputs,
  0,
  0);
VecSet(
  &iInputs,
  1,
  1);
VecSet(
  &iInputs,
  2,
  2);
VecLong2D iOutputs = VecLongCreateStatic2D();
VecSet(
  &iOutputs,
  0,
  0);
VecSet(
  &iOutputs,
  1,
  1);

NeuraMorph* nm =
  NeuraMorphCreate(
    3,
    2);

NeuraMorphUnit* unitA =
  NeuraMorphUnitCreate(
    (VecLong*)&iInputs,
    (VecLong*)&iOutputs);

NeuraMorphUnit* unitB =
  NeuraMorphUnitCreate(
    (VecLong*)&iInputs,
    (VecLong*)&iOutputs);

for (
  long iInput = 3;
  iInput--;) {

  VecSet(
    unitA->lowFilters,
    iInput,
    0.0);
  VecSet(
    unitA->highFilters,
    iInput,
    2.0);
  VecSet(
    unitB->lowFilters,
    iInput,
    0.0);
  VecSet(
    unitB->highFilters,
    iInput,
    2.0);

}
```

```
VecShort2D dim = VecShortCreateStatic2D();
VecSet(
  &dim,
  0,
  3);
VecSet(
  &dim,
  1,
  2);
unitA->transfer =
  BBodyCreate(
    1,
    &dim);
unitA->transfer->_ctrl[0]->_val[0] = 1.0;
unitA->transfer->_ctrl[0]->_val[1] = 2.0;
unitB->transfer =
  BBodyCreate(
    1,
    &dim);
unitB->transfer->_ctrl[0]->_val[0] = 2.0;
unitB->transfer->_ctrl[0]->_val[1] = 1.0;

float x = 1.0;
float y = 0.5;
float z = 1.5;
VecFloat* evalInputs = VecFloatCreate(3);
VecSet(
  evalInputs,
  0,
  x);
VecSet(
  evalInputs,
  1,
  y);
VecSet(
  evalInputs,
  2,
  z);

NMUnitEvaluate(
  unitA,
  evalInputs);
NMUnitEvaluate(
  unitB,
  evalInputs);

GSet units = GSetCreateStatic();
GSetAppend(
  &units,
  unitA);
GSetAppend(
  &units,
  unitB);

NMBurryUnits(
  nm,
  &units);

if (
  GSetNbElem(&units) != 0 ||
  nm->hiddens == NULL ||
  VecGetDim(nm->hiddens) != 4) {
```

```
  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMBurryUnits failed (1)");
  PBErrCatch(NeuraMorphErr);

}

VecLong2D checkA = VecLongCreateStatic2D();
VecSet(
  &checkA,
  0,
  0);
VecSet(
  &checkA,
  1,
  1);
VecLong2D checkB = VecLongCreateStatic2D();
VecSet(
  &checkB,
  0,
  2);
VecSet(
  &checkB,
  1,
  3);

bool isSameA =
  VecIsEqual(
    &checkA,
    unitA->iOutputs);
bool isSameB =
  VecIsEqual(
    &checkB,
    unitB->iOutputs);
if (
  isSameA == false ||
  isSameB == false) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMBurryUnits failed (2)");
  PBErrCatch(NeuraMorphErr);

}

float checkLowAa =
  VecGet(
    unitA->lowOutputs,
    0);
checkLowAa -=
  VecGet(
    nm->lowHiddens,
    0);
bool isSameLowAa =
  ISEQUALF(
    checkLowAa,
    0.0);
float checkLowAb =
  VecGet(
```

```
    unitA->lowOutputs,
    1);
checkLowAb -=
  VecGet(
    nm->lowHiddens,
    1);
bool isSameLowAb =
  ISEQUALF(
    checkLowAb,
    0.0);
float checkLowBa =
  VecGet(
    unitB->lowOutputs,
    0);
checkLowBa -=
  VecGet(
    nm->lowHiddens,
    2);
bool isSameLowBa =
  ISEQUALF(
    checkLowBa,
    0.0);
float checkLowBb =
  VecGet(
    unitB->lowOutputs,
    1);
checkLowBb -=
  VecGet(
    nm->lowHiddens,
    3);
bool isSameLowBb =
  ISEQUALF(
    checkLowBb,
    0.0);
float checkHighAa =
  VecGet(
    unitA->lowOutputs,
    0);
checkHighAa -=
  VecGet(
    nm->lowHiddens,
    0);
bool isSameHighAa =
  ISEQUALF(
    checkHighAa,
    0.0);
float checkHighAb =
  VecGet(
    unitA->lowOutputs,
    1);
checkHighAb -=
  VecGet(
    nm->lowHiddens,
    1);
bool isSameHighAb =
  ISEQUALF(
    checkHighAb,
    0.0);
float checkHighBa =
  VecGet(
    unitB->lowOutputs,
    0);
```

```
checkHighBa -=
  VecGet(
    nm->lowHiddens,
    2);
bool isSameHighBa =
  ISEQUALF(
    checkHighBa,
    0.0);
float checkHighBb =
  VecGet(
    unitB->lowOutputs,
    1);
checkHighBb -=
  VecGet(
    nm->lowHiddens,
    3);
bool isSameHighBb =
  ISEQUALF(
    checkHighBb,
    0.0);
if (
  isSameLowAa == false ||
  isSameLowAb == false ||
  isSameLowBa == false ||
  isSameLowBb == false ||
  isSameHighAa == false ||
  isSameHighAb == false ||
  isSameHighBa == false ||
  isSameHighBb == false) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMBurryUnits failed (3)");
  PBErrCatch(NeuraMorphErr);

}

VecSet(
  &iInputs,
  0,
  3);
VecSet(
  &iInputs,
  1,
  4);
VecSet(
  &iInputs,
  2,
  5);
VecSet(
  &iOutputs,
  0,
  4);
VecSet(
  &iOutputs,
  1,
  5);
NeuraMorphUnit* unitC =
  NMAddUnit(
    nm,
    (VecLong*)&iInputs,
```

```
    (VecLong*)&iOutputs);

for (
  long iInput = 3;
  iInput--;) {

  VecSet(
    unitC->lowFilters,
    iInput,
    0.0);
  VecSet(
    unitC->highFilters,
    iInput,
    20.0);

}

unitC->transfer =
  BBodyCreate(
    1,
    &dim);
unitC->transfer->_ctrl[0]->_val[0] = -1.0;
unitC->transfer->_ctrl[0]->_val[1] = -2.0;

NMEvaluate(
  nm,
  evalInputs);
float checkAout[2];
checkAout[0] =
  0.09375 -
  VecGet(
    nm->hiddens,
    0);
checkAout[1] =
  0.1875 -
  VecGet(
    nm->hiddens,
    1);
float checkBout[2];
checkBout[0] =
  0.1875 -
  VecGet(
    nm->hiddens,
    2);
checkBout[1] =
  0.09375 -
  VecGet(
    nm->hiddens,
    3);

bool isSameAa =
  ISEQUALF(
    checkAout[0],
    0.0);
bool isSameAb =
  ISEQUALF(
    checkAout[1],
    0.0);
bool isSameBa =
  ISEQUALF(
    checkBout[0],
    0.0);
```

```
bool isSameBb =
  ISEQUALF(
    checkBout[1],
    0.0);
if (
  isSameAa == false ||
  isSameAb == false ||
  isSameBa == false ||
  isSameBb == false) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NMEvaluate failed (1)");
  PBErrCatch(NeuraMorphErr);

}

x =
  VecGet(
    nm->hiddens,
    0);
y =
  VecGet(
    nm->hiddens,
    1);
z =
  VecGet(
    nm->hiddens,
    2);
float checkCout[2];
checkCout[0] =
  -0.976738 -
  VecGet(
    unitC->outputs,
    0);
checkCout[1] =
  -1.953476 -
  VecGet(
    unitC->outputs,
    1);

bool isSameCa =
  ISEQUALF(
    checkCout[0],
    0.0);
bool isSameCb =
  ISEQUALF(
    checkCout[1],
    0.0);
bool isSameCc =
  VecIsEqual(
    unitC->outputs,
    nm->outputs);
if (
  isSameCa == false ||
  isSameCb == false ||
  isSameCc == false) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
```

```
      "NMEvaluate failed (2)");
    PBErrCatch(NeuraMorphErr);

  }

  VecFree(&evalInputs);
  NeuraMorphFree(&nm);

  printf("UnitTestNeuraMorphBurryUnitsEvaluate OK\n");

}

void UnitTestNeuraMorph() {

  UnitTestNeuraMorphCreateFree();
  UnitTestNeuraMorphGetSet();
  UnitTestNeuraMorphAddRemoveUnit();
  UnitTestNeuraMorphBurryUnitsEvaluate();
  printf("UnitTestNeuraMorph OK\n");

}

void UnitTestNeuraMorphTrainerCreateFree() {

  GDataSetVecFloat dataset =
    GDataSetVecFloatCreateStaticFromFile("./Datasets/iris.json");
  NeuraMorph* nm =
    NeuraMorphCreate(
      GDSGetNbInputs(&dataset),
      GDSGetNbOutputs(&dataset));
  NeuraMorphTrainer trainer =
    NeuraMorphTrainerCreateStatic(
      nm,
      &dataset);
  bool isSame =
    ISEQUALF(
      trainer.weakUnitThreshold,
      0.9);
  if (
    trainer.neuraMorph != nm ||
    trainer.depth != 2 ||
    trainer.order != 1 ||
    trainer.nbMaxUnitDepth != 2 ||
    trainer.maxLvlDiv != 2 ||
    trainer.nbMaxInputsUnit != GDSGetNbOutputs(&dataset) ||
    isSame != true ||
    trainer.iCatTraining != 0 ||
    trainer.iCatEval != 1 ||
    trainer.dataset != &dataset) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphTrainerCreateStatic failed");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphTrainerFreeStatic(&trainer);
  NeuraMorphFree(&nm);
  GDataSetVecFloatFreeStatic(&dataset);
```

```
  printf("UnitTestNeuraMorphTrainerCreateFree OK\n");

}

void UnitTestNeuraMorphTrainerGetSet() {

  GDataSetVecFloat dataset =
    GDataSetVecFloatCreateStaticFromFile("./Datasets/iris.json");
  NeuraMorph* nm =
    NeuraMorphCreate(
      GDSGetNbInputs(&dataset),
      GDSGetNbOutputs(&dataset));
  NeuraMorphTrainer trainer =
    NeuraMorphTrainerCreateStatic(
      nm,
      &dataset);
  if (NMTrainerGetDepth(&trainer) != 2) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphTrainerGetDepth failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMTrainerGetOrder(&trainer) != 1) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphTrainerGetOrder failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMTrainerGetNbMaxUnitDepth(&trainer) != 2) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphTrainerGetNbMaxUnitDepth failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMTrainerGetMaxLvlDiv(&trainer) != 2) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphTrainerGetMaxLvlDiv failed");
    PBErrCatch(NeuraMorphErr);

  }

  if (NMTrainerGetNbMaxInputsUnit(&trainer) != GDSGetNbOutputs(&dataset)) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphTrainerGetNbMaxInputsUnit failed");
```

```
  PBErrCatch(NeuraMorphErr);

}

if (NMTrainerGetICatTraining(&trainer) != 0) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerGetICatTraining failed");
  PBErrCatch(NeuraMorphErr);

}

if (NMTrainerGetICatEval(&trainer) != 1) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerGetICatEval failed");
  PBErrCatch(NeuraMorphErr);

}

bool isSame =
  ISEQUALF(
    NMTrainerGetWeakThreshold(&trainer),
    0.9);
if (isSame != true) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerGetWeakThreshold failed");
  PBErrCatch(NeuraMorphErr);

}

NMTrainerSetDepth(
  &trainer,
  3);
if (NMTrainerGetDepth(&trainer) != 3) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerSetDepth failed");
  PBErrCatch(NeuraMorphErr);

}

NMTrainerSetNbMaxUnitDepth(
  &trainer,
  3);
if (NMTrainerGetNbMaxUnitDepth(&trainer) != 3) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerSetNbMaxUnitDepth failed");
  PBErrCatch(NeuraMorphErr);
```

```
}

NMTrainerSetOrder(
  &trainer,
  3);
if (NMTrainerGetOrder(&trainer) != 3) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerSetOrder failed");
  PBErrCatch(NeuraMorphErr);

}

NMTrainerSetNbMaxInputsUnit(
  &trainer,
  GDSGetNbOutputs(&dataset) + 1);
if (NMTrainerGetNbMaxInputsUnit(&trainer) != GDSGetNbOutputs(&dataset) + 1) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerSetNbMaxInputsUnit failed");
  PBErrCatch(NeuraMorphErr);

}

NMTrainerSetICatTraining(
  &trainer,
  3);
if (NMTrainerGetICatTraining(&trainer) != 3) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerSetICatTraining failed");
  PBErrCatch(NeuraMorphErr);

}

NMTrainerSetICatEval(
  &trainer,
  4);
if (NMTrainerGetICatEval(&trainer) != 4) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
    "NeuraMorphTrainerSetICatEval failed");
  PBErrCatch(NeuraMorphErr);

}

NMTrainerSetMaxLvlDiv(
  &trainer,
  3);
if (NMTrainerGetMaxLvlDiv(&trainer) != 3) {

  NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
  sprintf(
    NeuraMorphErr->_msg,
```

```
      "NeuraMorphTrainerSetMaxLvlDiv failed");
    PBErrCatch(NeuraMorphErr);

  }

  NMTrainerSetWeakThreshold(
    &trainer,
    0.5);
  isSame =
    ISEQUALF(
      NMTrainerGetWeakThreshold(&trainer),
      0.5);
  if (isSame != true) {

    NeuraMorphErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      NeuraMorphErr->_msg,
      "NeuraMorphTrainerSetWeakThreshold failed");
    PBErrCatch(NeuraMorphErr);

  }

  NeuraMorphTrainerFreeStatic(&trainer);
  NeuraMorphFree(&nm);
  GDataSetVecFloatFreeStatic(&dataset);

  printf("UnitTestNeuraMorphTrainerGetSet OK\n");

}

void UnitTestNeuraMorphTrainerRun() {

  GDataSetVecFloat dataset =
    GDataSetVecFloatCreateStaticFromFile("./Datasets/iris.json");
  GDSShuffle(&dataset);
  VecShort2D split = VecShortCreateStatic2D();
  VecSet(
    &split,
    0,
    130);
  VecSet(
    &split,
    1,
    20);
  GDSSplit(
    &dataset,
    (VecShort*)&split);
  NeuraMorph* nm =
    NeuraMorphCreate(
      GDSGetNbInputs(&dataset),
      GDSGetNbOutputs(&dataset));
  NMSetFlagOneHot(
    nm,
    true);
  NeuraMorphTrainer trainer =
    NeuraMorphTrainerCreateStatic(
      nm,
      &dataset);

  NMTrainerSetWeakThreshold(
    &trainer,
    0.99);
```

```
  NMTrainerSetDepth(
    &trainer,
    3);
  NMTrainerSetMaxLvlDiv(
    &trainer,
    1);
  NMTrainerSetNbMaxInputsUnit(
    &trainer,
    GDSGetNbInputs(&dataset));
  NMTrainerSetOrder(
    &trainer,
    1);
  NMTrainerRun(&trainer);
  NMTrainerEval(&trainer);
  printf("Bias (min/avg/max): ");
  VecPrint(
    NMTrainerResEval(&trainer),
    stdout);
  printf("\n");

  NeuraMorphTrainerFreeStatic(&trainer);
  NeuraMorphFree(&nm);
  GDataSetVecFloatFreeStatic(&dataset);

  printf("UnitTestNeuraMorphTrainerRun OK\n");

}

void UnitTestNeuraMorphTrainer() {

  UnitTestNeuraMorphTrainerCreateFree();
  UnitTestNeuraMorphTrainerGetSet();
  UnitTestNeuraMorphTrainerRun();
  printf("UnitTestNeuraMorphTrainer OK\n");

}

void UnitTestAll() {

  UnitTestNeuraMorphUnit();
  UnitTestNeuraMorph();
  UnitTestNeuraMorphTrainer();
  printf("UnitTestAll OK\n");

}

int main() {

  UnitTestAll();

  // Return success code
  return 0;

}
```

6 Unit tests output

```
UnitTestNeuraMorphUnitCreateFree OK
<0,0,0> -> <0,0> (0.500000)
```

```
UnitTestNeuraMorphUnitGetSetPrint OK
UnitTestNeuraMorphUnitEvaluate OK
UnitTestNeuraMorphUnit OK
UnitTestNeuraMorphCreateFree OK
UnitTestNeuraMorphGetSet OK
UnitTestNeuraMorphAddRemoveUnit OK
UnitTestNeuraMorphBurryUnitsEvaluate OK
UnitTestNeuraMorph OK
UnitTestNeuraMorphTrainerCreateFree OK
UnitTestNeuraMorphTrainerGetSet OK
Depth 1/2...
Nb available inputs: 4
Train units with 0001 inputs
Train units with 0002 inputs
Train units with 0003 inputs
Train units with 0004 inputs
Burry 2 out of 15 units
<0,1> -> <0,1,2> (1957747840.000000)
<0,1,2,3> -> <0,1,2> (2044897792.000000)
Depth 2/2...
Nb available inputs: 10
Train units with 0001 inputs
Train units with 0002 inputs
Train units with 0003 inputs
Train units with 0004 inputs
Train units with 0005 inputs
Train units with 0006 inputs
Train units with 0007 inputs
Train units with 0008 inputs
Train units with 0009 inputs
Train units with 0010 inputs
Add the last unit
<4,7,9> -> <6,7,8> (2147469824.000000)
UnitTestNeuraMorphTrainerRun OK
UnitTestNeuraMorphTrainer OK
UnitTestAll OK
```